Information and Communication Technologies (ICT) Programme

Project N^{o}: FP7-ICT- 248036

COAST

Deliverable D5.2

D5.2 - Media signalling specification for 3D video content types

Author(s): Karsten Grüneberg (HHI), Emanuele Quacchio (STM), Petr Verbitskiy (TUB), Robert Skupin (HHI), Davide Aliprandi (STM), Emiliano Piccinelli (STM), Gianluca Gennari (STM)

Status -Version: Final

Date: 14 October 2011

Distribution - Confidentiality: Public

Code: COAST_D5.2_HHI_V0.9_20111014.docx

Abstract:

This document specifies media signalling for different 3D video content formats and different transport protocols. It focuses on Dynamic Adaptive Streaming over HTTP for MVC. It also reports on related implementations which have been developed in Task 5.2 and provides a brief documentation on software modules provided to the project partners.

© Copyright by the COAST Consortium

Disclaimer

This document contains material, which is the copyright of certain COAST contractors, and may not be reproduced or copied without permission. All COAST consortium partners have agreed to the full publication of this document. The commercial use of any information contained in this document may require a license from the proprietor of that information.

The COAST Consortium consists of the following companies:

No	Participant name	Participant short name	Country	Country
1	ST Microelectronics	STM	Co-ordinator	Italy
2	Synelixis Solutions Ltd	Synelixis	Contractor	Greece
3	Yahoo Iberia SL	Yahoo	Contractor	Spain
4	NEC Europe Ltd	NEC	Contractor	UK
5	Telefonica Investigacion Y Desarrollo SA	TID	Contractor	Spain
6	Fraunhofer HHI	HHI	Contractor	Germany
7	Politecnico di Torino	Polito	Contractor	Italy
8	Technische Universitaet Berlin	TUB	Contractor	Germany
9	Fundacio Barcelona Media	BM	Contractor	Spain
10	University of California, Los Angeles	UCLA	Contractor	USA
11	Seoul National University	SNU	Contractor	S. Korea

The information in this document is provided "*as is*" and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

This page has intentionally been left blank

Table of contents

Disclaimer **2**

Abbreviations **5**

Executive Summary **7**

1. Signalling for 3D Video **8**

1.1. 3D Video Format Related Issues 8

1.1.1. Signalling of frame compatible stereo *8*
1.1.2. MVC Views and Decoding Dependencies *10*
1.1.3. Depth Maps for View Interpolation *11*

1.2. Transport Protocol Related Specifications 12

1.2.1. RTP Streaming *12*
1.2.2. HTTP Streaming *14*
1.2.2.1 MPD Design 16
1.2.2.2 Dependencies between Representations 17
1.2.2.3 DASH Multiple views scheme 17
1.2.2.4 HTTP Streaming Using MP4 File Format 18
1.2.2.5 HTTP Streaming Using MPEG-2 Transport Stream (TS) Files 20

2. Implementation **22**

2.1. Authoring Tools 22

2.1.1. Generation of Session Description protocol (SDP) for RTP *22*
2.1.2. Generation of Media Presentation Description (MPD) *23*
2.1.3. Generation of MPEG-2 Transport Stream (TS) segments *26*

2.2. HTTP Streaming 27

2.2.1. GStreamer media framework *27*
2.2.2. HTTP Streaming Client Module *28*
2.2.3. GStreamer based 3D video streaming receiver *30*
2.2.4. Demonstration System *31*

2.3. Client Module for View Interpolation 32

2.3.1. FVV rendering engine *33*
2.3.1.1 Virtual Camera 33
2.3.1.2 Smart warping 34
2.3.1.3 Blending 35
2.3.1.4 Occlusion solving 35
2.3.1.5 3D Packing 35

2.3.2. FVV integration in GStreamer receiver *35*

3. Conclusion **37**

References **38**

Abbreviations

AF	Adaptation Field
ATSC	Advanced Television Systems Committee
AU	Access Unit
AVC	Advanced Video Coding
AVCFF	AVC File Format
DASH	Dynamic Adaptive Streaming over HTTP
DES	Depth Enhanced Stereo
DRM	Digital Rights Management
DVB	Digital Video Broadcasting
DVD	Digital Versatile Disc
ES	Elementary Stream
FCN	Future Content-Centric Network
FVV	Free Viewpoint Video
GNU	GNU's Not Unix!
GPL	GNU General Public License
GPU	Graphical Processing Unit
GUI	Graphical User Interface
HTTP	HyperText Transport Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISO	International Organization for Standardization
ISOFF	ISO Base Media File Format
LGPL	Lesser GPL
MP4FF	MPEG-4 File Format (ISOFF, AVCFF or other derived specifications)
MPD	Media Presentation Description
MPEG	Moving Picture Experts Group
MTAP	Multi-Time Aggregation Packets
MTU	Maximum Transmission Unit
MVC	Multiview Video Coding
NAL	Network Abstraction Layer
NEM	Networked and Electronic Media
PAT	Program Association Table
PCR	Program Clock Reference

PES	Packetized Elementary Stream
PMT	Program Map Table
PSI	Program Specific Information (e.g. PAT or PMT)
PT	Payload Type
PTS	Presentation Time Stamp
RFC	Request For Comments (IETF standard)
RTP	Real-Time Transport Protocol
SAP	Stream Access Point
SDP	Session Description Protocol
SEI	Supplemental Enhancement Information
SI	Supplemental Information
SLA	Service Level Agreement
STAP	Single Time Aggregation Packets
SVC	Scalable Video Coding
SW	SoftWare
TS	Transport Stream (MPEG-2 systems)
URL	Universal Resource Locator
VRA	Video Retrieval Agent
WP	Work Package (of the COAST project)
XML	eXtensible Markup Language

Executive Summary

COAST aims to build a Future Content-Centric Network (FCN) overlay architecture able to intelligently and efficiently link billions of content sources to billions of content consumers, and offer fast content-aware retrieval, delivery and streaming, while meeting network-wide Service Level Agreements (SLAs) in content and services consumption.[1] According to this statement, COAST not only pays attention to common media formats which are used today in the majority of network applications, but COAST also anticipates novel media formats which can prosper in the near future due to the evolution of consumer electronic devices. Amongst these innovative formats, scalable and 3D video coding will play an important role.[2] Besides, such media formats need some attention due to their layered structure which has to be considered from the beginning in a content-centric approach. In its WP5, COAST analyses existing and emerging approaches of 3D video coding and provides solutions for content creation, storage, transmission, adaptation and decoding.

This document presents the result of Task 5.2 — *Session Streaming Signalling*.

The remainder of this document is structured as follows:

Section 1 introduces signalling aspects related to the transport of 3D video, where subsection 1.1 is dedicated to different 3D video formats while subsection 1.2 deals with transport protocols.

Section 2 discusses the modules that have been implemented in Task 5.2 and which have been delivered to the consortium. Authoring tools which are responsible to generate the proper signalling are described in section 2.1. Section 2.2 focuses on the receiver side, describing the client modules which receive the signalling information and how decisions can be based on such signalling. Within this section (2.2.4), a complete streaming chain is described. This is the first demonstration system for 3D streaming based on the DASH specification. Section 2.3 is dedicated to a module which uses depth maps for view interpolation.

Section 3 concludes this document and gives some information how the software modules fit into the integrated streaming system and the COAST architecture.

1. Signalling for 3D Video

The aim of media signalling is to allow for a communication between media source and media consumer about the format and related features in a way that e.g. a client can decide whether it is capable of decoding of a media stream and it can chose the proper decoder for a certain media type. Further, signalling is used to convey side information which might not be included in the encoded stream itself. In case of 3D video, the information that two video streams form a stereoscopic pair is not necessarily included in the bitstream format, because each of the stereoscopic views can be encoded independently, or the two views can be packed into a single video sequence and the information how to separate the views after decoding has to be signalled to the rendering device.

1.1. 3D Video Format Related Issues

As mentioned above, information needed for a proper presentation of the media at the client side depends on the format of the media. As discussed in D5.1, there are several options for storage and transmission of 3D video, which differ in terms of source format (e.g., the number of views) and video compression (e.g., the codec). The signalling aspects are discussed in the following sections for the different source formats and codecs.

1.1.1. Signalling of frame compatible stereo

Frame compatible transmission combines two views into the same video stream which can be encoded with any legacy codec because there is no need for the encoder to reflect the association of a subset of the pixels (e.g., the left and right halves in case of side-by-side frame packing) to different video sources. On the other hand, encoding is less efficient if the structural similarities of the two views are not exploited. There are different approaches to signal the fact that a video sequence uses a certain frame packing arrangement.

- The in-band approach adds some data to the coded bitstream which means that this data must be codec specific. Taking into account that video encoding standards usually do not specify such usage from the very beginning, this additional data needs to be encoded in a way that it does not disturb the operation of a legacy decoder, which at the same time means that a legacy client cannot derive any decision from this additional data, even less decode and present a single view as the only useful part of the media stream if the client is connected to a 2D display only.
- Out-of-band approaches rely on a systems layer to transmit side information which should be codec agnostic, as the separation of the views takes places after decoding. Nevertheless, also codec specific signalling is possible on the systems layer.

Two versions of in-band signalling for frame compatible stereo have been standardized or are in the process of standardization:

MPEG-4 AVC specifies Supplemental Enhancement Information (SEI) messages that are not necessary for the decoding of the bitstream (and thus may be ignored by decoders), but contain some useful information for its presentation. “Stereo video information SEI” messages have been introduced in 2005 as a first amendment together with AVC professional extensions. These SEI messages allow signalling of either alternating transmission of left and right view (temporal frame interleaving) or line interleaving of stereoscopic views, e.g., the left view being packed into odd lines, the right view into even lines. Another SEI message has been added recently (2010): “Frame packing arrangement SEI” messages allow for signalling a multitude of different frame packing arrangements also including side-by-side, above/below, column-based interleaving, checkerboard and quincunx arrangements. The syntax is shown in Table 2; the rightmost column shows the size (in parentheses, *v* stands for variable) and data format (*u* stands for unsigned, *e* for variable length encoded) of bit fields other than flags (always one bit).

Table 1 shows the syntax of this message; all flags are exactly one bit.

Another SEI message has been added recently (2010): "Frame packing arrangement SEI" messages allow for signalling a multitude of different frame packing arrangements also including side-by-side, above/below[1], column-based interleaving, checkerboard and quincunx arrangements. The syntax is shown in Table 2; the rightmost column shows the size (in parentheses, *v* stands for variable) and data format (*u* stands for unsigned, *e* for variable length encoded) of bit fields other than flags (always one bit).

Table 1 — Syntax of Stereo Video Information SEI message

```
stereo_video_info( payloadSize ) {
   field_views_flag
   if ( field_views_flag )
      top_field_is_left_view_flag
   else {
      current_frame_is_left_view_flag
      next_frame_is_second_view_flag
   }
   left_view_self_contained_flag
   right_view_self_contained_flag
   }
}
```

Table 2 — Syntax of Frame Packing Arrangement SEI message

```
frame_packing_arrangement( payloadSize ) {
   frame_packing_arrangement_id                    ue(v)
   frame_packing_arrangement_cancel_flag
   if (!frame_packing_arrangement_cancel_flag ) {
      frame_packing_arrangement_type                u(7)
      quincunx_sampling_flag
      content_interpretation_type                   u(6)
      spatial_flipping_flag
      frame0_flipped_flag
      field_views_flag
      current_frame_is_frame0_flag
      frame0_self_contained_flag
      frame1_self_contained_flag
      if ( !quincunx_sampling_flag
       && frame_packing_arrangement_type != 5 ) {
         frame0_grid_position_x                     u(4)
         frame0_grid_position_y                     u(4)
         frame1_grid_position_x                     u(4)
         frame1_grid_position_y                     u(4)
      }
      frame_packing_arrangement_reserved_byte       u(8)
      frame_packing_arrangement_repetition_period   ue(v)
   }
   frame_packing_arrangement_extension_flag
}
```

[1] In this document, we use the term „above/below" rather than „top/bottom" for consistency with D5.1 and in order to avoid confusion with an arrangement that uses the top and bottom fields for the stereoscopic views in a line-interleaved manner

For MPEG-2 Video, a similar approach is currently subject of standardization as Amendment 4 of [30]. In this case, it is proposed to use the user_data extension to the picture header for signalling which would not influence existing deployments but could be evaluated by future implementations. Here, only side-by-side and above/below arrangements will be supported.

Out-of-band signalling of frame-compatible stereoscopic video over MPEG-2 Systems is in the process of standardization. MPEG-2 Systems specifies Transport Stream (TS) and Program Stream (PS) format, the first being widely used for broadcast (e.g. using ATSC and DVB standards) and the second being commonly used on video DVDs. MPEG-2 TS will also be supported by MPEG-DASH (cf. section 1.2.2.4).

1.1.2. MVC Views and Decoding Dependencies

The Multiview Video Coding (MVC) specification is part of the MPEG-4 Advanced Video Coding (AVC) standard [22]. It allows to jointly encode (i.e. compress) a multitude of raw video sequences that contain different views of the same scene, exploiting similarities between (adjacent) views. If only two views are captured, the Stereo High profile can be used; more views would require the Multiview High profile. The standard itself does not require or make any assumptions whether the different views are showing the same scene, but coding efficiency gain can only be expected, if texture and related motion vectors in one view can be matched with blocks in the other view so that inter-view prediction makes sense. If a set of views is suitable for stereoscopic presentation, this is typically the case.

Inter-view prediction results in coding dependencies between views, i.e., the dependent view can only be decoded correctly if the complementary view (on which the prediction has been based) is available. In MVC, there is one backward compatible view (i.e., a sub-bitstream conforming to an AVC profile) which can be decoded independently from others. A second view typically depends on the base view, and each further view may depend on a specific set of views in a hierarchical relationship. One example is shown in Figure 1. Consequently, dependencies result in a specific decoding order of the data units related to the individual views which is signalled in the bitstream. One or more views can be discarded from the bitstream, as long as dependencies are respected, i.e., if no view in the remaining bitstream depends on any of the discarded views.

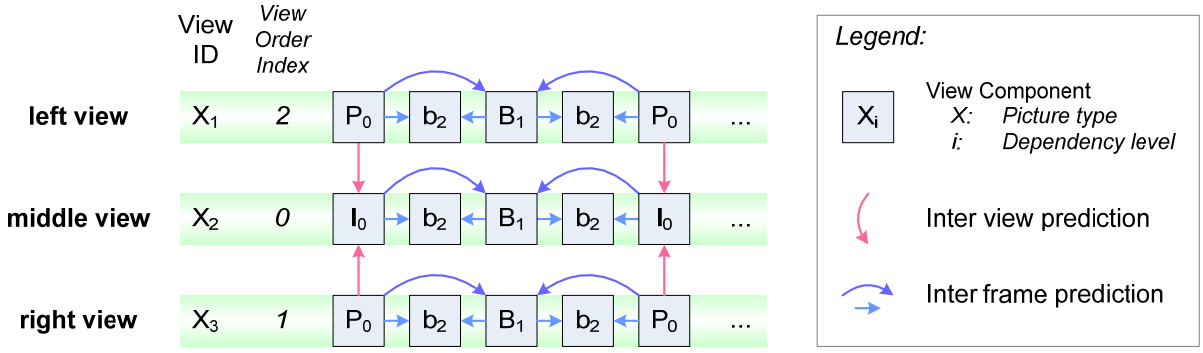

Figure 1 — Example for inter-view dependencies between multiple views

If data units are sent through different physical or logical channels, the original bitstream has to be reassembled after transmission and before decoding. Therefore, MVC specific signalling is required on the transport layer which indicates decoding dependencies and the proper decoding order.

The MVC standard allows for, but does not mandate, the in-band transmission of camera parameters, which could be used as indication how the different views should be presented to a viewer, i.e., if a pair of views would be suitable for the presentation to the left and right eye, and if so, which view would be the left and which the right view. This should also be signalled along the whole chain which includes video capturing, encoding, media storage and transmission, decoding and rendering.

1.1.3. Depth Maps for View Interpolation

Different technologies are available for 3D displays, some of which require the user to wear polarizing glasses in order to address the left and right eye separately. Other so-called “glassless” or “autostereoscopic” 3D display technologies use lenticular screens or parallax barriers in order to allow viewing of only a subset of the pixels from a certain perspective. In this case, different views are rendered on certain patterns of the pixels (e.g. columns) and are visible in certain zones with two of the zones located at eye distance if the user is watching the screen from a certain distance. Many autostereoscopic displays are well-suited to present more than two views to a viewer, at the cost that the native resolution of the panel (given by the number of pixels) is shared among the different views. This drawback will be overcome as soon as the technical progress allows panel manufactures to build displays with 4K resolution (i.e. about 2000 by 4000 pixels) at costs that result in affordable prices for consumers.

The scenario above is the background for new technologies that use view interpolation to display a multitude of different views to one or more users because transmitting many views in parallel over the network is less efficient than sending a reduced number of views and doing view interpolation for further intermediate views at the receiver [7]. High quality view interpolation can be achieved if the scene depth is known for each pixel of the two adjacent views [8]. More details are found in section 2.3.

1.2. Transport Protocol Related Specifications

1.2.1. RTP Streaming

RTP, a transport protocol for real-time applications [9] can be used for real-time transmission of many different media types. For each codec, a dedicated payload format is provided, some of which can be signalled directly using a dedicated Payload Type (PT) parameter [10], and many more being signalled using dynamical PT assignment. For AVC, the RTP payload format specification has been updated recently (RFC 6184, [11]). For SVC, RFC 6190 has just been published [12] with contributions by the COAST project. A dedicated payload format for MVC including COAST contributions is still under development [13].

The media signalling for RTP is specified by the Session Description Protocol (SDP [14]). Decoding dependencies between different media sessions can be signalled according to [15]. Table 3 shows an example SDP for an MVC stream with two views. For simplicity, other media description, e.g. audio, is not shown in this example.

Table 3 — Example Session Description for multi-session transmission of MVC streams

Line#	
[1]	v=0
[2]	o=mvcsrv 289083124 289083124 IN IP4 host.example.com
[3]	s=MULTIVIEW VIDEO SIGNALING Seminar
[4]	t=0 0
[5]	c=IN IP4 192.0.2.1/127
[6]	a=group:DDP V1 V2
[7]	m=video 40000 RTP/AVP 96
[8]	b=AS:2000
[9]	a=framerate:25
[10]	a=rtpmap:96 H264/90000
[11]	a=mid:V1
[12]	m=video 40002 RTP/AVP 97
[13]	b=AS:1200
[14]	a=framerate:25
[15]	a=rtpmap:97 H264-MVC/90000
[16]	a=mid:V2
[17]	a=depend:97 lay V1:96

In this example, the MVC bitstream is transported in two RTP sessions which are described in lines 7-11 (shaded grey in Table 3) and 12-17, respectively. Each block of a media description starts with an “m=” line and includes further a line with “mid” attribute (media stream identification [16]). As can be determined by the dynamic PT number 96 in line 7 together with the associated “rtpmap” attribute in line 10, the media labelled “V1” contains the AVC compatible base view, while the other media stream labelled “V2” is associated with PT 97 (line 12) for which line 15 signals the subtype “H264-MVC” as proposed in [13].

In line 17, we find the signalling for the fact that the second view (PT 97) depends on the Base View (V1, PT 96) in terms of a layered decoding dependency. The indication of a specific layered decoding dependency also corresponds to line 6 of the example SDP which specifies grouping type “DDP” for the media labelled “V1” and “V2”. When this SDP information is parsed by a streaming client, it will not request a server to send only the second view, but it will either request the Base View V1 (for which no dependency is signalled) or both V1 and V2.

RTP transmission and associated SDP signalling is supported in the COAST project for legacy reasons in a scenario as shown in Figure 2. However, some of the innovative features in terms of content caching will only be available for HTTP streaming as described in sections 1.2.2, 2.1.2 and 2.2 below.

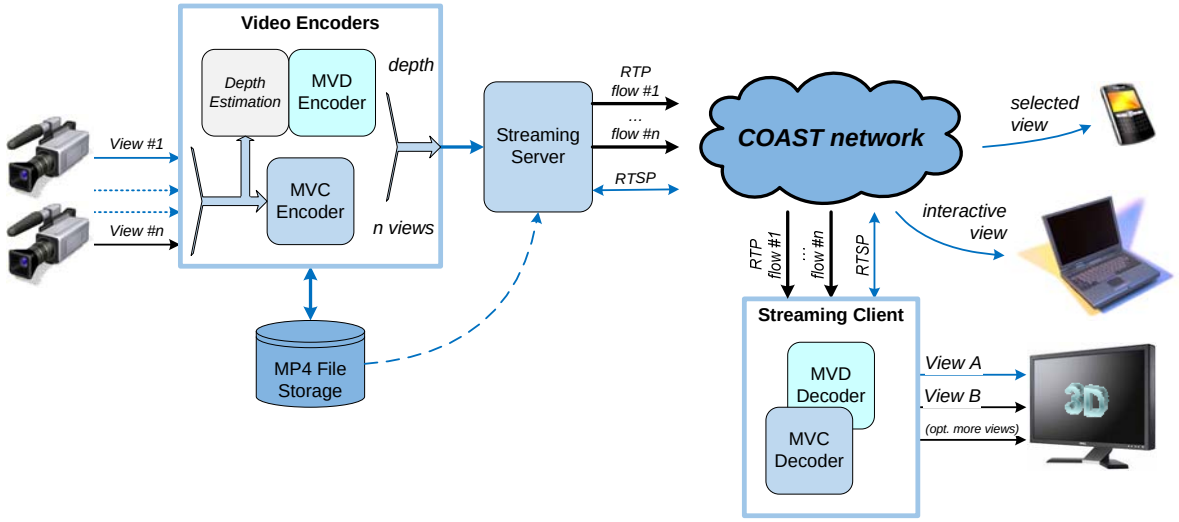

Figure 2 — RTP streaming chain transmitting multiple views

Figure 3 illustrates the RTP payload for MVC in a multi-session streaming scenario. In the top line we find the MVC bitstream which is composed of NAL units. Here, we show two access units (AU), each corresponding to four so-called view components which are numbered from 0 to 3. A view component is the coded representation for a single view at a certain time instant (i.e., a video frame). In our example, we assume that the first two views 0 and 1 are sent individually in their own RTP session S_1 and S_2, respectively, while the other two views 2 and 3 are sent together in a third RTP session S_3. For the encapsulation of MVC NAL units in RTP packets, different payload structures are specified in [11]. These different options are depicted in the figure.

Single NAL units can be most efficient, because the payload of the RTP packet is exactly one NAL unit. No start code is needed, because the start of the NAL unit coincides with the beginning of the RTP payload, and the first byte of the payload is the NAL unit header.

If the RTP payload size exceeded 1452 bytes, the resulting IP packet which contains the RTP packet would exceed the standard Ethernet MTU size of 1500 bytes, so larger NAL units should be fragmented and sent using multiple RTP packets in order to avoid IP fragmentation. Fragmentation Units, which serve for this purpose, are signalled by specific header bytes which differ from valid NAL unit header bytes, and a start-stop mechanism indicated by flags in the header bytes of each fragmentation unit signals which fragmentation units carry parts of the same NAL unit. In most cases, if one Fragmentation Unit is lost during transmission, which the receiver can detect checking the RTP sequence numbers, the whole sequence that carries a certain NAL unit has to be invalidated. If more than one packet is missing, the RTP timestamp can be used to identify the first Access Unit after the drop-out.

In order to reduce the RTP and IP overhead for very small NAL units, Aggregation Packets are specified which can carry more than one NAL unit. Aggregation Packets introduce a little overhead indicating the length of the NAL units that form the RTP payload. Two different forms of Aggregation Packets are specified: Single Time Aggregation Packets (STAP) and Multi-Time Aggregation Packets (MTAP). MTAPs carry additional timestamp values that allow detecting which of the NAL units belong to which Access Unit. For MTAPs, two different timestamp sizes can be used in order to keep the additional overhead as small as possible while also allowing an aggregation of NAL units from Access Units which differ very much in presentation time.

The example shows how two NAL units from AU 1 are packed together in one STAP, while two NAL units from different AUs (view 3 from AU 1 and view 2 from AU 2) are transmitted using an MTAP[2].

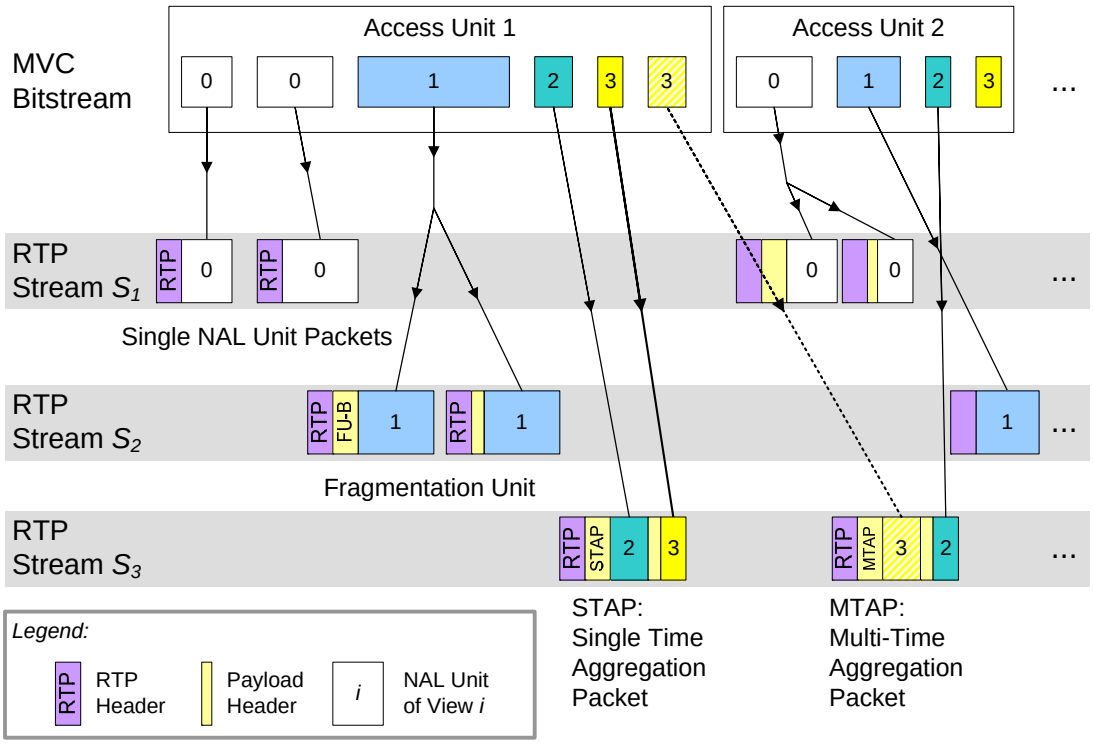

Figure 3 — Multi-session RTP streaming of MVC

1.2.2. HTTP Streaming

In HTTP streaming, media is transferred in portions on request of a client. On a server, these "portions" can be stored as individual files or as one large file of which certain byte ranges are fetched sequentially. In both cases, the client decides what data to request based on a description of all available media.

DASH (Dynamic and Adaptive Streaming over HTTP) is the novel MPEG proposal aiming to offer a standard and alternative solution to the several flavours of HTTP streaming technologies available in the market today. A review of existing commercial solution for video delivery over IP networks based on HTTP streaming can be found in [5]. The DASH client model is depicted in Figure 4.

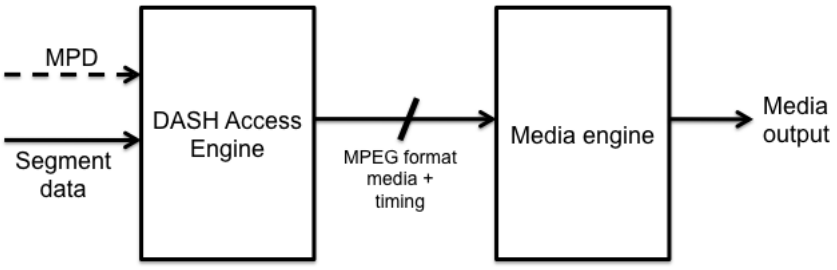

Figure 4 — DASH client model [28]

[2] In order to keep this example simple, we show two NAL units for view component 3 in Access Unit 1. More typical constellations can lead to the same result, e.g., if we assume either a small ***or*** a larger NAL unit of view 3 in AU 1. In the first case, we could create the STAP together with the preceding NAL unit of the same AU, in the second case, the maximum size might be exceeded, and an MTAP together with a NAL unit of the next AU could be beneficial.

MPEG-DASH standard extends specifications for adaptive HTTP streaming in mobile environment already defined in 3GPP (3rd Generation Partnership Project) [20] and primarily defines two kind of information:

- The Media Presentation Description (MPD), an XML file (more general: an XML document) which describes the media representations (i.e. bitrates, video resolutions, number of views etc.) available for the content and resource identifiers (which are exclusively HTTP-URLs);
- Segment properties which specify the way in which the media content need to be prepared in order to react to HTTP GET or partial HTTP GET requests issued by a client. Especially for MVC (or other layered codec like SVC), decoding dependencies between views (or layers in case of SVC) have to be respected as described in 1.1.2.

Further, the standard includes the definition of syntax elements to support delivery of layered contents, segmented media formats, stream alignment for bitstream switching, MP4FF and MPEG-2 TS support, definition of profiles, support for Digital Right Management (DRM) technologies etc.

The MPD has to provide all information needed by the client in terms of signalling before it starts downloading data portions. However, the MPEG-DASH standard does not describe a normative behaviour of the client once the MPD is received; it neither specifies the adaptation logic that drives the selection of the best representation according to terminal and network conditions.

The complete transmission chain for (adaptive) HTTP streaming for MVC is shown in Figure 5. On its left side, the content is captured, encoded and prepared for HTTP transmission, i.e., one or more media files and an appropriate MPD are generated and stored on a file system connected to an ordinary HTTP server.

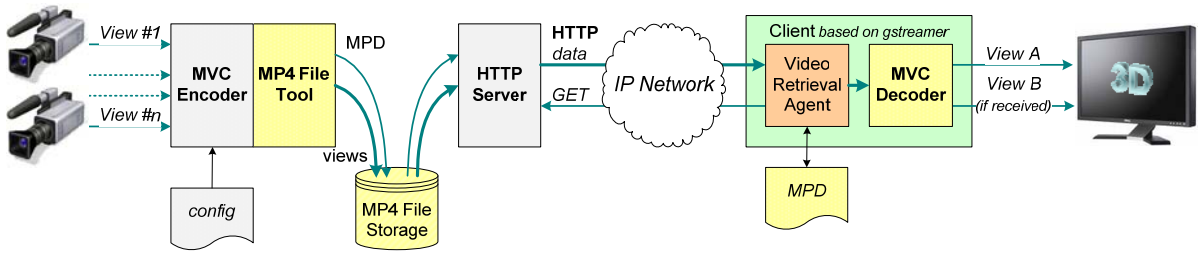

Figure 5 — HTTP streaming chain transmitting multiple views

As mentioned earlier, the MPD is an XML document; it contains XML elements that can contain further child elements and/or attributes. In the following sections, we describe some basic concepts and some details related to MVC.

1.2.2.1 MPD Design

The whole MPD schema specified in the MPEG-DASH standard is very complex, and it contains many optional elements and attributes.

Figure 6 gives an overview of mandatory XML elements and their nesting.

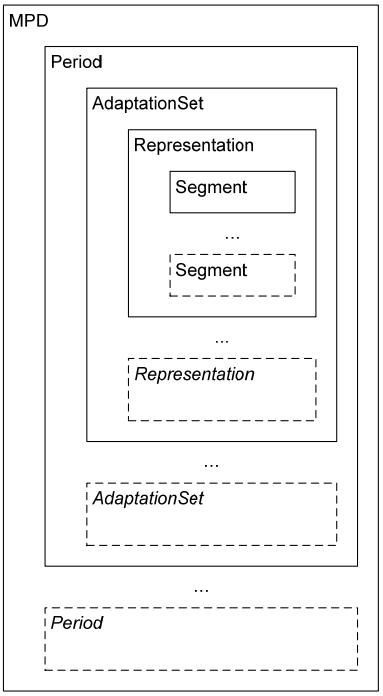

Figure 6 — Main MPD Elements

The top element is the Media Presentation Description which includes one or more Period elements. Periods describe parts that can be played sequentially. Inside a Period, the available media types are consistent, i.e., the set of available bitrates, languages, layers, views etc. does not change. Within a period, media components are arranged in Adaptation Sets which contain interchangeable versions of one or more media components. For example, an Adaptation Set can contain the audio component or a multiview video component. Within an Adaptation Set, alternative Representations are described which e.g. differ in terms of bitrate or number of views. In general, one Representation is sufficient to render the media component, so the client can select one out of the set of Representations, e.g. according to its capabilities, or switch dynamically between Representations, e.g. according to network conditions. One exceptional case is described in the following section1.2.2.2: If decoding dependencies exist, the client can choose one Representation, but also has to load data described in additional “complementary” Representations. Switching between Representations is done either at the boundaries of Segments, which are the innermost structure inside the MPD, or at an even lower level called Subsegments. Such Subsegments have no corresponding XML elements, but they are enumerated by MP4 File Format boxes within the media files (cf. section 1.2.2.4). Nevertheless, some Subsegment properties are given in the MPD using XML Attributes of the Segment elements.

Within the MPEG-DASH specification, six profiles are specified which pose restrictions on the availability of optional elements or which require certain values for specific attributes:

1. **Full profile** — *no restriction, either ISO Base media file format or MPEG-2 TS used;*
2. **ISO Base media file format On Demand profile** — *only one Segment per Representation;*
3. **ISO Base media file format live profile** — *aligned Segments;*
4. **ISO Base media file format main profile** — *no references to external document (@xlink[3]);*
5. **MPEG-2 TS main profile** — *no references to external document (@xlink[3]);*
6. **MPEG-2 TS simple profile** — *same PSI (cf. 1.2.2.5) is valid across Representations.*

In the list above, only one restriction per profile is mentioned to show a distinction between profiles.

1.2.2.2 Dependencies between Representations

One of the novel concepts that have been incorporated in MPEG-DASH is the definition of dependent and complementary representations.

dependent Representation:	Representation for which Segments from its complementary Representations are necessary for presentation and/or decoding of the contained media content components
complementary Representation:	Representation which complements at least one dependent Representation

These definitions are targeting layered codecs such as SVC and MVC which include a decodable sub-bitstream, e.g. the Base Layer resp. Base View. The first Enhancement Layer (EL) resp. a second view is regarded as a dependent representation of the video and Base Layer resp. Base View are the corresponding complementary representation. Further ELs resp. views may need more than one complementary representation, so the first EL would be a dependant representation (as explained above) and a complementary representation for higher ELs at the same time. One or more complementary representations which are necessary for decoding a dependent representation are indicated in the MPD using the @dependencyId[4]

1.2.2.3 DASH Multiple views scheme

For media content description of multiple views, a multiple views scheme is defined. This scheme can be applied to a ContentComponent element of type video or to an AdaptationSet element. Therefore, a Role element with @schemeIdURI equal to "urn:mpeg:dash:stereoid:2011" is specified which can take values that indicate left and right view of a stereo pair. If this scheme is applied to an AdaptationSet element containing multiple views, each media content component of type video shall have a ContentComponent element that contains the Role descriptor using this scheme.

Assumed that N views are available which can be combined into M valid stereo pairs. View indicators are text strings starting with either ‘l’ or ‘r’ followed by a non-negative decimal integer. A stereo pair i (0 <= i < M) is formed by views whose Role element includes the view indicator 'li' as the left view and a view whose Role element contains and the view indicator ‘ri’ as the right view. If a ContentComponent element can be used in different roles, e.g., it is the left view of pair i or the right view of pair j, the @value of the Role element contain a space-delimited list of the appropriate indicators.

As an example, for the camera setup shown in Figure 7; the view captured by the middle camera would be tagged with the Role element

```
<Role schemeIdUri="urn:mpeg:dash:stereoid:2011" value="l1 r0"/>
```

which indicates that this view could be used as the left view of pair 1 or as the right view of pair 0.

[3] the “full” profile allows a subset of W3C XLINK simple links for inclusion of remote elements
[4] the ‘@’ character is used as a prefix for attribute names (in line with the MPEG-DASH standard)

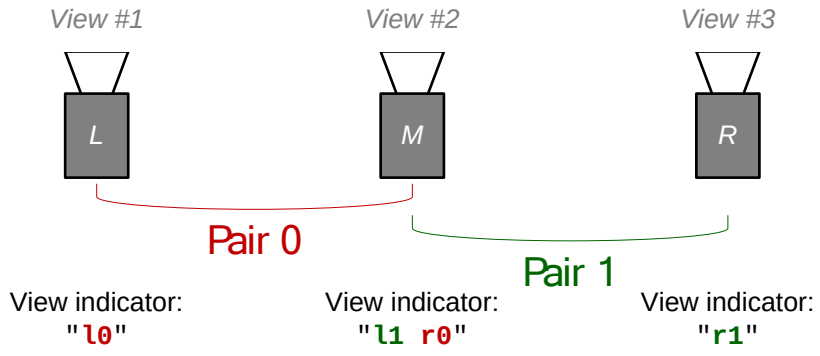

Figure 7 — Example camera setup (top view) with 3 cameras

A sample MPD which contains the complete syntax for such a configuration is given in the annex; section **Error! Reference source not found.**, **Error! Reference source not found.**.

The transmission of the MPD itself to the client is outside the scope of the DASH specification; we can assume that the MPD file is found as an HTML hyperlink on a webpage.

1.2.2.4 HTTP Streaming Using MP4 File Format

In the course of the DASH standardization activities, the MP4 file format played an outstanding role because on the one hand, the file format should provide information that a client needs about data location in the file that is cut into pieces for transmission, and on the other hand, the original media file or at least a standard compliant media file including a subset of the original information has to be re-assembled at the client. With older versions of the ISO Base Media File Format and its derivates, some issues had been detected which have been resolved by amendments to the standards.

A more detailed description of MP4 File Format features focussed on features which are used for HTTP streaming, e.g “Movie Fragments”, and mechanisms which are valuable for 3D content, such as the track dependencies and ‘auxv’ media handler is found in D5.1 [3] and therefore not repeated here.

The specification of the Segment Index box (‘sidx’) in [25] is one of the main MP4FF features related to signalling for HTTP streaming and for that reason described here in more detail.

This box allows signalling which byte ranges within an MP4 file constitute a Subsegment which can be used as the smallest entity for dynamic adaptation in the DASH standard. A Segment Index box can also point to another Segment Index box in a hierarchical way. Segment Index boxes can be located inside the file they are referencing, but also in an extra file which contains only indexing information. The latter can also be used for indexing into MPEG-2 TS files. Segment Index boxes contain data offsets for a list of Subsegments in binary format which makes this description rather compact compared to a SegmentList element in the MPD. In addition to that, a number of other useful properties for switching between Subsegments are included in the Segment Index box, namely the location of Stream Access Points (SAP) and related timing information. Six SAP types are defined which differ regarding their behaviour in the context of seamless switching:

SAP types 1 and 2: “Closed GOP random access points”, the random access picture being the first output picture for type 1 and a later picture for type 2;

SAP type 3: “Open GOP random access point” with “late” pictures (to be presented before the random access picture if the preceding Subsegment had also been decoded);

SAP type 4: "Gradual Decoding Refresh random access point” with “late” pictures;

SAP types 5 and 6: Random access point followed by pictures that depend on previous Subsegment.

Storing a whole media presentation within one file and indexing Subsegments thereof reduces complexity at the server side compared to switching on a per-Segment basis combined with the storage of many files when each file contains a single Segment.

The syntax of the Segment Index box is shown in Table 4.

Table 4 — Segment Index box

```
Line#
 [1]  SegmentIndexBox extends FullBox('sidx', version, 0) {
 [2]     unsigned int(32) reference_ID;
 [3]     unsigned int(32) timescale;
 [4]     if (version==0) {
 [5]        unsigned int(32) earliest_presentation_time;
 [6]        unsigned int(32) first_offset;
 [7]     }
 [8]     else {
 [9]        unsigned int(64) earliest_presentation_time;
[10]        unsigned int(64) first_offset;
[11]     }
[12]     unsigned int(16) reserved = 0;
[13]     unsigned int(16) reference_count;
[14]     for (i=1; i <= reference_count; i++) {
[15]        bit (1) reference_type;
[16]        unsigned int(31) referenced_size;
[17]        unsigned int(32) subsegment_duration;
[18]        bit(1) starts_with_SAP;
[19]        unsigned int(3) SAP_type;
[20]        unsigned int(28) SAP_delta_time;
[21]     }
[22]  }
```

Additional information on a Subsegment can be added using another MP4 File Format box called Subsegment Index box ('ssix') which allows further subdividing the Subsegment into a number of Levels where one Level can depend on another Level in a hierarchical manner. Levels are assigned in a Level Assignment box ('leva') which is a child box of the Movie Extends box located in the initialization segment for the whole Representation. A level can be assigned to a Sample Group, a Track or a Subtrack, which means that a certain media type, SVC layer or MVC view can be indexed using the Subsegment Index box if the Subsegment contains a multiplexed portion of multiple such entities.

Table 5 — Subsegment Index box

```
Line#
 [1]  SegmentIndexBox extends FullBox('sidx', version, 0) {
 [2]     unsigned int(32) subsegment_count;
 [3]     for ( i=1; i <= subsegment_count; i++) {
 [4]        unsigned int(32) ranges_count;
 [5]        for ( j=1; j <= range_count; j++) {
 [6]           unsigned int(8) level;
 [7]           unsigned int(24) range_size;
 [8]        }
 [9]     }
[10]  }
```

1.2.2.5 HTTP Streaming Using MPEG-2 Transport Stream (TS) Files

For legacy reasons, also MPEG-2 TS segments are used in HTTP streaming. The TS structure is shown in Figure 8. The TS has been designed for broadcast systems where it is necessary to accommodate different media stream rates and combine them to a multiplexed byte stream transmitted at constant data rate. A TS is composed of a sequence of TS packets of fixed size (188 bytes) of which the first four bytes are header bytes that, amongst other signalling information, contain a 13-bit packet identifier (PID).

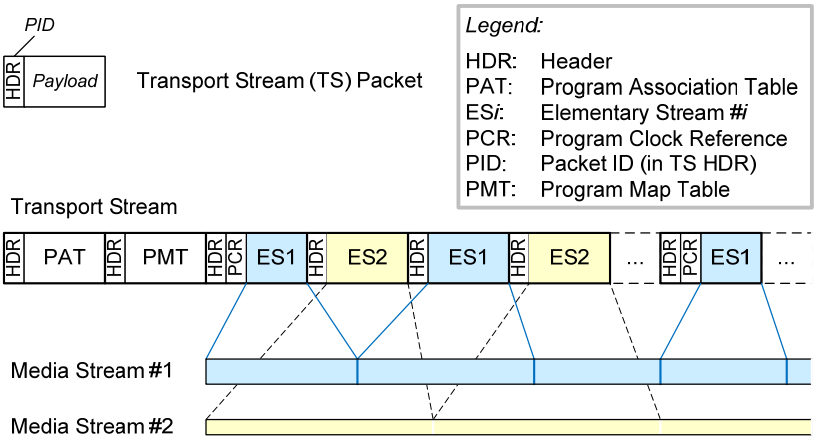

Figure 8 — Transport Stream carrying two Elementary Streams

Some PID values are assigned to so-called Program Specific Information (PSI) packets carrying general information about the multiplex which is needed by a TS demultiplexer (demux) in order to identify and reassemble the media streams inside the multiplex. Transport Streams can contain multiple programs, each consisting of multiple media streams, e.g. audio and video. The main PSI packets contain the Program Association Table (PAT) which lists all programs transmitted in the TS and the Program Map Table (PMT) which maps all PIDs belonging to a certain program and the related media streams. In some cases, TS contain only one program, especially if a single program is recorded and stored in a file. Another important feature is the transmission of time stamps called Program Clock Reference (PCR) which can be used to synchronize the receiver to the system clock of a program source, e.g., when PCR time stamps are received over a DVB transmission channel (i.e., in real time). However, when TS are stored, each PCR value loses its relationship to a certain instant in real time because the time it is encountered by the demux depends on the read access to the TS file.

Figure 8 also shows that different media streams (shaded blue and yellow at the bottom of the figure) are fragmented and inserted into the payload of TS packets. The width of each media stream line symbolizes its data rate; evidently the multiplex rate is larger than the sum of the media rates. The occurrence of TS packets that belong to a certain Elementary Stream (ES) corresponds to the associated media rate, i.e., the number of TS packets for a certain media stream in a certain time interval is proportional to the media rate.

In case of MVC, we can create separate ES for the different views. Before encapsulation of the ES in TS packets, a so-called Packetized Elementary Stream (PES) is generated by insertion of PES headers (this detail is neglected in Figure 8) which precede Access Units (AU) and contain Presentation Time Stamps (PTS). An example with three separate views is shown in Figure 9. It also shows another detail which applies at the end of each AU: The last TS packet contains some padding using the so-called Adaptation Field (AF).

Properties of each PES that belong to a certain program are indicated by so-called descriptors in the PMT. More details on descriptors used in the context of MVC are given in section 2.1.2.

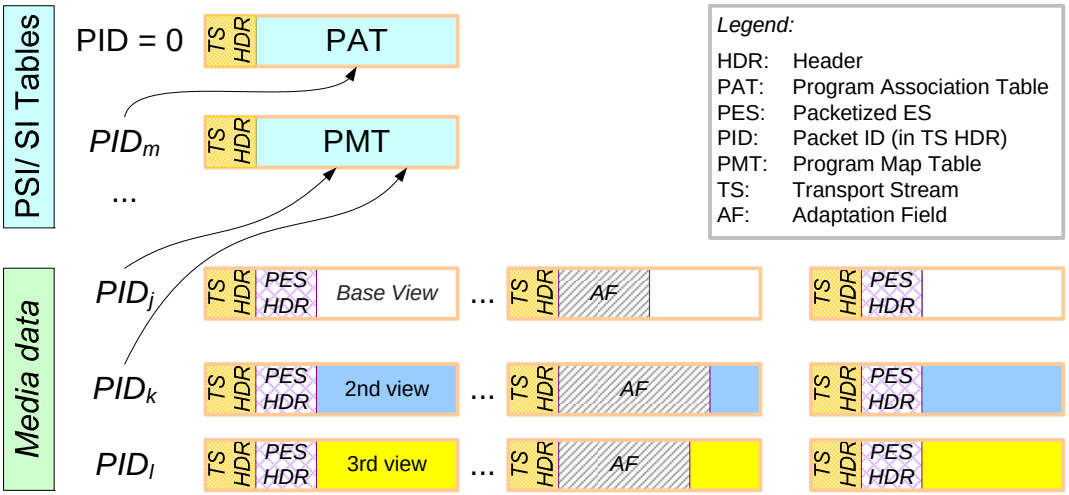

Figure 9 — Transport Stream packets carrying three Elementary Streams for MVC

TS files can be used for HTTP streaming; however, a number of issues have to be taken into account.

As mentioned before, TS files usually contain only one program; it certainly would be a waste of data rate if a streaming client downloaded a multi-program TS file if only a single program is shown to the user. Anyway, TS can contain audio in multiple languages of which only one is typically used, but the audio data rate can often be regarded as negligible. Alternatively, different TS can be generated which contain the same video together with only one language or a subset of all available languages; this would waste storage space on the server.

Similar reasoning can be applied to TS files that contain multiple views. More details on implementation issues are found in section 2.1.2.

2. Implementation

Software modules provided to the project partners are compiled binaries which are executable on a certain platform. Two sets of binaries are provided in order to support Linux as well as Microsoft Windows platforms. The software is protected by hardware devices which prevent unauthorized usage. The following chapters describe the different parts implemented by WP5. A directory listing is given in the **Error! Reference source not found.** in section **Error! Reference source not found.**.

2.1. Authoring Tools

2.1.1. Generation of Session Description protocol (SDP) for RTP

SDP information is stored in hint tracks in the MP4 file format, and the MP4FileTool is capable to analyse media tracks, generate appropriate SDP information (cf. Table 3 in section 1.2.1 and store it in the MP4 file in a standard compliant manner.

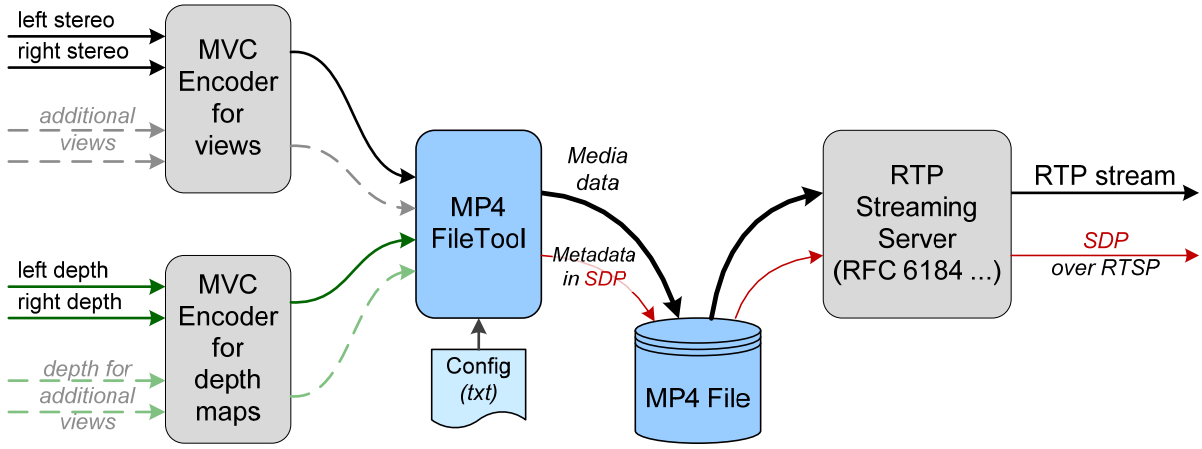

Figure 10 — Preparation of SDP signalling information for RTP streaming

In order to generate the metadata for the media tracks, the tool is executed with the following command line:

MP4FileTool <*media file name*> -h <*media track ID*> [packetization mode] –p <*parameter file name*>

This instructs the tool to create a so-called hint track which includes SDP information from the media track with the ID specified as the first argument of the –h attribute on the command line. An optinal second argument is used to specify the packetization mode (see [11]). The parameter file is used to specify if the media track is transmitted in a single RTP session or how views are transmitted in separate RTP sessions.

The syntax of this parameter file is not standardized but proprietary; an example is shown in Table 6. The media type “hint” is specified for the *MediaConfiguration* element in line 2, and in the track configuration (lines 3-10) there are two track definitions given. The first hint track (lines 4-6) is a pure AVC track (payload name is “H264”) which includes media data of tier 1 only. Numbers in the *TierList* element (line 5) refer to the *TierConfiguration* element (lines 11-18). Tier definitions are found in lines 12-14 and 15-17, and these tiers are numbered implicitly with natural numbers starting from 1. Consequently, the first tier definition (lines 12-14) is applied for the first hint track and only View ID 0 will be included in the first RTP session.

The second track is an MVC track (payload name is “H264-MVC” as specified in line 7) which includes media data of tier 2. Consequently, View ID 1 is sent in a separate RTP session. The tool automatically generates appropriate SDP lines “`a=group:DDP V1 V2`”, “`a=mid:V1`”, “`a=mid:V2`”, “`a=depend:97 lay V1:96`” as shown in Table 3 and explained in section 1.2.1.

Table 6 — Example Parameter File for Hint Track Generation

```
Line#
[1]  <?xml version="1.0" encoding="utf-8"?>
[2]  <MediaConfiguration MediaType="hint">
[3]     <TrackConfiguration>
[4]        <TrackDefinition PayloadName="H264">
[5]           <TierList Tiers="1"/>
[6]        </TrackDefinition>
[7]        <TrackDefinition PayloadName="H264-MVC">
[8]           <TierList Tiers="2"/>
[9]        </TrackDefinition>
[10]    </TrackConfiguration>
[11]    <TierConfiguration>
[12]       <TierDefinition>
[13]          <OperationPoint ViewID="0"/>
[14]       </TierDefinition>
[15]       <TierDefinition>
[16]          <OperationPoint ViewID="1"/>
[17]       </TierDefinition>
[18]    </TierConfiguration>
[19] </MediaConfiguration>
```

2.1.2. Generation of Media Presentation Description (MPD)

The MP4FileLibQtApp tool for MP4 file segmentation allows generating segments according to different policies, and the MPD is generated accordingly. In addition to that, the tool offers a number of parameters with respect to the MPD itself, e.g., if the URLs stated in the Presentation elements are constructed using a BaseURL attribute. This is somehow important in the COAST context, because it allows COAST network elements to alter the MPD at a single place in order to redirect the requests to a dedicated server or cache location.

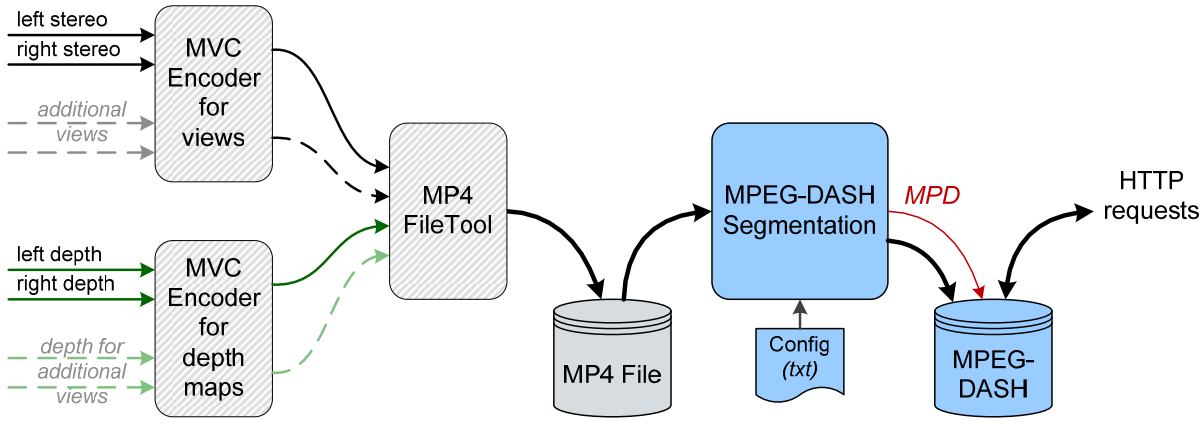

Figure 11 — Generation of MPD signalling information for HTTP streaming

Basic features of the segmentation tool have already been described in D5.1; below a more detailed description is found. The tool offers a GUI shown in Figure 12.

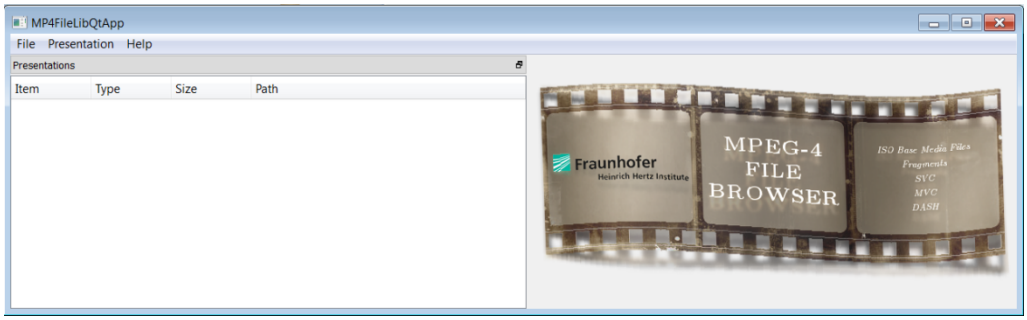

Figure 12 — Main GUI of the Segmentation Tool

When an MP4 file is opened using the "File" dialog (see Figure 13), the different tracks are shown in the left pane (see Figure 14).

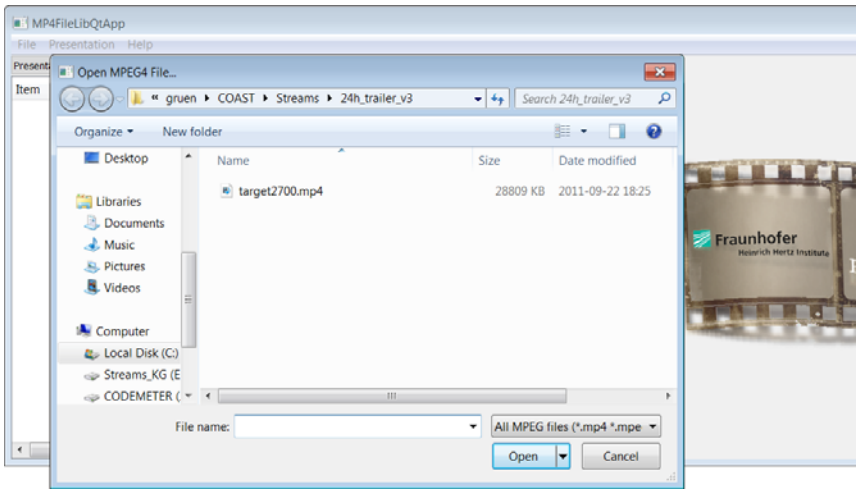

Figure 13 — File Open Dialog of the Segmentation Tool

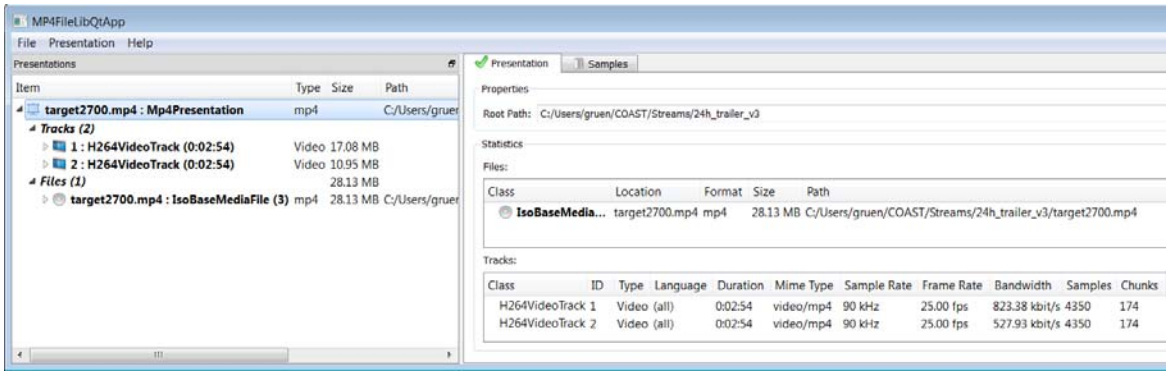

Figure 14 — Main View of the Presentation

Splitting the File into different presentations is initiated from the "Presentation" dialog. Three different tabs allow configuration of the DASH Presentation. On the Presentation Layout tab, tracks can be assigned to different Representations using drag-and-drop (cf. Figure 15).

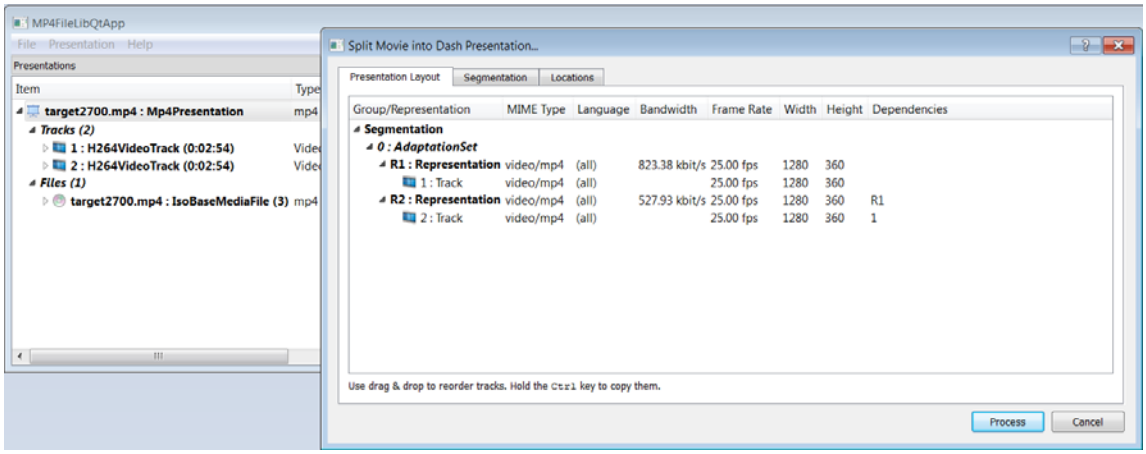

Figure 15 — Presentation Layout Tab

The Segmentation tab allows choosing different options; see Figure 16. If all Subsegments are stored in the same Segment, one or more 'sidx' boxes are inserted to allow access to single Subsegments (cf. section 1.2.2.4). In addition to that, it can be chosen whether Subsegments should be aligned with Stream Access Points. Finally, the Subsegment duration can be chosen. If also "Align Subsegments at RAP" is activated, an arbiter algorithm finds the nearest SAP to the desired Subsegment duration.

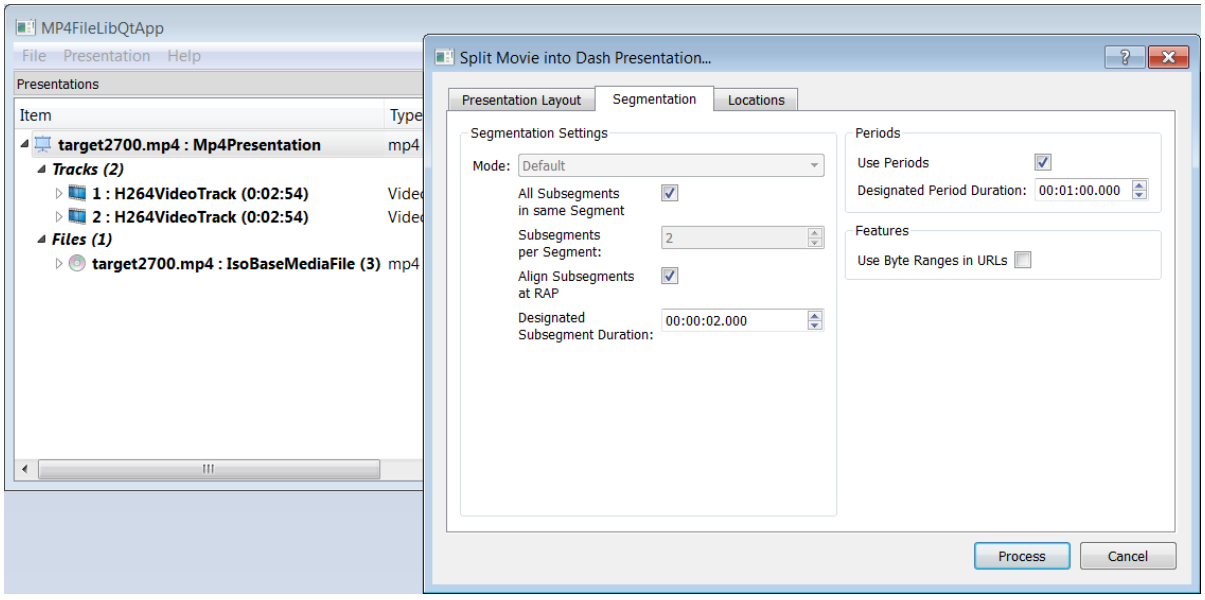

Figure 16 — Segmentation Tab

Figure 17 shows the choices offered by the Locations tab. A major choice with respect to the resulting DASH Presentation is given by the check box “Self-Initializing Segments”. If this box is checked, the Segments include also the ‘moov’ box which is necessary to access the movie fragments. If it is not checked, an extra file is generated which has to be loaded before any other segment can be decoded.

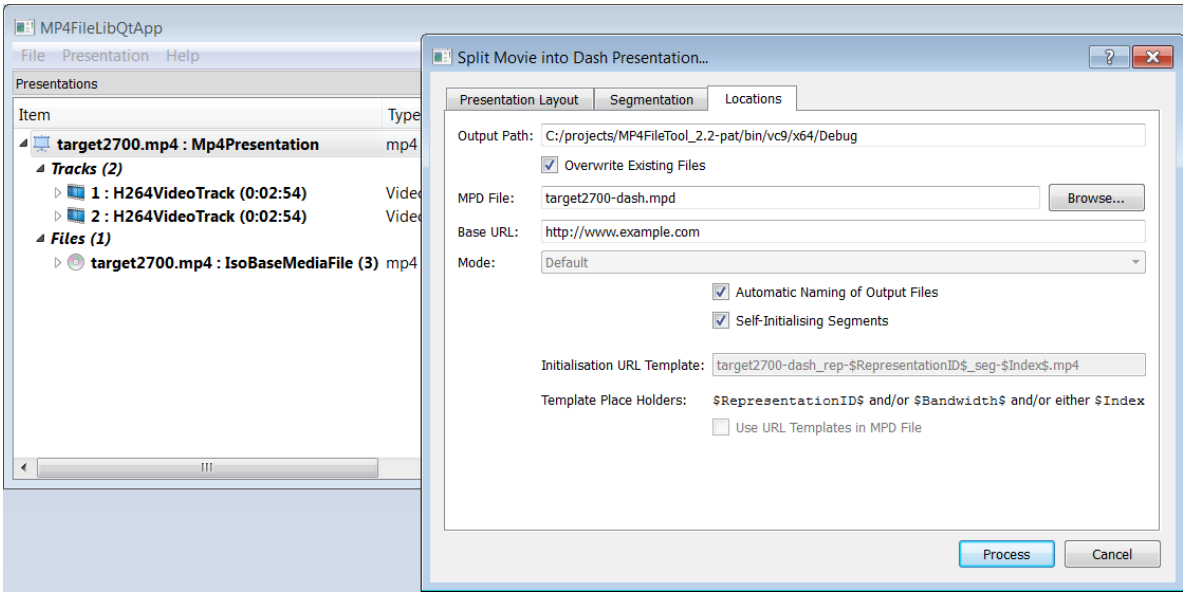

Figure 17 — Locations Tab

2.1.3. Generation of MPEG-2 Transport Stream (TS) segments

Due to the multiplexed structure of the TS, there are some issues to resolve when different views shall be accessible separately. If the server offers only the complete set of views to all clients, this would waste data rate if a client without 3D capabilities of can only use the Base View. Alternatively, TS files with a different number of views could be generated, but this would also waste storage space on the server. A third approach would resolve both issues: Partial TS files could be generated, each containing a certain view, which combined provide the number of views required by a client.

For this, the transport packets from different ES have to be demultiplexed so that segments contain only the data related to e.g. a single view. This corresponds to the notion of complementary representations which has been incorporated in the MPEG-DASH standard (see section 1.2.2). One difficulty is caused by the placement of the PCR time stamps mentioned earlier in this chapter. In the course of demultiplexing to partial TS at the server, data fields carrying the PCR may lose its proper location within the bitstream. A way out of these difficulties is re-multiplexing and, if needed, re-stamping of PCR values.

A first prototype of adaptive HTTP streaming based on TS files has been shown during the Annual Review (5 May 2011). Our implementation of the segment generator relies on some processing of downloaded segments before they are sent to the decoder, i.e., a software decoder rearranges the TS packets retrieved and interleaves them in order to get a timing of the data units that is very close to the original TS.

Further implementations for the COAST project focus on HTTP streaming using MP4 File Format (cf. section 2.2.2).

2.2. HTTP Streaming

In Work Package 5 of the COAST project, we developed an adaptive HTTP streaming client for 3D media compliant with the latest publicly available version of the emerging DASH standard (cf. section 1.2.2) – which has been integrated in the GStreamer open source multimedia framework. In the following sections, the GStreamer framework is introduced and the novel GStreamer plugins implementing DASH client and 3D decoder functionalities are detailed in sections 2.2.1, 2.2.2 and 2.2.3 respectively. Section 2.2.4 describes how such SW modules contributed to the implementation of technical demonstrators featuring 3D adaptive HTTP streaming which were publicly showcased during the Networked & Electronic Media (NEM) Summit held on September 28-29, 2011 in Torino, Italy [36].

2.2.1. GStreamer media framework

GStreamer is a SW framework that can be used to develop block oriented multimedia players. It is composed of several open source libraries (plugins) written in C programming language and released under the term of the GNU Lesser General Public Licence (L-GPL)[33]. The framework is available for multiple platforms; it is mainly used in Linux based operating systems, in particular to develop media player applications for the GNOME desktop environment (Totem, Rhythmbox, etc.). Software releases and documentation are continuously maintained by a wide community of developers and available for download from the GStreamer project web site [34].

GStreamer architecture is composed by a core package and a large set of plugins that extend the basic functionalities and enable processing of several media codec types. According to their features, plugins are grouped in three categories:

- *gst-plugins-good*: set of good quality plugins without license problems
- *gst-plugins-ugly*: set of good quality plugins but that may pose problems with licences
- *gst-plugins-bad*: set of plugins requiring more quality

As depicted in Figure 18, GStreamer plugins are linked together in order to create the so-called pipeline. In a typical pipeline, multimedia data flows from a source (disk, network etc.) to the final audio/video rendering device, through a set of modules that process the data. Each module can have one or more input interfaces (sink) and outputs (source), and can work in pull or push modality according to the functionalities implemented. By linking and mixing GStreamer components it is possible to support a wide range of audio/video formats, codecs, network protocols and video processing algorithms.

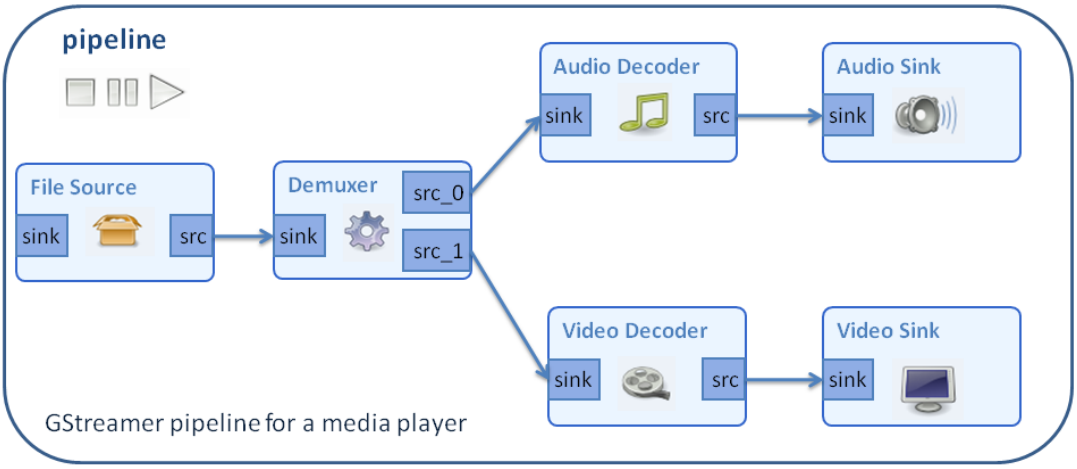

Figure 18 — Typical GStreamer pipeline for a media player

Once the GStreamer system libraries have been installed, pipelines can be tested in a Linux shell, using the *gst-launch* tool in the following way:

```
$ gst-launch-0.10 filesrc location=video.mp4 ! qtdemux ! ffdec_h264 ! xvimagesink
```

where the "!" symbol is used to link the plugins.

GStreamer provides high level elements named "*bins*". "*Bins*" simplify GStreamer-based application programming, as modules and pipelines are selected and created automatically according to the input source type. The most complete and powerful bin is named "*playbin*", which supports all basic media player functionalities. In short, *playbin* is an auto-plugger, an element which accepts input from a source (file or network), detects the protocol and the media type contained in the stream, and automatically selects and links demuxers, decoders, etc. Similarly "*decodebin*" automatically manage the selection of the best audio/video decoder module according to input stream format. Further information about GStreamer can be found in [4].

2.2.2. HTTP Streaming Client Module

In GStreamer the main network protocols stack has been developed on *gst-plugins-good* libraries. In particular, RTP[9] and RTSP[13] protocols have been implemented in a couple of plugins named respectively *gssrtspbin* and *gstrtpmanager*. *gstrtpbin* is a client bin that auto-plugs an internal pipeline for managing RTP sessions and packets arrivals. The RTP sessions can be controlled also using the *gssrtspbin* that implements RTSP specifications. HTTP progressive download of video contents is possible instead through the *souphttpsrc* plugin which is based on the SOUP library, an HTTP client/server library for GNOME [35].

Based on *souphttpsrc* module, in COAST a DASH compliant HTTP adaptive streaming receiver has been implemented. In particular, a new GStreamer "*bin*" was created, named *DASHbin* and implementing the DASH client model described in Figure 4. Similarly to other GStreamer "*bins*" such as *playbin* and *decodebin*, *DASHbin* include several components which are necessary to manage all the features of the adaptive streaming client (Figure 19).

The original *souphttpsrc* GStreamer module downloads the manifest file and each media fragment under the control of the Streaming Control Unit (SCU). When the MPD manifest is received through *souphttpsrc*, the *MPDParser* module parses the information contained. *MPDParser* also triggers all DASH client functionalities after checking whether the XML scheme in MPD file is compliant with DASH specification; as depicted in Figure 19, in case of progressive download of a video file, DASH processing is bypassed and the original *souphttpsrc* functionalities are replicated in DASHbin. The parameters extracted from the MPD (media type, number of groups and representations, etc.) are cached by the SCU and used to drive and temporize the download of the data segments. Segments are then buffered in a *queue* element that is continuously monitored by SCU in order to avoid under-run and guarantee a smooth playback. Data from queue element is then pushed to demuxing and decoder modules.

The SCU includes the Adaptation Logic (AL) sub-system, which is used in case different representations at different bitrates are described in the MPD file; by taking into account Network and Context information, device capabilities and buffer status, the SCU/AL selects the optimal representation, following an optimization model that is under development in Task 5.3. The adaptation decision resulting from SCU/AL is mapped to one representation among those available in MPD file, and the correspondent segments are then downloaded through SOUPhttp.

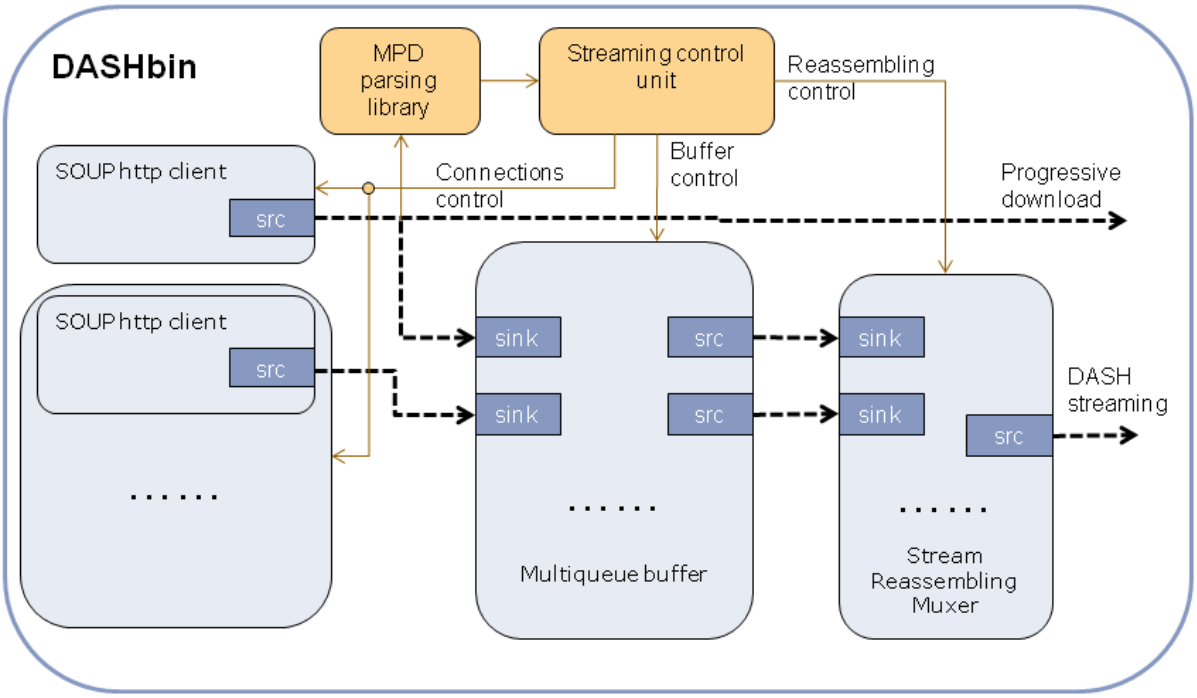

Figure 19 — DASH client structure

In case of MVC contents, the different views described in MPD need to be managed not as alternatives of the same content at different bitrates, but rather as different representations with interdependencies, which need therefore to be downloaded in parallel and composed at the receiver side. In this situation, multiple instances of SOUP http client libraries are created and managed in parallel by the SCU: one for the selected representation and the others corresponding to its various dependencies described in the MPD. Data segments for the representations are downloaded, buffered in different queue buffers and then merged according to dependencies in a Stream Reassemble Muxer (SRM) which finally forwards the composed audio/video flow to the demuxers and decoders.

The *DASHbin* comes in 2 SW packages:

- *gst-plugins-good-0.10.27* is a modified version of the "good" plugin package from GStreamer; it contains an extended version of souphttpsrc supporting DASH streaming;
- *gst-dashbin* is a package containing the newly developed plugin, composed by the DASH client and a library used to parse MPD file.

Being an extension of *souphttpsrc* element *DASHbin* replicates its basic functionalities and properties when a non-MPD file is downloaded. It can be safely used therefore to replace *souphttpsrc* in any GStreamer pipeline.

The other DASH specific properties of *DASHbin* are:

- *location*: can be either an "http://" or "https://" URL pointing to a remote MPD or a local file path starting with "file://";
- *max-bandwidth*: sets the max bandwidth in bit/s allowed for a DASH representation (0 = unlimited); all the available representations in the MPD file will be scanned to find the highest one not exceeding the required bandwidth;
- *upper-threshold*: sets the max number of buffers allowed in the queue; when the limit is reached, the download of further chunks of data will be blocked; data chunks have a small sizes (given by the "blocksize property", defaulting to 4 kB) and do not correspond to DASH segments;

- *output-mpd-file*: file name to save the parsed MPD file; please note that this is not an identical copy of the downloaded MPD file, but a new file generated from the values parsed by *MPDparser*; it can be used for testing purposes;
- *proxy, proxy-id, proxy-pw*: to set a proxy URI, along with the user id and password for authentication.

The following lines describe two typical GStreamer command lines that can be used to receive a standard 2D video streamed from a remote server:

```
gst-launch dashbin location=http://www-itec.uni-klu.ac.at/dash/wp-content/dash/mtbike.mpd
! decodebin name=decoder \ decoder. ! queue ! xvimagesink \ decoder. ! queue ! autoaudiosink
```

```
gst-launch dashbin max-bandwidth=500000 location=http://www-itec.uni-klu.ac.at/dash/wp-
content/dash/mtbike.mpd ! decodebin name=decoder \ decoder. ! queue ! xvimagesink \ decoder.
! queue ! autoaudiosink
```

The complete list of instructions to download, install and test GStreamer developed libraries are reported in **Error! Reference source not found.**.

2.2.3. GStreamer based 3D video streaming receiver

A GStreamer decoder plugin for MVC bitstreams (*mvcdec*) has also been developed in COAST. The plugin wraps MVC decoder library developed in Task 5.1 [3] and allows selection of different output video formats in order to interfaces different types of 3DTV monitors.

The *mvcdec* plugin is a filter plugin; it expects to receive a compressed flow of data in input to produce a sequence of decoded pictures in raw YUV format in output. The input flow need to be compliant with H.264/MVC elementary stream format, and composed by a continuous sequence of NAL units. The input data buffers received by the plugin must contain either full H.264/MVC Access Units or single NAL units. The NAL units can be either in Annex B format (that is, each NAL unit must start with a 3 or 4 byte start-code equal to 0x000001 or 0x00000001) or in MP4 boxed format (that is, each NAL unit is preceded by a NAL length field). Such information is typically signalled in the capabilities of the sink pad of each GStreamer element that receives data.

After decoding one Access Unit, the *mvcdec* module can compose the stereo views in the following way:

side-by-side:	the output views are mapped on the left and right part of the output picture, generating twice the width of a single view;
reverse-side-by-side:	same as *side-by-side*, with reversed left and right views;
line-interleaved:	two views are interleaved one line at a time, generating an output picture which is twice as high as each single view;
reverse-line-interleaved:	same as *line-interleaved*, with reversed left and right views;
above-below:	the left picture is mapped on the upper part of the output picture, and the right one on the bottom (output height is doubled);
below-above:	same as *above-below*, with reversed left and right views;
single-view:	only one view specified by the "output-view" property is displayed.

Once installed, *mvcdec* is also recognized and automatically loaded by *decodebin* when a stream compliant with H.264/MVC video format is received. Some changes in the MP4 demuxer plugin (*QTDemux*) were necessary in order to properly extract the MVC elementary stream embedded in the MP4 file format. The complete set of instructions to download and test the *mvcdec* plugin are given in Annex A.

When inserted in a GStreamer pipeline including DASH receiver, *mvcdec* permits implementation of an adaptive 3D video streaming receiver as shown in Figure 20.

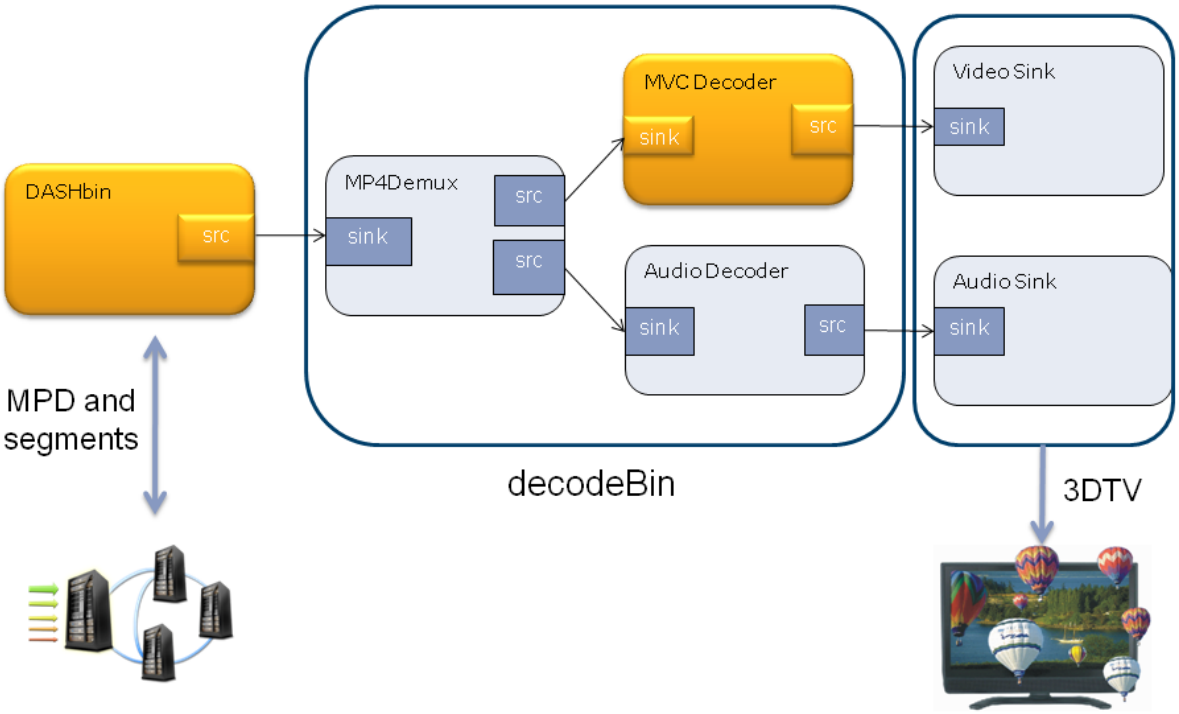

Figure 20 — 3D adaptive streaming receiver

2.2.4. Demonstration System

DASHbin and *mvcdec* GStreamer modules, MVC decoder libraries, the toolset necessary to generate the MP4FF fragmented contents and the correspondent MPD file have been used to develop a technical demonstrators of a 3D adaptive HTTP streaming framework showcased at the NEM summit held in Torino on September 2011 [36], [37], [38].

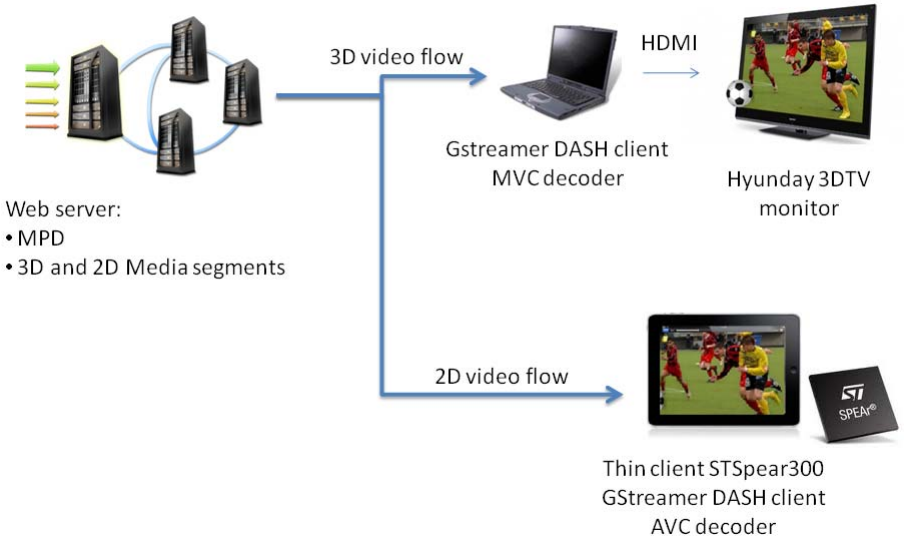

Figure 21 — Demonstration setup

Figure 21 describes the key elements of the demo setup:

- Web server hosting 3D and 2D media segments and the MPD file:
  Standard PC laptop running Ubuntu 10.10 OS and Apache Web Server 2.0;
- GStreamer based 3D adaptive streaming receiver:
  (Dell Precision T7400 desktop, Ubuntu 10.10 OS, GStreamer 0.10.32, GStreamer COAST plugins and MVC decoder library);
  the receiver is connected via HDMI to a 46'' 3DTV monitor (Hyundai S465D);
- A connected tablet:
  ST-Spear1340 ARM based platform [39], running Ubuntu 10.10 OS and GStreamer 0.10.32, plus COAST GStreamer plugin;
  Spear1340 is a general purpose ad low cost dual core ARM platform used to power Multimedia UI for printers, WiFi phones, low cost smartphones and tablets.

Streaming sessions to 3DTV and to ST Spear1340 board start by requesting the URL of the MPD file cached in web server, using the following GStreamer command line:

```
gst-launch –v dashbin location=http://coast-media-server.com/test_sequence.mpd
  ! qtdemux ! queue ! mvcdec output-format=3
  ! ffmpegcolorspace video/x-raw-yuv,format=\(fourcc\)YV12,
  width=1280,height=720,framerate=\(fraction\)25/1,pixel-aspect-ratio=\(fraction\)1/1
  ! videoscale
  ! video/x-raw-yuv,format=\(fourcc\)YV12,
  width=1920,height=1080,framerate=\(fraction\)25/1,pixel-aspect-ratio=\(fraction\)1/1
  ! sdlvideosink fullscreen=true
```

Content cached in web server is available in 3D stereo and 2D formats at different bitrates. By detecting the monitor capabilities (written in a text file for this demo purpose), and matching them with the description of each representation in MPD, the DASH client automatically select 3D representation in MPD for 3DTV receiver, and 2D representation for the Spear board. The DASH receiver is also capable of detecting and reacting to bandwidth fluctuations, exploiting the COAST adaptation algorithms. For demo purpose, bandwidth variations have been simulated using a linux tool for network bandwidth shaping [40].

2.3. Client Module for View Interpolation

Free Viewpoint Video (FVV) is defined as the capability of a rendering engine to dynamically change a viewpoint of a 3D video on user interaction, e.g. by creating new synthesized images from real views and depth information. In general, the quality of depth information is crucial in this scenario, since it may strongly affect the quality of the generated free viewpoint frames. From the point of view of the content fruition, FVV targets the rendering of a continuum of views like a kind of "time-slice effect" (frozen in time) and "bullet-time effect" (hyper-slow-motion). The user would be allowed to control the permutation of time and space (i.e. to move around the scene at a normal speed while events are slowed). The rendering process necessary to obtain this kind of effects is very time consuming, hence a Graphical Processing Unit (GPU)'s hardware acceleration is often needed to cope with such complexity. A GPU is a processor attached to a graphics card dedicated to computing mathematical operations commonly used in graphics rendering and realistic games. Modern GPUs also have support for stereo graphics and typically include digital video related functions. In COAST we developed the algorithmic chain on a common GPU by using the OpenGL ES2.0 libraries 0, and we are integrating this framework in the MVC video streaming receiver described in section 2.2.

The FVV rendering module is described in section 2.3.1, while section 2.3.2 details how such module has been integrated in the GStreamer based MVC receiver.

2.3.1. FVV rendering engine

The FVV rendering engine is responsible for processing views and depth map information and synthesizing the desired 3D image according to user choice.

The FVV rendering process developed is made up of the following steps:

1. *Virtual Camera*: computes the intrinsic and extrinsic parameters of the desired virtual camera
2. *Smart Warping*: projects the source views into the virtual targeted viewpoint and filter depth information
3. *Blending*: merge all the contributions weighted according to the selected viewpoint
4. *Occlusion solving*: filter and solve remaining occlusions
5. *3D Packing*: left and right generated viewpoints are merged together in any frame-compatible format, depending on the available 3D display

Steps 1 and 2 require complex mathematical and geometrical computations while steps from 3 to 5 are important to tune the quality of the rendered frames. In the following sections, further details for each step are given.

2.3.1.1 Virtual Camera

Calibration parameters determine the position, orientation and intrinsic characteristics of each camera. The first issue to solve in a view synthesis application is to find such parameters for the target virtual camera *C*. *C* parameters are classified in intrinsic parameters, i.e. the matrix K_c, and extrinsic parameters, i.e. the translation vector t_c and the rotation matrix R_c expressed with respect to the world reference system *W*.

In case of parallel rectified cameras, interpolating novel viewpoints is generally a matter of translating the virtual camera, as the rotation and the intrinsic matrix are kept equal for all the real cameras. In all the other cases (free camera position), it is necessary to define how to manage the virtual camera's trajectory; to address this situation a viewpoint *v* is defined as a set of seven parameters:

(Eq. 1): $v = <latitude, longitude, radius, roll, pitch, yaw, zoom>$

With this approach, the scene is in the center of a sphere and the virtual camera is located on an imaginary sphere's surface, at "geographical" coordinates specified by <*latitude*,*longitude*, *radius*>, oriented as described by the contribution of <*latitude*,*longitude*> plus <*roll, pitch, yaw*> angles, and set to a zoom factor identified by <*zoom*>.

By using such parameters it is possible for the user to move the viewpoint wherever in the 3D space, rotate the camera about its main axes, move close to or far from the scene (thus changing the perspective), or even zoom in or out of the scene (without changing the perspective).

The intrinsic matrix K_c can be evaluated as

- Linear interpolation from nearest cameras
- Average of all the real cameras parameters
- User specification

Computation of extrinsic parameters is a bit more complicated. Since the rotation matrix has to be orthonormal, it is necessary to compute the intermediate rotation matrices by a geometrical method that preserves this property. Literature suggests using a method derived from the theory of quaternion curves, named spherical linear interpolation (Slerp) [42].

Finally, once the rotation matrix of a specified viewpoint has been computed, the translation vector that describes the new position of the virtual camera can be obtained by means of a quaternion-vector multiplication.

In case of 3D output, to generate a three-dimensional view of the desired viewpoint, two virtual cameras have to be computed, one left and one right. Following the general "off-axis" stereoscopic principle for a better 3D view, the position of these two virtual cameras has to be computed as if the two cameras were parallel, and with a proper baseline (i.e. the distance between the cameras, e.g. 6.5 cm) to get a comfortable 3D viewing experience.

The user acts on two parameters to get the desired 3D effect:

- 3D Gain: the amount of 3D effect, e.g. ipo-stereo vs. iper-stereo;
- 3D Offset: the position of the 3D content along the depth axis, to make the scene pop in or out of the screen.

These values affect properly the virtual camera's parameters and so their projection matrices, thus leading to the final 3D view set by the user.

2.3.1.2 Smart warping

Smart warping is the process necessary to project the source views into the target viewpoint; this procedure requires the adoption of a particular geometrical model, named Pinhole Camera Model (PCM), plus the filtering of the depth information.

The PCM takes into account the information about intrinsic and extrinsic parameters and defines the so-called projection matrix as:

(Eq. 2): $$P_c = K_c[R_c|t_c]$$

The projection matrix allows definitely to find the 2D pixel point *m* at frame coordinates (*u*,*v*) in the camera reference system *C*, correspondent to the 3D point *M* located at three-dimensional coordinates (*x*,*y*,*z*) in world reference system *W* (and vice versa):

(Eq. 3): $$\alpha\, m_c = P_c\, M_w$$

where α is a scaling factor.

Since the PCM requires knowing the depth of each pixel, a per-sample depth map for each camera is needed. Programmable pipelines like the OpenGL 2.0, do not provide fixed operations on matrices and vectors like fixed graphic pipelines do. On the contrary, every operation among matrices and vectors can be easily written (programmed) and executed by means of shaders. Hence, once computed the projection matrix of the virtual camera and the inverse projection matrices of the selected source cameras (the nearest two), they have to be passed to the vertex shader which performs the warping step. We choose points as geometry elements to warp the image. The vertex shader receives the set of 2D coordinates corresponding to the pixels' position in the image, while the third coordinate (i.e. the depth *z*) can be passed to the GPU by texture lookup if the vertex shader has this capability. This operation is still OpenGL ES 2.0 compliant, but some implementations may not allow for this feature, so that the depth data has to be passed as vertex set (slower).

Let's consider a stereo sequence ("Breakdancers" sequence provided by Microsoft [43]). Suppose to set the camera C_3 as target viewpoint and project one camera for each side, i.e. C_2 (right) and C_4 (left). As can be seen in Figure 22, a frame of each source view is projected into the target viewpoint leaving unfilled many areas (painted green), due to corresponding occluded areas in the source frames. Moreover, a part of these unfilled pixels are due to integer approximation when rounding the position of the projected pixel, but this can be solved in the next step.

For some sequences, it may happen that the quality of the depth maps is not so good for the projection and this can lead to annoying artifacts around borders. In this case, before warping the image we perform a pre-filtering step capable of identifying the pixels that are likely to produce those artifacts and prevent them from being projected. This is done by a fragment shader that runs an edge-detector filter which marks only the background pixels as "not-to-be-projected", with strength customizable by the user.

Figure 22 – Projection of the cameras C_4 (a) and C_2 (b) to the C_3's viewpoint

2.3.1.3 Blending

Blending aims at merging the projected frames. In general, most unfilled pixels in the left projected view are filled in the right projected view, and vice versa. Different functions can be used to mix the frames, and the better one is probably to assign a view with a weight inversely proportional to its distance to the virtual viewpoint. All the distance-based methods, anyhow, allow for smoother transitions between the synthesized views while rendering free-viewpoint camera path.

2.3.1.4 Occlusion solving

To fill the remaining gaps, one of the best approaches is to search each occlusion boundary for the real background and paint the whole gap with the same colour. To simplify the step and make it running on a programmable GPU where the same kernel is executed on all the fragments at the same time, for each pixel of an occlusion we search the farthest pixel (i.e. with the smallest depth) among the eight neighbouring pixels and along with the four main directions (i.e. up, down, left, right). At last, we assign the colour of the identified pixel to the current one. Due to the local approach, this may propagate the colours to the pixels of an occlusion in a striped-like way, but the final effect looks quite realistic. It is worth noticing that we need the depth information of the final frame to perform this search and the complexity is kept low. Hence, during the warping step the two depth buffers are rendered to frames and then blended together.

2.3.1.5 3D Packing

To generate a three-dimensional view of the desired viewpoint the previous four steps have to be done twice, one for the left and one for the right virtual position. Please note that, in case of frame-compatible formats, one could even render the views at half the original resolution to benefit of some reduction in terms of complexity. After being generated, the two views are then packed together in any of the supported frame-compatible formats listed in Section 2.2.3, to fit the available 3D display.

2.3.2. FVV integration in GStreamer receiver

In order to add support for FVV rendering in the 3D streaming demonstrator described in section 2.2.4, in COAST we decided to integrate the FVV library in the GStreamer pipeline. This operation can be easily performed by using an existing GStreamer plugin, specifically designed to integrate OpenGL modules in the pipeline 0.

The resulting integrated SW is shown in Figure 23. The media flow downloaded from the web server is at least composed by two views and two depth flows; views and depth components are extracted by *mvcdec* and forwarded to the FVV library integrated in *gstOpenGL* plugin. The output from gstOpenGL is then forwarded to the GStreamer video sink for displaying on 3D TV monitor.

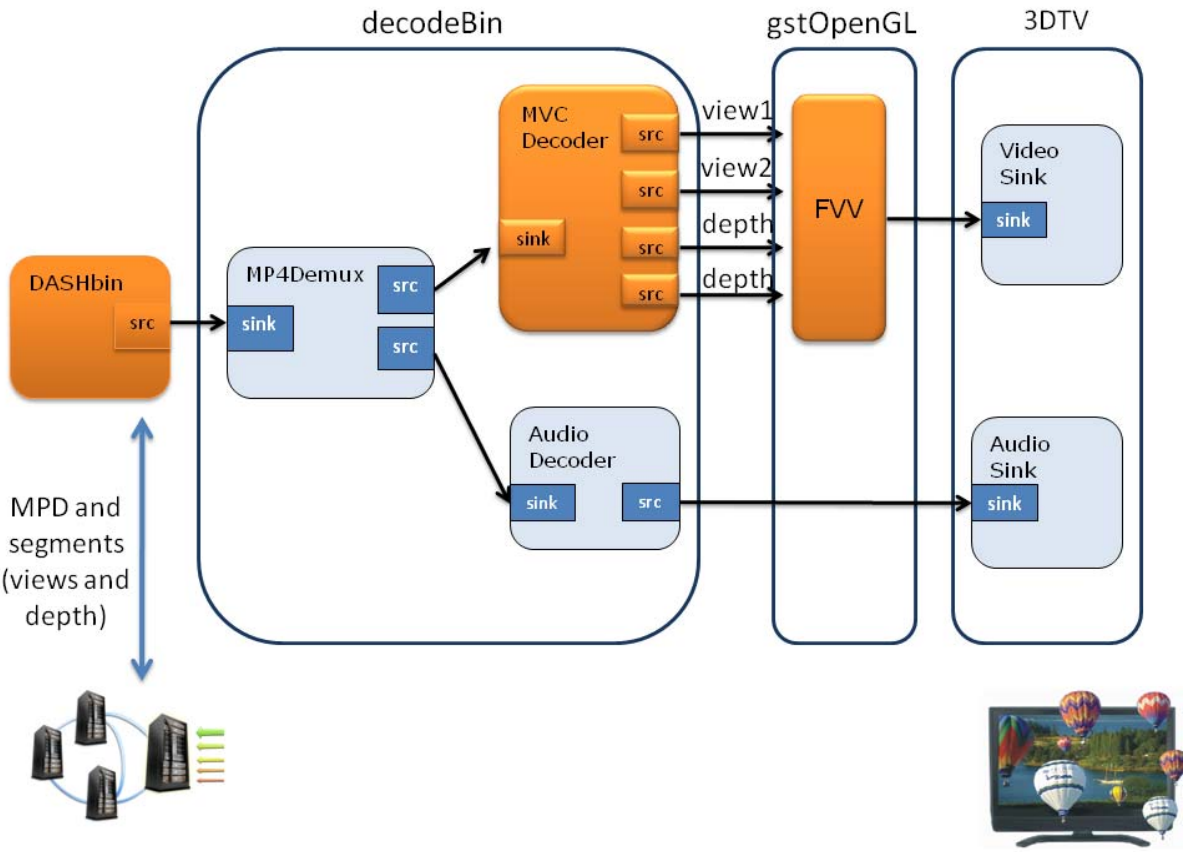

Figure 23 – FVV module integration in GStreamer 3D video receiver

The user can control the FVV rendering process by means of mouse and keyboard, both in terms of video playback and 3D viewpoint position; in particular the following interactions are possible:

- P key: Pause/Play the video;
- +/- keys: Zoom In/Zoom Out;
- S key: Enable/Disable 3D;
- F key: Switch among the 3D supported formats;
- Left mouse button + mouse movement: change the Viewpoint;
- Right mouse button + mouse movement: change 3D Gain;
- Center mouse button + mouse movement: change 3D Offset.

All these interactions do not affect the performance of the 3D rendering engine.

3. Conclusion

This document describes and specifies multiple methods of media signalling for multiple transport protocols which allow for transmission of novel 3D media formats. It enables communication between media server and client about the format and codec related features of the media streams. By this means, a client decides whether it can decode a media stream, and if so, chooses the proper decoder for that media stream.

This signalling differs with respect to the chosen codec; in this document, we focus on the state-of-the-art MVC codec which can be used for the compression and transmission of stereoscopic views as well as for multi-view processing of more than two views depending on the profile. We also address frame compatible stereo which poses different signalling issues; here, side information is conveyed which is not included in the encoded stream itself, namely the fact that two views are packed into a single video sequence. Moreover, the information how to separate the views after decoding has to be signalled to the rendering device. Finally, signalling for the transmission of depth maps is discussed and the use of depth maps for the rendering of intermediate views is described in detail.

Most signalling aspects also depend on the transmission protocol. In this project, the consortium has decided to concentrate on HTTP streaming, because we expect benefits related to the caching of media segments. Consequently, this document focuses on signalling aspects related to adaptive HTTP streaming, specifically on the emerging MPEG-DASH standard.

One basic observation is that the information whether two video streams or two views within an MVC bitstream form a stereoscopic pair of a 3D video transmission is not necessarily included in the video bitstream(s) itself.

The result of this work package is two-fold: On the one hand, we present differentiated specifications for the transmission of 3D video streams; on the other hand, we supply tools that provide and use such signalling. One authoring tool described in section 2.1 has already been used to generate test streams for a first public presentation during the NEM Summit (cf. section 2.2.4). The MPEG-DASH client application (cf. sections 2.2.2, 2.2.3) has also been shown at that occasion. Finally, it will be integrated in the COAST platform as part of WP7. The authoring tools will further be utilized for the generation of test material for the validation of the COAST platform.

References

[1] COAST Consortium, “COAST Description of Work,” Approved by EC on 13 November 2009

[2] FP7 Cooperation Work Programme: Information and Communication Technologies, ftp://ftp.cordis.europa.eu/pub/fp7/ict/docs/ict-wp-2009-10_en.pdf

[3] COAST Consortium, Deliverable D5.1, “File Format Specification and 3D video codec,” http://coast-fp7.eu/deliverables.html

[4] COAST Consortium, Deliverable D7.1, “Specification of the COAST service platform,” http://coast-fp7.eu/deliverables.html

[5] COAST Consortium, Deliverable D8.1, “Initial Assessment and Market Analysis,” http://coast-fp7.eu/deliverables.html

[6] T. Schierl, K. Grüneberg, and T. Wiegand, “Scalable Video Coding (SVC) over RTP and MPEG 2 Transport Stream in Broadcast and IPTV channels,” IEEE Wireless Communications Magazine, vol. 16, no. 5, pp. 64-71, October 2009

[7] Karsten Müller, Philipp Merkle, and Thomas Wiegand, “3-D Video Representation Using Depth Maps,” Proceedings of the IEEE, vol. 99. no. 4, pp 643–656, April 2011

[8] P. Kauff, N. Atzpadin, C. Fehn, M. Müller, O. Schreer, A. Smolic, and R. Tanger, “Depth map creation and image based rendering for advanced 3DTV services providing interoperability and scalability,” Signal Processing: Image Communication, Special Issue on 3DTV, vol. 22, no. 2, pp. 217–234, Feb. 2007.

[9] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications,” Internet Engineering Task Force (IETF) RFC 3550, February 2005, http://tools.ietf.org/html/rfc3550

[10] Internet Assigned Numbers Authority (IANA), “Real-Time Transport Protocol (RTP) Parameters,” September 2011 (regularly updated), http://www.iana.org/assignments/rtp-parameters

[11] Y.-K. Wang, R. Even, T. Kristensen and R. Jesup, " RTP Payload Format for H.264 Video,” Internet Engineering Task Force (IETF) RFC 6184, May 2011, http://tools.ietf.org/html/rfc6184

[12] S. Wenger, Y.-K. Wang, T. Schierl and A. Eleftheriadis, " RTP Payload Format for Scalable Video Coding,” Internet Engineering Task Force (IETF) RFC 6190, May 2011, http://tools.ietf.org/html/rfc6190

[13] Y.K. Wang, T. Schierl and R. Skupin, "RTP Payload Format for MVC Video,” IETF Draft — work in progress, September 2011, http://tools.ietf.org/html/draft-ietf-payload-rtp-mvc-01

[14] M. Handley, V. Jacobson, C. Perkins, " SDP: Session Description Protocol,” Internet Engineering Task Force (IETF) RFC 4566, July 2006, http://tools.ietf.org/html/rfc4566

[15] T. Schierl and S. Wenger, "Signaling Media Decoding Dependency in the Session Description Protocol (SDP),” Internet Engineering Task Force (IETF) RFC 5583, July 2009, http://tools.ietf.org/html/rfc5583

[16] G. Camarillo, G. Eriksson, J. Holler, H. Schulzrinne, “Grouping of Media Lines in the Session Description Protocol (SDP),” Internet Engineering Task Force (IETF) RFC 3388, December 2002, http://tools.ietf.org/html/rfc3388

[17] H. Schulzrinne, A. Rao, R. Lanphier, "Real Time Streaming Protocol (RTSP),” Internet Engineering Task Force (IETF) RFC 2326, April 1998, http://tools.ietf.org/html/rfc2326

[18] D. Hoffman, G. Fernando, V. Goyal, and M. Civanlar, "RTP Payload Format for MPEG1/MPEG2 Video," Internet Engineering Task Force (IETF) RFC 2250, January 1998, http://tools.ietf.org/html/rfc2250

[19] R. Gellens, D. Singer, P. Fröjdh, "The 'Codecs' and 'Profiles' Parameters for "Bucket" Media Types," Internet Engineering Task Force (IETF) RFC 6381, August 2011, http://tools.ietf.org/html/rfc6381

[20] 3rd Generation Partnership Project, Technical Specification and system Aspects, Transparent end-to-end Packet switched Streaming Service (PSS); Progressive Download and Dynamic Adaptive Streaming over HTTP (3GP-DASH), release 10

[21] ISO/IEC JTC1/SC29/WG11, Information technology — MPEG video technologies — MPEG C Part 3: Representation of auxiliary video and supplemental information, October 2007

[22] ISO/IEC JTC1/SC29/WG11, Information technology — Coding of audio visual objects — MPEG 4 Part 10: Advanced Video Coding, ISO/IEC 14496 10:2010 (6th Ed.), December 2010

[23] ISO/IEC JTC1/SC29/WG11, Information technology — Coding of audio visual objects — MPEG 4 Part 12: ISO base media file format, ISO/IEC 14496 12:2008 (3rd Ed.), October 2008

[24] ISO/IEC JTC1/SC29/WG11, Information technology — Coding of audio visual objects — MPEG 4 Part 12: ISO base media file format, Amendment 1: General improvements including hint tracks, metadata support and sample groups, ISO/IEC 14496 12:2008/AMD1, January 2011

[25] ISO/IEC JTC1/SC29/WG11, Information technology — Coding of audio visual objects — MPEG 4 Part 12: ISO base media file format, Amendment 3: DASH support and RTP reception hint track processing, ISO/IEC 14496 12:2008/FDAM3 (under ballot until 2011-12-07)

[26] ISO/IEC JTC1/SC29/WG11, Information technology — Coding of audio visual objects — MPEG 4 Part 15: Advanced Video Coding (AVC) file format, ISO/IEC 14496 15:2010 (3rd Ed.), 2010-06-01

[27] ISO/IEC JTC1/SC29/WG11, Information technology — Coding of audio visual objects — MPEG 4 Part 15: Advanced Video Coding (AVC) file format, Amendment 3: File format support for Multiview Video Coding, ISO/IEC 14496 15:2004/AMD3, 2009

[28] ISO/IEC JTC1/SC29/WG11, Information technology —Dynamic adaptive streaming over HTTP (DASH) — Part 1: Media presentation description and segment formats, ISO/IEC 23009-1 (2nd DIS under ballot until 2011-11-07) [5]

[29] ITU-T Rec H.222.0 (05/2006) Information technology – Generic coding of moving pictures and associated audio information: Systems | ISO/IEC 13818-1:2007 Information technology – Generic coding of moving pictures and associated audio information (MPEG-2) – Part 1: Systems, 2006

[30] ITU-T Rec H.262.0 (02/2000) Information technology – Generic coding of moving pictures and associated audio information: Video | ISO/IEC 13818-2:2000 Information technology – Generic coding of moving pictures and associated audio information (MPEG-2) – Part 2: Video, 2000

[31] T. Schierl, B. Berthelot, T. Wiegand (edts.); "ISO/IEC 13818-1:2007/AMD3 –Transport of SVC video over ITU-T Rec H.222.0 | ISO/IEC 13818-1," 2009

[5] The emerging MPEG-DASH standard is publicly available at: http://www.3gpp.org/LiaisonsDocs/Incoming_LSs/S4-meeting.htm

[32] GStreamer Open Source Multimedia Framework, http://gstreamer.freedesktop.org

[33] GNU Lesser General Public Licence, http://www.gnu.org/copyleft/lesser.html

[34] GStreamer open source and multimedia framework, http://gstreamer.freedesktop.org/

[35] SOUP HTTP client/server library for GNOME, http://live.gnome.org/LibSoup

[36] Networked and Electronic Media (NEM) Summit 2011, http://nem-summit.eu/

[37] STMicroelectronics Press Release, http://www.st.com/internet/com/press_release/t3217.jsp

[38] Fraunhofer Press Release, http://www.hhi.fraunhofer.de/index.php?id=4100

[39] http://www.st.com/internet/com/SALES_AND_MARKETING_RESOURCES/MARKETING_COMMUNICATION/FLYER/FLSPEAR1340511.pdf

[40] GNU Linux Traffic Control, http://tldp.org/HOWTO/Traffic-Control-HOWTO/

[41] The Industry's Foundation for High Performance Graphics, OpenGL, http://www.opengl.org/

[42] K. Shoemake, “Animation Rotation with Quaternion Curves”, *ACM SIGGRAPH*, San Francisco, USA, July 1985

[43] Microsoft@Research, http://research.microsoft.com/ivm/3DVideoDownload/

OpenGL GStreamer plugin, http://gstreamer.freedesktop.org/data/doc/gstreamer/head/gst-plugins-gl-plugins/html/