Noname manuscript No.
(will be inserted by the editor)

A URI-Based Approach for Addressing Fragments of Media Resources on the Web

Erik Mannens · Davy Van Deursen · Raphaël Troncy · Silvia Pfeiffer · Conrad Parker · Yves Lafon · Jack Jansen · Michael Hausenblas · Rik Van de Walle

Received: date / Accepted: date

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

D. Van Deursen
E-mail: davy.vandeursen@ugent.be Ghent University - IBBT
ELIS - Multimedia Lab
Ghent, Belgium
E-mail: davy.vandeursen@ugent.be

R. Troncy
EURECOM, Multimedia Communications Department
Sophia Antipolis, France
E-mail: raphael.troncy@eurecom.fr

S. Pfeiffer
Vquence
Sydney, Australia
E-mail: silviapfeiffer1@gmail.com

C. Parker
Kyoto University
Kyoto, Japan
E-mail: conrad@metadecks.org

Y. Lafon
W3C/ERCIM
Sophia Antipolis, France
E-mail: ylafon@w3.org

J. Jansen
CWI, Distributed Multimedia Languages and Infrastructures
Amsterdam, Netherlands
E-mail: Jack.Jansen@cwi.nl

M. Hausenblas
National University of Ireland, Digital Enterprise Research Institute - LiDRC
Galway, Ireland
E-mail: michael.hausenblas@deri.org

R. Van de Walle
E-mail: rik.vandewalle@ugent.be Ghent University - IBBT

Abstract To make media resources a prime citizen on the Web, we have to go beyond simply replicating digital media files. The Web is based on hyperlinks between Web resources, and that includes hyperlinking out of resources (e.g. from a word or an image within a Web page) as well as hyperlinking into resources (e.g. fragment URIs into Web pages). To turn video and audio into hypervideo and hyperaudio, we need to enable hyperlinking into and out of them. The W3C Media Fragments Working Group is taking on the challenge to further embrace W3C's mission to lead the World Wide Web to its full potential by developing a Media Fragment protocol and guidelines that ensure the long-term growth of the Web. The major contribution of this paper is the introduction of Media Fragments as a media-format independent, standard means of addressing media resources using URIs. Moreover, we explain how the HTTP protocol can be used and extended to serve Media Fragments and what the impact is for current Web-enabled media formats.

Keywords Media Fragments · W3C Standardization · HTML5

1 Introduction

Video clips on the World Wide Web (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 clips. The HTML5 specification is a game changer and all of the major browser vendors have committed to support the newly introduced `<video>` and `<audio>` elements [7][1]. However, in order to make video clips accessible in a transparent way, 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 video resource, and we should be able to easily show only specific regions-of-interest within this video resource. And last but not least, if we want to browse or scan several video resources based on (encapsulated) semantics, we should be able to master the full complexity of rich media by also enabling standardised media annotation [21,28]. Note that we can generalize the above observations to other media, such as audio resources. This way, media resources truly become "first-class citizens" on the Web.

The mission of the W3C Media Fragments Working Group (MFWG) [30], which is part of W3C's Video in the Web activity[2], is to provide a mechanism to address media fragments on the Web using Uniform Resource Identifiers (URIs) [6,9]. The objective of the proposed specification is to improve the support for the addressing and retrieval of sub-parts of so-called media resources (e.g. audio, video and image), as well as the automated processing of such sub-parts for reuse within the current and future Web infrastructure [19]. Example use cases are the bookmarking or sharing of excerpts of video clips with friends in social networks, the automated creation of fragment URIs

ELIS - Multimedia Lab
Ghent, Belgium
E-mail: rik.vandewalle@ugent.be

[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://www.w3.org/2008/WebVideo/Activity.html`

in search engine interfaces by having selective previews, or the annotation of media fragments when tagging audio and video spatially and/or temporally [18]. The examples given throughout this paper to explain the Media Fragments URI specification are based on the following two scenarios. In scenario (a), 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 that he himself performed in last Saturday's game. In scenario (b), Sarah –a video artist by profession– quickly previews the video footage in search of a particular high quality 10 seconds sub-clip to finalise editing her new video clip 'The Editors'.

In this paper, we present the rationale for a Media Fragments specification in Section 2. In Section 3, we outline the boundaries and semantics of a Media Fragment URI and show how the syntax should look like, whereas Section 4 elaborates on how a media fragment specified as a URI fragment can be resolved stepwise using the HTTP protocol [8]. We then identify the influence of the current media formats on fragment extraction in Section 5. Finally, we outline open issues and future work in Section 6 and give our conclusions in Section 7.

2 Related Work

Before video can become a "first-class citizen" on the Web, one urgently needs to provide standardised ways to localise the temporal, spatial, and track sub-parts of audio-visual media content [6,28]. Previous efforts to do so include both URI-based and non-URI based mechanisms.

In the non-URI based class of solutions, the SMIL specification over HTTP allows to play only a temporal fragment of the video by using the *clipBegin* and *clipEnd* attributes [1]. However, current implementations have to first get the complete media resource and then cut it up locally, which entails a terrible waste of bandwidth in case of large video resources. Using MPEG-7, a video is divided into *VideoSegments* that can be described by a *MediaTimePoint* and *MediaDuration* corresponding to the starting time and media duration respectively [10]. Using TV-Anytime, temporal intervals can be also defined, accessed, and manipulated through *segments* within an audio/video stream using MPEG-7 types for specifying the temporal boundaries [3]. For images, one can use either MPEG-7 or an SVG code snippet to define the bounding box coordinates of specific regions [4]. However, this approach implies an extra *indirection* since a semantic description of this region will actually be *about* a piece of an XML document just defining a multimedia fragment and not the fragment itself. As such, the identification and the description of the temporal fragment or region is intertwined (this use of indirection) and one needs to first parse and understand the metadata model in order to get access to the media fragment afterwards, which is not desirable in all circumstances. Finally, HTML ImageMaps can also define spatial regions (rectangles, circles, or polygons) via the `<area>` and `<a>` elements. However, the complete media resource has first to be downloaded to the user agent too [23].

As for the URI-based mechanisms, SVG has a spatial URI mechanism using the `#` that specifies the region of an SVG image to be viewed, but having the same limitations as SMIL. The temporalURI draft specification defines a temporal fragment of multimedia resources using the query parameter `?`, thus creating a new resource which is not desirable as fragments should still have a direct link to their "parent" resource [22]. An in-depth analysis of `#` versus `?` is discussed in Section 3.3. MPEG-21, on the other hand, specifies a normative syntax to be used in URIs for addressing parts of any resource using the #, but the supported media types are restricted to the MPEG formats only [15]. Furthermore, this specification has a very complex syntax which can be ambiguous. Four schemes – ffp(), offset(), mp(), and mask() – are defined and e.g. both mp() and ffp() can be used for example to identify tracks. Since our rationale is to find a scheme that is easy enough to get implemented and have a real impact on the web, our impression is that MPEG-21 Part 17 was over-designed even for the most simple use cases, thus preventing its adoption, as there are no real-world applications implementing this specification since it was launched in 2006. YouTube released a tool[3] to link to particular time points in videos and to annotate parts of those videos spatio-temporally. It uses the URI fragment marker #, but the entire resource must still be downloaded by the client in order to enable seeking in the media file afterwards. In contrast, the solution advocated by the MFWG is to only send the bytes corresponding to the media fragments requested and still be able to cache them. Finally, for fragment-aware protocols such as RTSP, we observe that this behavior is possible. While RTSP doesn't provide a URI scheme to address media fragments, it does allow temporal range requests in the PLAY method, track requests in the SETUP method, and mappings of named dimensions through the SDP service. With the media fragment URI specification, we want to be compatible with solutions already widely deployed as much as possible and enable for HTTP what is already possible with RTSP.

3 Media Fragments URIs

3.1 Media Resource Model

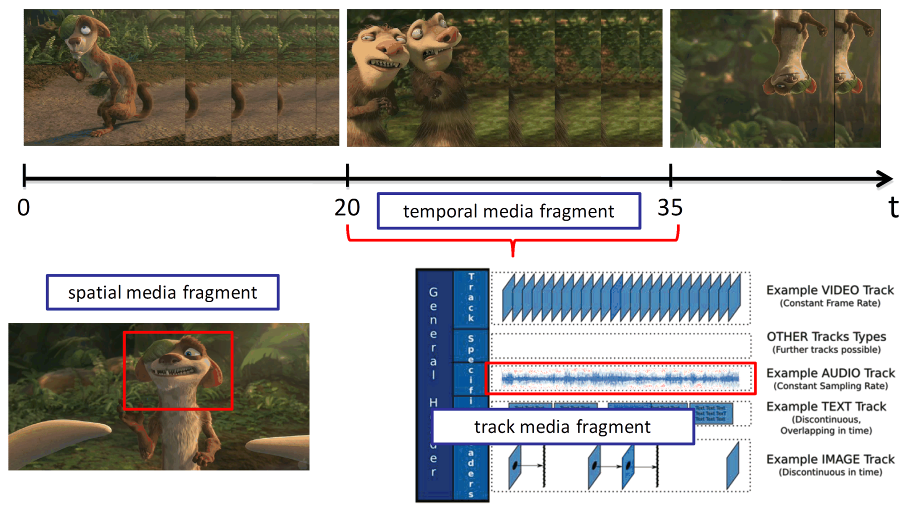
We assume that media fragments are defined for "time-linear" media resources, which are characterised by a single timeline (see also Fig. 1). Such media resources usually include multiple tracks of data all parallel along this uniform timeline. These tracks can contain video, audio, text, images, or 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.

3.2 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 fragment URIs.

- *Independent of media formats.* The media fragments URI specification needs to be independent of underlying media formats such as MP4 [11], Ogg [20] or WebM [27].
- *Fragment axes.* Media fragments need to be accessed along three different axes: temporal, spatial, and track. Additionally, media fragments could be identified through names.

[3] `http://www.youtube.com/t/annotations_about`



0
20
temporal media fragment
35
t
spatial media fragment
General Header
Track specific headers
Example VIDEO Track (Constant Frame Rate)
OTHER Tracks Types (Further tracks possible)
Example AUDIO Track (Constant Sampling Rate)
Example TEXT Track (Discontinuous, Overlapping in time)
track media fragment
Example IMAGE Track (Discontinuous in time)

Fig. 1 Example of media fragments and media resource model

- *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 the fragment is 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 [24].
- *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 not as media fragments.

3.3 URI Fragments vs. URI Queries

According to the URI specification [9], URI fragment markers # point at so-called “secondary” resources. Per definition, such a secondary resource may be “some portion or subset of the primary resource, some view on representations of the primary resource, or some other resource defined or described by those representations”. As such, it makes a viable syntax for Media Fragment URIs. A further consequence of the use of URI fragments is that the media type of the retrieved fragment should be the same as the media type of the primary resource, which means that a URI fragment pointing to a single video frame within a video results in a one-frame video, and not in a still image.

To make these URI fragment addressing methods work, only byte-identical segments of a media resource can be considered, as we assume a simple mapping between the media fragment and its elementary byte-aligned building blocks. Where byte-identity cannot be maintained and thus a form of transcoding of the resource is needed, the user agent is not able to resolve the fragmentation by itself and an extra server interaction is required. In this case, URI queries have to be used as they result in a server interaction to deliver a newly created transcoded resource.

The main difference between a URI query and a URI fragment is indeed that a URI query creates a completely new resource having no relationship whatsoever with the resource it is created from, while a URI fragment delivers a secondary resource that relates to the primary resource. As a consequence, URI query created resources cannot be mapped byte-identical to their parent resource (this notion does not even exist), and are thus considered a re-encoded segment. As of the aforementioned requirements *Context awareness* and *Fragment by extraction* described in Section 3.2, the use of URI fragments is preferable over the use of URI queries since byte range requests are an inherent part of HTTP including the caching proxy infrastructure, while providing a query mechanism requires extending the server software. We discuss how to resolve media fragments using the `#` in Section 4.

In the case of playlists composed of media fragment resources, the use of URI queries (receiving a completely new resource instead of just byte segments from existing resources) could be desirable since it does not have to deal with the inconvenience of the original primary resources - its larger file headers, its longer duration, and its automatic access to the original primary resources. On top of that, URI queries have to take care of an extra caveat of creating a fully valid new resource. This implies typically a reconstruction of the media header to accurately describe the new resource, possibly applying a non-zero start-time or using a different encoding parameter set. Such a resource will then be cached in web proxies as a different resource to the original primary resource.

3.4 Fragment Dimensions

3.4.1 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 SMPTE timecodes [24,26]. The time format is specified by name, followed by a colon, with `npt:` being the default. Some examples are:

```
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[
```

3.4.2 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 examples are:

```
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%
```

3.4.3 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 examples are:

```
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'
```

3.4.4 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). 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.

```
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'
```

3.4.5 Combined Dimensions

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, following fragments should be byte-identical:

```
http://example.com/video.ogv#t=10,20&track=vid&xywh=pixel:0,0,320,240
http://example.com/video.ogv#track=vid&xywh=0,0,320,240&t=npt:10,20
http://example.com/video.ogv#xywh=0,0,320,240&t=smpte:0:00:10,0:00:20&track=vid
```

4 Resolving Media Fragments with HTTP

We described in the previous section the Media Fragment URI syntax. We present in this section how a Media Fragment URI should be processed using the HTTP protocol. We foresee that the logic of the processing of 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 [9].
- 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 and track dimensions.

4.1 User Agent mapped Byte Ranges

As stated in Section 1 (scenario (a)), 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 fragment URI:

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

Since Steve does not want to blow 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.

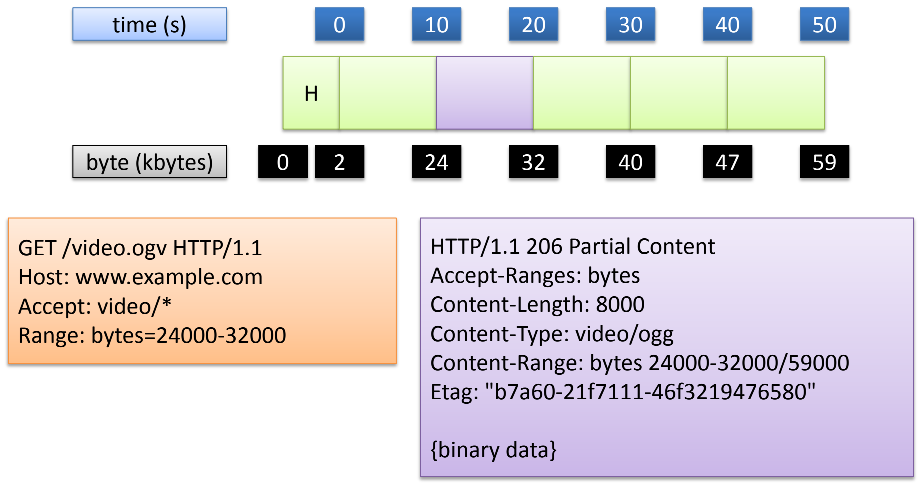
We assume that Steve's smart phone runs a smart user agent (e.g. an HTML5 compliant browser) that can resolve the temporal fragment identifier of the media fragment URI through an HTTP byte range request. A click on this URI triggers the following chain of events:

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 assume use of a user agent that knows how time is mapped to byte offsets for this particular Ogg media format[4]. 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 the 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

[4] This is possible by using Ogg Index, `http://wiki.xiph.org/Ogg_Index`.

the *HTTP Range request* header (see Fig. 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.



```
GET /video.ogv HTTP/1.1
Host: www.example.com
Accept: video/*
Range: bytes=24000-32000
```

```
HTTP/1.1 206 Partial Content
Accept-Ranges: bytes
Content-Length: 8000
Content-Type: video/ogg
Content-Range: bytes 24000-32000/59000
Etag: "b7a60-21f7111-46f3219476580"

{binary data}
```

Fig. 2 User Agent mapped Byte Ranges

In this scenario, media fragments can be served by traditional HTTP web servers that support byte range requests. However, a smart user agent is necessary to parse the media fragment URI syntax[5]. Furthermore, it requires knowledge about the syntax and semantics of various media codecs formats. 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 [19]. It is expected that this recipe will be supported for this kind of media fragment URI and the temporal dimension with the HTML5 media elements.

4.2 Server mapped Byte Ranges

We assume now that the user agent is able to parse and understand the media fragment 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.

[5] The Opera browser has announced that the next release will support this recipe, see `http://www.foms-workshop.org/foms2010OVC/uploads/philip_jagenstedt_opera.pdf`

4.2.1 Server mapped Byte Ranges including Header Information

In a first case, the server has to help the client to setup the initial decoding pipeline (i.e. there is no header information from the resource on client side yet). When Steve starts to play this “infamous” 10 seconds of video, the following events occur:

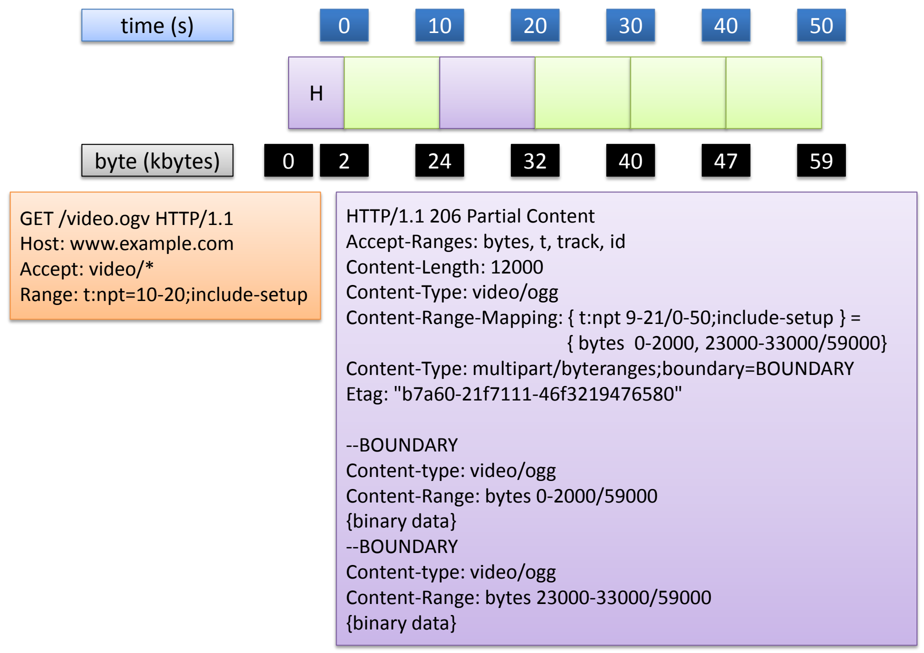
1. The user agent parses the media fragment identifier and creates an HTTP Range request expressed in a different unit than bytes, e.g. a time unit expressed in seconds. Furthermore, it extends this header with the keyword `include-setup` (see Fig. 3) in order to let the server know that it also requires 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 then wraps both the header information and the bytes requested in a multi-part HTTP 206 Partial Content response. As depicted in Fig. 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 response 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 (and not in bytes). The decoder might need extra data, before the beginning (hence npt 9), and/or after the end (hence npt 21) of the requested sequence (e.g., npt 10-20), since this initial requested 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 `/`, thus providing context information regarding the parent resource in case the HTTP Range request contained a temporal dimension. More specifically, the header contains the start and end time of the parent resource, so the user agent is able to understand and visualise the temporal context of the media fragment.
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.

A plugin named *Media Fragments 1.0* developed for the Firefox browser has successfully implemented this recipe and is now available in the Mozilla add-ons library.

4.2.2 Server mapped Byte Ranges with Initialized Client

On the other hand, if the server can assumes that the client already initiated the decoding pipeline, i.e., it knows about the header of the requested media resource, then the following events occur:

1. The user agent parses the media fragment identifier and creates a HTTP Range request expressed in a different unit than bytes, e.g. a time unit expressed in Normal Playing Time (npt) seconds (Fig. 4).
2. The server, which understands this time unit, resolves the HTTP Range request and performs the mapping between the media fragment time unit and byte ranges.



```
GET /video.ogv HTTP/1.1
Host: www.example.com
Accept: video/*
Range: t:npt=10-20;include-setup
```

```
HTTP/1.1 206 Partial Content
Accept-Ranges: bytes, t, track, id
Content-Length: 12000
Content-Type: video/ogg
Content-Range-Mapping: { t:npt 9-21/0-50;include-setup } =
                        { bytes  0-2000, 23000-33000/59000}
Content-Type: multipart/byteranges;boundary=BOUNDARY
Etag: "b7a60-21f7111-46f3219476580"

--BOUNDARY
Content-type: video/ogg
Content-Range: bytes 0-2000/59000
{binary data}
--BOUNDARY
Content-type: video/ogg
Content-Range: bytes 23000-33000/59000
{binary data}
```

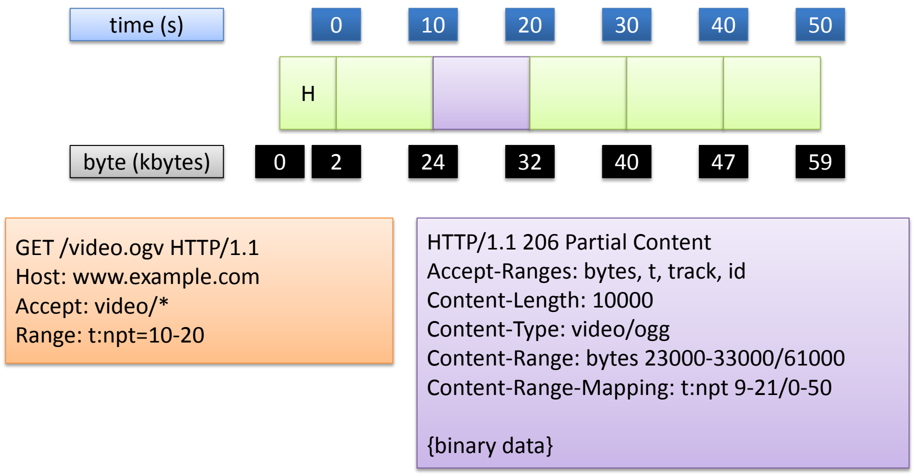
Fig. 3 Server mapped Byte Ranges including Header Information

3. Once the mapping is done, the server extracts the proper bytes and wraps them within a HTTP 206 Partial Content response. As displayed in Fig. 4, the response contains the additional `Content-Range-Mapping` header describing the content range in terms of time (ending with the total duration of the complete resource). Where multiple byte ranges are required to satisfy the HTTP Range request, these are transmitted as a multipart message body (with media type '`multipart/byteranges`'), as can be seen in Fig. 2.
4. Finally, the user agent receives these byte ranges and is able to decode and start playing back the media fragment requested.

If the server does not understand the HTTP Range request – which means it does not support media fragments –, it must ignore that header field – per the current HTTP specification – and deliver the complete resource. This also means we can combine both byte range and fragment range headers in one request, since the server will only react to the HTTP Range header it understands.

4.3 Proxy cacheable Server mapped Byte Ranges

To prevent wasting unnecessary bandwidth and to maximize throughput speeds, the current internet architecture heavily relies on caching web proxies. In the case of media fragment URIs that are sometimes resolved by servers as described in the previous section, existing web proxies have no means of caching these requests as they only understand byte ranges.



GET /video.ogv HTTP/1.1
Host: www.example.com
Accept: video/*
Range: t:npt=10-20

HTTP/1.1 206 Partial Content
Accept-Ranges: bytes, t, track, id
Content-Length: 10000
Content-Type: video/ogg
Content-Range: bytes 23000-33000/61000
Content-Range-Mapping: t:npt 9-21/0-50

{binary data}

Fig. 4 Server mapped Byte Ranges with initialized Client

In order to make use of the web proxies, the user agent should therefore only ask for byte ranges. To be able to do so, we foresee an extra communication between the server and the user agent. In a first step, the server will just redirect the original request giving extra hints of the byte ranges to request corresponding to the media fragment instead of replying with the actual data. In a second step, the user agent will re-issue another range request, this time expressed in bytes which will therefore be cacheable for current web proxies.

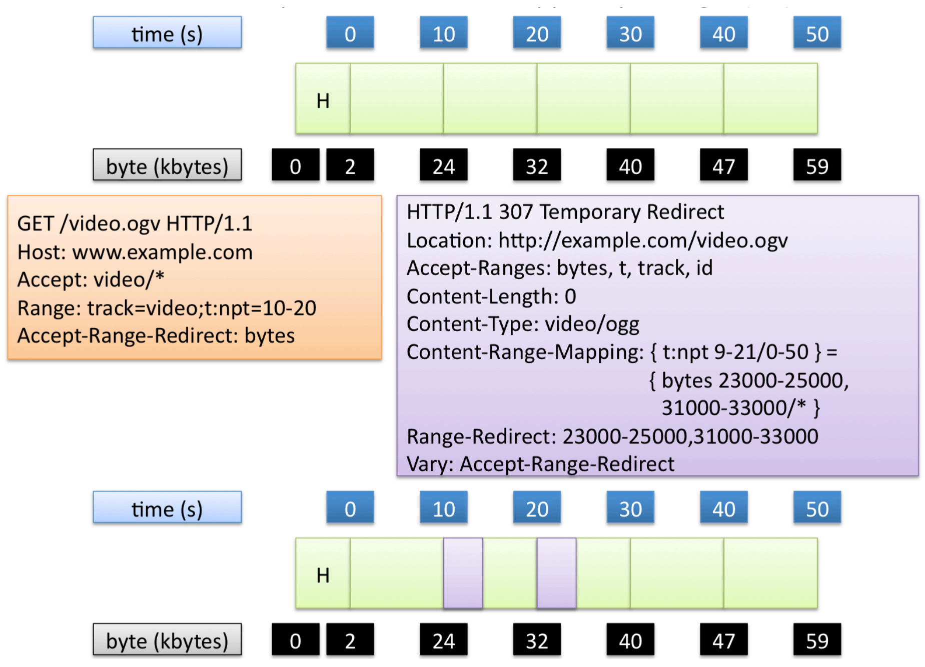
In the scenario (b) described in the Section 1, Sarah wants to finalise the new music clip of “The Editors”. Therefore, she scrolls through the video footage she has gathered in order to find the appropriate final scene. Since she is only interested in finding the right frames to start editing the high quality footage, she would like to watch only 10 seconds of the video track in low-resolution of those gathered clips, which can be translated into the following media fragment URI:

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

The chain of events for getting 10 seconds of the video track of this footage is as follows:

1. The user agent parses the media fragment identifier and creates a HTTP Range request expressing the need for 10 seconds of the video track and further requests to retrieve just the mapping to byte ranges, hence the extra `Accept-Range-Redirect` HTTP header (Fig. 5), which signals to the server that only a redirect to the correct byte ranges is necessary and the result should be delivered in the HTTP Range-Redirect header.
2. The server resolves the HTTP Range request, knows it only has to look for the correct byte ranges of the video track, and performs the mapping between the media fragment time unit and byte ranges.
3. Afterwards the server sends back an empty HTTP 307 Temporary Redirect that refers the user agent to the right byte range(s) for the previous requested correct time range. Note that for the sake of simplicity only 2 byte ranges are sent back, whereas interleaving normally means a lot more data chunks for one track.

4. After this initial indirection, the steps to follow by the user agent are the same as those we discussed earlier in Section 4.1.



```
GET /video.ogv HTTP/1.1
Host: www.example.com
Accept: video/*
Range: track=video;t:npt=10-20
Accept-Range-Redirect: bytes
```

```
HTTP/1.1 307 Temporary Redirect
Location: http://example.com/video.ogv
Accept-Ranges: bytes, t, track, id
Content-Length: 0
Content-Type: video/ogg
Content-Range-Mapping: { t:npt 9-21/0-50 } =
                           { bytes 23000-25000,
                            31000-33000/* }
Range-Redirect: 23000-25000,31000-33000
Vary: Accept-Range-Redirect
```

Fig. 5 Proxy cacheable Server mapped Byte Ranges

More advanced server side examples, e.g., server triggered redirects, can again be found in the Media Fragments specification [19].

5 Influence of Media Formats on Fragment Extraction

The extraction of media fragments is dependent on the underlying media format. Moreover, media formats introduce restrictions regarding possibilities to extract media fragments along different axes. For instance, very few coding formats support the extraction of spatial fragments without the need to re-encode the media resource. As such, depending on the fragment axis, there are different requirements regarding the extraction of media fragments (i.e., obtaining media fragments without transcoding operations).

– *Temporal*: random access points need to occur in a regular way. Random access refers to the ability of a decoder to start decoding at a point in a media resource other than at the beginning and to recover an exact or approximate representation of the decoded resource [5]. In case of video, random access points typically correspond to intra-coded pictures.
– *Spatial*: independently coded spatial regions are required. More specifically, (mostly rectangular) regions of the picture need to be coded independently from each other.

Typically, Regions Of Interest (ROIs) and a background are coded independently, which results in the possibility to extract these ROIs. Another, more advanced approach is "interactive" ROI. Interactive ROI extraction can be used in applications in which the ROI cannot be defined during the encoding phase of the video sequence [2]. In order to support interactive (rectangular) ROI scalability, a video frame has to be divided into different tiles that can be selected on an individual basis. Each tile has to be coded such that the tiles can be decoded independently of other tiles. This way, the tiles belonging to a certain ROI can be selected on-the-fly during the extraction process. Scalable Video Coding [25] (SVC) is an example of a video coding format supporting interactive ROI coding.

– *Track*: the way tracks are extracted from a media resource is dependent on the container format. In contrast to temporal and spatial fragment extraction, tracks are not "encoded" within a container format and can thus always be extracted without low-level transcoding operations. Based on the headers of the container format, it is possible to locate the proper chunks corresponding to the desired track.
– *Named*: whether named fragments are supported or not depends again on the container format. Typically, named fragments are supported through container formats by using other metadata formats describing these named fragments. An instance of such a metadata format can, for example, correspond to a separate track or can be included in the headers of the container format. Examples of such metadata formats are MPEG-4 TimedText [14] and Rich Open multitrack media Exposition[6] (ROE).

Hereafter, we provide more information regarding the extraction of media fragments applied to three popular media format combinations:

– H.264/AVC-encoded video [16] and AAC-encoded audio [13] packed in an MP4 container;
– Theora-encoded video [31] and Vorbis-encoded audio [32] packed in an Ogg container.
– VP8-encoded video [?] and Vorbis-encoded audio packed in a WebM container.

For each combination, we discuss how media fragments along different axes can be extracted, based on the above defined requirements.

5.1 H.264/AVC and AAC in MP4

MPEG-4 Part 14 or MP4 file format, formally known as ISO/IEC 14496-14:2003 [11], is a multimedia container format standard specified as a part of MPEG-4. It is able to store digital video and digital audio streams as well as other data such as subtitles and still images. The MP4 file format is built on top of the ISO Base Media File Format [12] (known as MPEG-4 part 12), which is in turn based on Apple's QuickTime file format[7]. H.264/AVC [16] is the latest block-oriented motion-compensation-based codec standard developed by the Joint Video Team[8] (JVT) and is used in applications such as Blu-ray Disc, YouTube and the iTunes Store. Advanced Audio Coding (AAC) is a standardised,

[6] `http://wiki.xiph.org/ROE`
[7] `http://www.quicktime.com`
[8] `http://www.itu.int/ITU-T/studygroups/com16/jvt/`

lossy compression and encoding scheme for digital audio and was designed to be the successor of the popular MP3 format[9]. AAC has been standardised by ISO and IEC, as part of the MPEG-2[10] and MPEG-4[11] specifications and is the standard audio format for Apple's iPhone, Sony's PlayStation 3, Android based phones, etc.

Temporal fragment extraction is both supported by H.264/AVC and AAC. For H.264/AVC, random access points correspond to Instantaneous Decoding Refresh (IDR) frames, which are intra-coded frames that are not referenced by any frames outside the current GOP. Thus, a temporal H.264/AVC fragment always starts with an IDR frame. An AAC audio stream consists of a stream of audio frames, each containing typically 1024 audio samples. Each audio frame corresponds to a random access point, resulting in a fine temporal granularity. The MP4 format provides support for headers containing information regarding random access points of the contained media streams. Hence, by interpreting the MP4 header, temporal fragments can be extracted on condition that random video access points occur in a regular way.

Spatial fragments extraction is only supported by H.264/AVC in a limited way. More specifically, ROI coding can be realized in H.264/AVC using the Flexible Macroblock Ordering (FMO) tool. By using H.264/AVC FMO type 2, rectangular regions can be coded independently from each other (i.e., each region corresponds to a slice group). Extracting a particular ROI corresponds to the detection of coded P and B slices located in non-ROI slice groups and the substitution of these slices with placeholder slices [17].

Tracks are described in the headers of the MP4 format. More specifically, the track box contains general information about the track (e.g., name, creation time, width, height, etc.) as well as references to the location of the bytes representing the track (through the sample table box). Hence, by interpreting these boxes, track fragments can be extracted, based on the names provided by the MP4 container.

Named fragments for MP4 can, for instance, be obtained using MPEG-4 Timed Text, which is the text-based subtitle format for MPEG-4. It was developed in response to the need for a generic method for coding of text as one of the multimedia components within audiovisual presentations. As such, it can be used to give names to temporal fragments.

5.2 Theora and Vorbis in Ogg

The Ogg container format is a multimedia container format and the native file and stream format for the Xiph.org multimedia codecs [20]. It is an open format, free for anyone to use, and is able to encapsulate compressed audio and video streams. Ogg is a stream-oriented container, meaning it can be written and read in one pass, making it a natural fit for internet streaming and use in processing pipelines. Note that this stream orientation is the major design difference over other file-based container formats such as MP4. Theora is a video codec whose bitstream format was frozen in July 2004 by Xiph.org. The standard codec is frequently found on the Web and used by large sites complying with the Wikimedia Commons like Wikipedia[12]. Vorbis, also developed by

[9] `http://www.iis.fraunhofer.de/EN/bf/amm/products/mp3/index.jsp`
[10] `http://mpeg.chiariglione.org/standards/mpeg-2/mpeg-2.htm`
[11] `http://mpeg.chiariglione.org/standards/mpeg-4/mpeg-4.htm`
[12] `http://www.wikipedia.org`

the Xiph.Org Foundation, is an audio format specification for lossy audio compression and has proven popular among supporters of free software. Both Vorbis and Theora contain solely royalty-free and open technology.

Similar to H.264/AVC, random access points in Theora streams correspond to I-frames. Random access in Vorbis streams is obtained through audio frames, similar to AAC (note that the number of samples is variable in case of Vorbis). Thus, temporal fragment extraction is both supported by Theora and Vorbis.

Spatial fragment extraction is not supported by Theora. More specifically, it does not provide support to encode spatial regions independent of each other.

An Ogg container is a contiguous stream of sequential Ogg pages. Ogg allows for separate tracks to be mixed at page granularity in an Ogg container. Tracks are identified by a unique serial number, which is located in each Ogg page header. Further, to link Ogg tracks to track names, again the ROE format can be used. ROE is a metadata format for describing the relationships between tracks of media in a stream. ROE descriptions are represented in an Ogg container by means of an Ogg Skeleton track. This way, track fragment extraction is supported by Ogg though fields containing the track names stored in Ogg Skeleton.

Named fragment extraction for temporal and track fragments in Ogg can be obtained by combining ROE, Skeleton, and Continuous Media Markup Language[13] (CMML). CMML allows a time-continuously sampled data file to be structured by dividing it into temporal sections (so-called clips) and provides these clips with some additional information. Thus, named fragment extraction can be done based on the CMML and ROE description of an Ogg container.

5.3 VP8 and Vorbis in WebM

The WebM container format, which is introduced by Google, is an open and royalty-free multimedia container format designed for the Web. WebM is based on the Matroska media container format. It uses Vorbis for the representation of audio streams while VP8 is used as video codec. VP8 [?], released by Google and originally created by On2 Technologies, is the latest open source video codec. Compression-wise, VP8 generally performs better than Theora.

The constraints implied by WebM for extracting temporal media fragments are very similar to the ones we discussed for Theora and H.264/AVC. More specifically, random access points in VP8 streams correspond to I-frames. Temporal fragment extraction is thus supported by VP8. Also, similar to Theora, spatial fragment extraction is not supported by VP8.

Tracks within WebM are identified by a track number and/or a track identifier. The track identifier can be used within a Media Fragment URI using the track axis, while the track number is used internally to indicate for each frame (audio or video) to which track it belongs. This way, we can address a track in a WebM container and also extract the track. Further, named fragments within WebM containers can be obtained by defining chapters. A chapter in WebM is a combination of a time segment and one or more tracks. The name of the chapter could then be used as an identifier for the named fragment. Note that WebM also provides support for adding tags to chapters. However, these tags cannot always be used as identifier for a named fragment because

[13] `http://wiki.xiph.org/CMML`

they could result in multiple time segments (e.g., two chapters having the same tag). Addressing multiple time segments within one Media Fragment URI is not supported, as discussed in Section 3.4.1.

6 Open Issues and Future Work

Currently, it is unclear for all the 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. Others have suggested that spatial media fragments should result in cropping the images. Finally, track selection could be done via dropdown boxes or buttons. Whether the media fragment URIs should be hidden from the end-user or not is an application implementation issue.

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 tracks. Another candidate to use is the *Media Multitrack API*[14] from the HTML5 Working Group. This is a JavaScript API for HTML5 media elements that allows content authors to determine which tracks are 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), the role this data plays (e.g., audio description, caption, sign language), and the actual language (e.g., RFC3066[15] language code).

With such media fragments addressing schemes available, there is still a need to hook up the addressing with 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 such as QuickTime, Flash[16], 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. On 6 May 2010, the WHATWG version of the HTML5 draft specification introduced the WebSRT format (Web Subtitle Resource Tracks), a format intended for marking up timed track resources. WebSRT will have a means of addressing segments of a media resource by name and also a means to include text with hyperlinks, though the exact scope of WebSRT and its inclusion as part of HTML5 is still under debate.

[14] `http://www.w3.org/WAI/PF/HTML/wiki/Media_MultitrackAPI`
[15] `http://www.ietf.org/rfc/rfc3066.txt`
[16] `http://www.flash.com`

7 Conclusion

In this paper, we presented the rationale for a Media Fragment URIs specification to make video a "first-class citizen" on the web and pinpointed the drawbacks/inconsistencies of existing solutions. We outlined the boundaries, requirements and semantics of a Media Fragment URI. We clearly defined the syntax of Media Fragments over the different possible fragment dimensions (i.e., temporal, spatial, track, and named) and showed how the defined Media Fragments can be resolved stepwise using the HTTP protocol. We then identified the influence of current media formats on such Media Fragments extraction.

We already have a HTTP implementation for the W3C Media Fragments 1.0 specification [29] in order to verify all the test cases defined by the working group. Furthermore, several browser vendors are on their way to implementing a HTTP implementation themselves. In the near future, we also foresee reference implementations for the RTSP, RTMP[17], and File protocols.

In the meantime this specification really opens up time-related media to the internet crowd and makes time-related annotation [19,18] feasible for hyperlinking into the media, thus providing the necessary support for this already omnipresent "third dimension" into the internet.

Acknowledgements The authors would like to thank the other W3C Media Fragments' participants: Eric Carlson (Apple, Inc.), Thierry Michel (W3C/ERCIM), Guillaume Olivrin (Meraka Institute, South Africa), Soohong Daniel Park (Samsung Electronics Co., Ltd.), David Singer (Apple, Inc.), Philip Jägenstedt (Opera Software) for their willingness to discuss the definition of media fragments and more generally their adequacy within the Web architecture. The research activities that have been described in this paper were partially funded by W3C/ERCIM, Ghent University, Interdisciplinary Institute for Broadband Technology (IBBT), the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT), the Fund for Scientific Research-Flanders (FWO-Flanders), the French Ministry of Industry (Innovative Web call) under contract 09.2.93.0966, EURECOM, CWI, Kyoto University, National University of Ireland, Mozilla Corporation, and the European Union within the 7th framework program (FP7/2007-2013) under agreement ICT-2007-214793.

References

1. Bulterman, D., Grassel, G., Jansen, J., Koivisto, A., Layaïda, N., Michel, T., Mullender, S., Zucker, D. (eds.): Synchronized Multimedia Integration Language (SMIL 2.1). W3C Recommendation. World Wide Web Consortium (2005). Available at `http://www.w3.org/TR/SMIL2/`
2. De Schrijver, D., De Neve, W., Van Deursen, D., De Bruyne, S., Van de Walle, R.: Exploitation of Interactive Region of Interest Scalability in Scalable Video Coding by Using an XML-driven Adaptation Framework. In: Proceedings of the 2^{nd} International Conference on Automated Production of Cross Media Content for Multi-channel Distribution, pp. 223–231. Leeds, United Kingdom (2006)
3. EBU/ETSI: ETSI TS 102 822-3-1: TV-Anytime – Part 3: Metadata (2008). Available at `http://www.tv-anytime.org/`
4. Ferraiolo, J., Jackson, D. (eds.): Scalable Vector Graphics (SVG) 1.1 Specification. W3C Recommendation. World Wide Web Consortium (2009). Available at `http://www.w3.org/TR/SVG11/`
5. Hannuksela, M.M., Wang, Y.K., Gabbouj, M.: Isolated Regions in Video Coding. IEEE Transactions on Multimedia **6**, 259–267 (2004)

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

6. Hausenblas, M., Troncy, R., Burger, T., Raimond, Y.: Interlinking Multimedia: How to Apply Linked Data Principles to Multimedia Fragments. In: Proceedings of the 2^{nd} Workshop on Linked Data on the Web (LDOW'09). Madrid, Spain (2009)
7. Hickson, I., Hyatt, D. (eds.): HTML5 – A Vocabulary and associated APIs for HTML and XHTML. W3C Working Draft. World Wide Web Consortium (2010). Available at `http://www.w3.org/TR/html5/`
8. Internet Engineering Task Force: RFC 2616: HyperText Transfer Protocol – HTTP/1.1 (1999). Available at `http://www.ietf.org/rfc/rfc2616.txt`
9. Internet Engineering Task Force: RFC 3986: Uniform Resource Identifier (URI) – Generic Syntax (2005). Available at `http://tools.ietf.org/html/rfc3986`
10. ISO/IEC: Information Technology – Multimedia Content Description Interface – Part 1: Systems (2002). ISO/IEC 15938-1:2002
11. ISO/IEC: Information Technology – Coding of Audio, Picture, Multimedia and Hypermedia Information – Part 14: MP4 file format (2003). ISO/IEC 14496-14:2003
12. ISO/IEC: Information Technology – Coding of Audio-visual Objects – Part 12: ISO Base Media File Format (2005). ISO/IEC 14496-12:2005
13. ISO/IEC: Information Technology – Coding of Audio-visual Objects – Part 3: Audio (2005). ISO/IEC 14496-3:2005
14. ISO/IEC: Information Technology – Coding of Audio-visual Objects – Part 17: Streaming Text Format (2006). ISO/IEC 14496-17:2006
15. ISO/IEC: Information Technology – MPEG-21, Part 17: Fragment Identification of MPEG Resources (2006). ISO/IEC 21000-17:2006
16. ITU-T and ISO/IEC: Advanced Video Coding for Generic Audiovisual Services (2003). ITU-T Rec. H.264 and ISO/IEC 14496-10 AVC
17. Lambert, P., De Schrijver, D., Van Deursen, D., De Neve, W., Dhondt, Y., Van de Walle, R.: A Real-Time Content Adaptation Framework for Exploiting ROI Scalability in H.264/AVC. In: J. Blanc-Talon, W. Philips, D. Popescu, P. Scheunders (eds.) Advanced Concepts for Intelligent Vision Systems, *Lecture Notes in Computer Science*, vol. 4179, pp. 442–453. Springer Berlin / Heidelberg (2006)
18. Mannens, E., Troncy, R. (eds.): Use Cases and Requirements for Media Fragments. W3C Working Draft. World Wide Web Consortium (2009). Available at `http://www.w3.org/TR/media-frags-reqs/`
19. Mannens, E., Troncy, R., Pfeiffer, S., Van Deursen, D. (eds.): Media Fragments URI 1.0. W3C Working Draft. World Wide Web Consortium (2010). Available at `http://www.w3.org/TR/media-frags`
20. Pfeiffer, S.: RFC 3533: The Ogg Encapsulation Format Version 0 (2003). Available at `http://www.ietf.org/rfc/rfc3533.txt`
21. Pfeiffer, S.: Architecture of a Video Web - Experience with Annodex. In: Proceedings of the W3C Video on the Web Workshop. Brussels, Belgium (2007)
22. Pfeiffer, Silvia and Parker, Conrad and Pang, A.: Internet Draft: Specifying Time Intervals in URI Queries and Fragments of Time-based Web Resources (2005). Available at `http://annodex.net/TR/draft-pfeiffer-temporal-fragments-03.html`
23. Raggett, D., Le Hors, A., Jacobs, I. (eds.): HyperText Markup Language (HTML) 4.01 – Specification. W3C Recommendation. World Wide Web Consortium (1999). Available at `http://www.w3.org/TR/html4/`
24. Schulzrinne, H., Rao, A., Lanphier, R.: RFC 2326: Real Time Streaming Protocol (1998). Available at `http://www.ietf.org/rfc/rfc2326.txt`
25. Schwarz, H., Marpe, D., Wiegand, T.: Overview of the Scalable Video Coding Extension of the H.264/AVC Standard. IEEE Transactions on Circuits and Systems for Video Technology **17**(9), 1103–1120 (2007)
26. 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`
27. The WebM Project: WebM Multimedia Container Guidelines (2010). Available at `http://www.webmproject.org/code/specs/`
28. Troncy, R., Hardman, L., Van Ossenbruggen, J., Hausenblas, M.: Identifying Spatial and Temporal Media Fragments on the Web. In: Proceedings of the W3C Video on the Web Workshop. Brussels, Belgium (2007)
29. Van Deursen, D., Troncy, R., Mannens, E., Pfeiffer, S., Lafon, Y., Van de Walle, R.: Implementing the Media Fragments URI Specification. In: Proceedings of the 19^{th} International World Wide Web Conference (WWW), pp. 1361–1363. Raleigh, USA (2010)

30. W3C Media Fragments Working Group: Media Fragments Working Group (2010). Available at http://www.w3.org/2008/WebVideo/Fragments/
31. Xiph.org Foundation: Theora Specification (2009). Available at http://theora.org/doc/Theora.pdf
32. Xiph.org Foundation: Vorbis I Specification (2010). Available at http://xiph.org/vorbis/doc/Vorbis_I_spec.pdf