

Deliverable

Project Acronym:	ImmersiaTV
Grant Agreement number:	688619
Project Title:	<i>Immersive Experiences around TV, an integrated toolset for the production and distribution of immersive and interactive content across devices.</i>

D3.1 Design Architecture

Revision: 0.9

Authors:

Joan Llobera (i2CAT)
Xavier Artigas (i2CAT)
David Cassany (i2CAT)
Maciej Glowiak (PSNC)
Szymon Malewski (PSNC)
Maciej Strozyk (PSNC)

Delivery date: M04

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement 688619		
Dissemination Level		
P	Public	x
C	Confidential, only for members of the consortium and the Commission Services	

Abstract: This deliverable defines the core technical concepts of ImmersiaTV project, it lays out the global architecture of the different software and hardware modules as well as the standards that will be used in the scope of the project. It establishes the architecture and the interfaces of the various functional blocks that comprises the immersive system. Functionality of the ImmersiaTV project is based on several modules that implement particular functions required for video acquisition, video processing and encoding, video production, content delivery and reception and display. All the main components are described in details in order to provide the reference for implementation and integration activities in the project. Finally, the Quality of Experience methodology and test scenarios are pointed out to support and formalize project results validation procedure.

The document provides overall system architecture but it is focused on Pilot 1 requirements collected and analysed in Deliverable D2.3.

REVISION HISTORY

Revision	Date	Author	Organisation	Description
0.1	8 Feb 2016	Llobera, Artigas, Cassany	I2CAT	First contribution of design containing metatata definition, hbbtv synchronization, user interfaces and transmission model
0.2	12 Apr 2016	Glowiak, Malewski, Strozyk	PSNC	Integration of partners' contribution for all tasks and modules
0.3	19 Apr 2016	Valente, Baari	VideoStitch, iMinds-ETRO	Description of Capture, QoE, Codecs and Production tools
0.4	20 May 2016	Glowiak, Malewski, Strozyk	PSNC	Refinement and restructuring
0.5	3 Jun 2016	Ebrahimi, Baari, Valente, Kelembet	EPFL, iMinds-ETRO, VS, CINEGY	Additional contribution added
0.6	9 Jun 2016	Glowiak, Malewski	PSNC	Final editing, integration of remaining text
0.7	10 Jun 2016	Pau Pamplona	i2CAT	Format review
0.8	8 Jul 2016	Luk Overmiere	VRT	Review
0.9	20 Jul 2016	Maciej Glowiak, Pau Pamplona	PSNC, i2CAT	Closing Review

Disclaimer

The information, documentation and figures available in this deliverable, is written by the **ImmersiaTV** (*Immersive Experiences around TV, an integrated toolset for the production and distribution of immersive and interactive content across devices*) – project consortium under EC grant agreement H2020 - ICT15 688619 and does not necessarily reflect the views of the European Commission. The European Commission is not liable for any use that may be made of the information contained herein.

Statement of originality:

This document contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

EXECUTIVE SUMMARY

This deliverable defines the core technical concepts and general architecture of the ImmersiaTV project for implementation and development purposes. Based on the state-of-the-art analyses, scope of the project pilots and requirements defined in D2.3 all partners of WP3 worked together and performed various actions to define the different modules and functionalities as well as interactions between these modules. The scope of this document is mainly related to pilot 1 and consequently the general architecture presented in Chapter 2 is focused on the requirements for off-line content acquisition, post-production and distribution. Besides this, the general architecture takes already into account to some extent the planned live scenarios for next pilots and makes the system ready for further extensions. The same accounts for other sections and module definitions, where possible, the document includes already some details regarding next planned Pilots and live scenario.

After the global architecture is defined and depicted, all the modules are characterised, defined and elaborated. First, in Chapter 2.1 the capture and stitching process is described, containing selected camera systems, their capabilities and possible usage in the different Pilots. This chapter also contains requirements for stitching software input and output as well as the architecture and functionality of the omnidirectional video acquisition and processing.

Chapter 2.2 describes the encoding process as well as contains a list of H.264 codec parameters agreed between modules and tools. The codec definition is a result of discussions between partners and the outcome from internal workshop that took place in Porto 21-23 April 2016.

Chapter 2.3 provides information on ImmersiaTV production tools. It concentrates on an off-line post production process for synchronized and interactive multi-platform 360° content across multiple devices using Adobe After Effects and Adobe Premiere. A basic definition of live production workflow is also provided, which is still under discussion and development.

Chapter 2.4 is dedicated to content distribution and describes the DASH concept that will be used in the ImmersiaTV distribution system for off-line and live workflows.

Chapter 2.5 provides information on the reception and display side, where multiple streams are synchronously transported to end-users' devices such as Head Mounted Display, tablets and TVs. The solution of synchronisation relies on HbbTV 2.0 concepts and describes how the different types of content (omnidirectional and directive) can be synchronized across devices and how interaction between both omnidirectional and directive content can be realised in an immersive display.

Finally, the Chapter 2.6 describes the Quality of Experience methodology and test scenarios. They are intended to support and formalize project results validation procedure.

CONTRIBUTORS

First Name	Last Name	Company	e-Mail
Wojciech	Kapsa	PSNC	kapsa@man.poznan.pl
Stephane	Valente	VIDEOSTITCH	stephane@video-stitch.com
Phillipe	Bekert	iMinds-EDM	philippe.bekaert@iminds.be
Adriaan	Barri	iMinds-ETRO	abarri@etro.vub.ac.be
Saeed	Mahmoudpour	iMind-ETRO	smahmoud@etro.vub.ac.be
Alexandr	Kelembet	CINEGY	kelembet@cinergy.com
Touradj	Ebrahimi	EPFL	touradj.ebrahimi@epfl.ch
Luk	Overmiere	VRT	Luk.Overmeire@VRT.BE

CONTENTS

Revision History.....	1
Executive Summary.....	3
Contributors.....	4
Contents.....	5
Table of Figures.....	7
List of TABLES.....	7
List of acronyms.....	8
1. Introduction.....	9
1.1. Purpose of this document.....	9
1.2. Scope of this document.....	9
1.3. Status of this document.....	9
1.4. Relation with other ImmersiaTV activities.....	10
2. Immersiatv system architecture.....	11
2.1. Capture and Stitching.....	13
2.1.1. Description.....	13
2.1.2. Interfaces.....	14
2.1.2.1. Camera rigs.....	14
2.1.2.2. VideoStitch stitching systems.....	15
2.1.3. Architecture.....	16
2.1.4. Workflow.....	17
2.2. Encoding.....	19
2.2.1. Description.....	19
2.2.2. Interfaces.....	19
2.2.3. Codec definition.....	20
2.2.4. Regions of interest.....	20
2.3. Production Tools.....	21
2.3.1. Description.....	21
2.3.2. Interfaces.....	22
2.3.3. Architecture.....	22
2.3.3.1. Synchronization.....	22
2.3.3.2. Portal video effect.....	23
2.3.3.3. Visualisation.....	24
2.3.3.4. ImmersiaTV package export.....	25

2.3.4.	Live content edition architecture.....	25
2.4.	Content Distribution.....	26
2.4.1.	Description	26
2.4.2.	Interfaces.....	26
2.4.3.	Architecture.....	26
2.5.	Reception, Interaction and Display	28
2.5.1.	Introduction.....	28
2.5.2.	Interfaces.....	29
2.5.2.1.	Metadata.....	29
2.5.3.	Software architecture	32
2.5.4.	Session management device	32
2.5.4.1.	Discovery	33
2.5.4.2.	Session Server.....	33
2.5.4.3.	Synchronization	34
2.5.4.4.	Receiver devices	34
2.5.4.5.	Session Client.....	35
2.5.4.6.	Metadata Reception.....	35
2.5.4.7.	Access control.....	35
2.5.4.8.	Data logging.....	35
2.6.	Quality of Experience	41
2.6.1.	Description	41
2.6.1.	Interfaces.....	41
2.6.1.1.	Logging information sent to QoE module	41
2.6.1.	Implementation Plan of QoE	43
3.	CONCLUSIONS	45

TABLE OF FIGURES

Figure 1: Relationship between different tasks	10
Figure 2: The general architecture of ImmersiaTV System designed for Pilot 1	11
Figure 3: The architecture of the modules of the ImmersiaTV system.	12
Figure 4: The architecture of creating off-line and live omnidirectional video streams in the ImmersiaTV system.	17
Figure 5: Details of capture and stitching workflow using VideoStitch Studio and Vahana VR. 18	
Figure 6: Schema of proposed editor workflow	22
Figure 7: Composition of the off-line production tools.	22
Figure 8: Live content edition workflow	25
Figure 9: Connectivity of devices in home network.....	29
Figure 10: Sample ImmersiaTV metadata file.....	31
Figure 11: Software architecture of a player	32
Figure 12: Logging module architecture	37
Figure 13: Logging sequence of Logging Module.....	38
Figure 14: Retrieving sequence of Logging Module.....	39
Figure 15: QoE Implementation Plan Overview.....	44

LIST OF TABLES

Table 1: Comparison of camera systems planned to be used for ImmersiaTV pilots.	15
Table 2: Off-line recording configuration	20
Table 3: <i>Portal video effect</i> parameters.....	24
Table 4: Preview mode parameters	25
Table 5: Example of an XML format of the logging information sent as input to the QoE module.	43

LIST OF ACRONYMS

Acronym	Description
HMD	Head Mounted Display
DASH	Dynamic Adaptive Streaming over HTTP
WP	Work Package
MPD	Media Presentation Description

1. INTRODUCTION

1.1. Purpose of this document

This deliverable documents in detail the architecture design for omnidirectional content creation, processing, transmitting and displaying in ImmersiaTV as investigated in Task 3.1. Platform design and architecture. The outcome of this task is a complete structured description of all the different components, the global architecture with diagram and workflows that will be implemented by all the tasks of WP3 (T3.2-T3.8).

1.2. Scope of this document

The objective of task T3.1 is to design an architecture for the overall ImmersiaTV system that will form the basis for the implementation and integration of all the software and hardware components. This document focuses on delivery of an architecture for pilot 1 and defines the full process chain of capturing, processing, encoding, content distribution up to users' display following the objectives of WP3, which consist of:

- to design a reliable and robust system architecture of the hardware and software platform and facilitate a smooth integration of all the project technical components.
- to design, set up and deploy an omnidirectional camera system capable of capturing off-line video. Capturing live high resolution high frame rate is the subject of further implementation for Pilot 2 and 3.
- to design and implement a real-time process to effectively encode multiple images from cameras into full omnidirectional video
- to design and implement the required functionalities to adapt the existing production tools to omnidirectional inputs and cross-device visualization and interaction.
- to design and implement the communication servers required to distribute omnidirectional content (incl. live stream) to remote users through existing and next generation access networks efficiently
- to design and implement the clients and libraries required to display omnidirectional video-based productions across devices (TV, second screen and HMD) maintaining coherence, synchronization, and responsivity in LAN environments.
- to integrate and test the different components in an end-to-end pilot and validate it in lab conditions.
- to document the overall process for other researchers and developers as well as produce lab demonstrations.

1.3. Status of this document

This is an intermediate version of D3.1 with delivery foreseen in M3. Other versions of this document will be delivered in M11 and M20.

1.4. Relation with other ImmersiaTV activities

The relationship between this task and the other WP3 tasks and relevant WP2 and WP4 tasks is shown below on Figure 1.

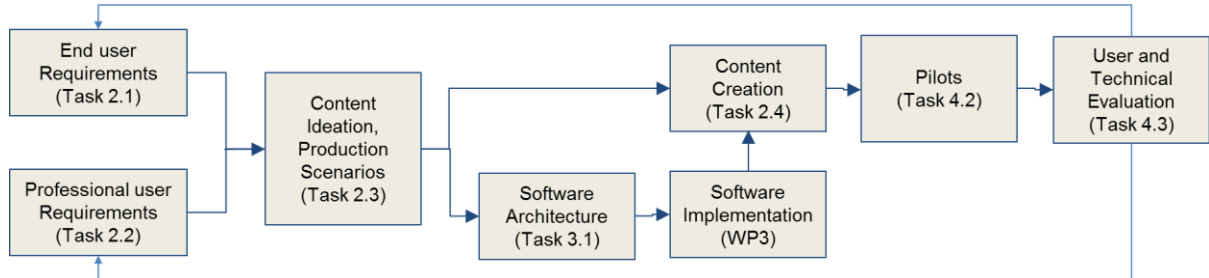


Figure 1: Relationship between different tasks

2. IMMERSIATV SYSTEM ARCHITECTURE

ImmersiaTV aims to distribute omnidirectional and directive audiovisual content simultaneously to head mounted displays (HMD), companion screens and the traditional TV.

The content distributed is constituted of one or more omnidirectional videos, complemented with several directive shots, and metadata detailing how to merge these streams in an immersive display, as well as how to select portions of the omnidirectional stream for traditional TVs and tablets.

The current development plan focuses on the implementation of the tools and modules required to demonstrate Pilot 1. In this offline scenario omnidirectional and directive streams will be captured, processed and aligned by editor using Adobe Premiere Plugin in order to prepare multi-platform omnidirectional view containing embedded portals to two-dimensional video streams. Parts of omnidirectional scene captured by several cameras will be processed and stitched together using VideoStitch Studio.

As the result of offline production action, several H.264 video files and metadata will be generated, transferred to the DASH server and then streamed to the end user's device. The user will need to run a dedicated player in order to display omnidirectional content on their HMD or tablet device as well as directive view on a TV set. All components required to achieve goals of Pilot 1 are depicted on Figure 2.

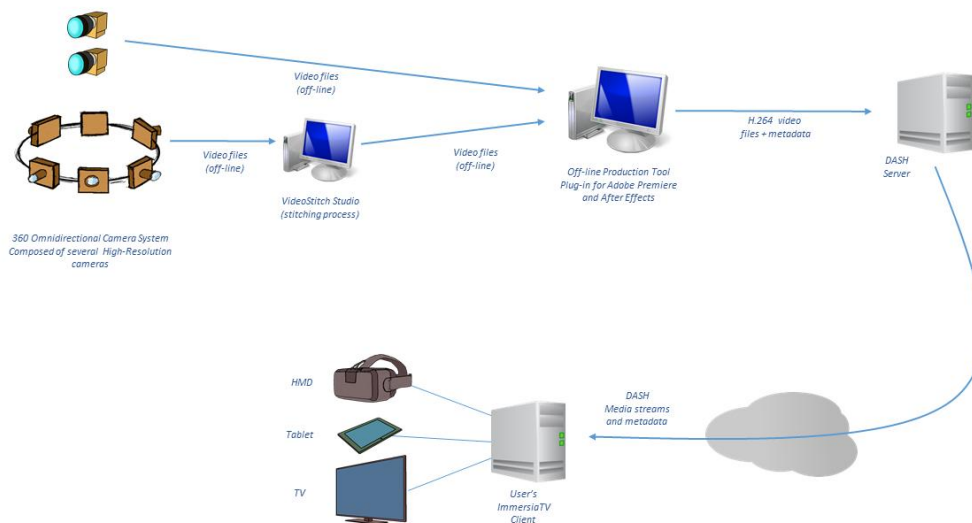


Figure 2: The general architecture of ImmersiaTV System designed for Pilot 1

The ImmersiaTV system consists of several modules that implement particular functionalities required for video acquisition, video processing and encoding, video production, content delivery and reception and display. General architecture is depicted on Figure 3.

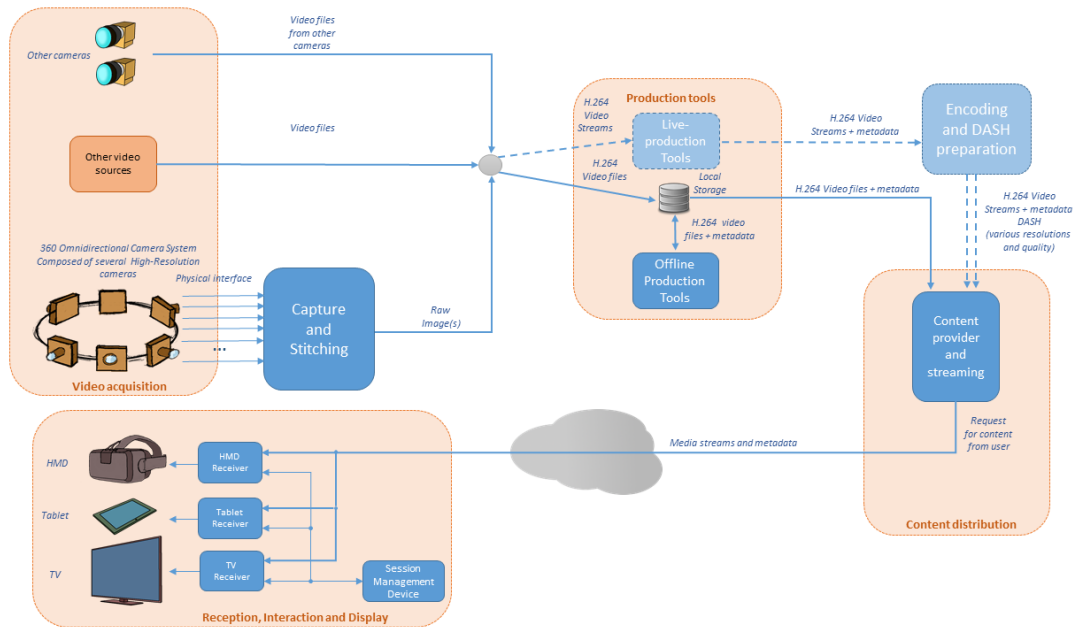


Figure 3: The architecture of the modules of the ImmersiaTV system.

The functional blocks and modules are:

- **Video acquisition** - this block is responsible for the physical capture of several video streams coming from 360 omnidirectional camera systems as well as other sources such as high resolution directive cameras, video clips, textual information and other metadata required for generating omnidirectional video enriched with audiovisual and auxiliary information in further stages. Omnidirectional video is the input for the Capture and Stitching module and other auxiliary data and video clips are used by production tools.
- **Capture and Stitching** - This block is responsible for grabbing and processing video images from 360 omnidirectional camera systems (constructed using multiple physical cameras). The main task of this block is to combine several video streams into one omnidirectional video stream (stitching process). Once the omnidirectional video is processed and prepared it's an input for the Encoding module.
- **Encoding** - Implementation of video codec used by other modules of the ImmersiaTV system such as Capture and Stitching and Production Tools.
- **Production tools** - A set of tools and plugins for offline and live video editing with functionality of synchronization and combining multiple 2D and omnidirectional video sources and auxiliary data together. These data come from Video Acquisition and Capture and Stitching blocks.
- **Content distribution** - This block handles the communication between offline encoded contents or live streams and the end-user's player. It encapsulates selected video streams into network protocols and provides synchronized video and auxiliary streams to the player.
- **Reception and display** - End-user's reception side and display. This block takes care of selecting proper video streams, receiving them, decoding and displaying them. It also handles the synchronization of multiple received streams in order to present them to the end-user.
- **Quality of Experience** – This module assesses the quality of content produced by the ImmersiaTV system by means of subjective and objective metrics and gives the

feedback and recommendations of changes to the Capture and Stitching, Encoding and Reception modules.

The architectural definition of all mentioned functional blocks is defined in the following sections.

2.1. Capture and Stitching

2.1.1. Description

The work related to capture and stitching for this document concentrates on off-line capture according to Pilot 1 requirements. For capturing and processing off-line visual content the ImmersiaTV system will use existing 360° camera rigs that are available on the market.

In later pilots, it will also rely on a distributed video capture architecture ground up designed for omnidirectional video in a live TV broadcasting context. In such solution (for later Pilots) we will address the issues of high equipment cost, too low perceived image resolution and frame rate, too low video processing performance and/or quality, lack of versatility in deployment of current systems. For Pilot 1 we will rely on hardware available on the market in order to test all other components of the workflow.

Two directions are foreseen:

- **Using off-the-shelf and professional cameras in conjunction with VideoStitch commercial products** (VideoStitch Studio and VahanaVR), that will be adapted to answer the constraints and requirements of broadcast-quality omnidirectional production workflows; VideoStitch Studio is dedicated to offline post-production, importing video files, stitching them together, and producing a 360° video file, while VahanaVR stitches live video streams. **These solutions will be deployed in pilot 1** (offline production). Although more oriented towards live productions, VahanaVR will be used in pilot 1, allowing the directors to have a live preview and direct feedback of the content during the shooting. In the whole content production and delivery chain of ImmersiaTV, VideoStitch Studio and VahanaVR will be the stitching nodes, interfacing with Cinegy's video servers;
- ImmersiaTV partners will also investigate a **dedicated distributed production camera architecture**, which will consist of edge capture, replay and per-camera processing units, and a central video processing unit. The edge units combine 1) raw camera data recording and instant replay and low-level image processing now usually available (in part) inside a camera or per-camera recorder, with 2) per-camera stages of pre-warping processing work such as demosaicing, colour correction, geometric lens distortion compensation. These units will be compact, light-weight, relatively silent and low-power consuming, allowing versatile positioning. The partial results from the edge units are ad-hoc coded and transported to the central video processor over affordable gigabit Ethernet long-range links, avoiding the need for much more costly higher bandwidth links. The central processing unit essentially relies on VahanaVR for the stitching, to allow for a seamless integration into the ImmersiaTV components and tools (in addition to common post-processing and control tasks) and thus can be much more lightweight than the full processing cluster that would now be needed in similar high-resolution live applications. The end-result is a modular and scalable architecture, with units adapted to the circumstances in which they will be placed (including concert stages, portable units for documentaries, and cycling sports tracks), and commodity data links in between. The developments will exploit existing 12-megapixel 180 fps

global shutter machine vision camera infrastructure at iMinds and other project partners, however without being limited to this type of cameras only. Due to its extremely high pixel count, this system will be able to generate 360° videos, as well as conventional rectilinear views chosen by operators, from a single capture device, as if conventional cameras had been placed on the set. Special attention is paid to keep pace with the rapid evolution in sensor and camera technology, data interfaces, processing architectures, in other words, to make sure the resulting system is not just addressing the needs of today, but will also be future proof. **This solution will be deployed in pilots 2 and 3** and will not be further detailed in this document.

2.1.2. Interfaces

The input and output formats are closely related to the cameras selected for content recording on the one hand, and VideoStitch software capabilities on the other. The section 2.1.2.1 presents the results of analysis of possible camera systems selected for Pilot 1 with the definition of their output formats. Section 2.1.2.2 describes requirements of the VideoStitch software.

2.1.2.1. Camera rigs

The following camera rigs will be used in the pilots:

The following camera rigs will be used in the pilots:

- **GoPro Hero 3 Black¹** camera rigs (3 or more sensors). The H3PRO6 rig enables to combine 6 GoPro Hero 3 Black cameras together for capturing omnidirectional video streams. Each piece of the camera has 12MPix CMOS sensor and produces H.264 encoded stream or provides HDMI live output. The cameras support storing on microSD/microSDHC cards in resolution up to 4K, although the frame rate in UHD resolutions is rather poor (12 or 15 fps). Each camera handles Full HD resolution in 60 fps (recording) or 30 fps (HDMI output)
- **Elmo QBIC²** rigs (4 sensors). QBIC Panorama X camera rig enables to combine 4 QBIC cameras for capturing omnidirectional video streams. Camera is equipped with CMOS sensor and supports resolution up to Full HD in 60p. (recording) or 30 fps (HDMI output). The camera has WiFi output.
- **Orah 4i camera³** (4 sensors). Contrary to previous cameras, Orah is an integrated camera in a single housing and does not require an additional rig for mounting several camera modules. Orah is equipped with 4 integrated lenses and SONY EXMOR sensors as well as Ethernet output, and does not require additional stitching software such as Vahana VR or Videostitch Studio (stitching is done by attached Live processing unit). Maximum resolution for stitched output is 4K in 30p. The output H264 video bit rate transmitted by RTMP varies between 5 and 25 Mbit/s. The Orah camera would be a perfect choice for ImmersiaTV capturing for Pilot 1 and may not be available by the time of pilot 1.
- **iMind's camera** (6 sensors). This camera will be specially developed for ImmersiaTV and will provide perfect image quality in UHD resolution. The specific design of the

¹ <http://www.cnet.com/products/gopro-hero3/specs/>

² <http://www.video-stitch.com/360-camera-rigs-elmo/>

³ <https://www.orah.co>

camera will be ready for next version of this document and will be used in pilots 2 and 3.

The most important parameters of the described camera systems with output capabilities are described and compared in Table 1.


Camera Rig	Number of sensors	Output resolution	How it is used
 <p>H3PRO6 Rig with 6 GoPro 3 Black cameras</p>	6	6x 1920x1080p/60 fps when recording on SD card 6x 1920x1080p/30 fps on HDMI output	Outputs video files on SD cards for VideoStitch Studio, or HDMI video for Vahana
 <p>Elmo QBIC</p>	4	4x 1920x1080p/60 fps when recording on SD card 4x 1920x1080p/30 fps on HDMI output	Outputs video files on SD cards for VideoStitch Studio, or HDMI video for Vahana
 <p>Orah 4i</p>	4	4x 1920x1440p/30 fps through an Ethernet cable	Associated to a Stitching Box, delivers 4K/30 fps to an SD card or an RTMP streamed output.
 <p>iMind's camera</p>	6	6x 4096x2880p/60fps through multiple HD-SDI cables	Each camera has a dedicated edge unit connected to a central stitching server.

Table 1: Comparison of camera systems planned to be used for ImmersiaTV pilots.

2.1.2.2. VideoStitch stitching systems

For the stitching process the VideoStitch products will be used. There are two tools to be used in the ImmersiaTV system:

VideoStitch Studio for off-line stitching and production:

- Input video files from cameras encoded with AVC, Baseline/Main/High profiles, 8 bits, progressive format, or Apple ProRes 10 bits (progressive)

- Computer requirements: Windows 7 or later, 64 bits, Linux Ubuntu 12.04.4 64 bits, Mac OSX 10.9 or later. Regardless of the OS, an nVidia graphics card with 4GB of GPU memory is needed.
- Output format: MPEG4/AVC/Apple ProRes 10 bits/Image sequences (Tif, jpg, png)

Vahana VR for live stitching:

- Video signal capture options:
 - Magewell HDMI video capture cards
 - BlackMagic Decklink cards for SDI
- Computer requirements: Windows 7 or later, 64 bits. Linux is not supported officially, but Linux Ubuntu 12.04.4 64 bits should work. Mac OSX is not supported.
- Output options:
 - RTMP streams
 - BlackMagic Decklink SDI cards

For post-production or live stitching, only the equirectangular 360° projection is currently supported

2.1.3. Architecture

As mentioned in Section 2.1.2.1, in the project there will be several camera systems tested and used in order to achieve the different goals of Pilots 1,2 and 3. For Pilot 1 and off-line content capturing the GoPro3 and Elmo QBIC rigs will be deployed. They will store several video streams on SD cards, and this content will be used in VideoStitch Studio which will do the stitching processing. Then the omnidirectional content will be available in off-line processing tools.

For Pilots 2 and 3, live content will be generated by using live outputs from GoPro3 (HDMI), Elmo QBIC (HDMI), Orah 4i (H.264 over RTMP) and finally the iMinds camera which produces 6 HD-SDI streams. GoPro3 and Elmo QBIC cameras will require Vahana VR stitching functionality before the omnidirectional video will reach live production tools. Orah 4i produces H.264 stitched video content that is directly streamed to production tools, and the iMinds camera will need additional edge units which convert multiple uncompressed HD-SDI streams into RTSP streaming accepted by the Central processing unit operating Vahana VR.

The workflow is depicted on Figure 4.

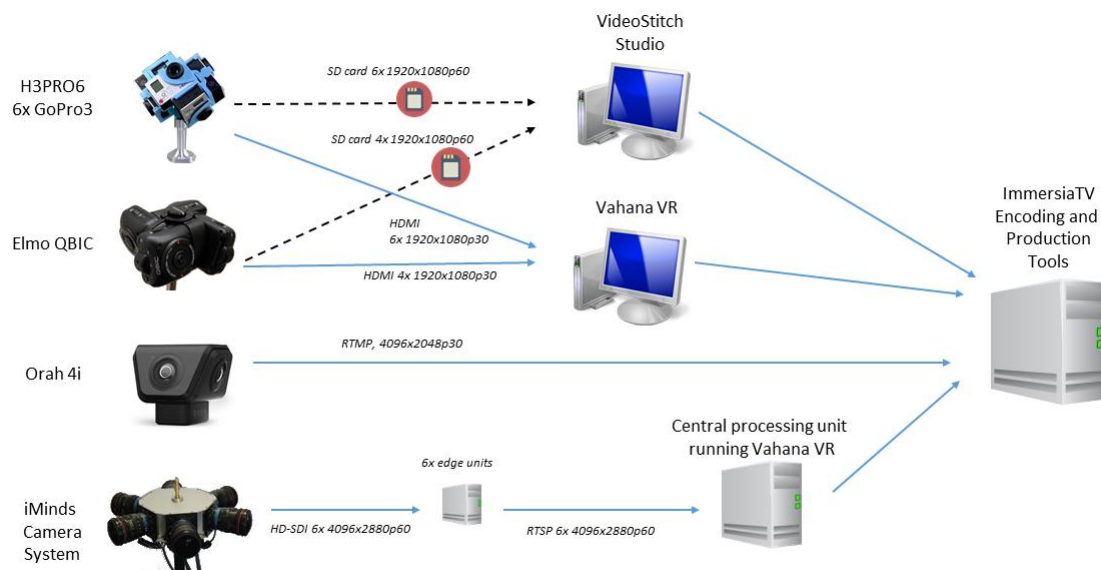


Figure 4: The architecture of creating off-line and live omnidirectional video streams in the ImmersiaTV system.

2.1.4. Workflow

The overall functionality consists of stitching various off-line or live input video sources. VideoStitch Studio and Vahana VR share a similar workflow:

- read the input sources coming from the various cameras on the 360° camera rig
- temporally synchronize them (**VideoStitch Studio only, Vahana VR assumes the frames on its physical or network input ports are already synchronized**)
- calibrate the camera rig geometry (through self-calibration of intrinsic and extrinsics parameters, but an offline calibration template can be imported)
- calibrate the camera rig photometry (to make up for various exposures, colour temperatures if the cameras are not properly controlled, and for lens vignetting)
- map each camera view onto a 360° equirectangular frame
- adjust the equirectangular frame orientation for horizon levelling (making sure the scene horizon maps to an horizontal line in the stitched output)
- export the stitched content. In Studio, it can take the form of individual picture files, or a compressed video stream. In Vahana VR, the output format can be a compressed video file stored on disk, or an uncompressed output on an HDMI or SDI port, or a compressed RTMP stream which can be sent to a video server.

The workflow described above is depicted on Figure 5.

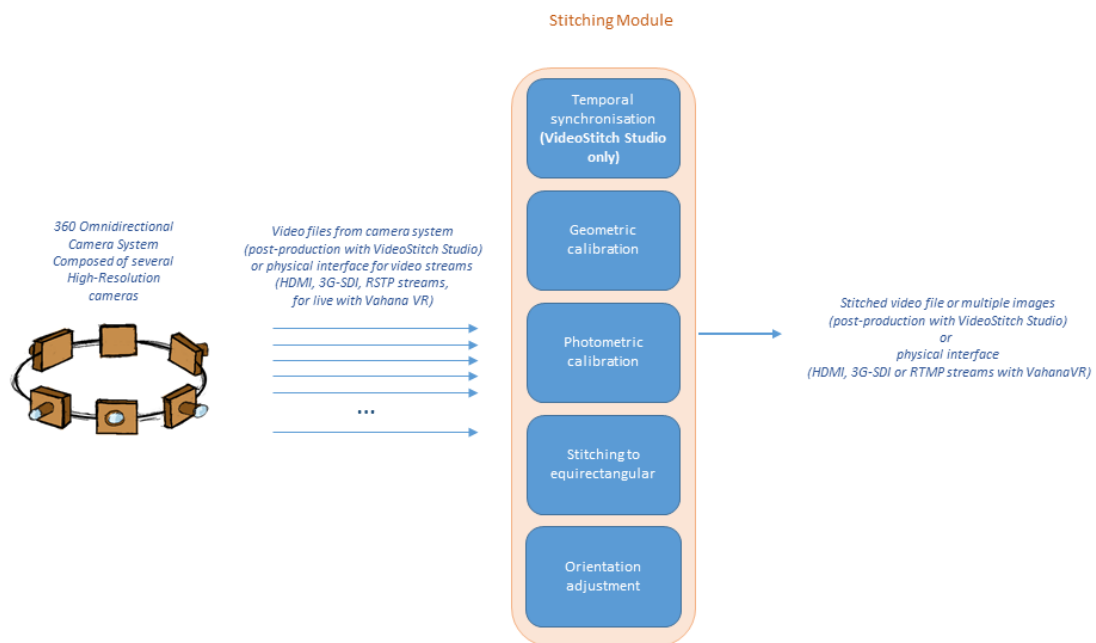


Figure 5: Details of capture and stitching workflow using VideoStitch Studio and Vahana VR.

As directions of development of third-party cameras are not clear, using new versions of omnidirectional cameras in the project is very risky. In order to have full control on all the parameters of the camera and capturing system, we decided to rely on dedicated solution being designed and integrated by iMinds. Concerning the dedicated production architecture using several edge units and a central unit for the stitching for pilots 2 and 3, a precise calibration procedure will be carried out for the 12-megapixel 180 fps global shutter cameras. A specific file format will be defined in order to allow the calibration parameters to be input into VideoStitch Studio and Vahana VR. The functionalities of Studio and Vahana VR will be tailored and improved to better interface with the other packages and use cases of ImmersiaTV (projections, supported formats, codecs, inputs and outputs).

- **Production tools:** In order to interface with the production and content-editing tools, one or several codecs and/or output interfaces will have to be agreed upon, if we need to depart from AVC encoded-files for post-production and media servers, and SDI/HDMI/RTMP outputs for live productions.
- **Encoding and Decoding:** Only progressive video is supported by the stitching software. One potential interaction with the codec tasks is the support of a lightweight codec for input/output capabilities, different colour spaces, RTP output streaming (if RTMP cannot be accepted) and alternative projections (like a cubic mapping). HDR imaging will require the support of larger bit depths and/or colour spaces.

Resolution is only limited by the computing power of the stitching device and available input cards. Interlaced video cannot be supported by the stitching, unless a deinterlacer is used.

2.2. Encoding

2.2.1. Description

The main functionality of the module is to provide an efficient and high quality encoding solution compatible with the general purpose equipment (TV, HMD, tablets) available on the market. At the moment, the most popular and widely supported codec in the consumer area is AVC/H.264 and it was chosen as a main solution for the Pilot 1 demonstration.

The Encoding module is responsible for implementation of video encoder and decoder functionality used by other modules of the ImmersiaTV system. As the general standard selected for **first iteration and pilot is H.264**, this section provides the guidelines regarding the video format and underlying parameters to be used within ImmersiaTV from capture to rendering and display, as well as all processing and streaming related matters such as corrections, stitching, transcoding, etc..

The specification of the codec format and parameters is focused on pilot 1 – off-line scenario. In particular for pilot 2 and real-time configuration, the constraints for video coding will largely depend on what is possible in rendering/display and streaming with the state of the art hardware and infrastructure available.

The main objective of this specification is to ensure that the complete chain of processing from creation to consumption is clear and as much as possible harmonized in order to reduce the number of transcoding and format conversions needed. Not only the quality of the content but also complexity issues arise if transcoding should be performed between different components in the processing chain.

2.2.2. Interfaces

The input to the Encoding modules is defined by the output formats supported by video acquisition equipment and Production tools, it can be raw data or pre-encoded video files. As the output the H.264 encoded video files or streams are generated.

The Encoding module has direct interaction with Video acquisition (Input side), Production tools (Input/Output side) and Content distribution (Output side) modules but the format and results of the encoding procedure influence all the stages in the ImmersiaTV processing chain.

The streaming of video is largely dependent on the available HD and infrastructure capabilities. A transcoding step will be necessary to convert the recorded content and to match it to what is possible.

On the reception site the decoder is responsible for decoding of H.264 encoded media streams received by the client player and sending raw data to the rendering module. Regarding decoding functionality of Reception, Interaction and Display module the rendering and display of video is largely dependent on the available HD and display capabilities. The final parameters of the encoding procedure will depend on the capabilities of both the streaming solution (see below) and display devices. As a general rule, the video format, bit rate, frame and rate will depend on the capabilities of a general purpose display as available on a typical HMD, a tablet or a smartphone.

2.2.3. Codec definition

In principle, the highest quality content should be produced when generating content for off line configurations (pilot 1). Therefore, the highest possible bit rate, frame rate, frame resolution, and colour sampling must be achieved. Where possible, the influence of compression on the quality should be minimized. It is however a fact that several existing devices and hardware do not allow for uncompressed content. Based on the feedback received from ImmersiaTV partners, for off-line recording of content, the following guidelines will be followed as far as the video format is concerned. Parameters of the codec were summarized in Table 2 below.

Parameter	Value	Comments
Encoder	AVC/H.264	If possible, uncompressed where possible
Decoder	AVC/H.264	Compressed bitstream should be decodable by any widely used and AVC/H.264 compliant decoder such as ffmpeg.
Profile	At least HiProfile (HiP)	If the capture device does not allow Main Profile is acceptable but not recommended.
Bit rate control	Variable rate	If constraints do not allow, constant rate is acceptable but not recommended
Color Space	RGB, YCbCr, YUV	YCbCr recommended
Frame/Field sampling	Progressive	Interlaced should be avoided
Color sampling	4:2:2	4:2:0 is acceptable if 4:2:2 not possible
Frame resolution	4K, UHD, HD	The higher the better
Frame rate	At least 30Hz	If capture device does not allow, then 25Hz is also acceptable but not recommended.
Bit depth per component	10bit (preferable) 8bits (acceptable)	The higher the better
Bit rate (total)	At least 10 Mbps per camera	The total bitrate corresponding to the number of cameras may arise various issues regarding their capture and could affect the 10Mbps lower limit.

Table 2: Off-line recording configuration

2.2.4. Regions of interest

Complementary to the adaptive approach of the DASH server, ImmersiaTV will implement an adaptive codec solution based on real-time feedback regarding regions of interest and quality of experience. As immersive content contains areas exciting peripheral regions of the human visual system, as well as areas not always visible to viewers consuming such content, region of interest coding seems an appropriate approach in order to prioritize the order in which the content is streamed or decoded, especially in environments with constraints in delay, computational complexity, bandwidth and power consumption.

Several options are available and under consideration in ImmersiaTV. The most popular approach consists of an adequate inverse mapping of the omnidirectional content into a geometry that is more suitable as representation of immersive content. Several of such inverse mapping mechanisms have already been explored in literature, such as cubic or pyramidal inverse mapping. However, these mainly deal with redundancy and statistical issues related to omnidirectional content. In addition to the above, in ImmersiaTV we will consider content consumption related characteristics such as focus of attention either obtained through a model or via direct measures of the gaze in order to identify areas which have to be coded according to quality that matches the human visual system characteristics. In particular we

extend models of focus of attention from conventional content and extend them to take into account not only the geometric distortions but also motion information.

Extension of conventional codecs to cope with the perceptual value of the pixels representing content will be then the next step, where additional transforms, quantization and rate control tools will improve the performance of such codecs to better cope with omnidirectional content.

Last but not least, perception models for objective metrics will be employed in order to optimize the rate distortion (quality) of the immersive content based on optimization of such metrics as opposed to Euclidean distance.

2.3. Production Tools

2.3.1. Description

Video editing in general is a complex process with many stages. In the project we extend typical media creation by adding new technical possibilities, however they also impose some restrictions for the content creation process.

The aim of this part of the project is to propose a workflow, elaborate techniques in existing and create missing tools that will allow users to create, in an easy and intuitive way, a wide variety of content suitable for the ImmersiaTV player.

The architecture of a set of tools for content creation is based on three elements:

- software requirements from the user scenarios (defined in Deliverable 2.3),
- output format and player capabilities (defined in 2.5 Reception, Interaction and Display),
- evolution of environmental capabilities.

While the first one defines requirements, the two others impose mostly limitations that have to be taken into account.

The proposed workflow should not differ much from the typical process of video editing. Editors should be able to follow their standard routine supplemented with additional steps and having in mind that sequences for three synchronized output destinations have to be prepared. All of the required additional functionality will be implemented and added to Adobe Premiere Pro as a set of plugins.

There are three main stages added to the standard editing workflow, as depicted in Figure 6:

- synchronization of media for different output destinations
- defining portals/transitions/interactive points
- export to different output formats

By adding three stages (depicted as yellow blocks) to the typical content edition workflow (dark blue), separate processes for each device (TV, tablet, HMD) are merged into one bigger process.



Figure 6: Schema of proposed editor workflow

2.3.2. Interfaces

Data acquired in previous modules are the input for production tools. Adobe Premiere Pro natively supports the formats described in the previous chapters.

Tools in this module will produce a package of files (media files and metadata) ready for distribution and compatible with ImmersiaTV player as depicted in Figure 7.

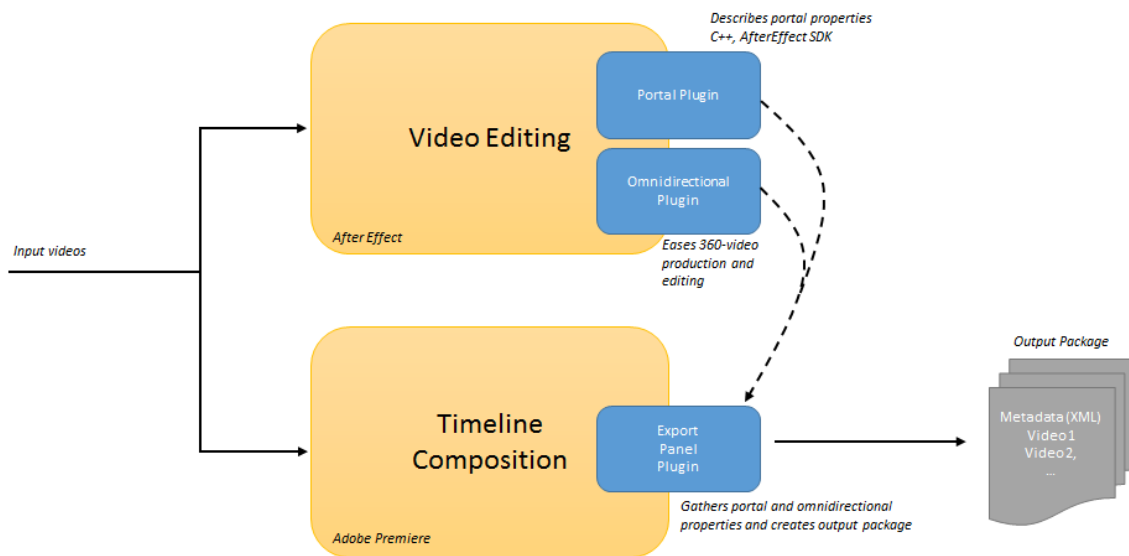


Figure 7: Composition of the off-line production tools.

2.3.3. Architecture

2.3.3.1. Synchronization

The first challenge is assembly and editing of content for three devices in one project. It is up to editors, if they create them in parallel or one by one, but the objectives of the project put strict requirements for the synchronization of output media. To achieve this goal, at this point of the workflow editors have to follow basic rules allowing to use the ImmersiaTV plugin. In

particular, clips for all types of devices should be edited together in a common sequence space, however on separate video tracks (layers). Each track will have to be labelled, indicating which device is the target. Tracks can be shared by more devices, if the same clips are intended for them.

This approach from editor perspective makes assembly, synchronization and editing of content for three devices similar to preparing picture-in-picture content for a single device. Generation of three separate contents will be done automatically in the export stage.

After tests editors may ask for additional tools to support labelling and managing tracks or enhancing preview capabilities, but these will be optional improvements, which will not influence general architecture.

2.3.3.2. Portal video effect

Portals are overlays (video or graphical) in omnidirectional content potentially with interactive options. In our approach inside of a portal there will be a separate video stream and the player will blend them dynamically. It can be used as a (conditional) transition, when the appearing portal covers the whole sphere.

To allow editors to create ImmersiaTV project with portals, there should be a Portal video effect implemented. It should be applied to the video clip that will be visible in the portal. It should describe portal parameters and visualize them in a preview. We assume that background space is omnidirectional with equirectangular projection. During the export portals parameters will be used to generate proper video files and metadata.

This part of the plugin will be created with After Effects CC 2015 Plug-in SDK (C++)⁴.

Portal parameters should be possible to be modified from standard Effects Control panel and on a preview. Table 3 lists all parameters of the *Portal video effect*.

Parameters	Control type	Animatable (Variable)	Details
Projection	list	no	none - Directive shots equirectangular - Omnidirectional shots
Shape	implicit	no	rectangle - Directive shots spherical cap - Omnidirectional shots
Reference	list	no	world/user
Longitude	360° (-180°-180°)	yes	position of a centre of a portal relative to 'Reference'
Latitude	180° (-90° - 90°)	yes	position of a centre of a portal relative to 'Reference'
Distance	implicit	no	Track number defines order of portals
Size	slider / implicit	yes	Directive shots - scales width of video to a defined size (keeping aspect ratio). 1.0 -> width of a background sphere Omnidirectional shots - always 1.0 -> 360°

⁴ <http://www.adobe.com/devnet/aftereffects/sdk/cc2015.html>

Luma matte	layer list	no	Layer defining transparency of a portal video (NOT related to the transition)
Transition:			
Visible	checkbox	yes	Possibility to switch on/off portal at keyframes.
Transition luma matte	layer list	no	Layer defining portal opening/closing transition
Additional action	list	yes	Action at a keyframe (currently only vibration)
Interactivity:			
condition on appearance	list	no	List of callbacks (click on, look at portal, shake the tablet, etc.)
condition on transition pause	list	no	List of callbacks (click on, look at portal, shake the tablet, etc.). Only affects the luma playout, not the content playout
condition on completion	list	no	List of callbacks (click on, look at portal, shake the tablet, etc.).
Separate switch	checkbox	no	Interactive area is different from portal content area (for tablets)
Switch longitude	-180° - 180°	yes	Position of interactive area
Switch latitude	-90° - 90°	yes	Position of interactive area
Switch width	slider	yes	Width of interactive area
Switch height	slider	yes	Height of interactive area
Switch reference	list	no	world/user

Table 3: Portal video effect parameters

2.3.3.3. Visualisation

A preview of an edited scene can be observed in the Program Monitor window. Selecting the Portal effect in the Effect Control window enables overlay in the preview, that visualizes parameters of a portal and allows their direct modification.

Additionally the Preview mode parameter of the Portal effect allows to change how video is rendered. There are 5 options in the Table 4:

Preview mode	Description
Outside	portal is not visible, only background
Inside	portal video is visible, without any modifications (projection, luma matte),
Luma matte	luma matte layer of a portal is visible
Combine	portal is composed into the background

Table 4: Preview mode parameters

2.3.3.4. ImmersiaTV package export

The final step of content creation is the generation of an ImmersiaTV package containing a set of media files and metadata describing their relations. When the user follows the workflow described earlier in this chapter, export to the ImmersiaTV format should be done automatically.

It will be accomplished by a Javascript script run from a ImmersiaTV panel plugin (based on Adobe Premiere Pro CC 2015 panel SDK⁵). A panel is a HTML document, which can use standard Javascript, additional libraries (e.g jQuery) and can interact with Premiere Pro API. It will include a form to specify export parameters and to launch the export process.

The export process will have three main stages:

1. **project analysis** - tracks' labels and portal video effects' parameters should allow to determine the structure of the final package.
2. **metadata generation** - from the results of the project analysis an XML file with metadata described in chapter 2.5.2.1 Metadata should be created.
3. **media files rendering and encoding** - an export of each of the media streams as defined in the metadata should be started.

The only required export parameter will be the device type (output path).

Additionally encoding parameters for each target device could be parameterized, however in first versions of a plugin they will be fixed.

2.3.4. Live content edition architecture

The following architecture for live production is initially foreseen as depicted on Figure 8:

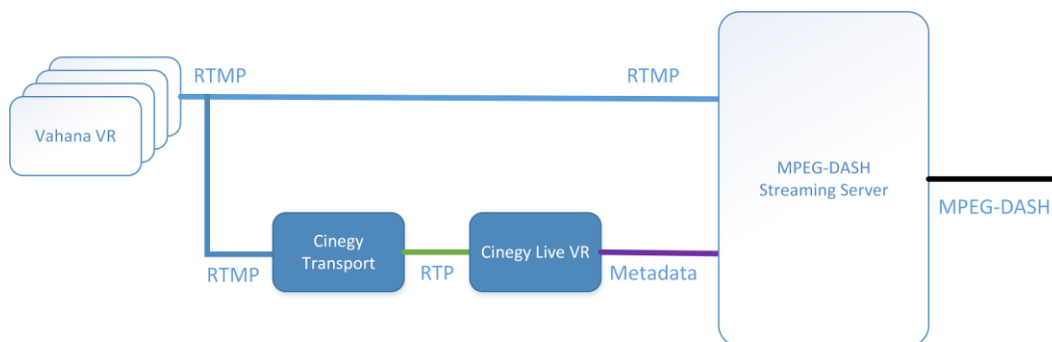


Figure 8: Live content edition workflow

- One or several instances of Vahana VR are connected to the cameras and perform the capturing and stitching of the incoming streams. Input format consists of RTP streams encoded with AVC, Baseline/Main/High profiles, 8 bits, progressive format. The resulting omnidirectional video is output as RTMP stream containing the ImmersiaTV scene XML segment

⁵ <https://github.com/Adobe-CEP/Samples/tree/master/PProPanel>

- When additional syncing is required the output RTMP stream is routed to Cinegy Transport that will allow adding the required delay to streams in order to sync them.
- The synced RTMP streams are sent to MPEG-DASH streaming server.
- At the same time synced RTMP streams are transformed by Cinegy Transport into RTP streams that can be received by Cinegy Live VR.
- Cinegy Live VR will display the streams preview and allow the operator to define the required scene change by activating one of the predefined scene update actions either by activating a corresponding button or activating the transition for editing and applying the updated one. The list of available presets should be configured beforehand in ImmersiaTV scene XML format.
- Cinegy Live VR based on operator input will generate the required XML segment with ImmersiaTV scene update and will send the corresponding metadata update request to MPEG-DASH streaming server (for example, via HTTP PUSH request).
- Cinegy Live VR will continuously send scene updates to MPEG-DASH streaming server according to the operator input.

2.4. Content Distribution

2.4.1. Description

This section specifies Content distribution mechanisms and deals with the transmission of all data (media and metadata) from the main server where the content is stored, through the Wide Area Network (Internet), and up to the user's Local Area Network, where synchronized playback among devices will take place. For multimedia delivery, or the ImmersiaTV content delivery, the streaming technique that will be used is the MPEG-DASH standard. MPEG-DASH is a recent standard, officially published in 2012⁶, and reviewed in 2014⁷. DASH is the acronym of Dynamic Adaptive Streaming over HTTP so which clearly denotes two of its main goals: being adaptive and use of HTTP as the network protocol..

2.4.2. Interfaces

The content distribution module as a communication service interacts mainly with two other modules. On the input site with Production tools including the encoding module which provides H.264 encoded content (video files or live streams) as well as the metadata. The data gathered from production tools are further encapsulated according to MPEG-DASH standard.

On the output site Content distribution module communicates with the receiver module included in the Reception, Interaction and Display functional block which receives H.264 encoded MPEG-DASH media streams (with metadata) distributed by the streaming server over the HTTP protocol. This module also deals with user feedback and passes requests generated by the player to the streaming servers in order to provide an adaptive approach and support the region of interests mechanism.

2.4.3. Architecture

⁶ http://standards.iso.org/ittf/PubliclyAvailableStandards/c057623_ISO_IEC_23009-1_2012.zip

⁷ http://standards.iso.org/ittf/PubliclyAvailableStandards/c065274_ISO_IEC_23009-1_2014.zip

This functional block handles communication between offline encoded content or live streams and the end-user's player. It encapsulates selected video streams into network protocols and provides synchronized video and auxiliary streams to the player.

MPEG-DASH emerged aiming to be reference standard for Segmented HTTP techniques, as before MPEG-DASH there were only proprietary or private approaches, like HLS (HTTP Live Streaming) from Apple (it has also being published as an IETF draft⁸), HDS (HTTP Dynamic Streaming) from Adobe or MSS (Microsoft Smooth Streaming) from Microsoft. All of them are HTTP based and adaptive solutions, but MPEG-DASH appears to be the only option that might get a wide adoption in the industry, as many of the main industrial actors already announced support to it (Microsoft, Adobe, Netflix, Google, etc.⁹). There are three main reasons to choose MPEG-DASH as the standard to follow in the ImmersiaTV project:

1. MPEG-DASH is getting adopted by the major players. This is a very important point in order to get ImmersiaTV close to the market. Ideally, the content providers using mature MPEG-DASH services would not need to drastically update their distribution scheme in order to provide immersive experiences.
2. It is based on HTTP which means it is easily supported by many CDN services that operate over the top and by any platform or infrastructure adapted for web content (i.e. mobile networks).
3. It is an adaptive standard. Being adaptive might be of special interest in ImmersiaTV as the project will handle different devices, screens and resolutions. In 360° video resolutions up to 4K must be considered, however there might be some client devices (i.e. tablets or smartphones) that are not capable of handling these high resolutions; in that case adaptation is useful. If via MPEG-DASH the server is offering a simpler version of the same content, a limited client might use it and be able to provide a lower quality experience instead of failing to provide any service or experience at all. Although, the quality can't be too low to prevent side effects of omnidirectional video feeling and perceiving. The right trade-off between the quality and effectiveness will be a subject of QoE feedback.

MPEG-DASH has been already used and tested with 360° immersive video nearly out of the box (just providing a specific player for 360, see for example a demo by BitMovin¹⁰).

MPEG-DASH is not a protocol, format neither a codec. As stated before it is an streaming technique that makes use of HTTP protocol, however it is codec and format agnostic. MPEG-DASH does not solve the HTML5 codec issues and does not describe the details about how a specific codec and container could be used in a way the result is MPEG-DASH compliant. In order to fill the gap the DASH Industry Forum provided several guidelines¹¹ about how to use MPEG-DASH together with H264 or HEVC codecs and ISO-BMFF container format (ISO/IEC 14496-12)¹². ImmersiaTV will stay as close as possible to those specifications.

The project is something more than just providing a single 360° video, as it has been already said, the project aims to provide a transversal immersive experience across devices. The impact is that there will be several synchronized streams, and all of them will be

⁸ <https://tools.ietf.org/html/draft-pantos-http-live-streaming-13v>

⁹ <http://dashif.org/members/v>

¹⁰ <http://www.dash-player.com/demo/adaptive-vr-360-video-html5-demo/>

¹¹ <http://dashif.org/guidelines/>

¹² http://standards.iso.org/ittf/PubliclyAvailableStandards/c068960_ISO_IEC_14496-12_2015.zip

contextualized with the metadata defined in 2.5.2.1 This metadata will be delivered using a simple HTTP download, to use the same mechanism as the media.

2.5. Reception, Interaction and Display

2.5.1. Introduction

This deals with the reception of the streams from the Wide Area Network (Internet), their redistribution in a local area network, and the integration with the interactive input of the end-user.

The player integrates the audio, video and data streams in a coherent omnidirectional scene, parses the user input and adapts the environment appropriately to the reactions expected. Examples of touch-based interaction include:

1. Browsing and selecting particular content
2. Starting/stopping the experience
3. Selecting a region of an omnidirectional video to share through social media
4. Zooming in or out

Examples of interaction based on movements include:

1. Moving the head in an immersive display should update consistently the portion of the omnidirectional video being displayed, to reflect basic sensorimotor correlations
2. Moving the tablet around should also enable the update the field of view.

Audio is played consistently across the different devices, either in stereo (TV), either in binaural format (tablet with headphones, as well as google cardboard and head mounted displays).

The chosen architecture involves two different kinds of connected devices, which synchronize and interact:

- Receiver devices (TV Set, HMD, Tablet)
- Session Management device

The receiver devices run the ImmersiaTV interaction and display software (in short, the ImmersiaTV player). This software is a multi-platform player targeting the general consumer. Consistently, this player is designed to be compatible with emerging broadcast synchronisation standards (like HbbTV 2.0¹³), and work on the main platforms available to deliver the ImmersiaTV experience.

The session manager is a device connected to the same local network as the players, which coordinates playback among them. It makes sure that all players synchronize to the same clock and watch the same content, among other things. All functions provided by the session manager can be integrated into the players, so any one of them can act as the session manager, thus removing the need for an additional device on the network.

The Figure 9 shows a diagram of the connectivity of all devices in the home network. Conceptual blocks inside each device are shown simplified. A more detailed view is given in the next section.

¹³http://www.etsi.org/deliver/etsi_ts/5C102700_102799%5C102796%5C01.03.01_60%5Cts_102796v010301p.pdf

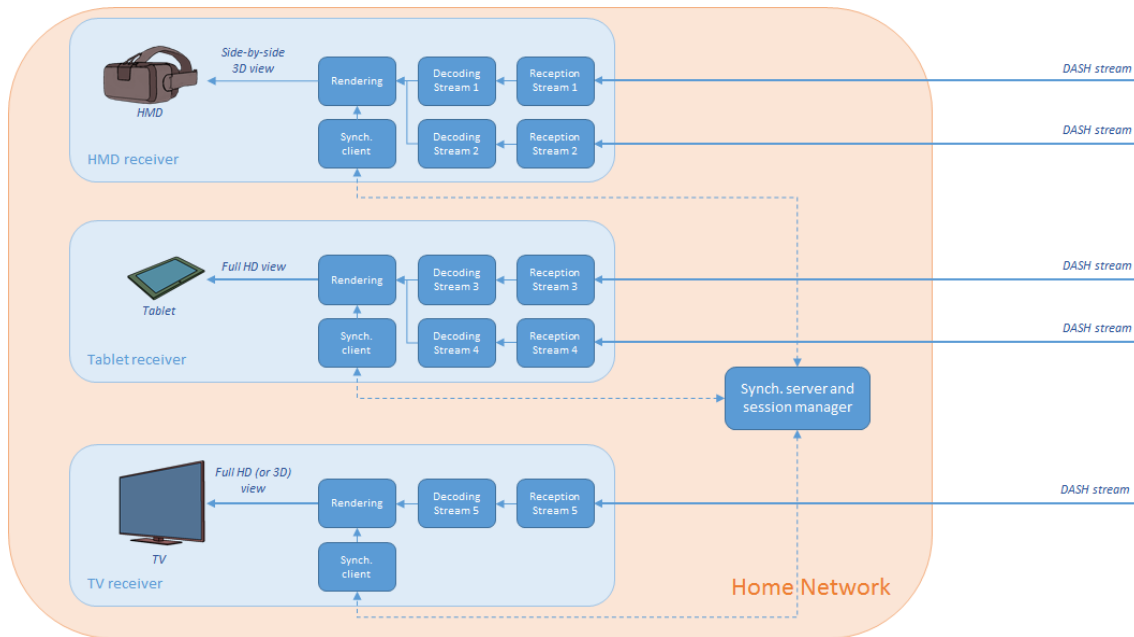


Figure 9: Connectivity of devices in home network

2.5.2. Interfaces

2.5.2.1. Metadata

The ImmersiaTV experience is based on 2 ideas: synchronized content across devices, and portals allowing interaction by blending different scenes, taken as traditional and omnidirectional shots in an immersive display.

All the metadata in ImmersiaTV will be sent in XML format, in a dedicated file. An event mechanism will be used, so the metadata can be added, removed or updated at pre-established times. Interaction is also defined in the XML file so metadata can be changed in response to user actions.

The ImmersiaTV Scene

The basic ImmersiaTV container will be called a Scene. Each Scene can contain the following elements:

- A unique string identifier (mandatory)
- A sequence of Shapes (defined below) showing some type of media
- A pointer to a CGI scene, containing additional geometry, textures, methods and other elements that may be involved in the scene rendering

The ImmersiaTV Shape

Each Shape can contain the following metadata:

- A unique string identifier (mandatory)
- The geometry describing this shape (rectangle, sphere, ...) and its size
- A list of Anchors (defined below) describing how this shape is to be situated in the scene, and a method to merge the different Anchors.
- A series of media file names, indicating the texture and optional transparency masks to render on the shape.

- A description of the projection used to turn omnidirectional media into a conventional flat video stream (if any).
- Cropping parameters, if desired

The ImmersiaTV Anchor

Anchors are points in the scene used to place Shapes. In the preparation towards pilot 1, each shape will have only one Anchor. Each anchor will contain the following metadata:

- A unique string identifier (mandatory)
- A reference frame (either the world or the camera)
- The polar coordinates (longitude, latitude and distance from the camera)
- A weight, used for merging different Anchors (not used in pilot 1)

The complete ImmersiaTV metadata specification

The above requirements have been implemented as an XML Schema Definition file (XSD), available at the following URL:

http://server.immersiatv.eu/public_http/metadata/ImmersiaTV.xsd

This definition is precise, allows checking for validity of XML files, and contains documentation, which can be processed to generate a human-friendly format, like the online pages available here:

http://server.immersiatv.eu/public_http/metadata/ImmersiaTV.html

The URL of the XSD file can be used in XML files so they can be validated automatically.

Metadata callback specification

The metadata format defined above specifies some methods on the player to be called after some user actions. The list of available methods is purposely left out of the definition of the metadata in order to render it more generally, and to ease expanding this list. The following table describes the accepted values for these callbacks (used, for example in the `onActivate` and `onDeactivate` attributes):

<code>toggleVisibility, shapeId</code>	Toggles the visibility of the shape with the indicated Id.
<code>setVisibility, shapeId</code>	Makes the shape with the indicated Id visible .
<code>unsetVisibility, shapeId</code>	Makes the shape with the indicated Id invisible .
<code>playTransition, shapeId</code>	Starts playing the transition indicated with the transitionFile attribute, on the shape with the indicated Id.
<code>pauseTransition, shapeId</code>	Pauses the transition indicated with the transitionFile attribute, on the shape with the indicated Id.
<code>playLimitedTransition, shapeId, seconds</code>	Starts playing the transition indicated with the transitionFile attribute, on the shape with the indicated Id, for the limited amount of time indicated in the mandatory <code>seconds</code> parameter.

The *shapeId* parameter is always optional. If no Id is given (the parameter is missing), the command affects the shape with the attribute.

A sample XML file containing ImmersiaTV metadata

The following is a sample ImmersiaTV metadata file (Figure 10). Please note the usage of `xsi:schemaLocation` in the root node to import the XML Schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<ITVEvents xmlns="urn:immersiatv:immersiatv01:2016:xml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:immersiatv:immersiatv01:2016:xml
    http://server.immersiatv.eu/public http/metadata/ImmersiaTV.xsd"
  type="static">
  <DefineScene id="0" device="hmd" externalScene="TestScene1.unity" time="0">
    <DefineShape id="0" type="rectangle" anchorMethod="simple"
      mediaFile="FILE1"
      mediaProjection="none" mediaCropX="0" mediaCropY="0"
      mediaCropWidth="1"
      mediaCropHeight="1" maskFile="FILE1MASK">
      <Anchor id="0" referenceFrame="world" longitude="0" latitude="0"
        distance="0.5" weight="0.8" maxAngularDeviation="45" />
      <Anchor id="1" referenceFrame="user" longitude="45" latitude="10"
        distance="0.5" weight="0.2" />
    </DefineShape>
    <DefineShape id="1" type="point" mediaFile="FILE2" />
    <DefineShape id="2" type="sphericalCap" size="1" mediaFile="FILE1"
      mediaProjection="equirectangular"
      onActivate="toggleVisibility,theatre_screen" />
    <DefineShape id="theatre_screen" type="mesh" mediaFile="FILE0"
      transitionFile="TRANSITION0" transitionState="paused"
      onActivate="playTransition" onDeactivate="pauseTransition" />
  </DefineScene>
  <DefineScene id="0" time="10">
    <DefineShape id="0">
      <Anchor id="0" latitude="10" />
    </DefineShape>
    <RemoveShape id="1" />
  </DefineScene>
</ITVEvents>
```

Figure 10: Sample ImmersiaTV metadata file.

This file defines two events: One creating a new scene and one updating it.

The Scene initially contains 4 Shapes: One rectangle with two anchors and a planar video with a mask, one point, one spherical cap with an equirectangular omnidirectional video and one external mesh with a planar video (because this is the default value). Moreover, the spherical shape (id "2") can toggle the visibility of the external mesh shape (id "theatre_screen") through user interaction. Also, the external mesh shape has a transition mask, which starts paused and can be played or paused through user interaction.

The second event triggers 10 seconds after the scene starts, and updates the latitude of one of the anchors of the first Shape and removes the second Shape.

2.5.3. Software architecture

The ImmersiaTV player running on the receiver devices is based on the Unity3D¹⁴ engine. This greatly simplifies deployment on a wide variety of end-user devices and adapts the experience to the particular characteristics of each device.

The processing of the media streams is performed using the GStreamer¹⁵ open-source framework. It receives and decodes different audio and video streams and delivers resulting frames to Unity3D for rendering. The connection between GStreamer and Unity is performed by a plugin developed within the ImmersiaTV project, unimaginatively called GStreamer-Unity Bridge (GUB for short)¹⁶ and publicly available.

The figure 11 depicts the complete software architecture (including both kinds of devices), with the most important blocks detailed in next sections.

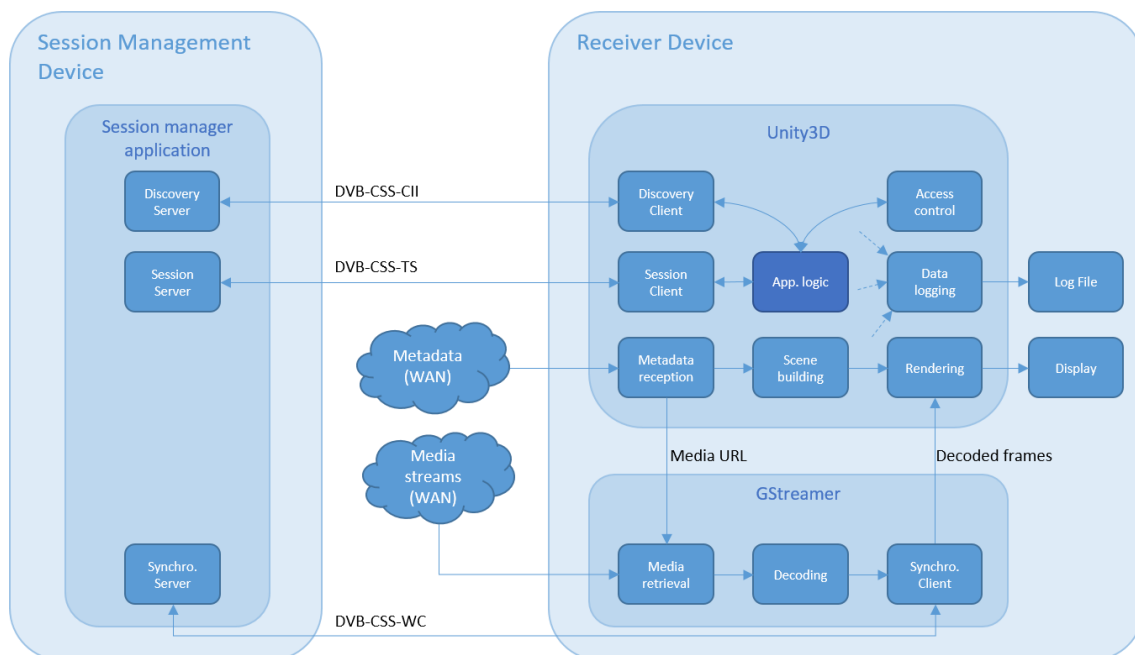


Figure 11: Software architecture of a player

2.5.4. Session management device

This is a device on the same local network as the players which coordinates the distributed playback experience. Initially it will be an application independent from the other players, running on a separate machine. The goal, however, is to integrate it with the player application, so any player can act as Session Manager, simplifying the setup for the user.

Its main functions are:

- Ensure all connected players see the same content
- When a player connects to a running session (there are previous players watching the same content already), it “catches up”, starting the playback at the point where the other players are.

¹⁴ <https://unity3d.com>

¹⁵ <http://gstreamer.freedesktop.org/>

¹⁶ <https://github.com/ua-i2cat/gst-unity-bridge>

More precisely, it provides:

- A master clock
- Session management: Distributing the BaseTime (the wall-clock time at which content playback started) and counting connected clients
- The media location (the remote MPD file URI)
- A discovery mechanism so the clients do not need to know the IP of the server.
- Optionally, a cache for the media files. Since many players might require the same content, huge bandwidth gains can be achieved by using a local media cache. The session manager, when running on a dedicated machine on the network, is the ideal location for this cache.

Communication between the Session Manager and the clients will be based on protocols from the DVB-CSS (Digital Video Broadcasting – Companion Screens & Streams) family to ease eventual interoperability with HbbTV 2.0 devices.

2.5.4.1. Discovery

To avoid having to provide each client with the server's address, the DVB-CSS-DA (Discovery and Association) discovery protocol will be used, in conjunction with the DVB-CSS-CII (Content Identification and Information) protocol. Combined, they provide the entry points for the other protocols and features (DVB-CSS-TS, DVB-CSS-WC and media location). DVB-CSS-DA uses the UPnP protocol, so there are plenty of available software libraries to aid its implementation.

The DVB-CSS-CII protocol will also be used to provide all players with the URL of the metadata file describing the scene, so this URL only needs to be stored in one place and can be easily changed.

Furthermore, it will easily allow caching, if this URL points to a local HTTP server in the same machine, for example.

The DVB-CSS-CII is very well suited for this task. However, it only provides a contentID which needs to be looked up in a Media Resolution Server (MRS) through HTTP to obtain the media URL. To ease implementation and reduce the requirements of the devices, we will embed the media URL in the CII response using private data, as already foreseen in the CII specification. This protocol uses JSON+WebSockets.

2.5.4.2. Session Server

This block will count the number of connected players and give each one the base time when they connect. When the first player connects, the base time is set to the current wall-clock time (so the clip starts from the beginning). Following players will see the clip has already started. When the last player disconnects base time is reset.

This functionality is very similar to the MSAS unit in DVB-CSS. Communication therefore will resemble the DVB-CSS-TS protocol, with the clients requesting a session through JSON+WebSockets, and the server replying with the current media time (base time). There are available C libraries to help the development:

- <https://libwebsockets.org/index.html>
- <http://www.json.org>

2.5.4.3. Synchronization

Multi-device synchronized playback should use standard protocols to achieve maximum interoperability. Particularly, to support [HbbTV 2.0](#)¹⁷ devices, the [DVB-CSS](#)¹⁸ (Digital Video Broadcast – Companion Screen & Streams) protocols family has been selected. The DVB-CSS-WC (Wall Clock) protocol is interesting, since it synchronizes the clock of all devices, so they all provide the same time.

GStreamer, though, lacks support for DVB-CSS-WC synchronization which needs to be added. This library is modular and plugin-based by design and could be used in implementation. Also, this allows contributing the work back to the GStreamer open source community, extending the project's dissemination.

There are elements already in GStreamer that allow inter-device synchronization, although they use a different protocol and therefore cannot be directly used. Their code, however, can serve as basis to implement support for DVB-CSS-WC. These elements are [GstNetTimeProvider](#)¹⁹ and [GstNetClientClock](#)²⁰. For a usage example, take a look at the [gst-rtsp-server](#)'s²¹ [test-netclock](#)²² and [test-netclock-client](#)²³.

This work will provide two new GStreamer libraries, a DVB-CSS-WC Server and a DVB-CSS-WC Client, which will perform the same functions as the already present [GstNetTimeProvider](#) and [GstNetClientClock](#), using the DVB-CSS-WC protocol.

The source code will be hosted in a GIT repository, forked from GStreamer, to ease contributing back to the original project. The library will be written in C and follow the GStreamer naming conventions. There is an open source Python implementation by the BBC of the DVB-CSS-WC protocol available [here](#)²⁴ which can also be used to help implementation.

These libraries will then be used by the Synchronization Server and Synchronization Client blocks.

2.5.4.4. Receiver devices

These are the players which display the immersive content to the user. They are programmed using the Unity3D game engine to allow interoperability on a wide range of devices. Therefore, most of the software modules are made in C#, with occasional calls to C when needed (for GStreamer operation, for example).

This is the normal operation of an ImmersiaTV player:

- Upon powering on, the player tries to discover a Session Manager on the network using the Discovery Client. It receives from the Session Manager entry points for the rest of protocols and the URL of the media being played.

¹⁷ www.hbbtv.org

¹⁸ www.dvb.org/resources/public/factsheets/dvb-css_factsheet.pdf

¹⁹ gstreamer.freedesktop.org/data/doc/gstreamer/head/gstreamer-lib/html/GstNetTimeProvider.html

²⁰ gstreamer.freedesktop.org/data/doc/gstreamer/head/gstreamer-lib/html/GstNetClientClock.html

²¹ cgit.freedesktop.org/gstreamer/gst-rtsp-server/tree

²² cgit.freedesktop.org/gstreamer/gst-rtsp-server/tree/examples/test-netclock.c

²³ cgit.freedesktop.org/gstreamer/gst-rtsp-server/tree/examples/test-netclock-client.c

²⁴ bbc.github.io/pydvbcss/docs/latest/wc.html

- The obtained URL points to a metadata file describing the scene, which needs to be downloaded (by the Metadata reception module).
- The metadata is parsed and used to build the scene inside Unity3D (Scene building module). The parts of the scene which require displaying media will instantiate GUB objects as required (which, in turn, will instantiate GStreamer pipelines) and provide them with the appropriate media URL.
- The player starts synchronizing its internal clock to the remote master clock using the Synchronization Client.
- The player instantiates a Session Client which will inform the Session Manager that a new client is connected, and in return, it will obtain the Base Time (This is, the wall clock time at which playback of the current media started).
- Unity3D will take care of rendering the scene onto the display.
- During the whole session, the Data Logging block can retrieve information from any module and produce a log file, to monitor the Quality of Experience.
- For testing sessions in which a questionnaire must be filled in before the experience, the Access Control module ensures all information has been received before starting playback.

Modules whose function is not clear from previous descriptions are described next.

2.5.4.5. Session Client

From the client perspective, this module is only needed to retrieve the current Base Time so it knows at which point in the media playback has to start. This is done through the DVB-CSS-TS protocol (only a small subset of it will actually be required). This small DVB-CSS-TS interaction, though, is more interesting for the server, since it allows it to count the number of connected clients.

Session Clients will periodically poll the server, as the protocol states, and this will also allow the server to know when a client has been disconnected (via a timeout mechanism).

Also, each Session Client must have a unique ID (unique within the local network) so the server can keep track of them.

2.5.4.6. Metadata Reception

The metadata describing the scene contents, including media and interactions, will be contained in an XML file hosted on a remote server. The Metadata Reception module has to retrieve this file, through a simple HTTP GET request and provide it to the rest of the player.

The only input to this module is the URL of the XML file, which will be provided by the application logic, after retrieving it from the Discovery Client.

2.5.4.7. Access control

Some of the tests require that the users fill in a questionnaire before the experience can start. This questionnaire will be online and, upon completion, will provide a token (an alphanumeric string, for example). The first screen in the player must ask for this token, which then needs to be validated against a remote database. Only tokens which have an associated questionnaire stored in the database will allow entering the experience.

2.5.4.8. Data logging

Data can include static information like device, session and user characteristics, or dynamic information like user view direction, network state, CPU usage, frame losses or the selected adaptive bitrate.

The Data Logging module will continuously monitor this data (polling the required modules) and write the logs to a file or database.

In order to provide reliable quality evaluation methods it is required to have detailed and credible live data as well as aggregated statistics representing the key parameters of the modules used in scope of ImmersiaTV project. The Logging module should provide all necessary data required by QoE module.

Architecture

The logging module architecture is composed of two main components

- *Logging Client Library* which defines functions implementing a flexible logging interface available for all other ImmersiaTV modules. As most of the communication is handled using JSON, Logging Client Library may be implemented easily in various languages for all the ImmersiaTV components. We foresee the C# and C/C++ implementation.
- *Logging Server*. On the *Logging Server* side there is a database system that collects all log messages delivered by *Client Library* via JSON interface. Data stored in database can be retrieved in defined format (e.g. XML) by functions provided by *Logging Client Library* for further use in QoE analyses.

The general architecture of Logging module is depicted on Figure 12:

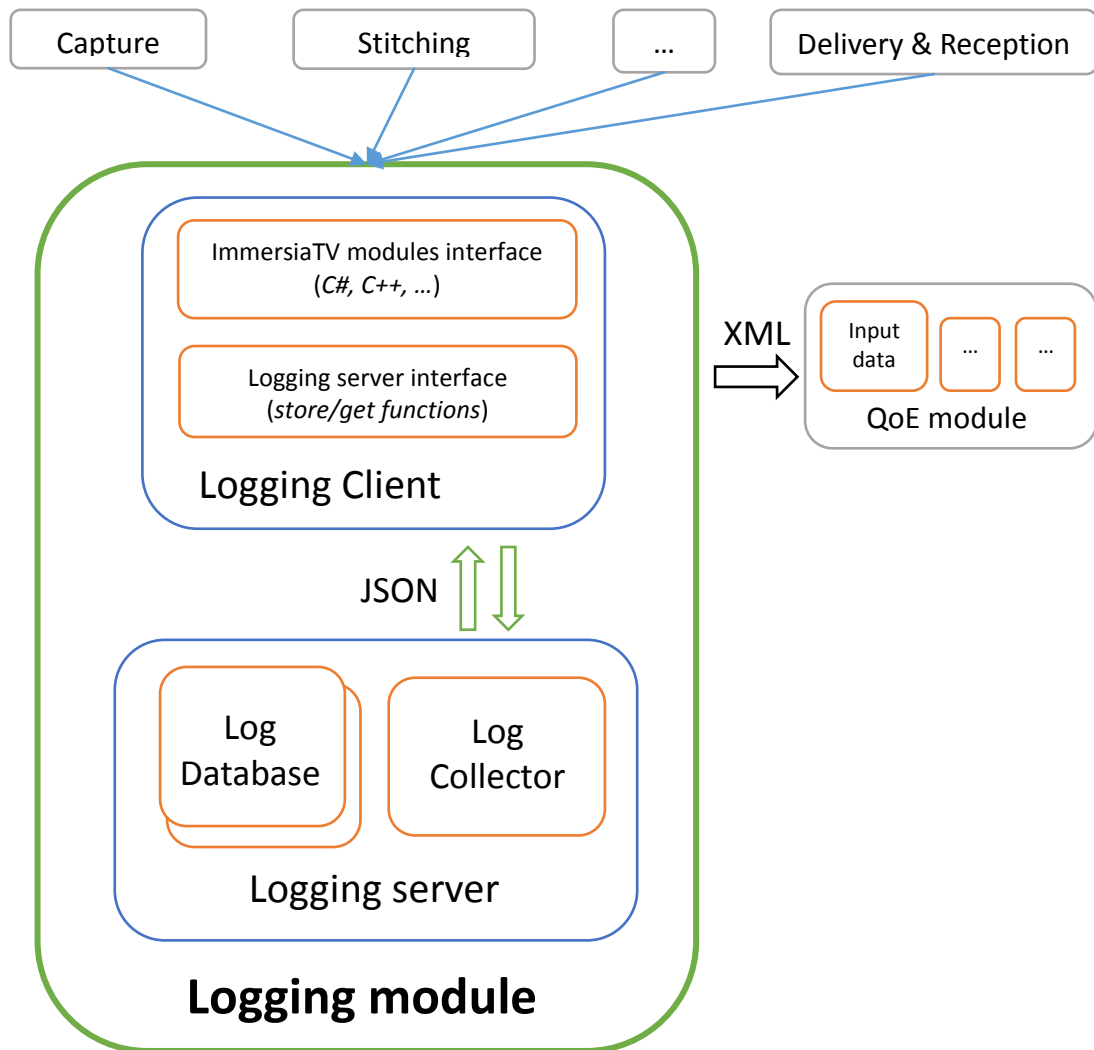


Figure 12: Logging module architecture

Workflow

All log data messages generated by different ImmersiaTV modules are stored in Log Database implemented as a part of Logging Server component. Messages are collected in a database as a sequence of records and are identified by the *sessionID* and *Timestamp*.

Logged data includes static information such as device, session and user characteristics, or dynamic information such as user view direction, network state, CPU usage, frame losses or the selected adaptive bitrate.

From the *Logging module* point of view each session is limited in time by *startSession Timestamp* and *endSessionTimestamp* messages and different modules, in scope of this session, continuously logs data with the same *sessionID* and appropriate timestamp.

Each session logged in the database starts with the record that includes timestamp pointing out the beginning of the session, after that static information is stored and then sequence of dynamic information automatically generated in defined intervals (e.g. 1 second, 1 frame) is logged. The *endSession* message with final timestamp closes collection of the session records.

The basic logging sequence during the Session is depicted on Figure 13:

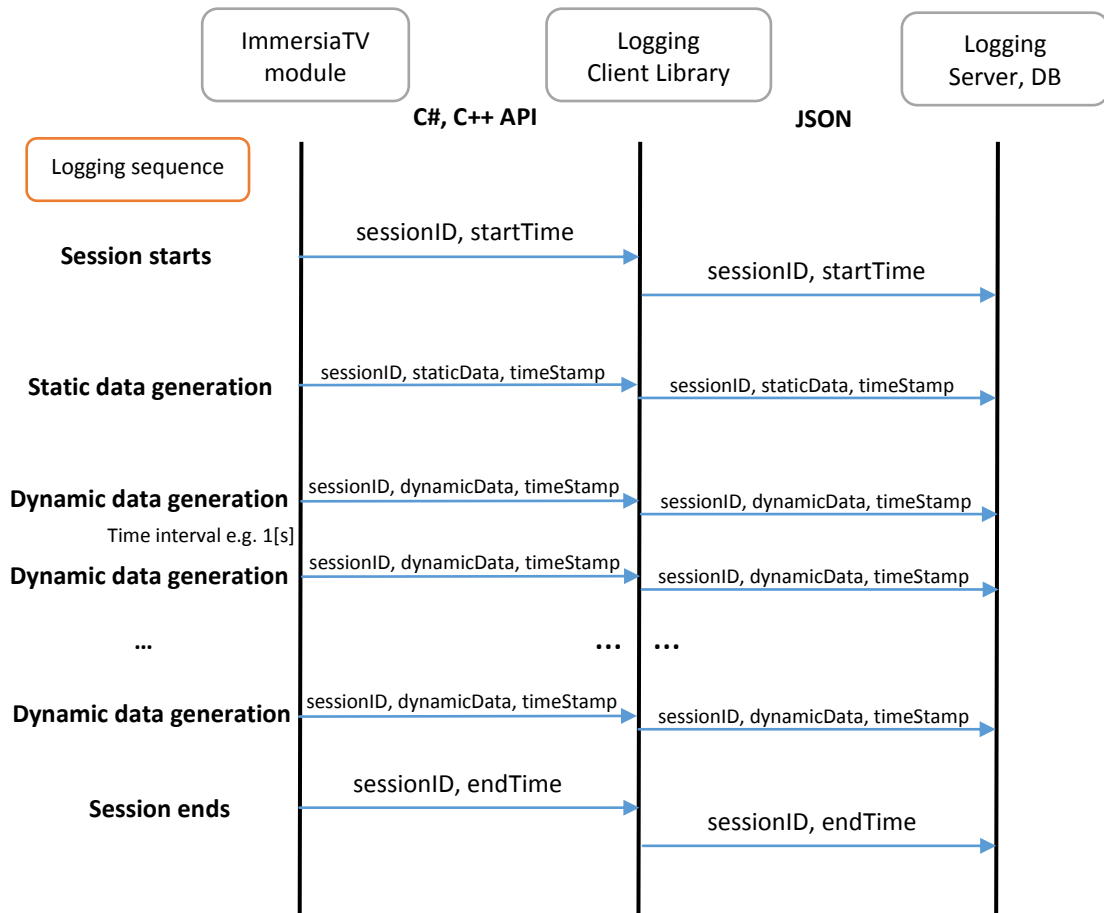


Figure 13: Logging sequence of Logging Module

When the data related to the particular Session are stored in the Log database they can be retrieved by QoE module (or any other) using functions provided by Client Library (functions are defined in section 3.6). The client module (e.g. QoE) can get all data related to the session, just static data or just dynamic data, in this case time period can be also specified. All functions returns requested data in XML format. The basic retrieving sequence is depicted on Figure 14:

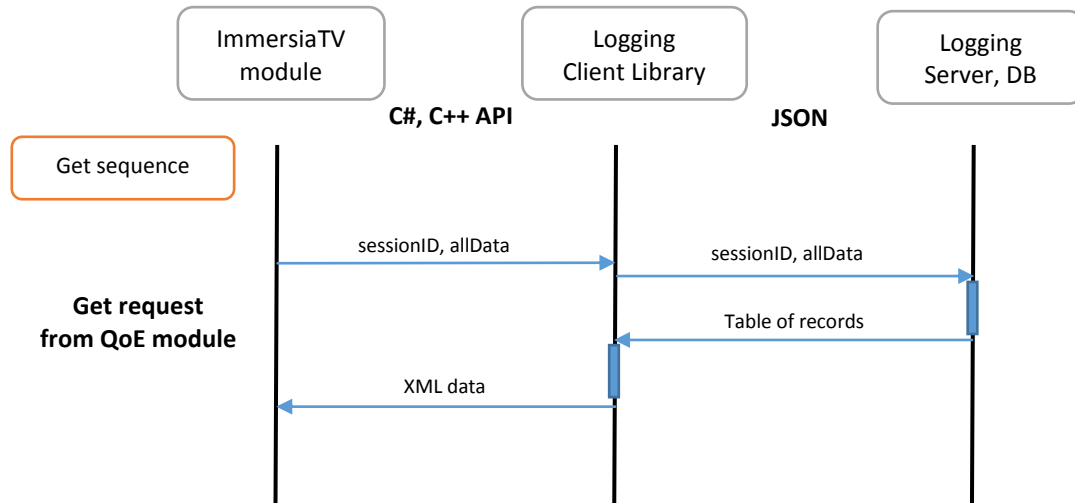


Figure 14: Retrieving sequence of Logging Module

Logging server

Logging server is a central element that provides web interface for storing and retrieving log messages from the database (e.g. MySQL). Data exchanged between *Client Library* and *Logging Server* are formatted as a JSON objects.

Example of dynamic information (*Frame_level_QoS_parameters*) JSON array of objects:

```

{"frameLevelQoSparameters": [
  {"instantaneous_bit_rate": "value",
   "network_delay": "value",
   "packet_loss": "value",
   " frame_rate ": "value"}

  {"instantaneous_bit_rate": "value",
   "network_delay": "value",
   "packet_loss": "value",
   "frame_rate": "value"}
...
]}

```

Client library

Client library provides interface which supports logging mechanisms for all *ImmersiaTV* modules. The API is implemented as a shared libraries for C# (integrated with Unity 3D) and C++ programming languages. It provides one function for logging and three dedicated retrieve functions mainly dedicated for QoE module, the API can be later extended according to the specific requirements of the other modules.

Client library functions:

- *logMessage(logDataMessage logData, Timestamp timestamp);*

Logs a data message with local timestamp into database.

- *XML getSessionAllData (int sessionId);*

Returns all logged data messages related to sessionId

- *XML getSessionStaticData (int sessionId);*

Returns only static logged data messages related to sessionId

- *XML getSessionDynamicData (int sessionId, [Timestamp start], [Timestamp end]);*

Returns all logged data messages related to provided sessionId or data in range defined by optional start and/or end session timestamp parameters.

As the log data messages are strongly related to the timestamps and time correlation between logged events is very important it is required to ensure the same time for all modules using logging mechanisms. It could be NTP protocol or any other time synchronization mechanism.

Logging data structure

The logging data have an open structure in order to ensure flexibility and extensibility. Each module can define own different parameters because the structure is based on *field- value* schema.

The logging structure is composed of seven fields:

- sessionId – unique identifier of the session
- deviceId – type of the device (e.g. tablet, TV, HMD)
- userId – user identifier
- component – name of the module/component which generated the message
- field – name of the parameter
- value – value of the parameter
- description – additional description or unit of measure

```
struct logDataMessage {  
    string sessionId;  
    string deviceId;  
    string userId;  
    string component;  
    string field;  
    string value;  
    string description;  
}
```

2.6. Quality of Experience

2.6.1. Description

Quality of Experience (QoE) represents the degree of delight or annoyance of the immersive visualization at the end-user's side. The QoE module in the ImmersiaTV platform is a piece of software that provides QoE estimations of the audiovisual content shown on the primary display device (tv screen) as well as the immersive display device (head-mounted display, smartphone, or tablet). These QoE estimations will be made available to other components in the ImmersiaTV platform, and can for example be used to steer the parameters inside the codec.

The QoE module will receive the additional logging information described in the Section 2.6.2, which will be used to adapt the QoE estimations to the viewing context (display device specifications, room setup, frame-rate, etc.)

2.6.1. Interfaces

The QoE module will have access to all audiovisual content from which it will extract perceptually relevant features. Examples of these features are artefact detectors (synchronisation issues between audio and video, delays, stuttering, compression artefacts, etc.) and image attributes (local contrast, spatial/temporal information, brightness, colorfulness, etc.).

The QoE module will pass the extracted feature values to an integrated objective quality metric, an algorithm that uses machine learning to process the available data into objective scores indicating the QoE. The machine learning parameters will be predetermined based and will remain constant in the QoE module.

The output is one floating point stream per display device, whose values vary between 0 and 1. The floating points represent QoE estimations at equidistant time intervals (the higher the floating point value, the higher the QoE at that time interval). The length of the intervals will be parameterized in the software.

2.6.1.1. Logging information sent to QoE module

The QoE module will require both static and dynamic logging information as input. The static information will be sent once when setting up the QoE module. The dynamic information needs to be continuously recorded at specific time intervals.

The logging information will be provided in XML format (Table 5). The format should be backwards compatible if one of us decide to add new parameters, also it should allow recording only a subset of full parameters' list. The logging module will be directly programmed in Unity, C# and a repository will be created and shared for the QoE module. Ultimately, it will be open-sourced. The targeted platforms are Windows, MacOS X, iOS, and Android.

Below there is a list of all static and dynamic information stored by the logging module.

- **Static Information** (sent once)
 - a. **User information** (provided by experimenter):
 - user ID
 - legal information (name, consent form, terms and conditions)

- pre-experimental questionnaire (gender, background, mood, age, visual acuity, color vision, other)
 - post-experimental questionnaire (subjective quality ratings, qualitative feedback, other)
 - b. **Setup of the room** (provided by experimenter):
 - condition (lab, semi-open, open), viewing distance from monitor, other
 - c. **Specifications of display/audio devices and playback software** (logged automatically with amendments from experimenter):
 - external monitor (display size, resolution, frame rate, other)
 - audio device (headphone/earbud/speaker, # speakers, other)
 - immersive display (phone/tablet/HMD, resolution, pixel density, other)
 - ambient light conditions (intensity, color temperature)
 - playback software version
 - d. **Content identifiers and service-level metadata** (logged automatically)
 - codec specifications
 - transmission channel specifications (total bandwidth, transmission protocol)
 - content descriptors
- **Dynamic Information** (continuous recordings)
 - e. **Position of user in virtual world** (logged automatically)
 - a. **Movement acceleration of HMD** (logged automatically)
 - b. **Viewport/field of view** (logged automatically) i.e. displayed portion on HMD, smartphone or tablet
 - c. **Audiovisual monitoring of subject** (video recorded in lab) position and actions of user
 - d. **Frame-level Quality-of-Service parameters** (logged automatically in gstreamer) instantaneous bit rate, network delay, packet loss, frame rate, etc.
 - e. **Timestamps**

```
<?xml version="1.0" encoding="UTF-8"?>
<Logging_information>
  <Static_information>
    <User_Information>
      <user_ID>ID</user_ID>
      <legal_information>
        <name>Enter Name</name>
        <consent_form>Enter consent form</consent_form>
        <terms_and_conditions>Enter terms</terms_and_conditions>
      </legal_information>
      <pre_experimental_questionnaire>
        <gender>Male</gender>
        <age>24</age>
      </pre_experimental_questionnaire>
      <post_experimental_questionnaire>
        <subjective_quality>Enter values</subjective_quality>
        <qualitative_feedback>Enter feedback</qualitative_feedback>
      </post_experimental_questionnaire>
    </User_Information>
  </Static_information>
</Logging_information>
```

```

<Setup_of_the_room>
  <condition>Enter condition</condition>
  <viewing_distance>Enter distance value</viewing_distance>
</Setup_of_the_room>

<Specifications_of_display_audio_devices_and_playback_software>
  <external_monitor>
    <display_size>Enter display size</display_size>
    <resolution>Enter resolution</resolution>
    <frame_rate>Enter frame rate</frame_rate>
  </external_monitor>
  <audio_device>Enter audio device type</audio_device>
  <immersive_display>Enter display type</immersive_display>
  <ambient_light_conditions>
    <Intensity>Enter Intensity</Intensity>
    <color_temperature>Enter temperature
      </color_temperature>
    </ambient_light_conditions>
  <software_version>Enter version</software_version>
</Specifications_of_display_audio_devices_and_playback_software>

<Content_identifiers_and_service_level_metadata>
  <codec_specifications>Enter specifications</codec_specifications>
  <channel_specifications>
    Enter specifications
  </channel_specifications>
</Content_identifiers_and_service_level_metadata>
</Static_information>

<Dynamic_information>
  <Position_in_virtual_world>Enter position</Position_in_virtual_world>
  <Movement_acceleration>Enter acceleration</Movement_acceleration>
  <Viewport_field_of_view>Enter value</Viewport_field_of_view>
  <Audiovisual_monitoring>
    Enter position and action
  </Audiovisual_monitoring>
  <Frame_level_QoS_parameters>
    <instantaneous_bit_rate>Enter bit rate</instantaneous_bit_rate>
    <network_delay>Enter delay</network_delay>
    <packet_loss>Enter packet loss</packet_loss>
    <frame_rate>Enter frame rate</frame_rate>
  </Frame_level_QoS_parameters>
  <Timestamps>Enter Timestamps</Timestamps>
</Dynamic_information>

</Logging_information>

```

Table 5: Example of an XML format of the logging information sent as input to the QoE module.

2.6.1. Implementation Plan of QoE

The Implementation Plan of the QoE, depicted in Figure 15 involves four sub-modules as follows:

INPUT DATA module, FEATURE EXTRACTION module, MODELING module and TESTING module.

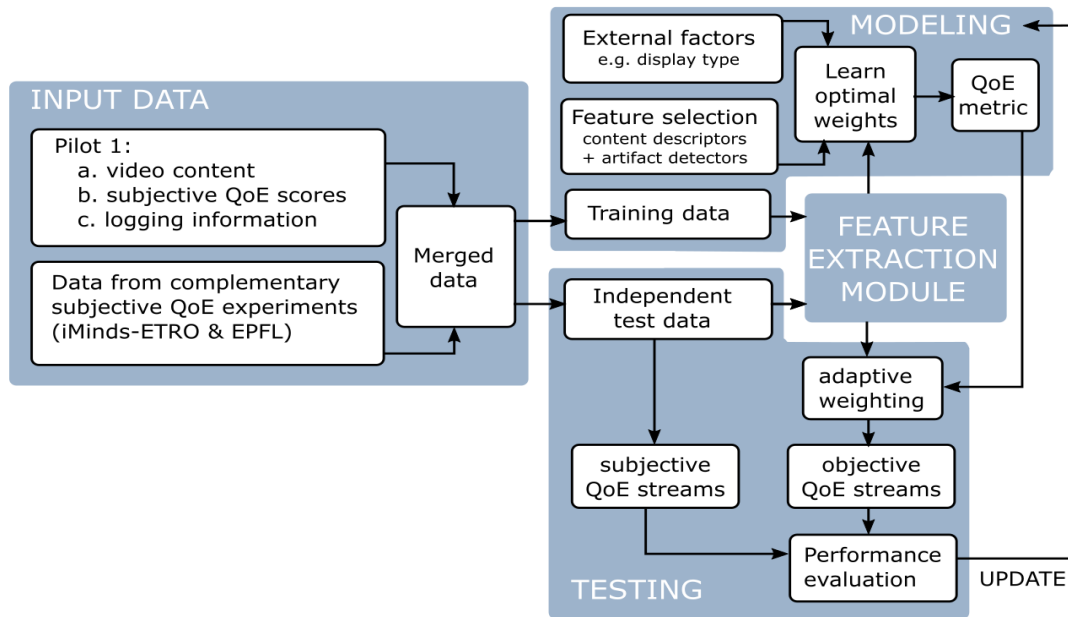


Figure 15: QoE Implementation Plan Overview

In the INPUT DATA module, the subjective scores and logging information from Pilot 1 are merged with complementary data that will be obtained through subjective QoE experiments conducted by iMinds/Etro & EPFL. This data will be divided into two disjoint data sets (one used for training, the other for testing).

Both the training and test data sets are subjected to the FEATURE EXTRACTION MODULE, which computes a huge pool of perceptually relevant features (content descriptors and/or artefact detectors).

In the MODELING module, the QoE metric is designed in two steps. First, the training data is used to make a sparse vector that contains the most appropriate features for quality assessment. Second, the selected features are adaptively combined based on the video content as well as external factors such as the display type and room setup.

In the TESTING module, the performance of the QoE metric is analyzed, based on which the MODELING module is updated. To this end, the QoE metric is evaluated on the independent test data and the resulting objective QoE streams are compared with the ground-truth subjective QoE streams.

3. CONCLUSIONS

This document presents the architecture design of ImmersiaTV system being a base for all components and tools. Although, the document focuses on implementation for Pilot 1, it contains some details also for further implementation towards Pilot 2 and 3. As this document is iterative, next months whole team of the project will be working on further definition of architecture of hardware and software components that are missing.

Although the market of the omnidirectional cameras is very dynamic and new models are presented by various vendors, most of development still focuses on low-resolution home devices for amateurs. The only way we can foresee to achieve goals of ImmersiaTV (high resolution high frame rate live camera) is to design and construct own camera that meets all the requirements. Some parts of the document (Chapter 2.1) reflect these concerns.

Similarly, regarding the codec and QoE for Pilot 1, these issues are not fully covered in this version of the D3.1, however we defined the codec that is common for cameras, processing modules, production tools, as well as streaming and receiving applications. In next version, more details on the specific codec and QoE module will be provided. For Pilot 1, the most important thing is to define what statistics will be gathered from all components and how to analyse them by QoE tools.