Deliverable D2.1 Specification of the Media Fragment URI scheme

Raphaël Troncy / EURECOM
Pieter van Leeuwen, Jechiam Gural / Noterik

20/04/2012

Work Package 2: Linking hypervideo to Web content

LinkedTV
Television Linked To The Web
Integrated Project (IP)
FP7-ICT-2011-7. Information and Communication Technologies
Grant Agreement Number 287911

Dissemination level	PU
Contractual date of delivery	30/03/2012
Actual date of delivery	20/04/2012
Deliverable number	D2.1
Deliverable name	Specification of the Media Fragment URI scheme
File	`D2.1.tex`
Nature	Report
Status & version	Released & v1.0
Number of pages	21
WP contributing to the deliverable	2
Task responsible	EURECOM
Other contributors	Noterik
Author(s)	Raphaël Troncy / EURECOM Pieter van Leeuwen, Jechiam Gural / Noterik
Reviewer	Daniel Stein / Fraunhofer
EC Project Officer	Manuel Carvalhosa
Keywords	Media fragments, video URL, media delivery, media servers, implementations
Abstract (for dissemination)	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 deliverable 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. We present early implementations of this technology and in particular how the LinkedTV player will handle media fragments URI.

History

Table 1: History of the document

Date	Version	Name	Comment
20/03/2012	v0.1	Troncy, EURECOM	first version of the deliverable
26/03/2012	v0.2	van Leeuven, Noterik	add Noterik implementation
14/04/2012	v0.3	Troncy, EURECOM	complete implementation section
20/04/2012	v0.4	Stein, Franuhofer	review
20/04/2012	v1.0	Troncy, EURECOM	finalize the deliverable

0 Table of Content

0 Table of Content **3**

1 Introduction **4**

2 Media Fragment URIs **5**
2.1 Media Resource Model . . . 5
2.2 Requirements . . . 5
2.3 URI Fragments vs. URI Queries . . . 5
2.4 Fragment Dimensions . . . 6
2.4.1 Temporal Axis . . . 6
2.4.2 Spatial Axis . . . 7
2.4.3 Track Axis . . . 7
2.4.4 Named Axis . . . 7
2.4.5 Combined Dimensions . . . 7

3 Resolving Media Fragments with HTTP **8**
3.1 User Agent mapped Byte Ranges . . . 8
3.2 Server mapped Byte Ranges . . . 9
3.2.1 Server mapped Byte Ranges including Header Information . . . 9
3.2.2 Server mapped Byte Ranges with Initialized Client . . . 10
3.2.3 Proxy cacheable Server mapped Byte Ranges . . . 10

4 Implementations **13**
4.1 Compliant Implementations . . . 13
4.2 Early Implementation of Media Fragments URI in the LinkedTV player . . . 14
4.2.1 Temporal fragments . . . 17
4.2.2 Spatial fragments . . . 18

5 Conclusion **20**

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[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. 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), 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). 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. Example use cases are the bookmarking or sharing of excerpts of video clips with friends in social networks, 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. The examples given throughout this deliverable 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 finalize editing her new video clip "The Editors".

In this deliverable, we present the boundaries and semantics of a Media Fragment URI and show how the syntax should look like (section 2). We describe how a media fragment specified as a URI fragment can be resolved stepwise using the HTTP protocol (section 3). We then present some implementations of the Media Fragment URI technology (section 4). Finally, we outline future work in (section 5).

[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`

2 Media Fragment URIs

2.1 Media Resource Model

We assume that media fragments are defined for ”time-linear” media resources, which are characterised by a single timeline (see also 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.

2.2 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, Ogg or WebM.
- *Fragment axes*. Media fragments need to be accessed along three different axes: temporal, spatial, and track. Additionally, media fragments 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 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.
- *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.

2.3 URI Fragments vs. URI Queries

According to the URI specification, 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.

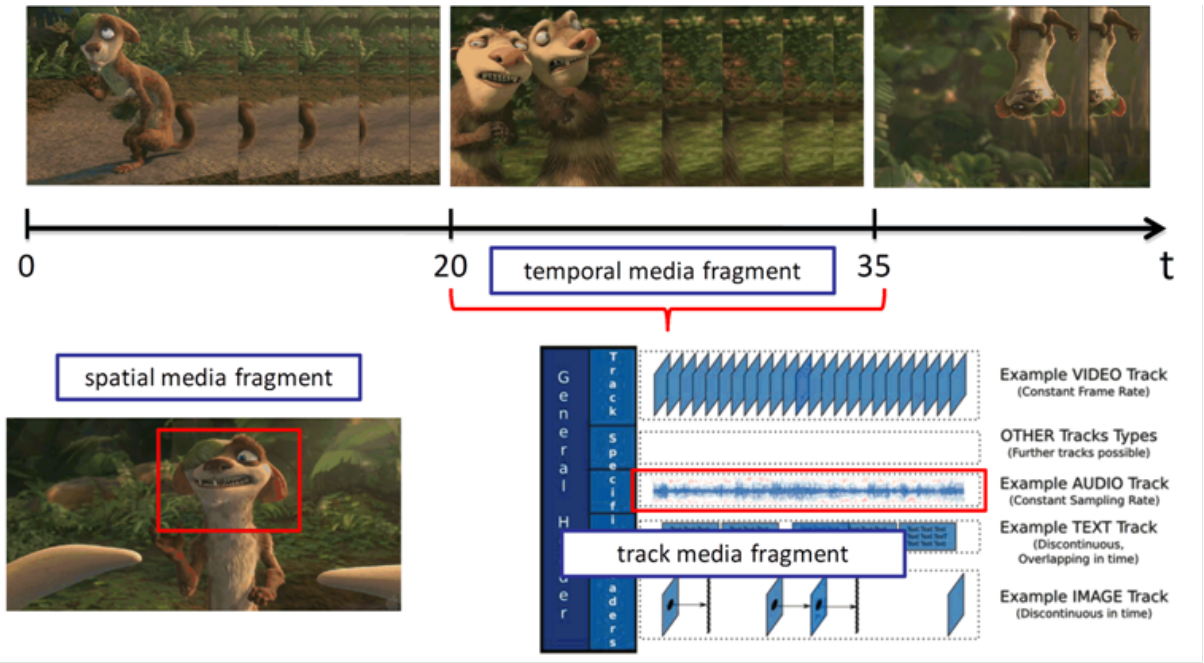

Figure 1: Example of media fragments and media resource model

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 2.2, 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 3.

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.

2.4 Fragment Dimensions

2.4.1 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. 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[
```

2.4.2 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%
```

2.4.3 Track Axis

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'
```

2.4.4 Named Axis

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 temporal axis. 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'
```

2.4.5 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, the 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`

3 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.
- 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.

3.1 User Agent mapped Byte Ranges

As stated in Section 1 (scenario (a)), Steve can now show his play using his smartphone 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 smartphone 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 the use of a user agent that 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 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 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.

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. 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. 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.

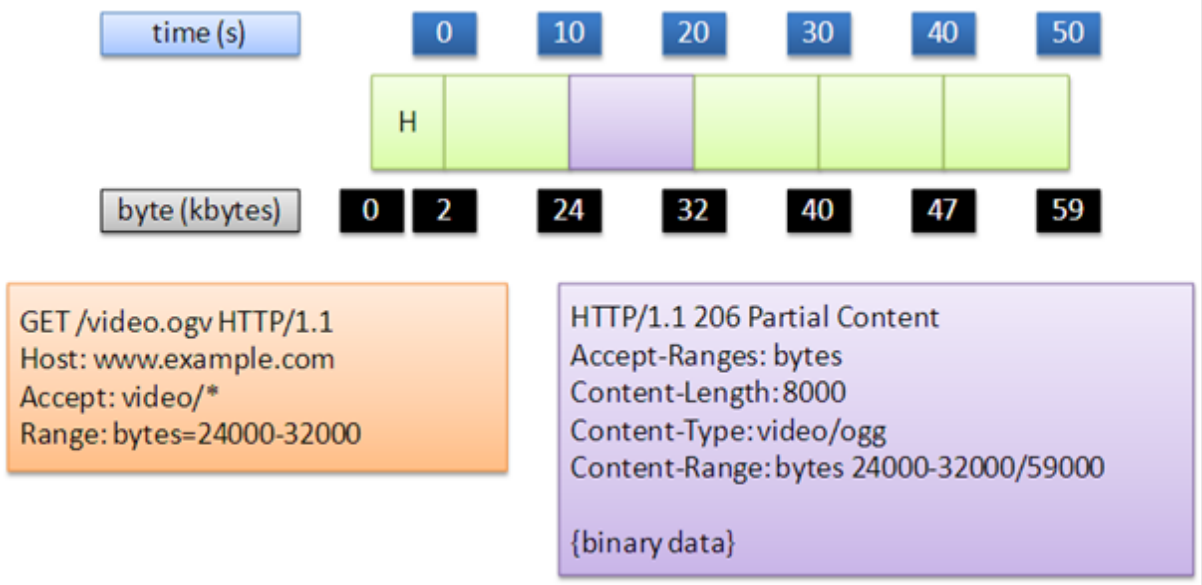

Figure 2: User Agent mapped Byte Ranges

3.2 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.

3.2.1 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 10 seconds sequence of the 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 then wraps 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 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.

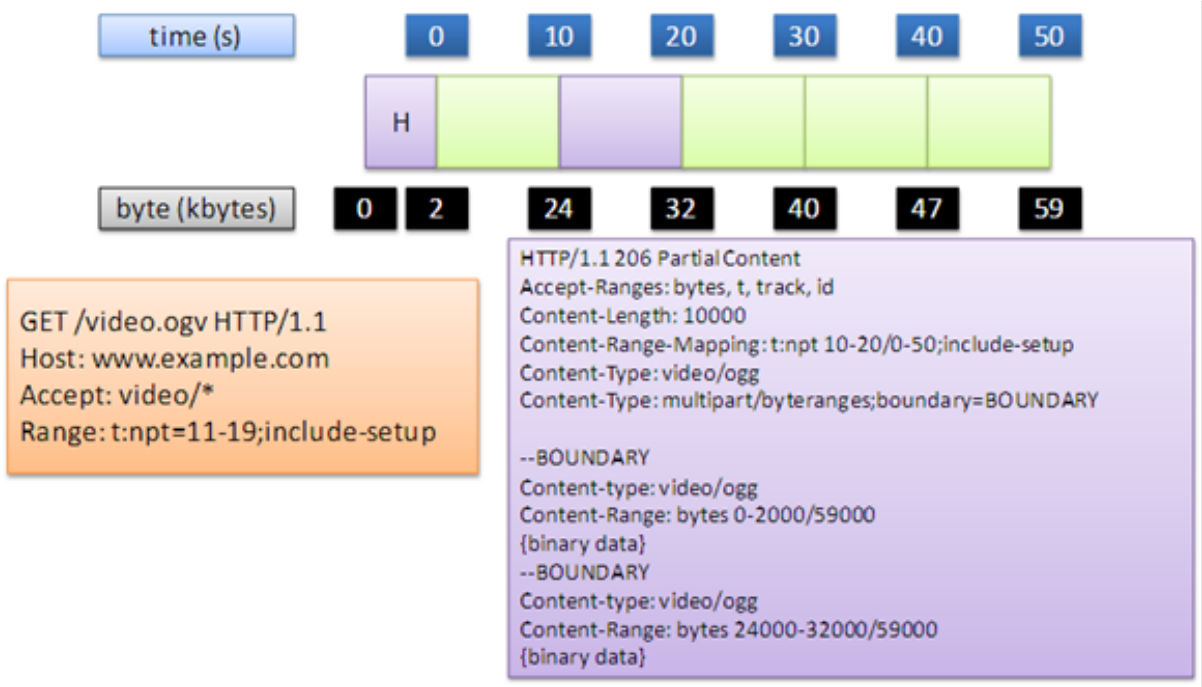

Figure 3: Server mapped Byte Ranges including Header Information

3.2.2 Server mapped Byte Ranges with Initialized Client

If the server can assume 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 npt seconds (Figure 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.
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 Figure 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 Figure 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.

3.2.3 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.

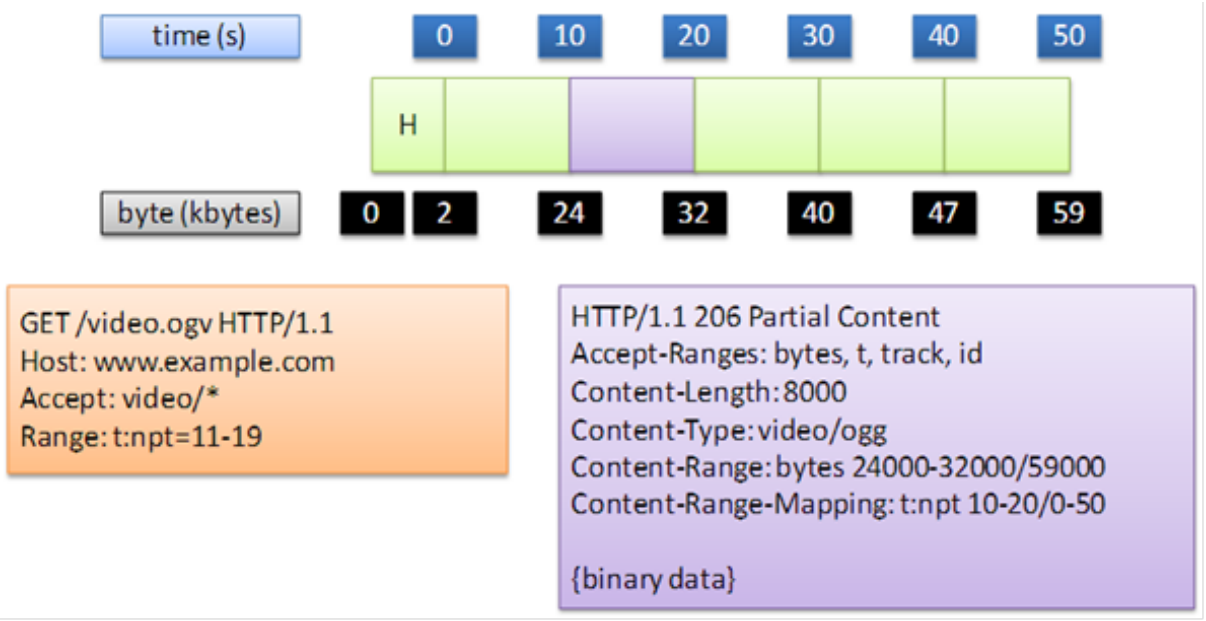

Figure 4: Server mapped Byte Ranges with initialized Client

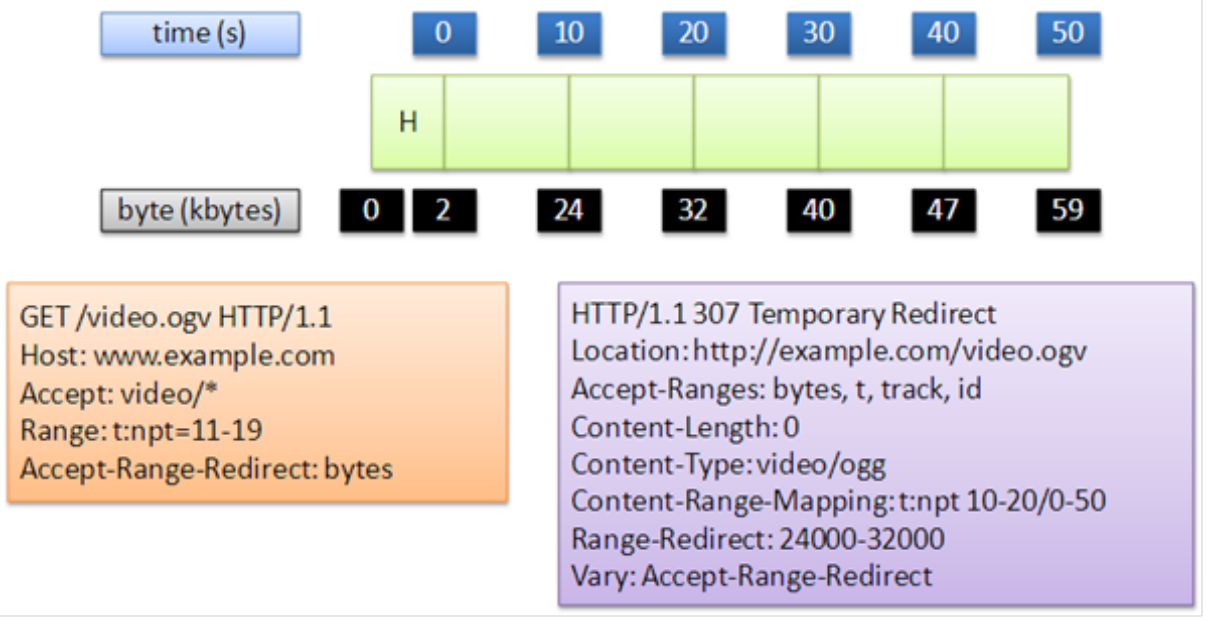

Figure 5: Proxy cacheable Server mapped Byte Ranges

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 (Figure 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 2.

4 Implementations

In this section, we present various implementations of the Media Fragments URI specification. In order to check the conformance of these implementations, we also create 145 test cases covering the four dimensions by which a media fragments can be addressed. The test cases for a conforming **user agent** implementation of the media fragments URI specification are available at `http://www.w3.org/2008/WebVideo/Fragments/TC/ua-test-cases` while the test cases for a conforming **server** implementation of the media fragments URI specification are available at `http://www.w3.org/2008/WebVideo/Fragments/TC/server-test-cases`.

The test cases are represented in both HTML and RDF (both Turtle and RDF/XML representations) are available. For example, the first test case for the user agent is represented in RDF by:

```
<http://www.w3.org/2008/WebVideo/Fragments/TC/ua-test-cases#TC0001-UA>
  a mftc:UATestCase;
  rdfs:label "TC0001-UA";
  ctag:tagged mftc:SafeTag, mftc:InvalidURITag, mftc:TemporalSegmentTag, mftc:SyntaxTag;
  dc:title """#t=,""";
  tc:purpose """Syntax error, not allowed according to the ABNF. The media fragment is ignored.""";
  mftc:media <http://www.w3.org/2008/WebVideo/Fragments/TC/media#spatial_30fps_webm>;
  mftc:mediaFragmentString "t=,";
  mftc:expectedRequest :simple-request, :byterange-request;
  mftc:expectedVisualResult "Playback of the entire media resource";
    tc:reviewStatus tc:approved ;
  mftc:decision <http://www.w3.org/2011/04/06-mediafrag-minutes.html#item03> .
```

Thus the approved test cases can be accessed by executing the following query:

```
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns\#>
prefix tc: <http://www.w3.org/2006/03/test-description\#>
select * where {
  ?tc rdf:type tc:TestCase ;
      tc:reviewStatus tc:approved .
}
```

To evaluate the coverage of an implementation's features, the Media Fragments User Agent Test Cases are used as a point of reference. The results of the implementation tests have been published at `http://www.w3.org/2008/WebVideo/Fragments/WD-media-fragments-impl/`. They are splitted over two sections:

- a section with test results for the features described in Media Fragments Basics: the temporal and spatial dimensions;
- a section with test results for the features described in Media Fragments Advanced: the track and id dimensions.

4.1 Compliant Implementations

Firefox has been an early implementor of the Media Fragments URI specification. It has been officially part of the browser since the version 9 released on December 20, 2011[3]. At the moment, it supports only the temporal dimension and does not save bandwith.

The Figure 6 shows a demonstration of the native HTML5 player of firefox playing only a fragment of an audio file encoded in WebM. The source code if this page reads:

```
...
<video id="v" src="AudioAPI.webm\#t=50,100"
  onloadedmetadata="update()" onpause="update()" onplay="update()" onseeked="update()"
  controls>
</video>
```

Synote is a Web 2.0 application allowing users to embed audio-visual resources from other domains and make synchronised annotations [LWO+12]. It supports also temporal media fragment URI but does not save bandwidth. Figure 7 shows a demonstration of the Synote player applied to a YouTube video. The media fragment is appended to the YouTube page URI which is propagated to the video element contained in this page.

Ligne de Temps is a French project developed by IRI[4]. It is a two-part software: a backend and an interface. The backend enables the user to save his work (notes, editing, rough-cut edits) into a file. It runs the codecs through which videos are imported. The video is encoded and thus readable via the interface. The system generates a timeline out of the movie which organizes the shots and sequences

[3] See the announcement at `http://lists.w3.org/Archives/Public/public-media-fragment/2011Nov/0017.html`

[4] `http://www.iri.centrepompidou.fr/outils/lignes-de-temps-2/?lang=en_us`

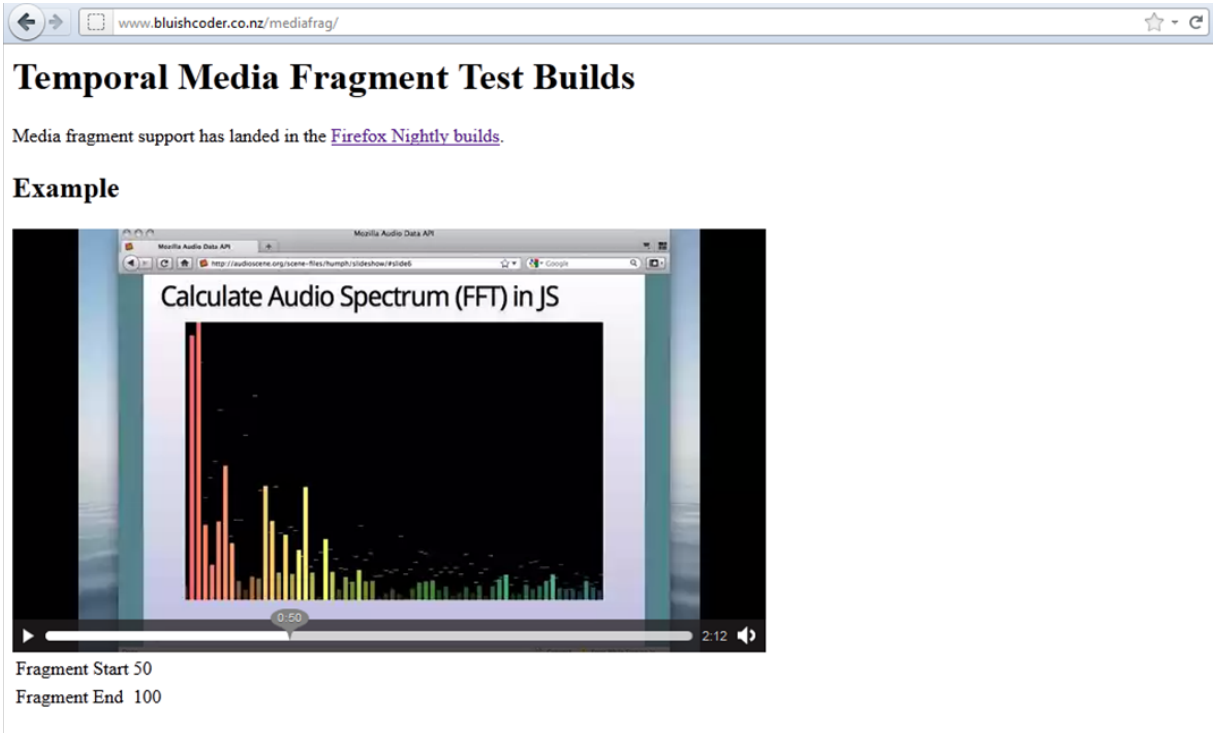

Figure 6: Implementation of Media Fragments URI in Mozilla Firefox 9+, demo available at `http://www.bluishcoder.co.nz/mediafrag/`

like a score, it can also give access to frequencies and sawtooth patterns. The interface is the visible part of the software through which the user can access and edit content. The interfaces is divided into a menu bar (File, Tool, etc) and three windows: information, video player and timeline(s). It also supports temporal media fragments without any bandwidth saving. The Figure 8 shows a screenshot of the software with a video being annotated.

Ninsuna is a fully integrated platform for format-independent multimedia content adaptation and delivery based on Semantic Web technologies [VVD+10]. It supports the temporal, spatial and track dimensions of media fragments URIs. In addition, it does save bandwidth for some video codecs and containers formats such as MP4 (H.264) and WebM (VP8). The Figure 9 shows a screenshot of media fragments being played from second 12 to second 21 while highlighting the bounding box defined by the top left coordinates (247,156) with a width of 129 pixels and a height of 206 pixels.

Finally, yuma is a javascript media annotation toolkit that makes content on a web page annotatable with just a few lines of code. It supports spatial media fragments and has plan to support temporal media fragments for videos (Figure 10.

4.2 Early Implementation of Media Fragments URI in the LinkedTV player

As we have seen, media fragments provide a standardized way of addressing subparts of a media resource by using the URI scheme. The media fragments can be either temporal, spatial or a combination of both. Within media fragments we can distinguish between URI fragments and URI queries (section 2.3). The first gives direct access to the selected subpart of the media resource within the context of the original resource while the second one creates a new resource based on the requested subpart leaving out the context of the original resource.

To make the media fragments URI compatible with the current fsxml (file system xml) that Noterik is using for video playout the parameters from the media fragments URI will be translated into fsxml properties. These properties are not stored in the fsxml resource itself, there only added dynamically once a media resource is requested with media fragment parameters.

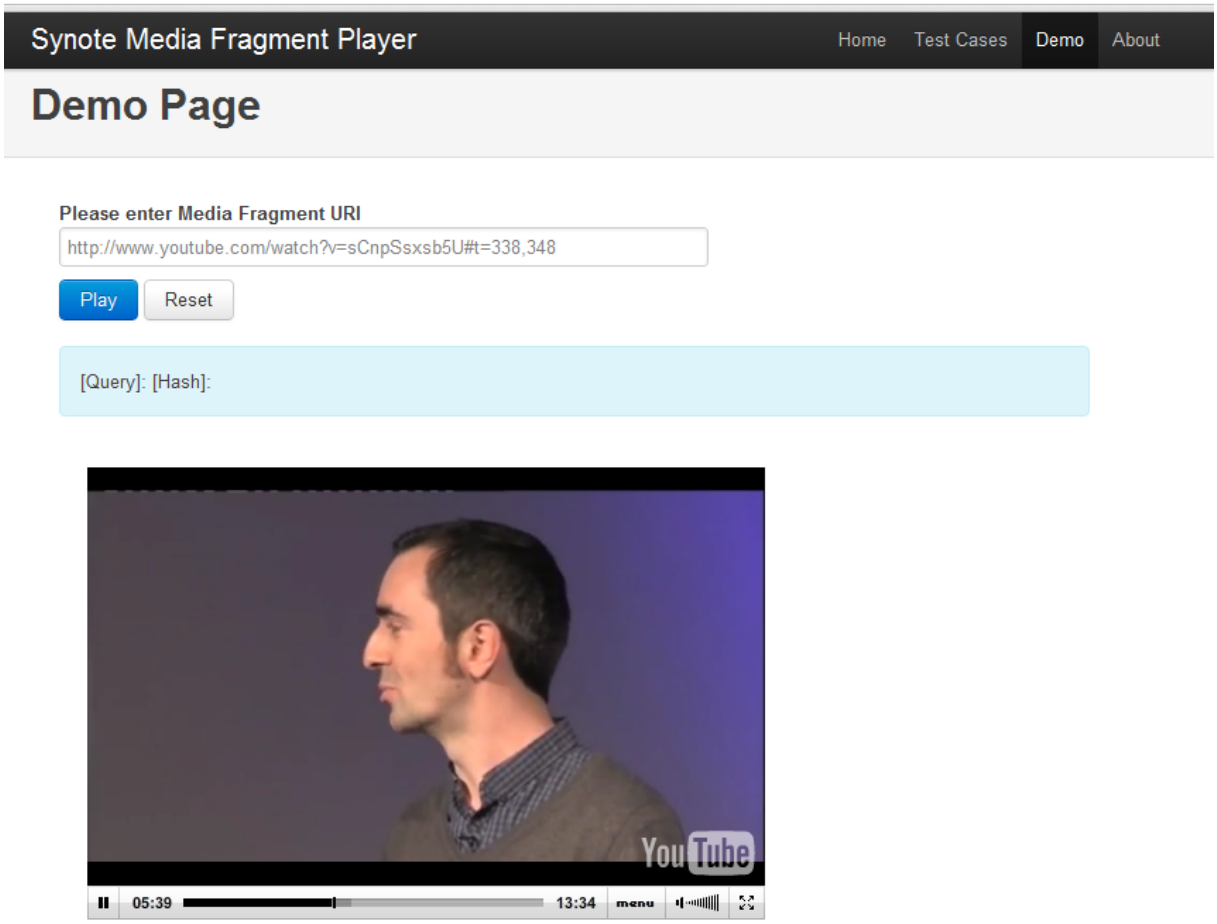

Figure 7: Implementation of Media Fragments URI in Sysnote, demo available at `http://synote-server.ecs.soton.ac.uk:8000/media-fragment-player/demo.html`

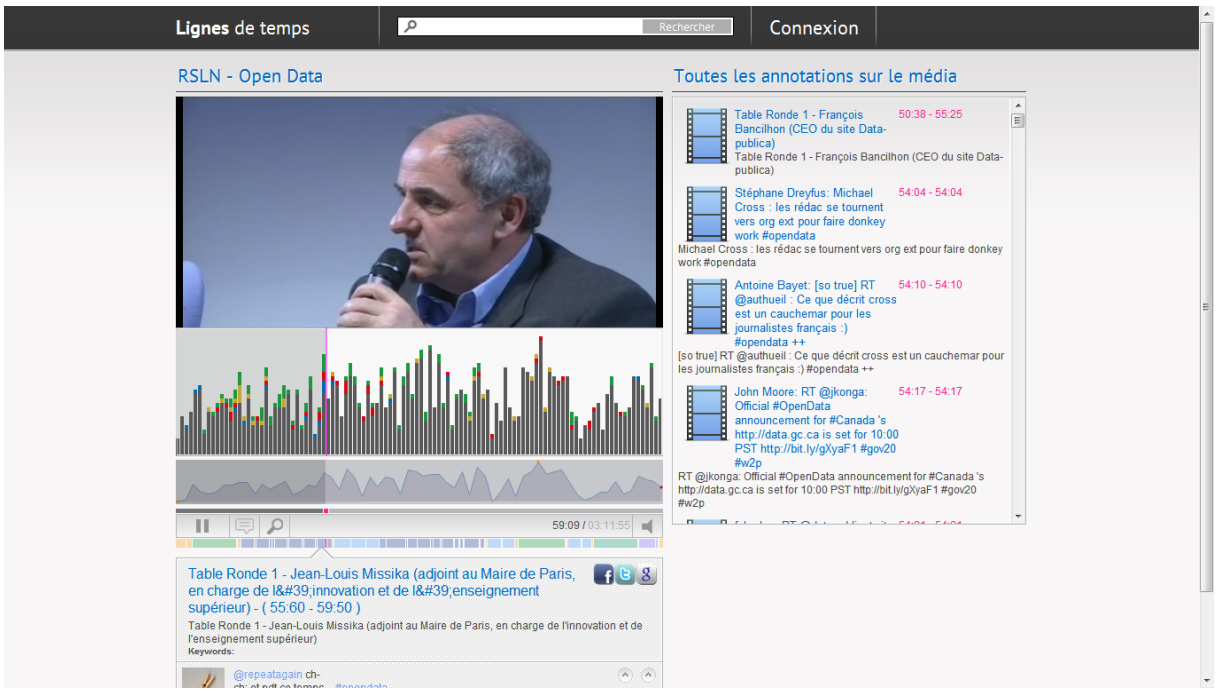

Figure 8: Implementation of Media Fragments URI in LigneDeTemps, demo available at `http://ldt.iri.centrepompidou.fr/ldtplatform/ldt/`

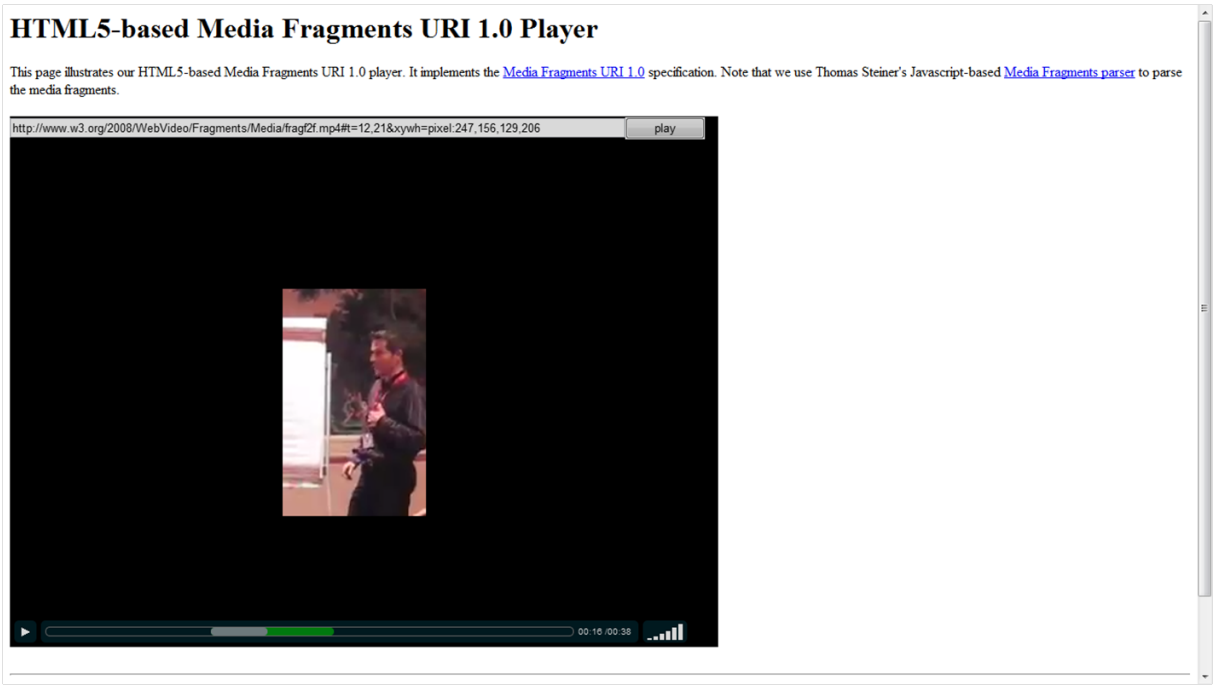

Figure 9: Implementation of Media Fragments URI in Ninsuna, demo available at `http://ninsuna.elis.ugent.be/MFPlayer/html5`

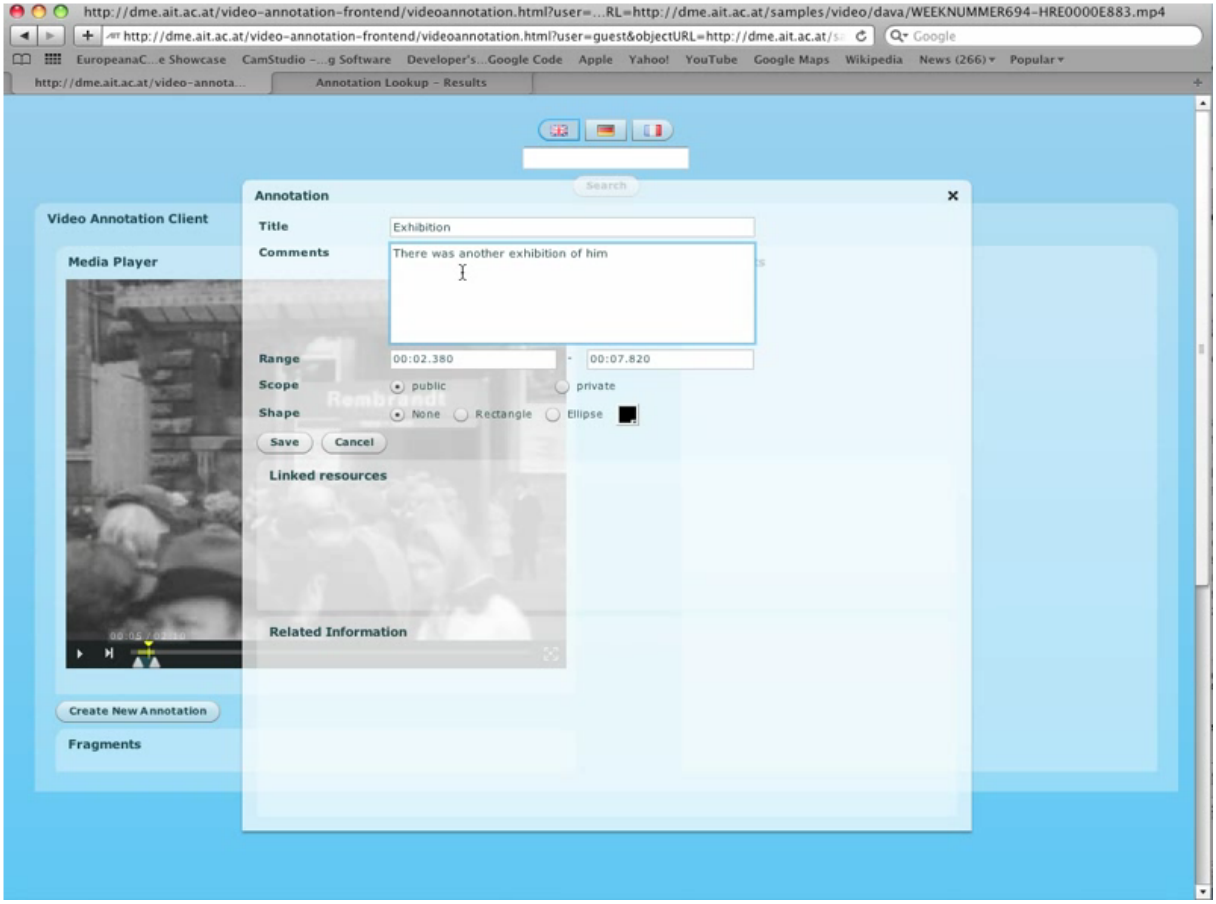

Figure 10: Implementation of Media Fragments URI in Yuma, demo available at `http://yuma-js.github.com/demos.html`

4.2.1 Temporal fragments

Temporal fragments are media fragments that cover a certain time span from the original media resource. Providing at least a start time or end time creates a temporal fragment. When a temporal URI **fragment** is requested it will start to play from the provided start time until the provided end time. However the transport bar allows to see the entire resource with the fragment being highlighted directly under the transport bar. The syntax of a temporal URI fragment is: `http://<mediaresourcepath>#t=starttime,endtime`. The corresponding fsxml syntax of the temporal URI fragment is:

```
<fsxml>
  <video id="1">
    <properties>
      <starttime>starttime</starttime>
    <duration>duration</duration>
      <context>true</context>
    </properties>
  </video>
</fsxml>
```

Note: the endtime is replaced by the duration (endtime - starttime) for compatibility with the current system.

Figure 11: Implementation of Media Fragments URI for the temporal dimension and the fragment parameter in the LinkedTV player

When a temporal URI **query** is requested it will start to play from the provided start time until the provided end time. The transport bar only shows the fragment, leaving out the original context, therefore no highlighting is needed. The syntax of the temporal URI query: `http://<mediaresourcepath>?t=starttime,endtime`. The corresponding fsxml syntax of the temporal URI query is:

```
<fsxml>
  <video id="1">
    <properties>
    <starttime>starttime</starttime>
    <duration>duration</duration>
    </properties>
  </video>
</fsxml>
```

Note: the endtime is replaced by the duration (endtime - starttime) for compatibility with the current system

Figure 12: Implementation of Media Fragments URI for the temporal dimension and the query parameter in the LinkedTV player

4.2.2 Spatial fragments

Spatial fragments are media fragments that cover a rectangular selection from the original media resource. To create a spatial fragment the x and y position together with the width (w) and height (h) of the rectangle needs to be provided. When a spatial URI **fragment** is requested the player shows only the requested rectangle. The surrounding context of the resource is available but blacked out. The syntax of the spatial URI fragment `http://<mediaresourcelocation>#x,y,w,h`. The corresponding fsxml syntax of the spatial URI fragment is:

```
<fsxml>
  <video id="1">
    <properties>
      <x>x</x>
      <y>y</y>
      <width>w</width>
      <height>h</height>
      <context>true</context>
    </properties>
  </video>
</fsxml>
```

When a spatial URI **query** is requested the player shows the requested rectangle scaled to fit the player dimensions. The context of the original resource is not visible any more. The syntax of the spatial URI query: `http://<mediaresourcelocation>?x,y,w,h`. The corresponding fsxml syntax of the temporal URI fragment is:

```
<fsxml>
    <video id="1">
        <properties>
  <x>x</x>
  <y>y</y>
  <width>w</width>
  <height>h</height>
        </properties>
    </video>
</fsxml>
```

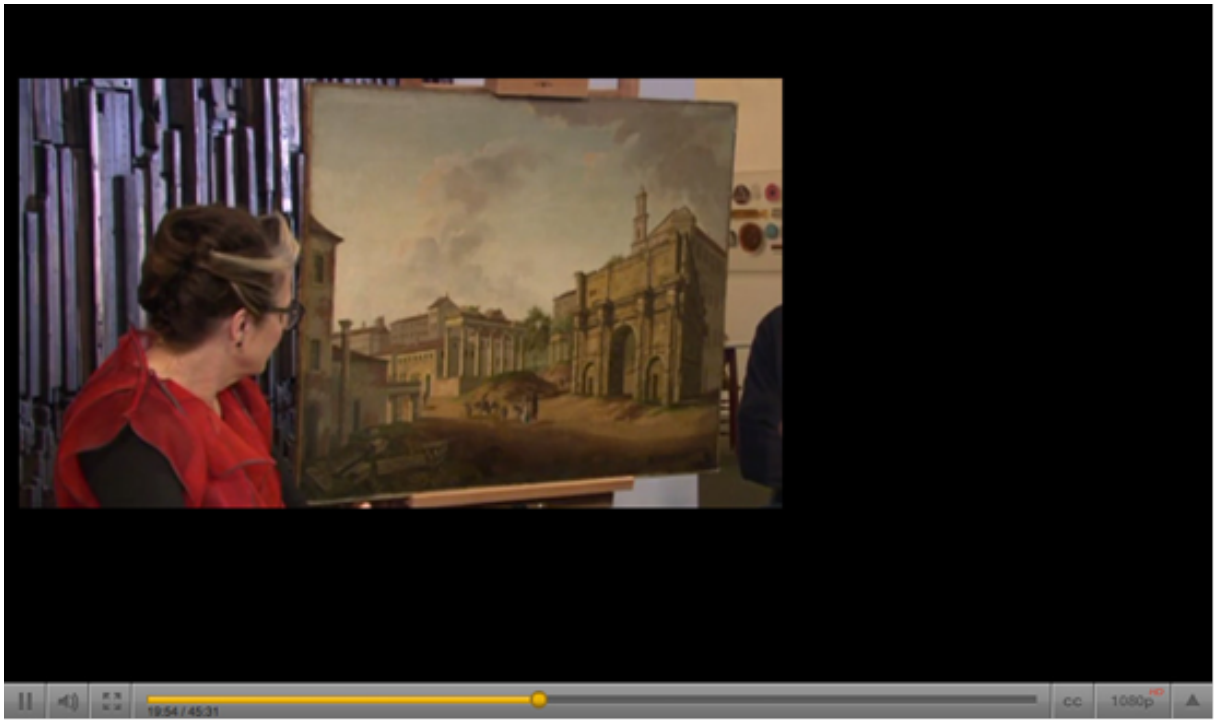

Figure 13: Implementation of Media Fragments URI for the spatial dimension and the fragment parameter in the LinkedTV player

Figure 14: Implementation of Media Fragments URI for the spatial dimension and the query parameter in the LinkedTV player

5 Conclusion

In this deliverable, we presented the rationale for a Media Fragment URIs specification to make video a “first-class citizen” on the web. 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 already have a HTTP implementation for the W3C Media Fragments 1.0 specification in order to verify all the test cases defined by the working group. Furthermore, several browser vendors have implemented a HTTP implementation themselves including Mozilla Firefox and WebKit powering both Safari and Chromium. In the near future, we also foresee reference implementations for the RTSP, RTMP, and File protocols.

References

[HTRB09] Michael Hausenblas, Raphaël Troncy, Yves Raimond, and Tobias Bürger. Interlinking Multimedia: How to Apply Linked Data Principles to Multimedia Fragments. In *2nd Workshop on Linked Data on the Web (LDOW'09)*, Madrid, Spain, 2009.

[ISO03] ISO/IEC. Information technology – Coding of Audio, Picture, Multimedia and Hypermedia Information – Part 14: MP4 file format. ISO/IEC 14496-14:2003, December 2003.

[LWO+12] Yunjia Li, Mike Wald, Tope Omitola, Nigel Shadbolt, and Gary Wills. Synote: Weaving Media Fragments and Linked Data. In *5nd Workshop on Linked Data on the Web (LDOW'12)*, Lyon, France, 2012.

[Pfe] Silvia Pfeiffer. RFC 3533: "The Ogg Encapsulation Format Version 0," Available on `http://www.ietf.org/rfc/rfc3533.txt`.

[Pfe07] Silvia Pfeiffer. Architecture of a Video Web - Experience with Annodex. W3C Video on the Web Workshop, 2007.

[THvOH07] Raphaël Troncy, Lynda Hardman, Jacco van Ossenbruggen, and Michael Hausenblas. Identifying Spatial and Temporal Media Fragments on the Web. W3C Video on the Web Workshop, 2007.

[TM09] Raphaël Troncy and Erik Mannens, editors. *Use cases and requirements for Media Fragments*. W3C Working Draft. World Wide Web Consortium, November 2009.

[TM12] Raphaël Troncy and Erik Mannens, editors. *Media Fragments URI 1.0*. W3C Proposed Recommendation. World Wide Web Consortium, March 2012.

[VVD+10] Davy Van Deursen, Wim Van Lancker, Wesley De Neve, Tom Paridaens, Erik Mannens, and Rik 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*, 46(2-3):371–398, January 2010.