Grant Agreement No.: 687645
Research and Innovation action
Call Topic: H2020 ICT-19-2015

Object-based broadcasting – for European leadership in next generation audio experiences

D2.4: Final Reference Architecture Specification and Integration Report

Version: v1.0

Deliverable type	R (Document, report)
Dissemination level	PU (Public)
Due date	28/02/2018
Submission date	07/03/2018
Lead editor	Michael Weitnauer (IRT)
Authors	Michael Weitnauer (IRT), Chris Baume (BBC), Andreas Silzle (FHG), Nikolaus Färber (FHG), Olivier Warusfel (IRCAM), Nicolas Epain (BCOM), Tilman Herberger (MAGIX), Nikolaus Färber (FHG), Benjamin Duval (TRI), Niels Bogaards (ECANDY), Andrew Mason (BBC), Marius Vopel (MAGIX)
Reviewers	Simon Tuff (BBC), Werner Bleisteiner (BR)
Work package, Task	WP2, T2.1, T2.2, T2.3, T2.4, T2.5
Keywords	Architecture, workflow, specification

Abstract

This document describes the final reference architecture of ORPHEUS, a completely object-based, end-to-end broadcast and production workflow. It has been the subject of intensive discussions and several iterations over the duration of the project and has been shaped by considering typical channel-based broadcast workflows as well as the knowledge gained and lessons learned from the pilot phases. The architecture is format and interface agnostic and, as far as is possible, it should be applicable to a range of different infrastructures and ecosystems. Additionally, the pilot implementation and integration is also summarised.

[End of abstract]

Document revision history

Version	Date	Description of change	List of contributor(s)
v0.1	09/01/2018	Initial version	IRT
v0.11	26/01/2018	MAGIX text in 2.2.2 revised	Marius & Tilman
v0.2		Multiple contributions from partners	BBC, BCOM, BR
v0.3	01/02/2018	Merged contributions from partners for section 2 and section 3	IRT, ECANDY, BBC
V0.31	01/02/2018	Agreed changes during the WebEx	
V0.32	01/02/2018	Input to 2.2.5	FHG
v0.4	08/02/2018	Contributions to 2.2 and 3	BCOM, BBC, MAGIX, FHG, ECANDY
v0.5	15/02/2018	New section 2.3 and major updates for section 3.3	FHG, BBC, IRT, ECANDY
v0.6	21/02/2018	Edits in Section 2.2.1, 2.2.6, 3.5, 3.6, Review of all sections up to and including 2.2 from Werner Bleisteiner	BCOM, BR, ECANDY, IRCAM
v0.7	26/02/2018	Merged contributions from FHG and BBC; review of Werner Bleisteiner	FHG, BBC, BR
v0.8	28/02/2018	Addressed feedback from reviewers; prepared for PMC	BBC, BR, IRT
v1.0	07/03/2018	PMC feedback, Final editing by Eurescom	IRCAM, IRT, EURES

[1] http://creativecommons.org/licenses/by-nc-nd/3.0/deed.en_US

Executive Summary

One of the major objectives of ORPHEUS was the specification of a reference architecture for end-to-end object-based production and broadcast workflows. The intention of which is to provide guidelines for other content producers and broadcasters to build an object-based production and distribution workflow from scratch or to adapt an established infrastructure to incorporate object-based components.

To achieve this, the ORPHEUS consortium first specified the pilot implementation architecture which was the basis for the pilots in the project. Using an iterative process and based upon the findings and lessons learned from these ORPHEUS pilots, the reference architecture was developed. Unlike the pilot implementation architecture the reference architecture is more format and interface agnostic, allowing it to be more easily used as a general guideline for other broadcasters.

This document forms an ORPHEUS project deliverable and is an update of D2.2, which reported our architecture specification activities until M14. It also contains descriptions of any significant results from work packages 3-5, as WP2 is the ‘umbrella’ work package of ORPHEUS. This is where all the technical developments are integrated and combined to build the object-based workflows necessary for our pilots.

Each stage of the reference architecture is described in detail, with identified components, interfaces and protocols for a complete object-based audio workflow.

Additionally, the implementation and integration of all pilot phases is reported in this document in detail.

Table of Contents

List of Figures

List of Tables

Abbreviations

AAC	Advanced Audio Coding
AAX	Avid Audio eXtension
ADM	Audio Definition Model
AES67	Audio Engineering Society standard for audio over IP
API	Application Programming Interface
ATSC	Advanced Television Systems Committee
BW64	Broadcast Wave 64 Bit
CDN	Content Distribution Network
CSS	Cascading Style Sheets
DAB+	Digital Audio Broadcasting +
DASH	Dynamic Adaptive Streaming over HTTP
DASH-IF	DASH Industry Forum
DAW	Digital Audio Workstation
DRC	Dynamic Range Control
DVB-S	Digital Video Broadcasting Satellite
EBU	European Broadcasting Union
FIR	Finite Impulse Response Filter
GPS	Global Positioning System
gRPC	Google remote procedure call
HOA	Higher Order Ambisonics
HTML	Hyper Text Markup Language
IIR	Infinite Impulse Response Filter
IP	Internet Protocol
ITU	International Telecommunication Union
JSON	JavaScript Object Notation
MP4	MPEG-4 File Format
MPD	Media Presentation Description
MPEG	Moving Pictures Expert Group
MSE	Media Source Extension
NGA	Next Generation Audio
NMOS	Networked Media Open Standards
OBA	Object-based Audio
PCM	Pulse Code Modulation
RFID	Radio Frequency Identification

RTP	Real-Time Transport Protocol
UI	User Interface
UMCP	Universal Media Composition Protocol
XML	Extensible Markup Language
VBAP	Vector Base Amplitude Panning
VST	Virtual Studio Technology

1 Introduction

This document forms an ORPHEUS project deliverable and is an update of D2.2, which reported our architecture specification activities until M14. It contains the final reference architecture specification of ORPHEUS, which has been the subject of intensive discussions and several iterations over the duration of the project. The architecture described here has been shaped by taking into account typical channel-based broadcast workflows but, more importantly, by also including the knowledge and lessons learned during the pilot phases and stages

1.1 Motivation

The object-based approach to media gives the most fundamental opportunity to re-imagine the creation, management and enjoyment of media since the invention of audio recording and broadcasting. One may argue that the core of the object-based approach to media is not new in itself, and that object-based audio has not been fully adopted so far, despite the fact that past and recent developments ([1][2][3]) have utilised some variation or parts of the object-based concept. This is due to the fact that previous attempts did not cover the full creative and technical process including planning, editing, production and leading to the consumption of object-based audio. This has led to isolated and partial solutions, which did not take into account enough of the whole system required to unlock the full potential of an object-based approach.

The ORPHEUS project therefore opts for an integrated method, incorporating the end-to-end chain from production, storage, and play-out to distribution and reception. Only through this approach can it be ensured that the developed concepts are appropriate for real-world, day-to-day applications and are scalable from prototype implementations to large productions. In order to achieve this, the ORPHEUS project structure has been designed with a full media production chain in mind.

1.2 Relation between Pilot Implementations and Reference Architecture

One major objective of WP2 was the definition and specification of an overall valid *Reference Architecture* for object-based audio broadcasting.

This *Reference Architecture* is to provide general specifications for the required interfaces and components for an end-to-end production chain, with the intention that it will be beneficial beyond the lifetime of the project, since it can serve as a general guide to setup, integrate or migrate an object-based production workflow. Additionally, a *Reference Architecture* should be format, interface and protocol agnostic as much as is practicable, thereby making it useful to a large number of broadcasters, who’s infrastructures can vary substantially. To this end, a list of different options for formats, tools or components, is included.

In order to achieve this goal in a practical way, using the object-based tools from consortium partners, many of which were under development during the course of the creation of pilot material, we had to take an intermediate step, by creating and applying multiple *Pilot Implementations*.

This was a practical approach, based on established infrastructures and workflows of the broadcast partners in the project, but integrating occasionally newly developed object-based tools of our consortium members into the production process.

WP2 and T2.1 focussed on the specification of the *Pilot Implementation Architecture* with implications for the *Reference Architecture* in mind.

It is worth emphasising that the ORPHEUS *Reference Architecture* is based on existing channel-based broadcaster and production workflows and by using these existing workflows, which have been perfected over many years, as the starting point it helps to ensure the quality and cost effectiveness of the final design.

Within the ORPHEUS project a complete end-to-end object-based broadcasting workflow based on open standards has been assembled for the first time. Hence, for the definition and specification of a *Reference Architecture*, the actual pilot implementations were essential to gather the knowledge and experience for the universal, implementation-agnostic reference architecture. This has ultimately led to a step-by-step development of the final *Reference Architecture*.

2 Reference Architecture Specification

2.1 Basic components of a radio broadcast workflow

The typical audio broadcast workflow contains five components, organized in macroblocks:

- Recording
- Pre-production and Mixing
- Radio Studio
- Distribution
- Reception

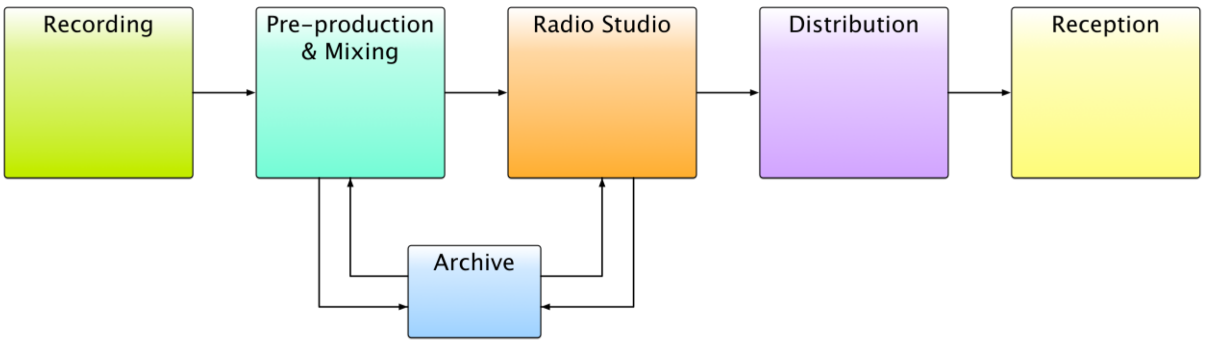


Figure 1: Basic radio broadcast workflow

The well-known workflow shown in Figure 1 is the result of the long-term experience of broadcasters and content producers and is proven to deliver reliability, efficiency and quality in both the content and technical aspects for the final programme.

The ORPHEUS project has used this model as the starting point for both pilot implementations and the reference architecture, as it was more practical and efficient than creating a completely new system from scratch.

This approach also helped to demonstrate very early on how feasible it would be to adapt existing broadcast systems to implement object-based broadcasting

However there are still challenges in delivering the full capability of new object-based media systems and, in order to deliver the full listening experience and achieve the best representation, various features (technical parameters, additional metadata) have to be (automatically) generated and converted, (manually) created and finally injected into the object-based media stream at various points in to the traditional workflow.

In the following chapters, we describe in detail our considerations, approaches and, occasionally bespoke (single) solutions to resolve these challenges within the limited field of our project.

Thus, inevitably, object-based broadcasting will cause the workflows in existing legacy broadcasting systems to evolve.

To make this eco-system work it appears that we have an example of the classic "chicken or the egg" problem. What do we need first? Is it production tools or reproduction devices? But unlike other media digitisation challenges the capability to replay object-based media is already widely available, as it is present in the majority of smart phones and personal computers. The distribution infrastructure, in the form of IP networks, is also universal.

This leads us to conclude that enabling production to work with object-based audio is the main challenge.

2.2 Reference Architecture

This section describes the actual ORPHEUS reference architecture, which is illustrated in a high-level diagram in Figure 2.

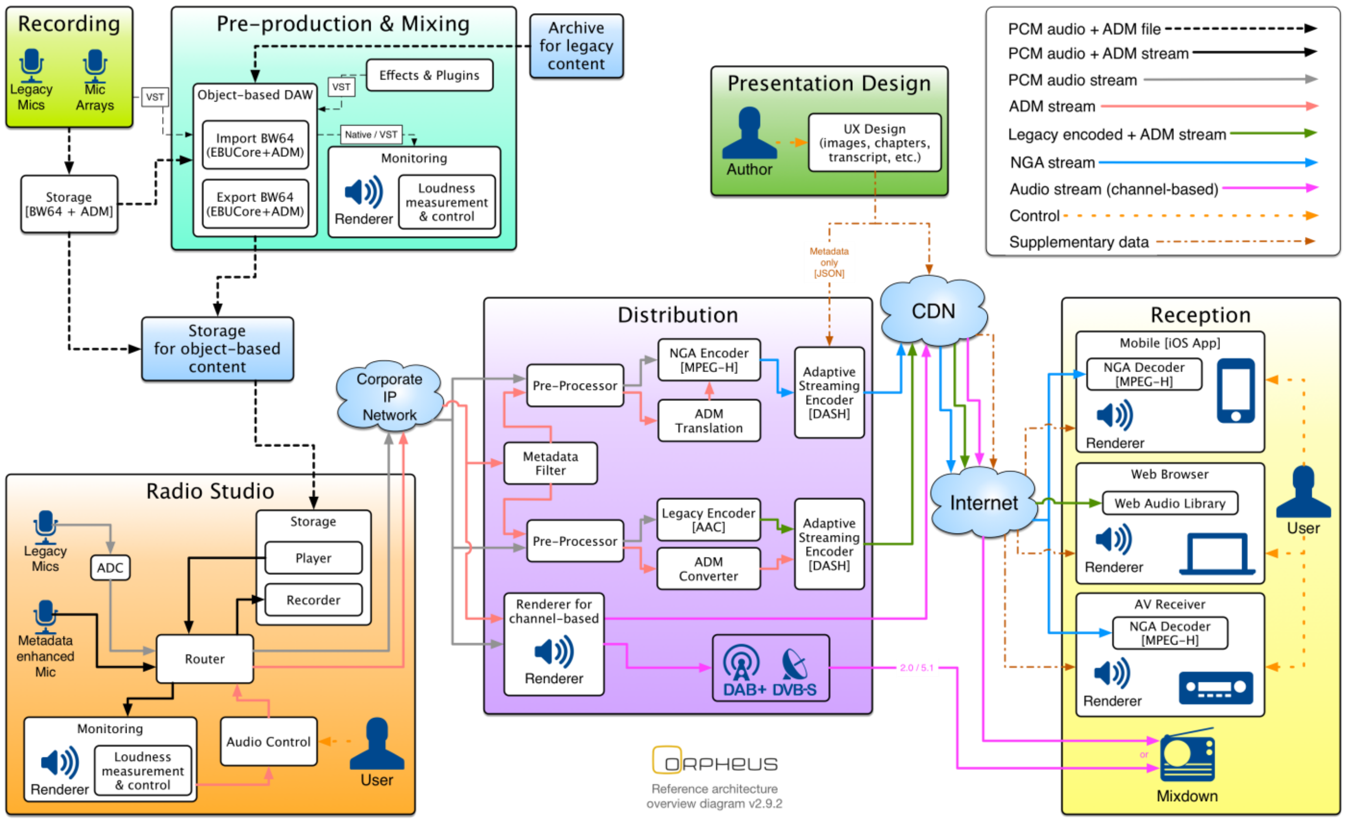


Figure 2: ORPHEUS Reference Architecture

2.2.1 Recording

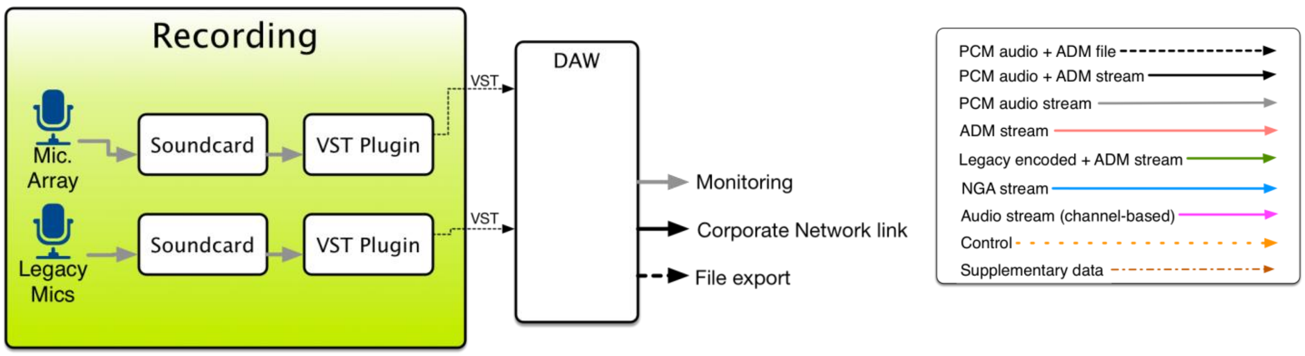


Figure 3: Detailed Recording block structure of the ORPHEUS reference architecture.

The purpose of the recording macroblock is to provide the tools and infrastructure required to conduct recordings for the production of object-based audio content. Currently object-based audio content, as described by the ADM standard, may consist of audio data in the following three formats:

- Pure object-based audio, i.e. mono or stereo tracks along with metadata describing the position, width, etc. of the object.
- Scene-based audio, i.e. a number of audio tracks describing a scene or sound field. In the

scope of this project, this format would be HOA (Higher Order Ambisonics).

- Channel-based audio, i.e. a number of audio tracks corresponding to specific speaker positions. This includes stereo but also more spatialised formats like 4+7+0[2].

Sound engineers typically record audio using a Digital Audio Workstation (DAW). In this context, recording with specific equipment, such as microphone arrays, or in specific formats, requires the use of audio plugins (additional software running within the DAW). The two most frequently used formats for audio plugins are Steinberg's VST and Avid's AAX. Note that both VST and AAX are proprietary formats implemented using SDKs that are available from the respective company websites. The VST interface is supported by many more DAWs than the AAX interface, thus the former should be favoured over the latter. However, as Protools (Avid), which supports only the AAX interface, is widely used also in broadcast production environments, it is difficult to make a clear recommendation here.

In the case of a live production, or when the content producers wish to mix the audio signals prior to recording, the recording block should share an interface with the pre-production and mixing block. This interface should typically occur within the DAW, which is used to monitor and/or mix the recorded signals, and possibly add or edit the corresponding metadata. As explained above, the interface should then be either VST or AAX depending on the DAW.

In addition, in the case of non-live productions, the audio data produced by the recording macroblock may be stored for later use. The audio signals should then be stored as BW64 files with ADM metadata. ADM is the standard used for describing object-based audio content in the ORPHEUS project.

<u>Formats, protocols and Interfaces to other components</u>

The Recording macroblock has the following output-interfaces:

File formats

[BW64]	Long-form file format for the international exchange of audio programme materials with metadata, Recommendation ITU-R BS.2088-0.
[ADM]	Audio Definition Model, Recommendation ITU-R BS.2076-1.

Audio plugin interface standards

[VST]	Virtual Studio Technology, the Steinberg GmbH interface for audio plugins (See the Steinberg website for more information).
[AAX]	The Protools (Avid Technology) audio plugin format. Information about the interface is available from Avid Technology by signing in as a developer.

[2] See Recommendation ITU-R BS.2051-1: This new nomenclature is adapted throughout this document

2.2.2 Pre-production and Mixing

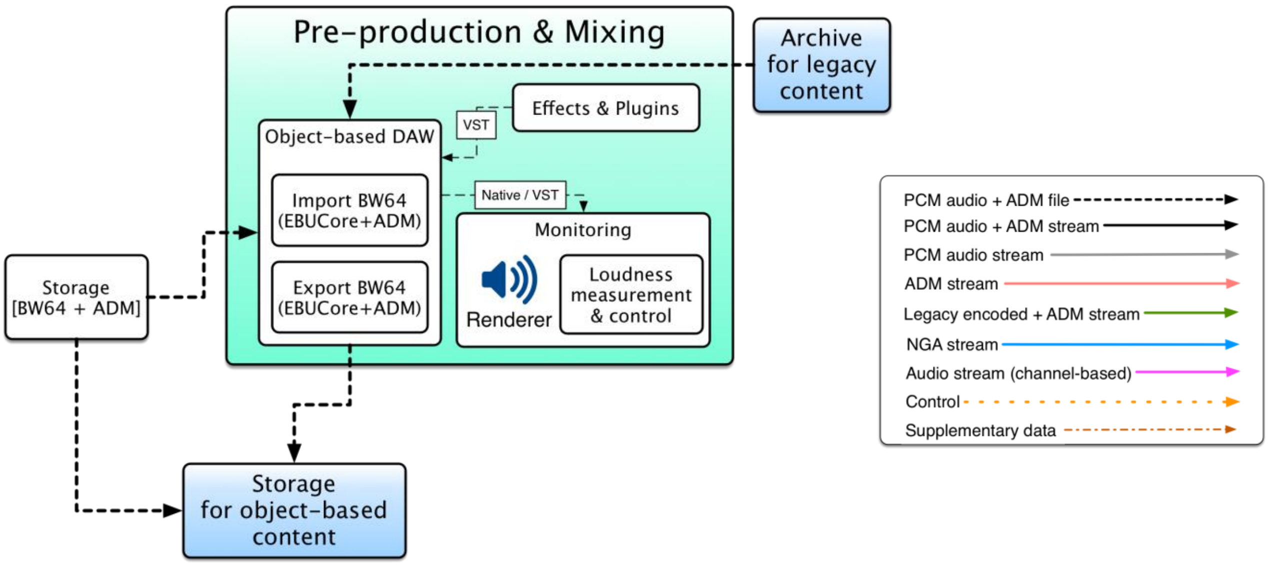


Figure 4: Detailed Pre-production & Mixing macroblock structure of the ORPHEUS reference architecture

The purpose of the pre-production and mixing block is to deliver tools for editing existing object-based content or creating such content from legacy audio material or other sources. The core of this block is the object-based DAW, which is extended by several tools and workflows to import, edit, monitor and export object-based content. These tools and workflows were developed and implemented in task T3.2. The DAW interacts with the components presented in the section below and illustrated in Figure 4.

Description of DAW components:

The following main components of the object-based DAW were identified:

Recording

The recording of object-based content may be done independently from the DAW. In this case the recorded content is transferred in form of multichannel BW64 files with ADM metadata. Depending on the established broadcast architecture, these files are stored in a temporary **ADM enabled storage** or directly in the **storage for object-based content.**

Additionally, there is the option to record object-based content directly within the DAW. This implicates further technical requirements for the DAW (e.g. 32 input channels for the microphone array processing described in the recording macro block).

Storage for object-based content

The DAW needs to be able to import and export object-based content in the form of uncompressed multichannel BW64 files with ADM metadata. This involves **decoding** and **encoding** these files including their metadata. Editing existing ADM metadata or additional metadata required for broadcasting may be implemented directly in the DAW or in form of stand-alone tools.

Archive for legacy content

For integration into an established broadcasting architecture, the access to existing content is very important. For this purpose, the DAW can at least convert legacy content to object-based content by adding the required object-based metadata to it manually. This process could be automated partially.

Effects and plugins

A typical usage for a DAW is the application of effects or plugins onto audio tracks. Basically, applying an effect consists in taking a given sound signal and changing it somehow. Typical effects used in audio production are filtering effects (changing the frequency content of a sound) or dynamic effects (limiter or compressor, to change the dynamic of an audio track). Plugins are extensions for the DAW to be used with audio tracks such as special filter effects.

Usually effects and plugins can be applied to audio content within the DAW. Due to the nature of the ADM format there are some limitations to consider here. Especially there is no final mastering stage for a certain channel format, but effects can be applied to individual audio objects only.

For the scope of ORPHEUS, it was decided that any relevant plugins were made available as VST 2 plugins. This also included format conversions e.g. converting 3D microphone input to HOA format or converting HOA format to channel-based surround formats. Plugins for these use cases already exist and are delivered e.g. by b<>com.

A special use case is applying reverb to the audio signal. Currently reverb could only be included as a fixed part of the individual objects or as an additional audio object, which partially limits the possibilities of ADM. An object-based reverb algorithm and inclusion into ADM format was investigated by IRCAM in the scope of T3.2.

To deal with 3D spatial audio, the panning engine of the DAW needs to provide a suitable interface for handling 3D automation curves and to convert them from and to the ADM objects as well as to the renderer plugin.

Monitoring

For previewing the object-based content and simulating the user experience, it is important to allow monitoring with different speaker setups (including binaural monitoring). For this purpose, an object-based renderer needs to be implemented in the DAW. The ITU-R WP6C is currently in the process of standardizing one or multiple options for a production renderer. For ORPHEUS, however, the MPEG-H renderer was chosen, since the ITU-R standardization is still ongoing. But it should be mentioned that the MPEG-H renderer is one of the candidates for this Recommendation.

An essential part of the monitoring is **loudness measurement and control.** This may be either supported by additional VST plugins in the DAW and is researched in T3.4 or implemented as part of the rendering solution.

In addition to pure audio monitoring, other parts of the user experience may be simulated within the DAW, e.g. transport control, projects with variable length or exchanging language specific objects. This was researched in T3.4 and in T5.4.

<u>Formats, protocols and Interfaces to other components</u>

[BW64 + ADM]	Audio Definition Model (ADM): BS.2076-1, June 2017, ITU.
[WAV][AAC][MP3]	or other common audio formats to be imported from legacy content archives
[VST]	Virtual Studio Technology, the Steinberg GmbH interface for audio plugins

2.2.3 Radio Studio

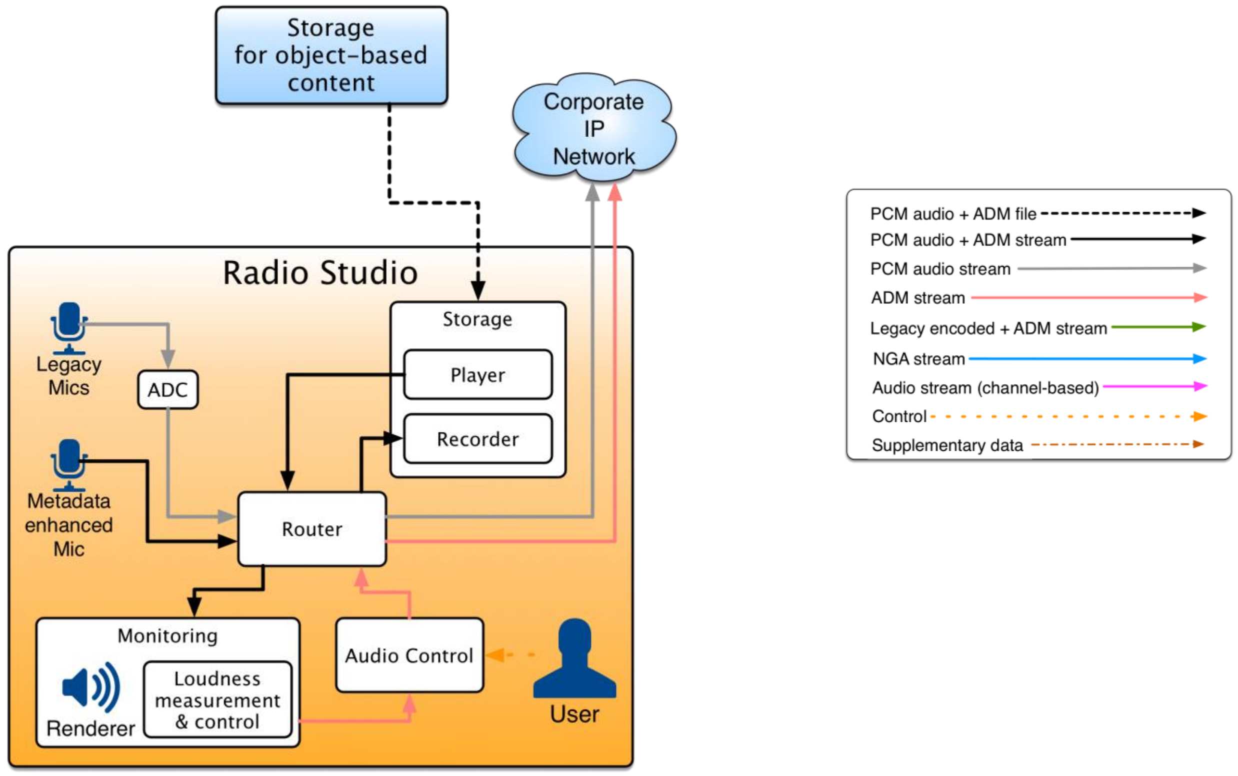


Figure 5: Detailed Radio Studio structure of the ORPHEUS reference architecture

There are three main aims to the radio studio macroblock: capture and control, monitoring, and record/replay. Each of these is explained below, with detail about the functionality within each. We then describe the interfaces used to communicate between components.

Capture and control

In the studio macroblock, we want to be able to capture audio sources, and describe how these and other audio sources should be combined into a single programme, or 'composition'. In a traditional studio, these audio sources would be mixed together to produce a new audio source, however with the object-based approach, the audio sources remain separate and we generate metadata to describe how they should be combined.

In describing a composition, the following information must be captured, stored, and transmitted alongside the audio:

- **Sources**: Which audio sources are parts of the composition, and which audio source the metadata relates to.
- **Gain and position**: How much gain should be applied to each audio signal, and where the audio should be panned within the auditory scene.
- **Labelling**: A description of what the audio source is. This will vary based on the context of how the audio is used. For example, the labelling could describe whether the audio source is a foreground or background sound, or identify a person that is speaking. Some of this information could be automatically gained using devices that identify themselves and their location (e.g. a networked microphone, enabled with an RFID reader)

We have chosen to use the NMOS specifications for our radio studio. This provides open standards for mechanisms that handle identification, timing, discovery and registration. We are also using the Audio Definition Model (ADM) to describe audio metadata such as gain, position and labelling.

Audio control interface

To capture and transmit this control data, there must be a user interface that the engineer or producer can use to achieve this. The features of the user interface must include the following:

- **Creating a new composition**: Generating a blank programme that will be populated with audio objects.
- **Adding objects**: Selecting a new audio source and including it in the composition.
- **Controlling gain/position:** Being able to set the gain/position of each object dynamically.
- **Labelling:** Setting labels for each audio object appropriate to the context of the use case.

Traditionally, the audio control interface would be a mixing desk, made up of a series of faders and knobs. However, to be able to capture the rich metadata required for an object-based production, it may be necessary to replace the mixing desk with a graphical user interface, or at the very least, use a combination of physical controller and graphical interface.

Pre-production interface

In a live workflow, some of this data may have to be entered on-the-fly, such as gain and position, however much of the data can be prepared in advance. For example, the identity of contributors is often known in advance, so this information should be captured in pre-production. This pre-production information could be captured using a specialised interface designed for the producer, who generally looks after the editorial side of the programme. Examples of data to be captured using this interface include:

- **Programme title/description**: An overall view of the composition
- **Running order**: A sequence of chapters or scenes in the programme, with descriptions for each
- **Contributors**: A list of people who are involved in the production, including producer, engineer, writer, presenter and guests.
- **External Links**: This could include URLs related to the content, or a link to a related image.

Once pre-production information has been captured, then its use must be triggered at the appropriate moment. This could be done manually through the audio control interface with the engineer pressing a button. Otherwise, it could be done automatically using other inputs such as ID badge readers, or using signal processing to work out when a microphone is being used.

Metadata transmission

Methods for the transport of audio over networks is well specified (e.g. AES67), however work on defining techniques for transporting audio metadata is in its early stages. We are investigating using the Universal Media Composition Protocol (UMCP), which is being developed at the BBC for use in media metadata transport.

UMCP provides a way to describe ‘compositions’, i.e. how various NMOS-described sources can be combined into a single experience. This protocol provides a flexible and sustainable way to link metadata to audio streams. It can either be carried over WebSocket connections, or using RTP packets. UMCP uses a server model to store incoming compositions and distribute them to any receivers that have subscribed to updates.

Monitoring

When producing an audio experience, it is important to be able to monitor the audio output and ensure it is complying with the appropriate technical and editorial standards. This can be done using two methods: first by listening to a rendered version of the output, and secondly – only for technical compliance - by using metering to visually monitor each object and the output.

Audio monitoring

A renderer is a system that takes an object-based audio composition and generates a set of output signals to drive loudspeakers or headphones. For monitoring the output of a production in a radio studio, this typically involves rendering to a set of loudspeakers so that the engineer and producer can listen to the output. Alternatively, it could be rendered to headphones for individual monitoring.

A variety of rendering systems are available, but it is up to each organisation to select the one most appropriate to them. There are currently efforts in the ITU-R to standardise a 'production renderer', however this falls outside the timeline of the ORPHEUS pilot. For the pilot, we used a VBAP-based renderer, starting with the BBC's own implementation, succeeded by the MPEG-H rendering system.

Audio metering

Metering involves processing an audio source using an algorithm to calculate the level of some property of the signal. The levels that we use in ORPHEUS are loudness and true peak, as described by EBU R128. Each audio object is routed through a processing node to calculate the levels, and these can then be displayed on the audio control interface. This allows the engineer to monitor the audio signals of each object. The output of the studio renderer is also metered to ensure the overall output levels are appropriate.

Recording/replay

The ability to record and replay material is an essential aspect of audio production. However, current tools only offer a way to record the audio data. For object-based production, we need to be able to record and replay both audio and metadata. To achieve this, we are using a 'sequence store' in IP Studio, which records audio and metadata flows. These are recorded into a database along with the timestamps of each grain in the flow.

To interact with the sequence store, we must use an API. For the ORPHEUS pilot, we are using a BBC-developed interface called the 'Media Access API'. This allows systems to trigger the recording and replaying of streams.

The API to the sequence store can also be expanded to allow for importing/exporting of various formats. For the ORPHEUS pilot, we have implemented an import script, which can read BW64+ADM files, convert them to a stream of audio and metadata, then import that into the store.

Formats, protocols and Interfaces to other components

The Radio Studio macroblock has the following output interfaces:

Live output to Distribution

This macroblock exports an object-based stream as a UMCP composition containing ADM metadata of NMOS audio sources. An overview of these protocols is described below:

[NMOS] Networked Media Open Standards are a set of open specifications, which define four important aspects of interfacing object-based audio over IP:

1. **Identity** – How to identify each individual audio source and stream of data ('flow') from those sources.
2. **Timing** – How to define when audio was sampled, when metadata changed, or when events occurred.
3. **Discovery and registration** – How devices announce themselves as being available, and advertise what streams they can offer.
4. **Routing** – How streams get routed between devices on the network, and how devices can request to receive streams.

The NMOS specification is available at http://nmos.tv

[ADM] Audio Definition Model is used as a data model as a standard way to describe a sound scene made up of audio objects. ADM has been used in two ways:

1. in combination with BW64 to describe the contents of object-based audio files
2. as a data model to describe audio parameters over UMCP for streaming of object-based audio.

The ADM specification is described by Recommendation ITU-R BS.2076 “Audio Definition Model”.

[UMCP] Universal Media Composition Protocol is used to link the ADM metadata to NMOS-described audio sources. Currently (January 2018), ADM is used in a file format and can only describe audio sources as channels within the file. In a live production, the audio objects are distributed on a network so cannot be described as ‘channel 1’, for example. UMCP solves this problem. UMCP links into the NMOS specifications we are using to identify audio sources, and to describe the timing of events. Although a UMCP stream could be sent directly from a transmitter to a receiver, it is normally routed through an API that records the stream and distributes it to any receiver that has subscribed to it. UMCP is currently being developed into an open specification for publication.

For further information about used subsets or specific profiles of the identified interfaces, see Deliverable D4.2.

2.2.4 Presentation Design

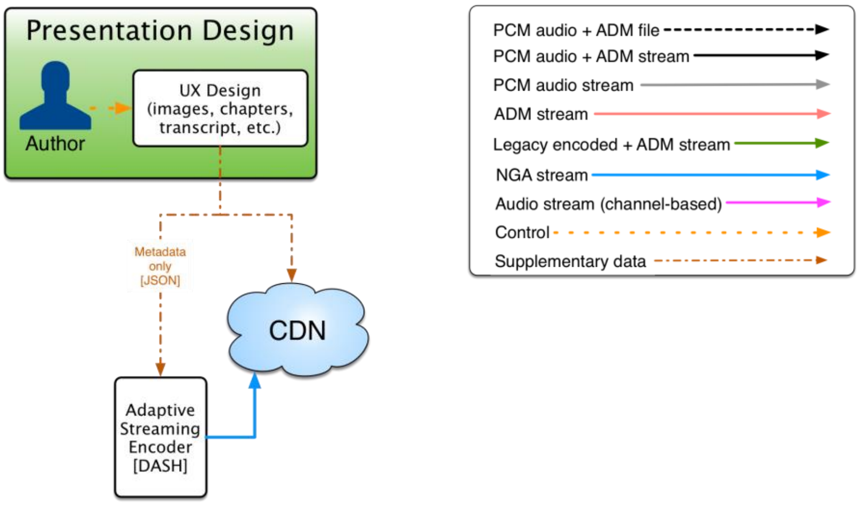


Figure 6: Detailed Presentation Design structure of the ORPHEUS reference architecture

A key motivation for object-based broadcasting is the desire to transition from monolithic ‘one-size-fits-all’ streams that are fixed in both content and presentation to programs that can be personalised to a listener’s interests, wishes and playback environment.

Object-based broadcasting can enable a wide variety of novel interactions with the audio broadcast content, both in the determination of what and in how it is rendered. For a particular program and a particular user or audience, a specific subset of the possible interaction features may be selected. To make this interaction simple, understandable and effective, specific user interfaces are to be designed. As the design of these user interfaces can have a profound effect on the total experience, it is a component to be considered already during creative and editorial processes.

Besides interaction with the audio objects in the broadcast, such as language selection, level adjustments, non-linear playback and spatialisation, OBA is very well suited to be complemented with additional information and metadata, such as complementary text and images relating to narrated text and speakers, multi-lingual program information and content-related tags.

Formats, protocols and Interfaces to other components

The formats and protocols that can be attached to an object-based audio stream are determined mainly by the presentation that the broadcaster wants to provide: for a highly interactive program in a browser standard web technologies like HTML can be used, if the goal is to just add some extra labels or textual information, a stream of inline JSON data within MPEG-DASH will suffice, as it is shown in Figure 6.

2.2.5 Distribution

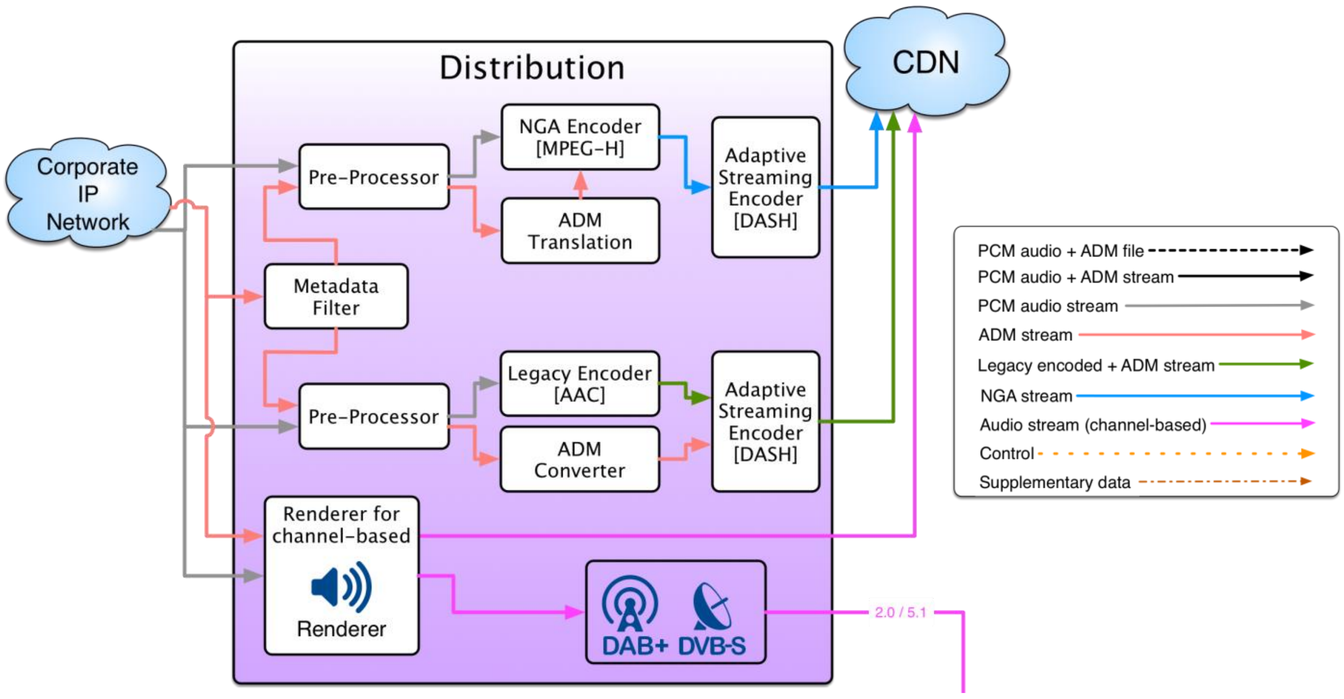


Figure 7: Detailed Distribution structure of the ORPHEUS reference architecture

2.2.5.1 Scope and functionality

This macroblock contains the modules and tools needed for the distribution of object-based audio from the broadcaster to the end user. It converts the production format as used within the broadcast infrastructure (AES67, BW64, ADM) into a more efficient format that is suitable for the transport over transmission channels and the reproduction in receiver devices (AAC, MPEG-H, DASH). The main distribution channel is the Internet (IP, TCP, HTTP) but, for backward compatibility, also legacy systems (DAB+, DVB-S, Shoutcast) are considered in the Reference Architecture.

The distribution-macroblock receives its input from the studio-macroblock via a private IP network as illustrated in Figure 2. For live production, the object-based audio is received as PCM via AES67 plus an ADM-based metadata stream. Both macroblocks use UMCP as the underlying protocol to establish and control data streams between each other (including audio and metadata). As the input interfaces and UMCP are already described in Section 2.2.3, we focus on the modules within the distribution-macroblock and its output-interfaces in the following.

1. **Internet Distribution:** The ORPHEUS Reference Architecture includes two options for distribution over the Internet: An AAC-based distribution to HTML5 browsers and one for clients with support for MPEG-H 3D Audio (iOS-App, AV-Receiver). Both paths are made available via the public Internet and are based on HTTP/TCP/IP as the underlying transport- and network-protocols of the World Wide Web. In addition, both use DASH as the most widely adopted streaming protocol as of today. As a consequence, a Content Distribution Network (CDN) can be used for scaling the service to many users and several geographical regions. As the usage of a CDN is transparent to the services defined in this document it is not covered in more detail.
2. **Distribution to legacy devices:** Though legacy devices, such as DAB+ and/or DVB receivers will never support the full functionality of object-based audio, they are important for backward compatibility to mass market legacy broadcasting systems. To enable this additional distribution path it is necessary to render object-based audio to channel-based versions, e.g. as a simultaneous downmix into 2, 0+5+0 surround and binaural. In addition, alternative language versions can be mapped into audio sub-channels and a subset of the audio-metadata can be mapped into existing metadata models of legacy systems. A similar functionality is required when emission via a classic Internet radio system, such as Shoutcast, is required.
3. **Filtering of private metadata:** For object-based audio, specific metadata has to be transmitted to the end-users as part of the distribution. However, not all metadata available within production is intended for the end-user. Hence, any information that is 'private' or used solely for internal purposes needs to be removed before distribution.
4. **Pre-Processing for Distribution:** Depending on the capabilities of the distribution format (e.g. MPEG-H), the produced/archived audio content and metadata may need to be pre-processed or converted. For example, the ADM metadata might need to be either translated into a different metadata representation or a new/modified ADM metadata stream needs to be created for emission. This process may also include a reduction of the total number of objects or a combination of several objects into one audio track or channel-bed. Such a conversion process typically translates a more generic ADM into a more constrained *ADM-Profile*, such that conversion into MPEG-H becomes more predictable.
5. **Conversion of ADM to MPEG-H Metadata:** Though the Pre-processor can significantly reduce the complexity of an ADM-based content, there still needs to be a final conversion from ADM into the metadata model of the emission codec, e.g. MPEG-H. This step must be as transparent and predictable as possible and should be a direct mapping of metadata elements. This, however, can only be achieved if a rather constrained and well defined subset of ADM is used as input, i.e. an ADM-Profile for emission via NGA codecs such as MPEG-H.

2.2.5.2 Output Interface Specification

In this section we describe the two main distribution paths to the end user, which are addressed in the ORPHEUS Reference Architecture. After providing a textual overview, we list the relevant specifications that define the output interface of the distribution-macroblock. We focus on the audio-related distribution but note that additional metadata is typically needed to achieve a good overall user experience, e.g. additional program information or text and images etc. The definition and transport of such additional metadata is described in Section 2.2.4.

Distribution to HTML5 Browsers

In order to reach a broad audience it is desirable to support commonly available browsers as reception devices. However, as current browsers do not yet support Next Generation Audio (NGA) audio codecs such as MPEG-H, it is necessary to implement the required functionality as a *Web Application* based on JavaScript and other available browser APIs.

Web applications for object-based audio have already been demonstrated successfully and use HTML, CSS, and JavaScript for implementation. When the user visits the web site of a broadcaster, the web application is automatically downloaded and executed by the browser as part of the web site. It then downloads the Media Presentation Description (MPD) to start a DASH stream. The MPD links to the actual media, which is then downloaded as a series of fragmented MPEG-4 (fMP4) file segment. For decoding the fMP4 segments there are two approaches:

1) For audio programmes up to 8 Mono objects the audio stream can be decoded using the Media Source Extensions (MSE) in the browser.

2) For programmes with more than 8 channels, multiple streams must be used currently as the browser decoders support only up to 8 channels per stream. As no reliable synchronisation between multiple MSE browser decoders can be achieved, the WebAudio API is used to decode the media into AudioSourceNode objects.

These can then be scheduled for playback with sample accuracy and linked to a timeline. Finally, the WebAudio API is used to render the audio using the JavaScript rendering engine, e.g. to generate a binaural stereo signal from multiple audio objects. In order to do so, the web application also needs the audio-metadata, e.g. the position of objects in 3D space. Those are also streamed via DASH as file segments including ADM-metadata, which can either be transmitted as XML or encoded as JSON. Though the exact implementation of the browser-based web application is out of scope for the distribution-macroblock, the named browser APIs implicitly define the required output-interfaces (JavaScript, CSS, HTML, MSE, WebAudio) and are therefore mentioned here.

It is worth noting that the exact definition of the streaming protocol does not have to be defined as long as the client implementation itself is downloaded as “mobile code” along with the data. For example, the exact protocol of how audio-metadata is represented and transmitted does not have to be defined as long as the JavaScript-based web application knows how to receive, parse, and interpret it correctly in order to drive the WebAudio API for rendering. In this sense, the distribution-macroblock only has to make sure that the encoded fMP4 segments are compatible with the MSE API and that the provided web application uses the MSE and WebAudio API correctly.

Figure 2 illustrates how the following functionality is implemented in the macroblock for the distribution path to browsers: First, the ADM metadata has to be received from the studio-macroblock and interpreted, e.g. to configure the AAC-Encoder with the appropriate number of objects. The AES67 stream (or BW64 file) is received and the de-capsulated PCM audio is fed into a multi-channel AAC-Encoder. The compressed AAC bit-stream is encapsulated into fMP4 file segments and stored on a DASH-Server who is also responsible for generating the MPD. In addition, the file segments with XML-or JSON-encoded ADM-metadata are generated, which are also streamed via DASH as a parallel data stream.

The distribution-macroblock has the following output-interfaces to the reception-macroblock when streaming to HTML5 browsers:

[HTML5]	HTML5, A vocabulary and associated APIs for HTML and XHTML, W3C Recommendation 28 October 2014, https://www.w3.org/TR/html5/
[CSS]	Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification, W3C Recommendation 07 June 2011, https://www.w3.org/TR/CSS2/
[JavaScript]	ECMA-262, ECMAScript 2016 Language Specification, 7th Edition / June 2016, http://www.ecma-international.org/publications/standards/Ecma-262.htm
[MSE]	Media Source Extension, W3C Recommendation 17 November 2016, https://www.w3.org/TR/media-source/

[WebAudio]	Web Audio API, W3C Working Draft 08 December 2015, https://www.w3.org/TR/webaudio/
[AAC]	ISO/IEC IS 14496-3:2009, Information technology - Coding of audio-visual objects - Part 3: Audio
[DASH]	ISO/IEC 23009-1, Information technology - Dynamic adaptive streaming over HTTP (DASH) - Part 1: Media presentation description and segment format

Distribution to Clients supporting MPEG-H 3D Audio

Though the browser-based distribution is a flexible solution, which is immediately deployable, it has several shortcomings compared to a “true” NGA audio codec such as MPEG-H. First, the implementation in JavaScript is less efficient than a native C-implementation and therefore the computational complexity can become a problem, especially on mobile phones. Secondly, rendering via the WebAudio API has the inherent limitation that audio content is not protected but “in the clear”, which can become a problem for premium content. Furthermore, MPEG-H supports not only objects but a flexible combination of channel-, object-, and scene-based (HOA) audio formats – all of which with a significantly improved coding efficiency compared to an AAC-based approach. Finally, MPEG-H provides a well-defined behaviour based on an open and interoperable standard, which can reduce the implementation and maintenance effort for broadcasters.

Because MPEG-H is an open standard and already adopted in several application standards, the output-interfaces for this path are well defined and documented. In addition to the actual MPEG-H 3D Audio standard, which defines the overall audio codec [MPEG-H], it is typically required to define a subset for a specific application. The ORPHEUS Reference Architecture follows the *Profile and Level* that has been defined in ATSC 3.0 and DVB, namely the *Low Complexity Profile at Level 3* [ATSC-3]. Those application standards also include further definitions and clarifications on the usage of MPEG-H, which shall also apply to the Reference Architecture. In addition to the audio codec, the output-interface also covers the usage of MPEG-DASH [DASH], which itself is a flexible standard that needs further profiling and clarification. Here, the ORPHEUS Reference Architecture follows the recommendations of the DASH Industry Forum (DASH-IF), which not only published its Implementation Guidelines but also provides test data for interoperability testing [DASH-IF].

Also in Figure 2 we see how the following functionality is implemented in the macroblock for the distribution path to end-user clients that support MPEG-H 3D Audio: First, the ADM metadata has to be received from the studio-macroblock and interpreted, e.g. to configure the MPEG-H-Encoder with the appropriate number of objects or channels. This requires a conversion of metadata from ADM to MPEG-H. The AES67 stream (or BW64 file) is received and the de-capsulated PCM audio is fed into the MPEG-H-Encoder. In addition, dynamic ADM-metadata (if any) has to be received, converted and fed into the MPEG-H-Encoder. The compressed MPEG-H bit-stream is encapsulated into fMP4 file segments and stored on a DASH-Server who is also responsible for generating the MPD.

Any client supporting MPEG-H and DASH according to the output-interface defined above can receive object-based audio from the distribution-macroblock. Within ORPHEUS two client implementations have been implemented and thus verified feasibility: An iOS-App to be executed on an iPhone and an AV-Receiver for immersive reproduction at best quality.

The distribution-macroblock has the following output-interfaces to the reception-macroblock when streaming to clients supporting MPEG-H 3D Audio:

[MPEG-H]	ISO/IEC 23008-3:2015, 2015, Information technology - High efficiency coding and media delivery in heterogeneous environments - Part 3: 3D audio
[ATSC-3]	A/342 Part 3:2017, MPEG-H System, March 2017, http://atsc.org/wp-content/uploads/2017/03/A342-3-2017-MPEG-H-System-1.pdf
[DASH]	ISO/IEC 23009-1, Information technology - Dynamic adaptive streaming over

HTTP (DASH) - Part 1: Media presentation description and segment format

[DASH-IF] Guidelines for Implementation: DASH-IF Interoperability Points, Version 4.0, DASH Industry Forum, December 12, 2016

For further information about used subsets or specific profiles of the identified interfaces, see Deliverable D4.2.

2.2.6 Reception

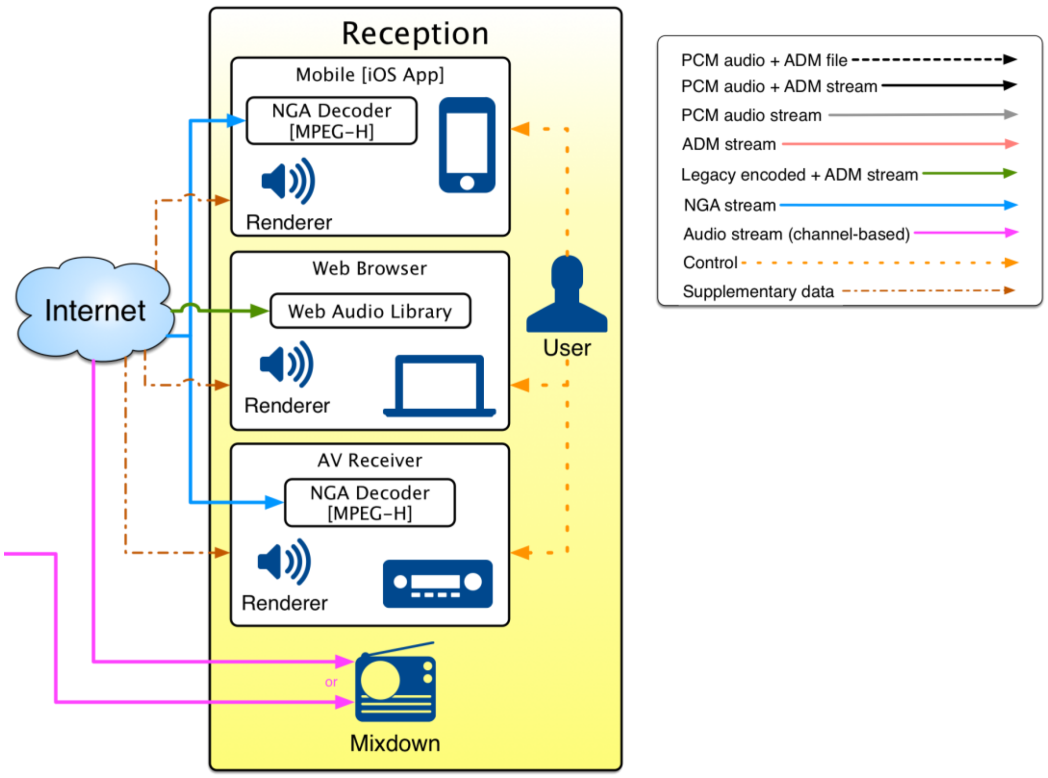


Figure 8: Detailed Reception block structure of the ORPHEUS reference architecture

The purpose of this macroblock is to provide solutions for the reception, personalisation and reproduction of object-based audio content for the end-users. Hardware and software solutions are considered in order to meet different segments of the consumer electronics market, as well as different end-user listening habits and audio content consumption contexts. These solutions differ in terms of rendering capabilities, according to the available network bandwidth, number of audio output channels and processing power. They also differ in terms of user interface and proposed personalisation/interactivity features since they are not addressing the same listening contexts (e.g. domestic use vs mobility) and are equipped with different sensors and input devices (screens and keyboard, touch screens, built-in microphones, GPS sensors etc.).

All solutions are comprised of two major components, a **decoding/rendering module** and a **personalisation module**. The decoding/rendering module receives and decodes the compressed audio streams and adapts the play-out of the content according to their type (channel-, scene- (HOA) and object-based) and to the end-user environment's setup (e.g. binaural rendering over headphones or over conventional as well as ad-hoc multi-channel loudspeaker setups).

The personalisation module provides means for displaying and altering received metadata related either to mixing/spatial features as well as semantic properties. For instance, the user interface provides means for adapting the audio quality to the rendering setup or to the listening context (e.g. 3D immersive audio over headphones or loudspeakers, more conventional play-back, compression level). The user interface displays "content-metadata" such as music title, performer, composer, author, name of panellists as well as live text transcription. When applicable, it provides also means for selecting the preferred language.

To provide the best possible user experience, some of the personalisation features need to go beyond traditional content authoring. A radio drama, for example, can be enriched with a special user interface as it was done with the Pilot 1 Stage A (see Figure 15). Additional images, the chapterisation, transcripts and other tested user features are further examples and require an additional authoring & creation process, a *Presentation Design*, as introduced in this reference architecture in Section 2.2.4.

Optionally, an environmental adaptation is proposed with inputs from environment sensors (head-tracking for binaural rendering, compensation for background noise, rendering setup alignment/equalisation).

The general features of the decoding/rendering module and user interface are described hereafter for each hardware or software solution.

Mobile device

Smartphones and tablets are the most dominant platforms for future media consumption. On mobile devices, it is common to install custom apps instead of using the browser as it allows for the integration of more efficient renderers. An iOS app to be executed on an iPhone is being designed and implemented by elephantcandy to highlight and exploit innovative features of object-based broadcasting, with specific attention paid to the personalisation of rendering and reception, based on user preferences and environmental situations. Mobile devices can be connected to a wide range of types of reproduction hardware, ranging from headphones and internal speakers to (wireless) loudspeakers and multi-channel surround systems. Therefore, a mobile object-based audio app should also support multiple audio output formats, such as mono, stereo, binaural and 0+5+0 surround sound.

The reception macroblock for the mobile phone device is depicted on Figure 8. It receives MPEG-H 3D audio streams over DASH (See 2.2.5). The decoder/renderer of the MPEG-H client implements the Low Complexity Profile Level 3 of MPEG-H standard. In its current implementation, it is limited to a maximum of 16 simultaneously active (=rendered) signals out of a total of up to 32 channels, objects, and/or HOA signals in the bitstream as specified in the standard.

For a typically individual device, personalisation features are important. Object-based audio allows for personalisation on the level audio rendering, with features such as foreground/background balance adjustment to increase intelligibility or clarity of the sound, dynamic range compression to compensate for background noise or customised spatialisation settings.

In addition to user determination of how the audio is rendered, the rich metadata in an object-based audio stream can also give the listener increased control of what is played. Sections can be skipped or relayed through interaction with Points of Interest or textual transcripts. Automatic selection, based on user-selected criteria can be used to achieve variable-length playback, where a radio program is reassembled on the mobile device in a personalised way.

As not all metadata required for a particular app or broadcast may be included in the MPEG-H stream, additional JSON streams can be streamed in parallel, possibly linking to resources outside the object-based audio architecture specified in this document.

Audio hardware device

The hardware receiver designed and built by Trinnov is a CPU-based device. The audio processing is a closed software library running on an embedded computer. Some third-party libraries such as audio decoders may be included as independent modules, linked to the main library and run from the main signal processing function. The hardware receiver is designed for luxury “home cinema” rooms. Thus, the typical reproduction systems that will be connected to it are multi-loudspeaker set-ups. To elicit the sensation of immersion, most common setups have several layers of loudspeakers, including some elevated ones. This is a chance to render in a very accurate way spatialised audio objects that can be static or moving around the 3D space.

The reception macroblock for the audio-video device is depicted on Figure 8. It receives MPEG-H 3D audio streams over DASH (see 2.2.5). As for the mobile device, the decoder/renderer of the MPEG-H client implements the Low Complexity Profile Level 3 of MPEG-H standard. In its current implementation, it is limited to a maximum of 16 simultaneously active (=rendered) signals out of a total of up to 32 channels, objects, and/or HOA signals in the bitstream as specified in the standard.

The main audio playback formats supported by the renderer are binaural over headphones, VBAP or HOA decoding over conventional as well as ad-hoc 2D or 3D loudspeakers layouts and up to a maximum of 32 output channels.

The loudspeaker setups differ most of the time more or less widely from the recommendations in terms of speaker positioning. Thus, the rendering has to deal with improper loudspeaker distributions, including misplaced or even missing loudspeakers, and the content has to be adapted to these configurations. This adaptation is done in 2 steps:

- First, the closest configuration is selected by the user, and the decoding/rendering module provides ad hoc contents for the channels, HOA scenes and/or objects;
- Secondly, the personalisation module, after an acoustic calibration of the system, performs an optimisation to compensate as much as possible the deviations from the reference, starting from the level and delay adjustments for all the drivers.

The personalisation module can also compensate for the acoustic dispersion of the loudspeakers, since they are rarely all similar in real-life installations: people usually favour higher quality loudspeakers for front channels and lower cost transducers for surround or elevation ones. Furthermore, multiway loudspeakers are often used for the front channels whereas they are hardly ever used for other positions. This disparity can cause bad rendering for instance of objects that should be panned between some loudspeakers of different type or quality.

When this environmental adaptation is performed, the content personalisation is also performed, as for the mobile app described above: foreground/background balance adjustment to increase intelligibility, dynamic range compression to compensate for background noise or customised spatialisation settings are offered to the user in the interface.

Web browser

Today, a HTML5 capable web browser can be considered as the most universally available reception system. Its ubiquity means that a very large audience can be exposed to object-based broadcasting without any extra effort required (such as downloading plugins, installing apps, configuring speaker setups etc.). Browsers can be used on devices with a large variety of reproduction hardware, ranging from headphones and internal speakers to (wireless) loudspeakers and multi-channel surround systems. Therefore, the web renderer should also support multiple audio output formats, such as mono, stereo, binaural and 0+5+0 surround sound. At present, web browsers do not readily support object-based audio rendering. The ORPHEUS consortium, however, worked on the implementation of a rendering system based on JavaScript and the Web Audio API. While a native solution will not be available for all browsers immediately, growing support for and maturity of the Web Audio API makes this a promising option.

The reception macroblock for the Web Browser is depicted on Figure 8. It receives AAC audio streams JSON-encoded ADM metadata over DASH (See 2.2.5). The DASH receiver decodes the encapsulated AAC bit-streams using the MSE API. The multiple audio objects are rendered according to transmitted ADM metadata using the WebAudio API.

The Web Audio API[3](WAA), developed within the World Wide Web Consortium (W3C) is a platform dedicated to audio processing within web applications. It provides specifications and description of high-level JavaScript (JS) API for different audio tasks such as mixing, processing and filtering. The WAA provides a number of native audio node objects (e.g. *AudioBufferSourceNode*, *GainNode*, *PannerNode*, *ConvolverNode*, *AudioDestinationNode*…) that can be connected together to form an audio graph (see Figure 9).

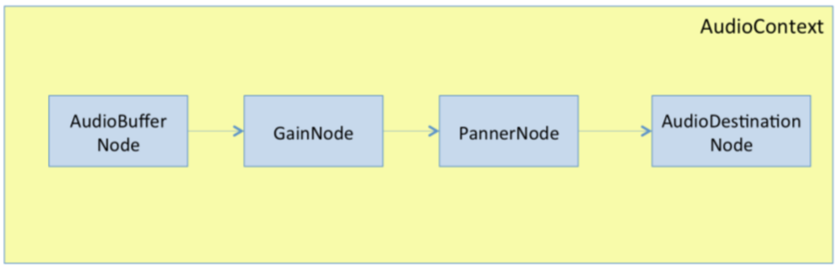


Figure 9: A basic example of an audio processing graph using the native audio node objects provided by the WAA.

The native audio *PannerNode* provides basic panning functions for stereo output as well as for binaural rendering over headphones with pre-encoded HRTFs. However, this binaural implementation presents several limitations. More efficient implementations have been proposed[4] and provide also means for loading personalised HRTFs encoded under the AES69 standard or HRTFs from publicly available database. More complex rendering algorithms over 3D loudspeakers setups may require a specific audio rendering implementation as well. The WAA provides means for writing custom audio effects directly in JS or WebAssembly. For a couple of years, such customised JS code would be handled by the *ScriptProcessorNode*, the major drawback of which was that it would run in the main UI thread, thus lacking efficiency (asynchronous, sensitive to UI interactions…). Progressively, solutions are being developed to process the custom audio script within the audio rendering thread, using the *AudioWorklet* interface[5] (see Figure 10), thus providing better efficiency and accurate synchronisation between audio tasks.

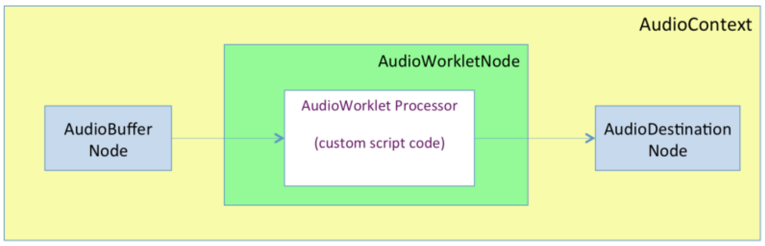


Figure 10: Custom audio processing script code can be run within an AudioWorkletNode

The main audio playback formats currently supported by the renderer are binaural over headphones, and VBAP over conventional 2D or 3D loudspeakers layouts.

Even though browsers on mobile devices nowadays have many interfaces to the sensor APIs such as velocity (e.g. for head-tracking) and GPS, it has to be taken into account that those are not always available on notebooks or desktop computers. Hence, the personalisation features need either to be dependent on the actual end device capabilities, or the user interface and the personalisation

[3] https://webaudio.github.io/web-audio-api/

[4] Carpentier, T., Binaural Synthesis with the Web Audio API, in proceedings of Web Audio Conference, Paris 2015

[5] https://developers.google.com/web/updates/2017/12/audio-worklet

features need to be the lowest common denominator, such as foreground/background balance.

Additional metadata for a particular program user interface, can be streamed in parallel to the AAC+ADM stream as additional JSON streams, possibly linking to resources outside of the object-based audio architecture specified in this document. For the sake of reducing redundant work, these additional JSON streams should be not tied to the specific NGA distribution codec capabilities, but rather independent from them. So it could be used for any kind of object-based distribution (in the ORPHEUS case: AAC+ADM and MPEG-H).

2.3 For Further Study

Even though the ORPHEUS consortium tried to take into account as many real-world requirements and use cases as possible for the design and development of the reference architecture, some areas and scenarios could not, or could only partly, be explored during the project. The reasons for this are varied, and include missing standards, the lack of required infrastructure or limited capacities. However, the following paragraphs contain at least some indication and expertise for those areas.

STORAGE: Archive systems

A decisive component in broadcast workflows is a media asset management or archive system. For the migration to object-based audio it should support in any case ADM metadata and uncompressed audio formats. For audio-only archives, the BW64 container should be a reasonable choice, but depending on the product and workflow, other container formats like RF64[6] would also fulfil these requirements. For audio-with-video archives, several choices are possible including MXF[7], IMF[8] or even BMF[9]. For MXF, however, Recommendation ITU-R BS.2076 contains some information how to integrate ADM metadata. For BMF, the specification is currently being extended to support ADM.

RADIO STUDIO: Scheduling lists for consecutive programmes

Typically, radio programmes are individually composed of a sequence of programme elements. These are planned in advance and played out according to a running order from a special component (hardware or software) in the radio studio. This component should support both file-based ADM and serialised ADM.

Several radio programmes are normally presented, according to a schedule, through the day. The play-out may be triggered by an automation system according to the schedule, or manually from a radio studio. The play-out of these entire programmes, when pre-recorded, requires similar capabilities for file-based ADM and serialised ADM, as well as advanced integrated metadata models (EBUCore) and adjacent descriptions (JSON as described in 2.2.4).

DISTRIBUTION: Alternative formats and protocols

Alternative format options exist in several places, but especially for the distribution of object-based content. For distribution to the end user, there is in particular HTTP Live Streaming (HLS) as an alternative to MPEG DASH. Considering Next Generation Audio (NGA) codecs, besides MPEG-H 3D

[6] https://tech.ebu.ch/docs/tech/tech3306-2009.pdf

[7] http://ieeexplore.ieee.org/document/7292073/

[8] http://ieeexplore.ieee.org/document/7560857/

[9] http://bmf.irt.de/

Audio, there is also the Digital Audio Compression (AC-4) format, and DTS:X. In any case, all of those formats need to provide a suitable interface to ADM. The fact that these are not covered in the ORPHEUS reference architecture in the same level of detail does not mean that they could not be used. However, within the ORPHEUS project, the consortium did not have the resources to investigate alternative formats in detail.

DISTRIBUTION: existing broadcast systems over non-IP platforms

Even though the consumption of content on IP-based platforms is growing extensively, traditional broadcast emission like DAB (Digital Audio Broadcasting), DVB (Digital Video Broadcasting) or ATSC (Advanced Television Systems Committee) need to be taken into account for a holistic object-based workflow. Those emission channels are, and will continue to be, crucial for an unpredictable time period due to both technical and audience-acceptance reasons. At the present time, no activities within DAB were observable that one or more of the NGA technologies or codecs would be integrated in a new revision of the standard. For TV broadcast, however, both ATSC and DVB standards bodies published new revisions of their specification, which contain one or more NGA codecs[10] [11] which will be implemented in TV devices. The broadcast infrastructure needs then to be updated with corresponding encoder and multiplexer devices. For terrestrial UHDTV services in Korea, ATSC 3.0 with MPEG-H 3D Audio as the only NGA codec has been standardized and is being used during the Winter Olympics 2018 in PyeongChang. For this purpose, professional encoders from KaiMedia, PixTree and MediaExcel were developed and TV-Sets from Samsung and LG with support of MPEG-H 3D Audio are available in stores.

DISTRIBUTION: Variable and various bitrates for distribution

To reduce the average bandwidth required for distribution or emission, a variable bitrate is often used for the encoded audio. This feature was not used for the pilots, so no further details could be reported in the reference architecture. Nevertheless, the MPEG-H codec provides this functionality. It is clear that the use of audio objects that appear and then disappear might be a scenario where variable bitrates become important. Moreover, although MPEG-DASH was used, the pilots were encoded and streamed in only one quality: this could be easily done with MPEG-H, or other NGA codecs. The implications of a reducing bitrates for individual audio channels are well known and didn’t require detailed investigations.

DISTRIBUTION: additional content metadata

Because a full and satisfactory object-based media user-experience exceeds the present capabilities of a purely object-based *audio* production and distribution format, additional metadata and matching presentation design is necessary (as described in 2.2.4). The metadata that is required, or desirable, includes basic information about the programme, e.g. the schedule, title, description of the content, titles of elements included (music, artists, authors, contributors), as well as things such as visual elements (pictures) for illustration and navigation. Various approaches and solutions have been developed to enhance and enable legacy digital and analogue broadcast distribution systems. They offer similar user-experience features through integrated or hybrid (IP-connected) technologies.

Within the ORPHEUS project we’ve taken a ‘design-oriented’ approach, by identifying the requirements and features, regardless of what might already exist, in order to create and deliver the

[10] https://www.atsc.org/standards/atsc-3-0-standards/

[11] http://www.etsi.org/deliver/etsi_ts/101100_101199/101154/02.03.01_60/ts_101154v020301p.pdf

optimum for our pilots. Even so, it is likely that some of the basic features applied here could also be implemented with existing broadcast metadata technologies like EPG[12], Dynamic Label Plus[13] or RadioDNS[14]. Further investigation and evaluation is needed in order to identify their possible applicability and other practical solutions for future real-life broadcast implementation.

REPRODUCTION: Object-based loudness

The ORPHEUS partners conducted investigations to define an object-based loudness calculation and normalisation method but those have not been adopted by a standards organisation yet. Hence, no formal recommendation could be made for the reference architecture. It is foreseeable, however, that the ITU-R will publish a Recommendation in the future, which will provide the necessary methodology. A loudness calculation and normalization of rendered loudspeaker feeds according to ITU-R BS.1770-4[15] was used for the pilots, which provided an appropriate workaround.

[12] http://www.etsi.org/deliver/etsi_en/300700_300799/300707/01.02.01_60/en_300707v010201p.pdf

[13] http://www.etsi.org/deliver/etsi_ts/102900_102999/102980/01.01.01_60/ts_102980v010101p.pdf

[14] http://www.etsi.org/deliver/etsi_ts/103200_103299/103270/01.02.01_60/ts_103270v010201p.pdf

[15] https://www.itu.int/rec/R-REC-BS.1770-4-201510-I/en

3 Implementation and integration of Pilots

This section details the implementation and integration activities for the project pilots.

3.1 Pilot Phase 1 Stage A

This section describes the final implementation of the live object-based audio broadcast that formed "Stage A" of our Pilot Phase 1. In this stage, we performed a live broadcast of our radio drama "The Mermaid's Tears". A full description of the drama and set-up can be found in Section 2 of Deliverable 2.3. Below we provide a brief outline of this stage, and its implementation.

3.1.1 Introduction

Stage A was a live broadcast of a specially-written, 9-minute long object-based radio drama. It was produced and transmitted on 6th July 2017 using a fully IP-based workflow. The drama was commissioned especially for the project, and designed to show off some of the functionality that the object-based approach can offer, whilst simultaneously demonstrating the possibility of live production.

The drama was produced in partnership with BBC Radio Drama, who ran the editorial side of the production. The ORPHEUS partners developed the technology for production, distribution and reception. We also worked with several freelancers such as a writer, graphic designer and web developer.

In the drama, there are three main characters – two police officers (Dee and Bill) and a mother (Lesley). The story takes place in the mother's apartment. The police officers are investigating a poisoning, and interviewing the mother about what happened.

The three characters in the story move around the apartment into various rooms at different times. This introduces multiple parallel audio streams where action is happening simultaneously (see Figure 11). The users were given the option to choose which character they follow. Their experience of the story will therefore vary depending on who they follow, which gives a different perspective on what really happened.

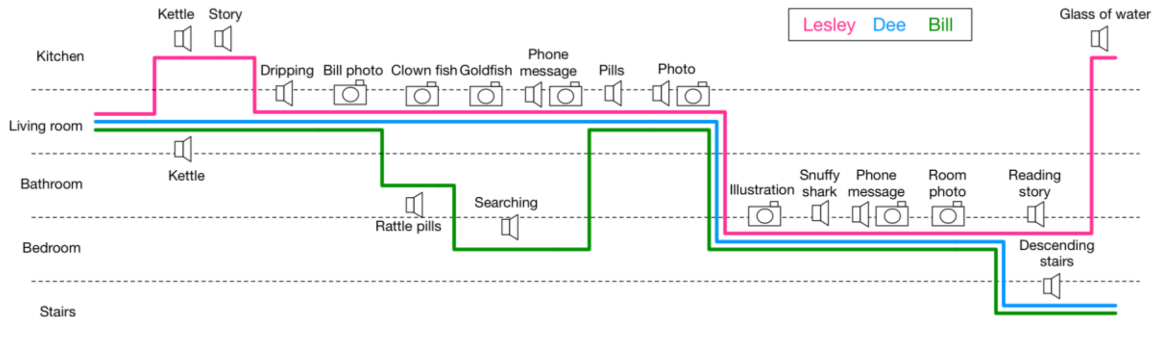


Figure 11: Draft diagram of character movements in the drama. Sound effects are marked with a speaker symbol, and images with a camera symbol.

Photos and images appeared at different points in the drama. These were used to enhance the story by giving the audience a glimpse into the fictional world of the story, and to supplement the audio (e.g. displaying a text message). The images were prepared in advance and triggered in the studio at the right moments.

Figure 12 shows the system diagram of our implementation of Stage A. The production took place in our object-based radio studio in BBC Broadcasting House in London. A range of sound effects were pre-produced using ProTools, then imported into a playout system. We used the BBC's IP Studio platform to route the audio/metadata over IP, play pre-produced audio, record the audio/metadata, encode the audio and create the DASH stream. The drama was distributed by encoding the audio using AAC, and making the audio and metadata available as a DASH stream. The drama could be experienced using a web browser with our custom web interface.

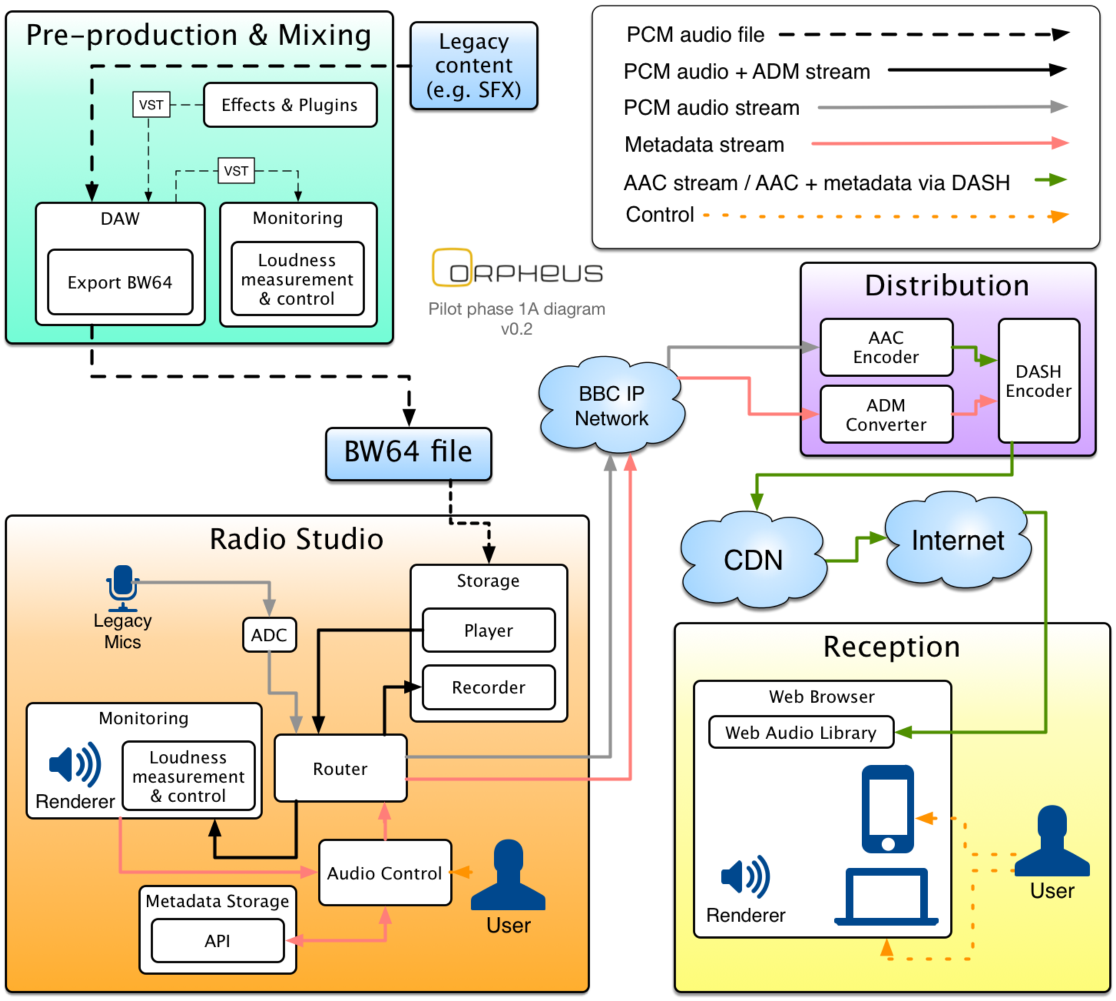


Figure 12: Pilot Phase 1 Stage A implementation architecture

In the rest of this section, we will discuss the audio capture and control, the metadata capture and control, and the distribution and reception.

3.1.2 Audio capture and control

Audio signal acquisition and control was centred on BBC R&D's IP Studio platform, as shown in Figure 13. Put simply, microphone signals and pre-recorded audio clips go into an IP Studio system, which creates two outputs: one is a rendering for the loudspeakers in the studio, and the other is a stream for encoding for delivery to the audience.

The microphones were connected to several Axia xNode units (an off-the-shelf product from the Telos Alliance). These have AES67 interfaces, which were configured to send to the IP network to which the IP Studio PC is connected. The RTP streams sent by the xNodes were received by RTP software modules within IP Studio.

Software modules in IP Studio calculate audio level metering data and send it through a websocket to be received and displayed wherever is convenient (by a separate PC on the network, in the same or a different room, for example). The level meter data was incorporated directly into the production control interface described in Section 3.1.3.

Other software modules render the audio objects to signals for loudspeakers in the studio (through a soundcard in the IP Studio PC). This was configured for a 4+7+0 layout.

The signal flows and processing blocks were set up using a "configurator" GUI in IP Studio. The configuration can be saved and reloaded on demand.

Existing WAV files were imported into a "*sequence store*" in an IP Studio system, which allows the audio to be replayed over the IP network on-demand. Simple cueing and triggering of replay of these files was integrated into the audio control interface. A separate machine was used for this, to ensure that the live pilot did not suffer from problems caused by overloading of the machine doing the acquisition and rendering.

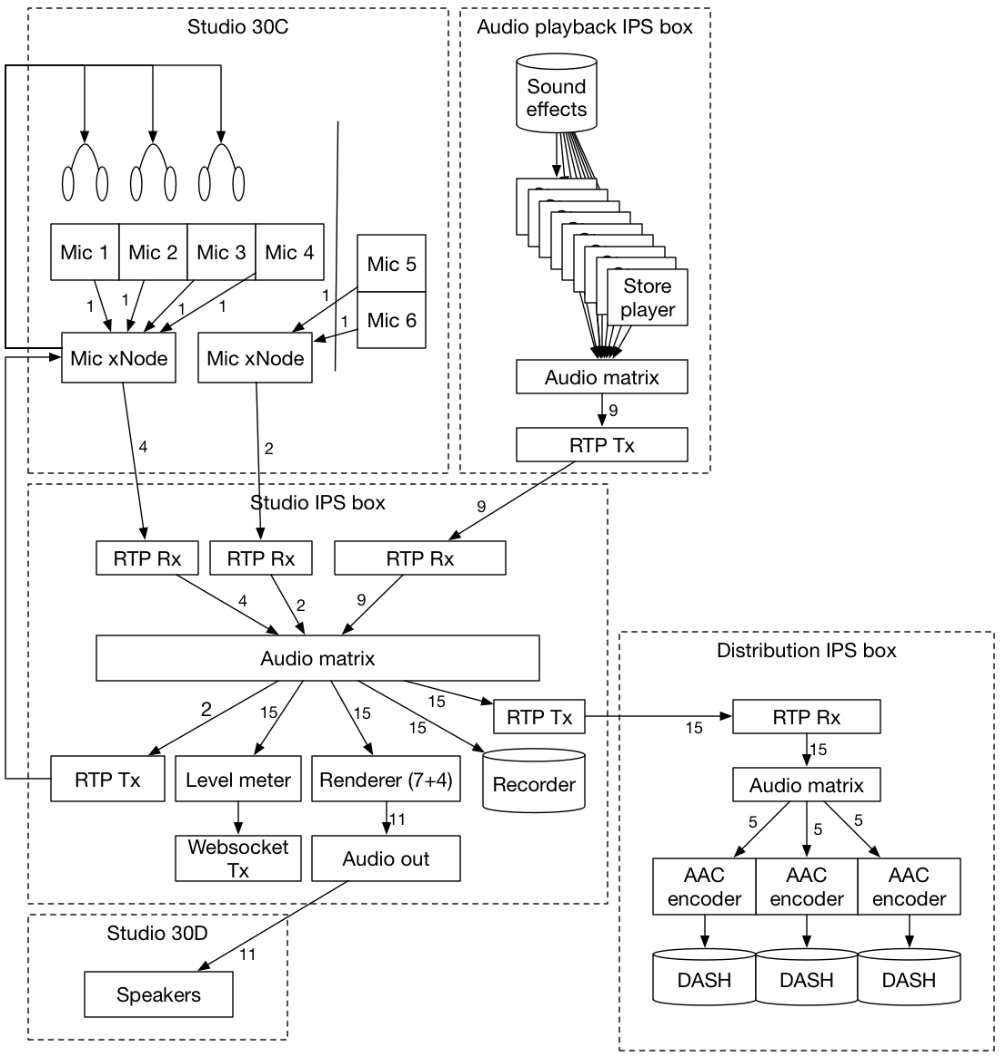


Figure 13: Audio signals and systems in the live production environment

3.1.3 Metadata capture and control

During the live programme, metadata to indicate the positions of audio objects and the gain to be applied to them is created under the control of the studio manager, using a custom-made interface, a screenshot of which is shown in Figure 14. The triggering of images and the replay of pre-recorded audio clips were triggered by the same interface. The audio control interface is fully described in Section 3 of Deliverable 3.6.

The final result of the production control interface was to create a stream of UMCP data. The UMCP data controls the rendering of the audio objects for audition in the studio, and was used in the encoding process for DASH to generate the metadata to accompany the AAC streams.

UMCP allows nesting of compositions, which can be used to create a hierarchical programme structure. The parallel stories of the drama were created by producing a composition for each character, and individual compositions for each room (living room, kitchen, bathroom, bedroom), with the room compositions being a 'child' of each character composition. The coloured faders in Figure 14 show the composition for each character. During the drama, the audio control interface was used to switch the room compositions that fed the character composition.

The pop-up images were controlled through the audio control interface by configuring a 'trigger' element. Pressing a trigger would send a packet of metadata to the user's device, which in this case controlled the appearance of an image on the web browser. The triggers can be seen in purple on the faders in Figure 14.

The setting up of audio and metadata flows, the processing of them, recording, monitoring, and encoding, was all done within IP Studio.

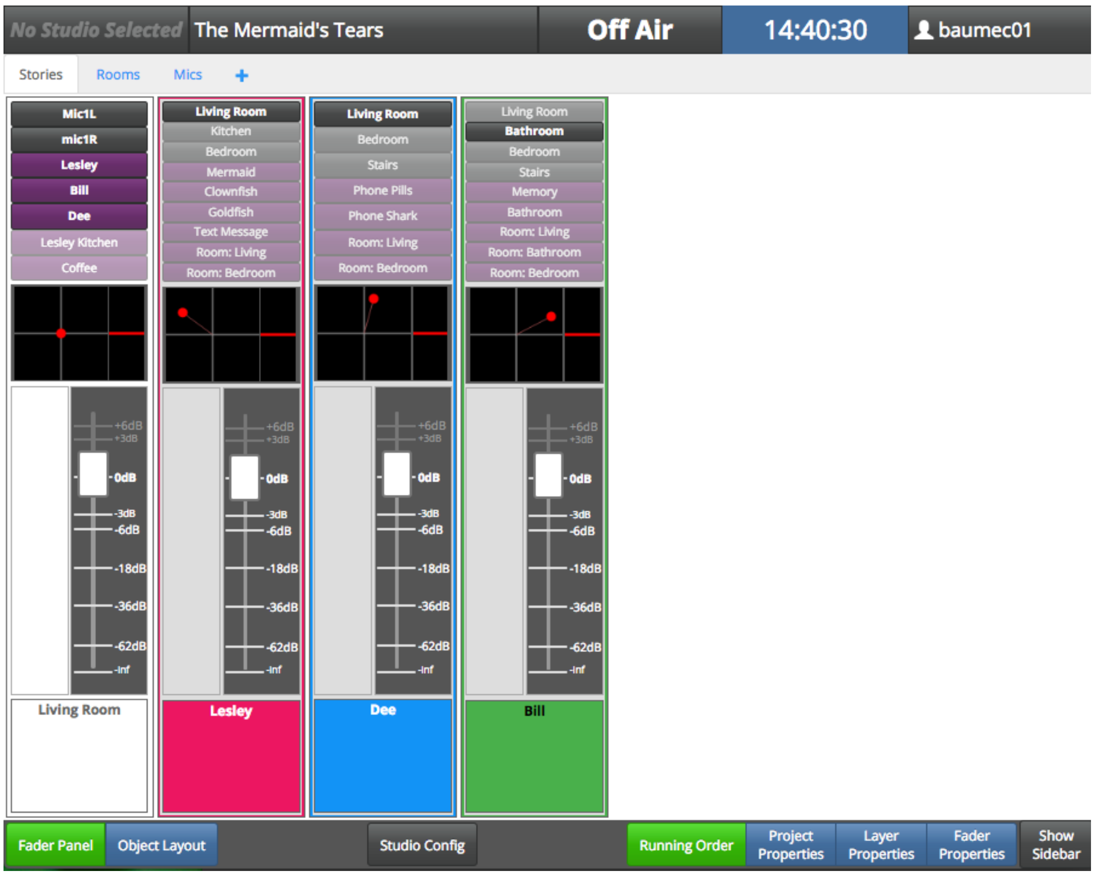


Figure 14: Screenshot of the audio control interface used to generate the metadata stream for the live object-based broadcast.

3.1.4 Distribution and reception

The audio and metadata streams that were produced were encoded (bit rate reduced) for distribution. We achieved this through a combination of IP Studio and Amazon Web Services (AWS).

The AAC encoding was performed by a module in IP Studio that received audio via RTP, and created encoded segments for MPEG-DASH. The UMCP metadata was received via a Websocket and converted into individual JSON files, each representing a 5-second segment of the programme that represented a DASH stream. A script was used to filter the data in these files so that only the public-facing information was included.

We set up a public-facing web server using AWS, to which we uploaded our DASH segments and a DASH manifest from our IP Studio machine, and served them to browsers via a CDN. A process running on our IP Studio machine continuously monitored the sequence store containing the segments, and uploaded the new data as it appeared.

The DASH segments were received and rendered in the browser using a JavaScript library. The library integrated with a front-end we developed that allowed the user to switch between different characters, and displayed the triggered pop-up images. The interface is shown in Figure 15.

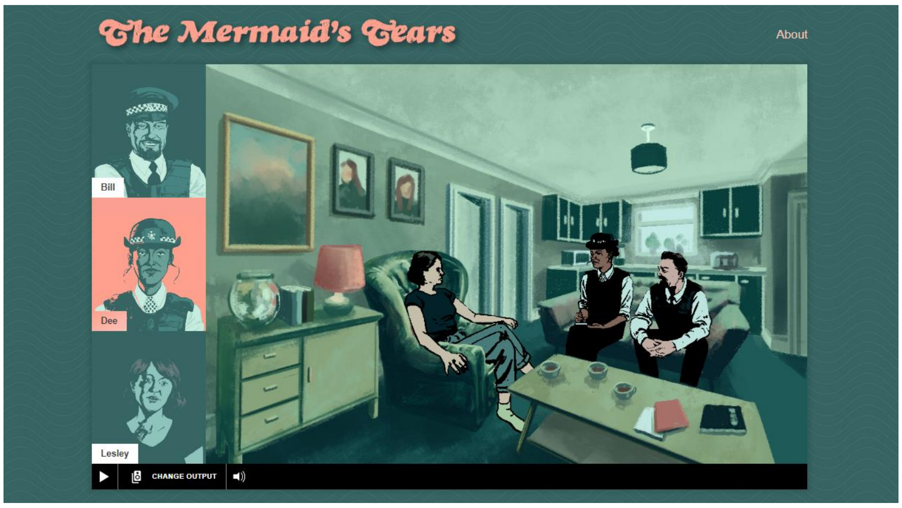


Figure 15: User interface for The Mermaid's Tears radio drama.

The use of UMCP and ADM metadata was still evolving during our pilot production, and so the specification of the transport and payload for the metadata that will accompany the audio may change in future. Initial work on a serialised form of ADM by the ITU-R could enable the development of alternative ways of transporting the metadata from within IP Studio to the web browser.

3.2 Pilot Phase 1 Stage B

Stage B of Pilot 1 is referred to ‘as live’. The goal for this stage of the pilot was to complete the reception part of the MPEG-H based transmission system, such that the full user experience can already be demonstrated. The work focused on the iOS-App by elephantcandy as a client. The term “as-live” refers to the fact that the iOS-App operates exactly as if the DASH-stream would have been operated live. In other words, the IP-interface “looks” exactly like a live-stream to the client. The overall implementation architecture for Stage B of Pilot 1 is illustrated in Figure 16 and explained in more detail below. Besides the **iOS-App**, an **As-Live DASH-Source** was developed to emulate the backend and a major effort was put into the **MPEG-H Test Content Generation**.

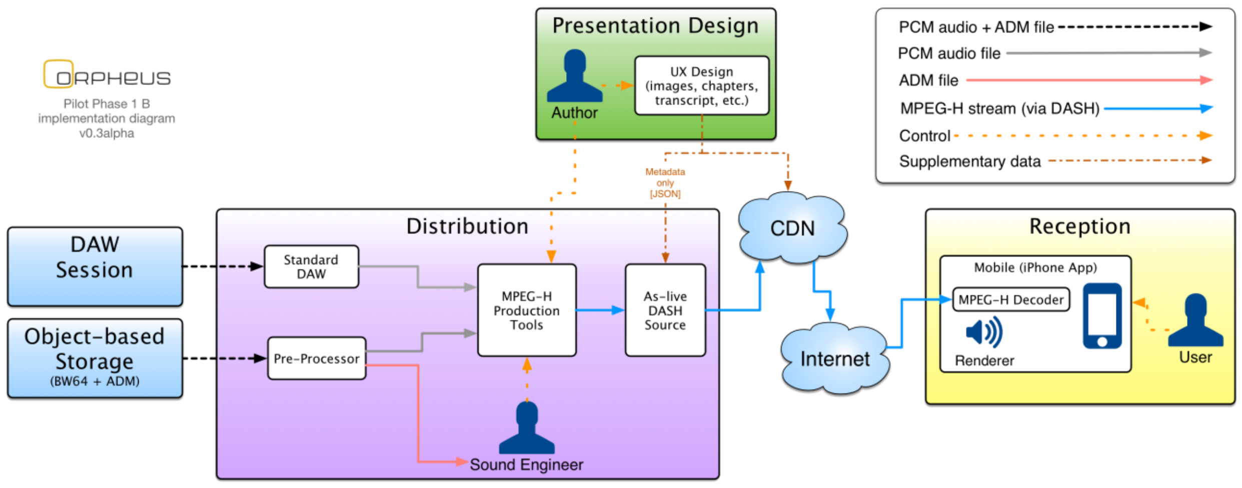


Figure 16: Pilot Phase 1 Stage B implementation architecture

iOS-App

For the iOS-app, elephantcandy integrated the *MPEG-H Decoder* from Fraunhofer and developed a *DASH-Receiver*. For the former, Fraunhofer provided the iOS-Decoder-Lib and a simple example player app for iOS to illustrate the APIs and calling sequence. This approach was particular useful for the User Interactivity API of the MPEG-H Decoder, which allows the user to interact with the audio rendering. Besides those audio-layer components, a major effort for the development of the iOS-app is the GUI design and the generation, transport, and display of additional program-metadata (text, images, etc.). In more detail, the following rendering, interaction and personalisation features were implemented:

Output rendering: All audio content can be rendered to various output formats:

- mono (headphones, internal speaker)
- stereo (headphones, internal speaker, AirPlay)
- binaural (headphones)
- 0+5+0 multichannel surround (Lightning / HDMI)

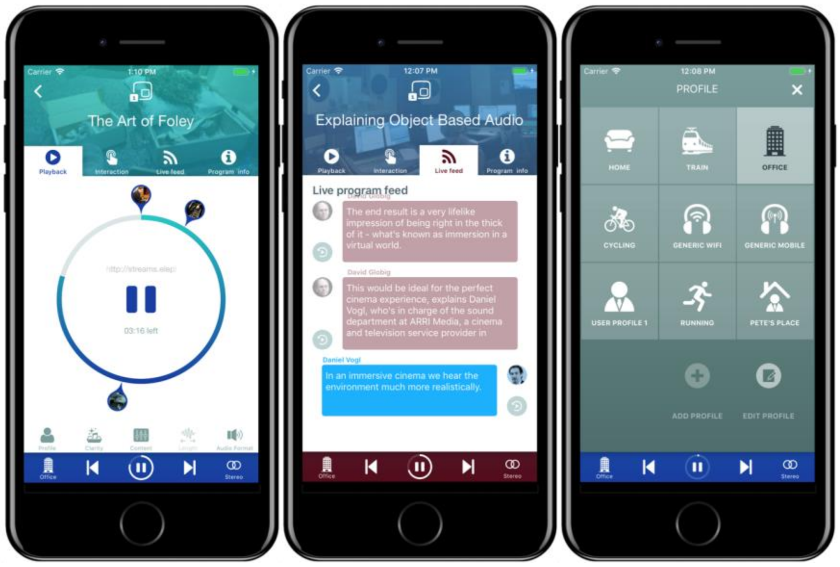
Audio object interaction: In the MPEG-H streams, objects can be marked for interaction. The programs used in Pilot 1 Stage B featured a range of interaction possibilities, such as:

- language selection
- audio object prominence (for instance for foreground/background balance adjustment)
- positioning of an object in 2D or 3D (when using binaural rendering) space
- selection of spatialisation configurations (narrow stage, wide stage, etc.)

Program navigation: Besides the selection of a program, the programs also contained timed metadata such as Point of Interest markers and a live textual transcript. These metadata not only serve as complementary information to the audio, but also allow going backward and forward in time in a more content-related way than traditional scrubbing.

Playback profiles: To facilitate the adoption of the many new options that object-based broadcasting can provide, the concept of *profiles* was introduced. A profile consists of a combination of rendering settings (output format, preferred language, dynamic range control) and environmental conditions (the user’s current location, activity, available bandwidth, environmental noise level etc.). This system allows quick transition from for instance a Home setup (0+5+0 surround rendering, high bandwidth, low environmental noise) to a commuting situation (stereo headphones rendering, lower bandwidth and compensation for the higher environmental noise).

As a result, a very appealing app for the iPhone was available, which enabled a realistic user experience, as can be seen in Figure 17.



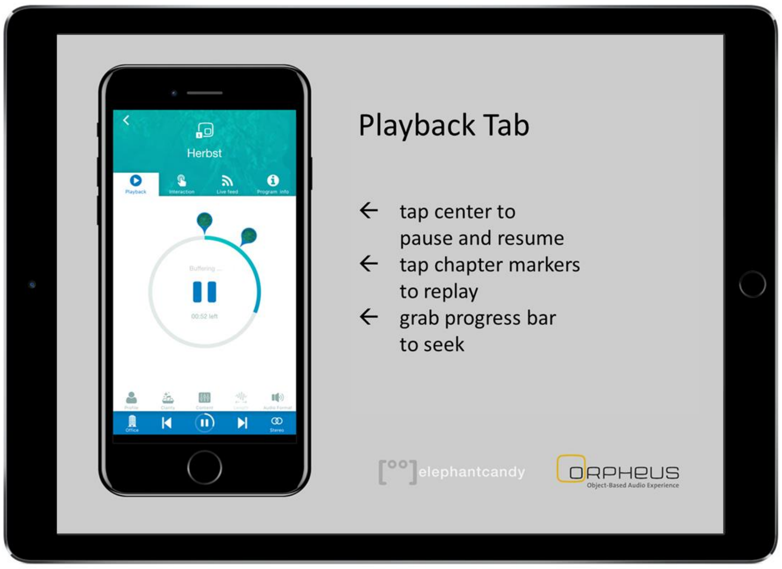
David Globig: The end result is a very lifelike impression of being right in the thick of it - what's known as immersion in a virtual world.

David Globig: This would be ideal for the perfect cinema experience, explains Daniel Vogl, who's in charge of the sound department at ARRI Media, a cinema and television service provider in

Daniel Vogl: In an immersive cinema we hear the environment much more realistically.

Figure 17: Screenshots of iOS App for Stage B

Finally, to allow autonomous exploration of the app and the possibilities of object-based broadcasting, an iPad version of the app was developed for use at trade shows. It consists of an encapsulated version of the iPhone app combined with explanatory text to guide the user, see Figure 18. This setup was used extensively for demonstrations during workshops and trade shows and for user experience testing by BCOM and FHG (the latter also at JOSEPHS[16]).



Playback Tab

- ← tap center to pause and resume
- ← tap chapter markers to replay
- ← grab progress bar to seek

Figure 18: iPad version of iOS-App for autonomous exploitation at trade shows.

As-Live DASH-Source

In order to support the development of the iOS-App (and Trinnov AVR) Fraunhofer provided test

[16] http://www.josephs-service-manufaktur.de/

content and an *As-Live DASH-Source*. The latter allows setting up a DASH-stream easily from a local HTTP server, which then acts identical to a live streaming server. The As-Live DASH-Source makes use of DASH-segments (fragmented MP4 files, fMP4) and a Media Presentation Description (MPD), which can be encoded offline. In addition, it included a PHP-Script, which modifies the MPD dynamically and controlled the availability of segments. In summary, this allowed to stream offline encoded content as if it would be live encoded, i.e. make the client believe it decodes a live stream.

MPEG-H Test Content Generation

Besides software development, a major effort in Stage B was the generation of MPEG-H test content. As an automatic import of ADM to MPEG-H was not yet available and tools for producing and encoding MPEG-H are still under development, a lot of manual pre-processing and authoring was required. The Bayrischer Rundfunk (BR) was particular active in providing content and OBA productions. This content was then often provided as DAW-sessions and re-imported into MPEG-H Production Tools by Fraunhofer. Some content, such as e.g. “The Art of Foley”, had to be pre-rendered by IRT from many objects to a 4+7+0 channel bed before import to MPEG-H. As a result, 6 items with various levels of immersion and interactivity features (e.g. language selection, object movement) were available for the IBC 2017 trade-show, at which the iOS-App was demonstrated by several partners. Table 1 provides a summary of the six items and their features as demonstrated at IBC. Two more items have been added at a later stage (including a live-stream in stage C) as illustrated in Figure 19, which shows 4 screenshots from the start screen of the iOS-App from which the items are selected.

Figure 19 also illustrates the need for additional metadata, such as program title, images, and live transcripts (the latter is visible in the middle screenshot of Figure 17). In order to describe and transport this metadata, a lightweight JSON-based content format was developed by elephantcandy that encapsulates program metadata, interaction setup and links to the MPEG-DASH streams. This JSON-file was served together with the MPEG-H Test Content.

#	Content	Input PCM			Metadata							Measured Bitrate
		Channels (last = ControlTrack)	Duration	Duration in s	Channel Bed	Objects transmitted	Objects reproduced at a time	Layout CICP	Groups	Presets	Interactivity Features	kbit/s
1	Explaining Object Based Audio	15	00:13:02	782	7.1+4 (12 Ch)	2	1	19	0=Channel Bed; AllowOnOff; 12ch 1=Englisch, AllowGain -8 .. +8, 1ch 2=German, AllowGain -8 .. +8, 1ch	Broadcast, Dialog+	-Prominence -Fore-/background balance -Language selection	516
2	German Forest	16	00:01:15	75	5.1 (6 Ch)	3	2	6	0 , Ambience, AllowOnOff, 6ch 10, German, AllowOnOff, AllowGain -20...12, 1ch 11, Englisch, AllowOnOff, AllowGain, -20...12, 1ch 12, Humming Bee, AllowOnOff, AllowGain -20...12, AllowPositionAz -49.5...49.5, AllowPositionEl 0...48.0, DistFactor 1...1, 1ch (other channels empty)	Default, Dialog+, Ambience	- Prominence, azimuth and elevation - Fore-/background balance - Language selection	414
3	Mermaid Tears	7	00:08:34	514	2.0 (2 Ch)	3x2	2	2	0, Lesley, AllowOnOff, 2ch (w. ambience) 1, Dee, AllowOnOff, 2ch (w. ambience) 2, Bill, AllowOnOff, 2ch (w. ambience)	Switch Characters	-Switch characters -Fore-/background balance	281
4	Soccer Match	16	00:06:24	384	5.1 (6 Ch)	1	1	6	0, , AllowOnOff, 6ch 10, Kommentator, AllowGain 0..+10, 1ch (other channels empty)	Default	-Mute commentator -Fore-/background balance	273
5	The Art Of Foley	25	00:15:58	958	7.1+4 (12 Ch)	3x4	1x4	19	0, ChannelBed, AllowOnOff, 12ch, 1, English, AllowGain -8...8, 4ch 2, German, AllowGain -8...8, 4ch, 3, French, AllowGain -8...8, 4ch	Dialog, Broadcast, Dialog+	-Fore-/background balance -Language selection	1219
6	Turning Forest	15	00:04:39	279	7.1+4 (12 Ch)	2	1	19	0, Bed, AllowOnOff, 12ch 1, Narrator en, AllowGain -12...+12, 1ch 2, Narrator ger, AllowGain -12...+12, 1ch	Default Preset, Dialog+	-Fore-/background balance -Language selection	516

Table 1: Summary of Content-Items produced in MPEG-H for Pilot-1 Stage-B

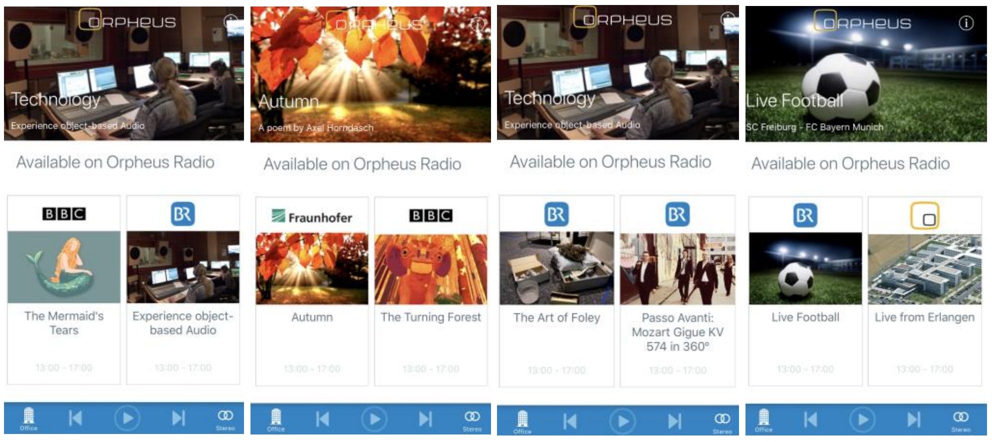


Figure 19: Screenshots from the iOS-App showing the items that can be selected from the start screen

3.3 Pilot Phase 1 Stage C

The focus of Pilot 1 Stage C was on MPEG-H based live transmission and was therefore referred to as “MPEG-H Live”. The main differences between the previously completed Stage A (see Section 3.1) was that MPEG-H was used as an emission codec instead of AAC with ADM metadata. This also meant that the two MPEG-H based clients were used as reception devices, i.e. the iOS-App (elephantcandy) and the AV Receiver (Trinnov). With reference to Stage B (see Section 3.2), the main difference was that the production and encoding was operated in real time, i.e. “live”. Hence, Stage C combined the feature sets of the previouly achieved stages and aimed at the most complete integration scenario of Pilot Phase 1.

The overall implementation architecture of Pliot 1 Stage C is illustrated in Figure 20 and explained in more detail below. Before describing the integration work in the individual macroblocks, we first describe the live production which was used to demonstrate and verify the implementation work of Stage C.

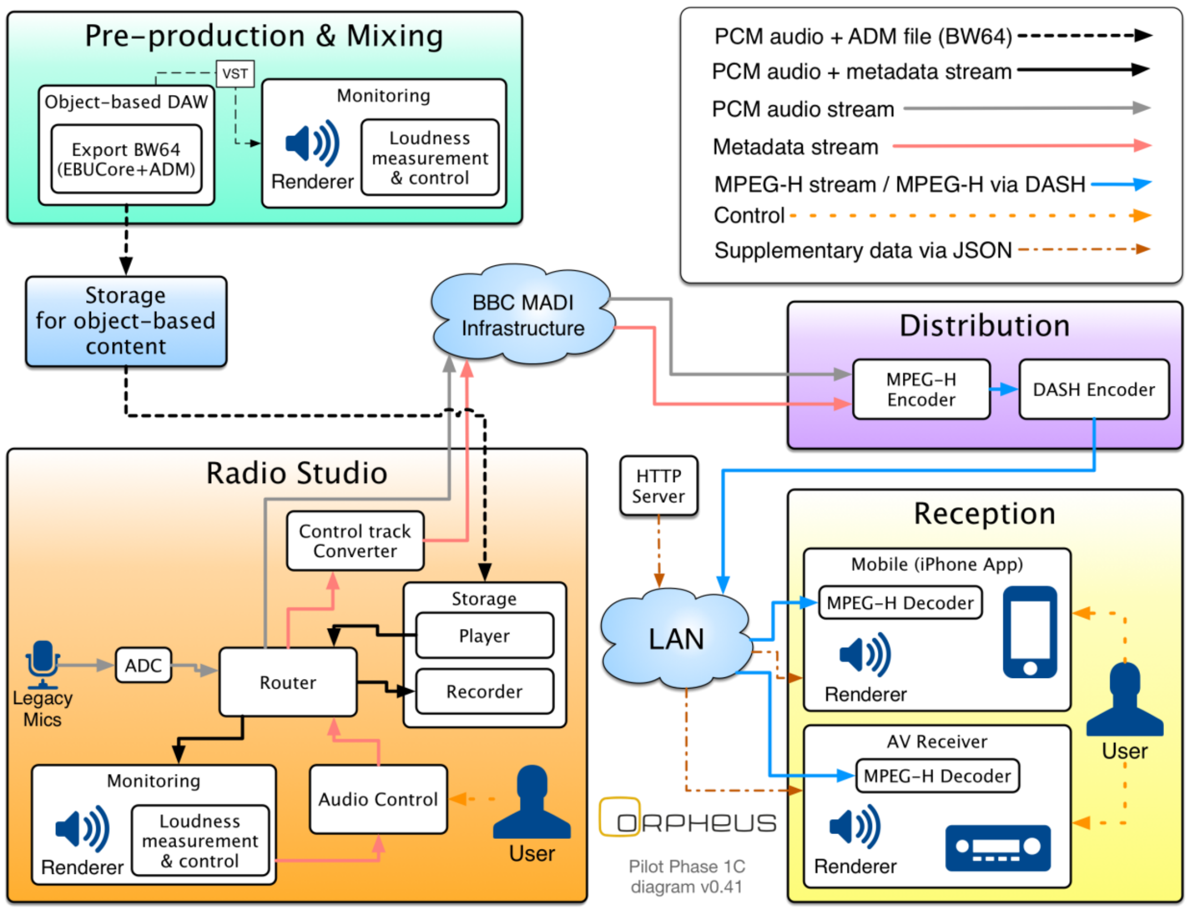


Figure 20: Pilot Phase 1 Stage C implementation architecture

3.3.1 Live Production Scenario

In order to demonstrate the scope of Pilot 1 Stage C, the following live event was produced. The scenario was inspired by a radio show of BBC Radio 4 called “Desert Island Discs” in which the presenter (Kirsty Young) interviews a prominent person and invites her guest to share the soundtrack of their lives by asking, “Eight tracks, a book and a luxury: what would you take to a desert island?”. More specifically, the show in which the English singer and song writer Steven Patrick Morrissey participated was used as a template.

To reproduce this scenario, two staff members read the transcript of the show, one acting as Kirsty and one as Morrissey. Their voice was recorded live in the studio and ingested as two audio objects in the IP Studio system. The position and gain of the readers’ voice could be controlled live by the sound engineer during the show. In addition, the concept of the show allowed alternation between the spoken word and the selected music items. This was used to replay archived material and an immersive recording of a classical concert was used for the pilot. In terms of the MPEG-H system, this setup corresponded to a 4+7+0 immersive channel setup (7.1 surround plus 4 height speakers) with two dynamic audio objects.

Though the demonstration scenario follows a relatively simple structure, it combined channel-based and object-based audio in a realistic live production scenario. Hence, it was sufficient as a proof of concept. The architecture implemented, however, could have supported more complex scenarios as well.

The live production was streamed using MPEG-H during the review meeting on Dec. 14 2017 with a representative of the European Commission and independent assessors.

3.3.2 Integration in IP Studio

In Stage C, the renderer which already existed in IP Studio (and used in Stage A) was exchanged with the renderer used for MPEG-H production. In general, a renderer is needed in the studio for monitoring, i.e. to know that the audio that is being created will sound as it should when it reaches the audience. As the distribution in Stage C was done with MPEG-H it was more appropriate to also use the corresponding renderer, such that the monitoring signal provides a better prediction.

An annotated screen-shot of the main signal pipeline in IP Studio is shown in Figure 21. The pipeline was created using a tool – the “configurator” - within IP Studio that enables the construction of flows of signals or data between processors. The yellow highlighting and text labels have been added by hand to the screen-shot.

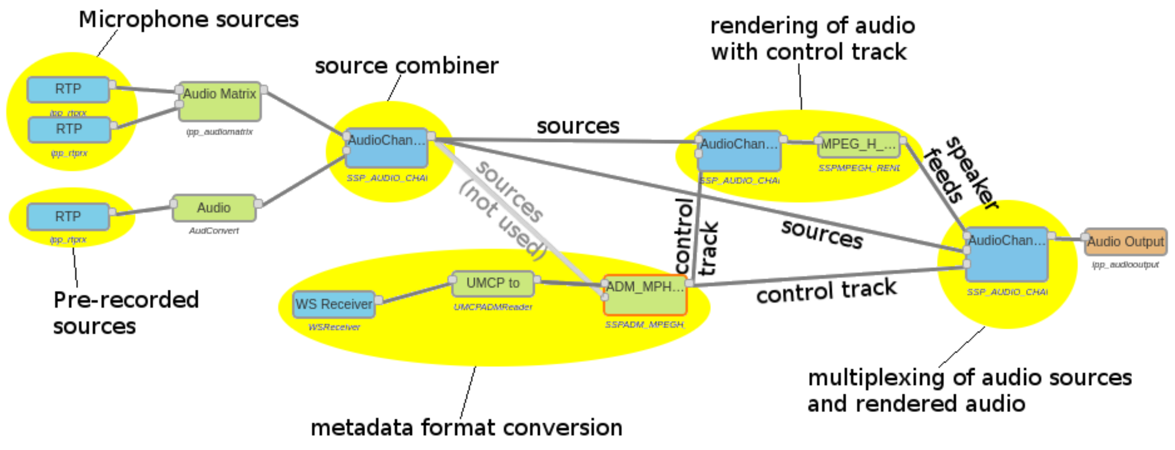


Figure 21: Main pipeline from IP Studio used for pilot 1, stage C

On the left are the RTP receivers for audio sources. Microphone sources are carried as AES67 streams from Axia xNodes, the pre-recorded sources are carried as RTP streams entirely within IP Studio. These multiple flows are multiplexed into one flow for ease of handling, by a source combiner. The pipelines that provide the signals are shown in Figure 22.

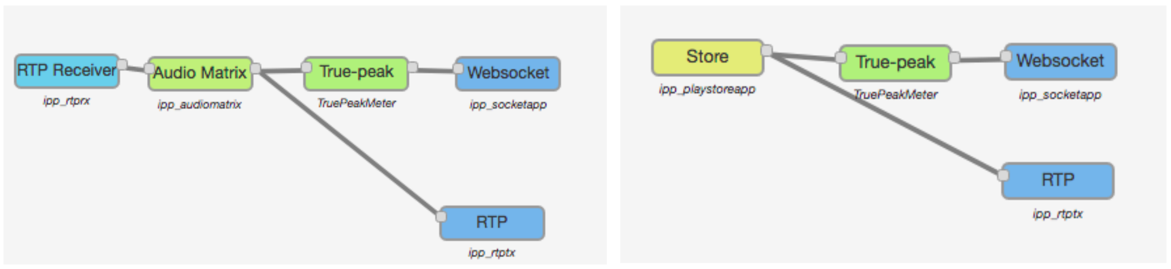


Figure 22: Pipeline showing RTP receiver for incoming signals from microphones using AES67, with metering (left) and player of pre-recorded clips, with metering (right).

Metadata, carried using UMCP and received from the live production interface over web sockets, is converted into MPEG-H style and the MPEG-H production library from FhG is used to create a “control track” (a PCM signal, treated as audio, that carries the metadata encoded within it, see also Section 3.3.3). The production library is invoked by the component labelled “ADM_MPH...”in the figure.

The rendering of the audio, with the control track, is implemented in an IP Studio component shown in the upper right of the figure, labelled “MPEG_H_...”. This component receives a flow of audio sources and control track, and invokes the MPEG-H renderer library from FhG, passing it blocks of audio and control track. The library functions render the audio and return blocks of audio for loudspeakers.

In the implementation for pilot 1 stage C, the connection to the loudspeakers in the studio was through the same 64-channel MADI interface as used for the connection to the MPEG-H live encoder. A final multiplexing process (on the right of the figure) combines the rendered audio with the original sources and the control track for output over MADI.

Because the structure of the audio scene was relatively simple in the Stage C pilot, it was not necessary to implement a fully dynamic meta-data handling system capable of dealing with any possible object hierarchy or level of interactivity. Instead, knowledge of the nature of production was used to initialise the renderer component when the pipeline was started. More specifically, the used audio scene consisted of an immersive 4+7+0 channel bed with two dynamic audio objects, which was maintained throughout the live transmission without any configuration changes. Note that the readers were transmitted as audio objects with dynamic metadata (position, gain) generated in the studio. Though interactivity by end users would have been possible with the implementation architecture, it was not used. However, the audio scene that was used did allow flexible rendering with the two receiver devices. While the AV Receiver from Trinnov reproduced the stream to the available 4+7+0 speaker setup, the iOS-App by elephantcandy used the binaural rendering mode of MPEG-H to produce an immersive audio experience on headphones. In both cases, dynamic objects were rendered to the appropriate playout situation (speaker/binaural).

3.3.3 MPEG-H DASH Live Encoder

For Pilot 1 Stage C, it was necessary to integrate MPEG-H Live Encoding into IP Studio. The integration work was split between FhG and BBC by following a modular and pragmatic design, in which the MPEG-H encoding is performed on a dedicated hardware machine with clear input- and output interfaces. This approach allowed better control of real-time operation, more rigid testing before delivery, and easier on-site integration (hardware instead of software). This approach also follows traditional deployments in broadcast studios in which professional, rack-mounted hardware devices are used for encoding. Therefore, the selected architecture also allowed testing of how object-based audio can be integrated into legacy broadcast infrastructure, which are not IP- and software-based.

The integration architecture is illustrated in Figure 20 and is explained as follows. Fraunhofer provided a dedicated Windows PC with an external MADI card for live audio input (RME MADIface, 128-Channel MADI USB interface). The PC runs the *MPEG-H DASH Live-Encoder*, which implements multi-channel audio input, MPEG-H encoding and the generation of DASH content, consisting of MP4 fragments (fmp4) and the Media Presentation Description (mpd). The DASH content is then served from a local HTTP server, which connects to the Local Area Network (LAN) at BBC. The clients (elephantcandy iOS-App, Trinnov AVR) accessed the content via a Wireless LAN access point.

The input of the MPEG-H DASH Live-Encoder is provided from the IP-Studio and includes 16 channels of uncompressed PCM audio, transmitted over MADI. One critical point is the transmission of audio-related metadata. For that purpose, Fraunhofer has developed the *MPEG-H Control Track* (not within ORPHEUS), which allows the transport of MPEG-H metadata over a PCM channel using a data modem. This Control Track is carried over PCM channel #16. In order to generate the Control Track, BBC has integrated the *MPEG-H Production Library*, which includes the *Control Track Writer*. The H-Production-Lib needs to be initialized correctly with a description of the audio scene configuration (static metadata) and also has runtime API to change the position and gain of audio objects (dynamic metadata). Those APIs were used by the IP-Studio software which offered a similar audio control interface as illustrated in Figure 14 for controlling the position of audio objects by the sound engineer. The important point of the Control Track is that object-based audio can be transported over legacy broadcast infrastructure, such as MADI or SDI. It therefore allows an easier integration of OBA into existing workflows and studio infrastructure.

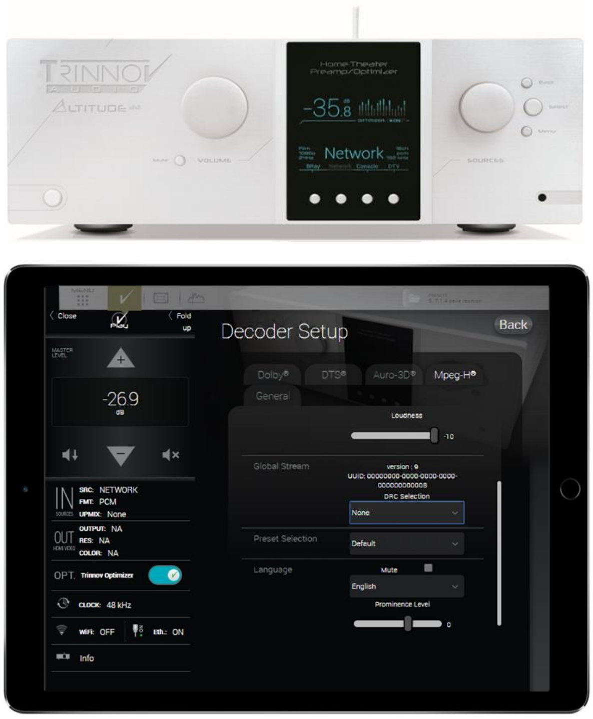
The MPEG-H Control Track is described in more detail in [8].

3.3.4 MPEG-H based Reception Devices

As can be seen in Figure 20, two reception devices for the MPEG-H stream were used in Stage C: The iOS-App developed by elephantcandy and the AV Receiver developed by Trinnov. For the iOS-App, only minor changes had to be implemented compared to Stage B and we therefore refer to Section 3.2 for the description.

The integration of MPEG-H decoding and DASH reception into the AV Receiver by Trinnov was successfully completed for the Stage C pilot. After refactoring and optimizing the software architecture of the AVR, it was possible to run the MPEG-H decoder on the processor. Comprehensive tests have been carried out with the test bit streams provided by FhG to ensure correct behaviour. A modified version of the free GStreamer media framework was used by Trinnov and FhG provided a DASH-Receiver as a GStreamer-Source-Plugin as the basis for DASH reception. The result was a fluent reception and decoding of the streamed content, and a correct rendering of it over any selected loudspeaker configuration among the targeted ones.

An automatic GUI is constructed when the streaming starts, to allow adjusting stream specific parameters in real time. For example, this allows switching the language between dialogue objects, changing the volume and/or the prominence of the audio objects (the dialogue vs. ambience), and changing the position of an audio object within a range authorized by the sound designer. The GUI is using web technology and can be accessed (after authorization) from any browser that connects to the WiFi-Access Point of the AVR. Figure 23 shows the Trinnov-AVR Altitude32 next to the automatic GUI running on a tablet.



Loudness: -10
Global Stream: version : 9
UUID: 00000000-0000-0000-0000-00000000000B
DRC Selection: None
Preset Selection: Default
Language: English
Mute
Prominence Level: 0
SRC: NETWORK
FMT: PCM
UPMIX: None
OUTPUT: NA
RES: NA
COLOR: NA
OPT. Trinnov Optimizer
CLOCK: 48 kHz
WiFi: OFF
Eth.: ON
Info

Figure 23: Trinnov-AVR Altitude32 with Web-GUI running on a tablet

3.4 Pilot Phase 2

Pilot 2 focusses on on-demand delivery and non-linear playback. As such, all content is pre-produced offline. New metadata in the form of timed markers is added to the audio-objects in the Pre-production stage and converted to distribution formats in the Distribution stage. The receiving clients can then provide means to control the selection and ordering of segments to play based on this metadata.

To produce the markers, MAGIX added new functionality to the Sequoia DAW that writes timed markers in an EBUCore chunk of the BW64 files. The markers can contain information about the relative importance of a segment, keywords, topics etc.

When the ADM+BW64 files are converted to MPEG-H for distribution, these new markers are separated out from the BW64 and processed by an EBUCore to JSON converted developed by elephantcandy. The resulting JSON stream is then paired with the MPEG-H stream into an MPEG-DASH mpd file that is made available on the CDN.

On the receiving side, the timed markers can be used in various ways to decide which segments of audio to download and play back. The iOS app can skip sections of a program that do not match the listener's interests (comparing the topics and keywords in the metadata to the user's profile) or in a variable-length scenario by evaluating the markers' level-of-importance values for a user's preferred content 'depth' or the time available for listening.

It is important to note that the use of specialised user interfaces for detailed non-linear playback, which is required to take full control of the on-demand rendering, is not mandatory: the Trinnov AV receiver does not feature such a user interface that is fully compatible with the content produced in Pilot 2.

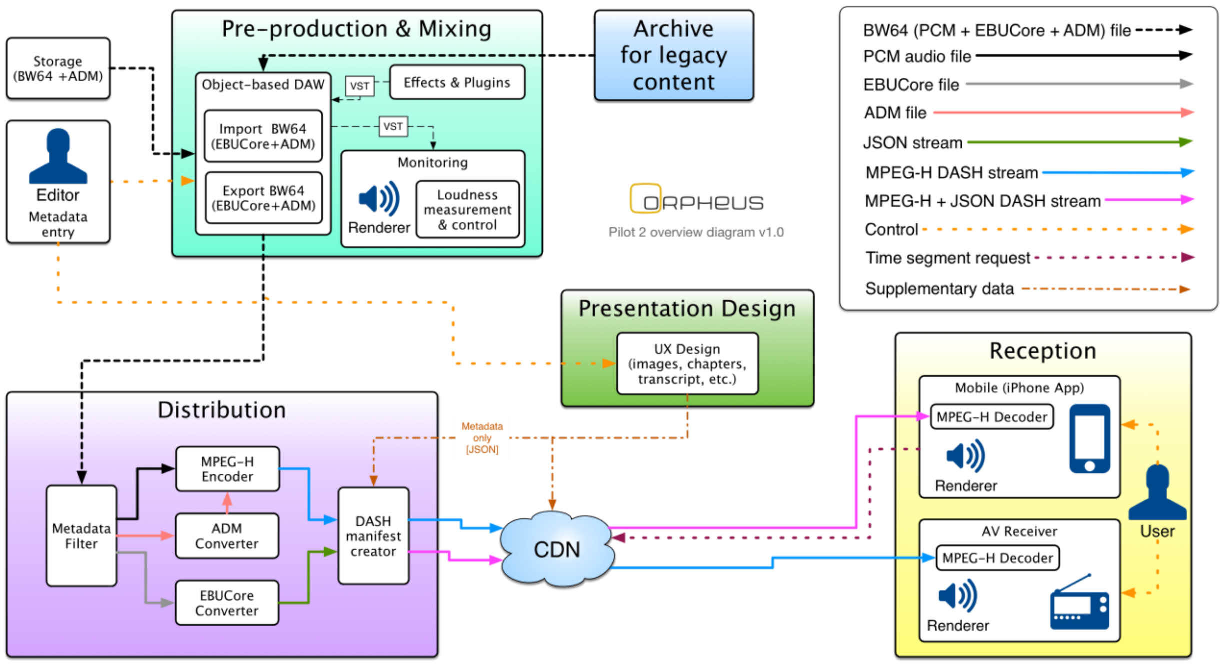


Figure 24: Pilot Phase 2 implementation architecture

3.5 Presentation Design

For the pilots in ORPHEUS, three user interfaces were developed: a 3D interactive spatialisation component to position audio objects in the Trinnov receiver, a mobile app allowing personalized rendering and content interaction by elephantcandy and a website providing visuals and character selection to accompany The Mermaid's Tears by the BBC.

In the ORPHEUS Reference Architecture, the Presentation Design block can feed streams of (meta) data into the distributed MPEG-DASH streams, or make data available on the Internet / CDN to be integrated by the Reception devices. JSON was selected as container format for the high flexibility in the types of information it can carry and the widespread software support.

It is important to remember that specifically designed presentations may not provide an identical experience on various reception clients. For example, the interaction on website for *The Mermaid's Tears* was designed and implemented explicitly for modern web browsers and therefore could not be represented identically in the iOS app or the Trinnov receiver. Likewise, the interaction and presentation in the app cannot easily be replicated in a browser.

To make optimal use of the rich interaction and user experience enabled by object-based broadcasting, it is recommended to consider Presentation Design at an early point in the editorial process so relevant metadata can be collected and attached to the audio objects as soon as possible; ideally already during the capture phase. Also, some of the key functionalities of this Presentation Design process should ideally be considered for integration in future DAW applications.

3.6 Trials

The sections above detailed the architecture that was designed and implemented for the different stages of the ORPHEUS pilots. Note that, in addition to those, the ORPHEUS partners envisioned other scenarios that were not retained in the pilots for practical reasons.

One outstanding example of these “trial” architectures concerns the recording macroblock. For Pilot 1, the envisioned scenario was that the producers of the radio drama would mix mono or stereo objects, recorded using traditional proximity microphones, with a channel-based or scene-based bed recorded with a microphone array. In this scenario, the objects would have consisted of the sources of interest in the scene (the characters) while the multi-channel bed would have constituted the ambient part of the scene. For the mono and stereo sound objects, the recorded signals would have been sent directly from the sound card used to digitize the signals to the DAW, using the sound card driver (typically an ASIO driver). For the multichannel bed, however, signals recorded using the microphone array would have been processed to produce scene-based or channel-based content. This would have been done using the plugins developed by FhG and b<>com during the course of the ORPHEUS project, which interface with the DAW via the **VST** standard.

Below is a list of the VST plugins that were developed over the course of the ORPHEUS project and tested during such trials:

- b<>com’s MicProcessor, a plugin that converts signals recorded by microphone arrays to HOA signals.
- b<>com’s Render Hoa2Spk, which renders HOA signals to speaker signals. This plugin would be used to produce a channel-based bed.
- b<>com’s MainSpot, a plugin that estimates the position (direction and distance) of a proximity (spot) microphone relative to a microphone array (main microphone). This is to help producing object metadata.
- b<>com’s HoaScope, a monitoring plugin which analyses recorded HOA signals and displays a map of the acoustic energy as a function of the direction of arrival.
- FhG’s 3D audio capturing plugin, which processes signals recorded by FhG’s compact circular microphone array to speaker signals (channel-based format). This plugin would be used to produce a channel-based bed.

4 Conclusions

Broadcasting architectures and infrastructures have been developed, refined and optimised over the last 100 years. Up to now, the final product delivered to the audience, was assembled edited, mixed and configured as an invulnerable entity, leaving little or no options for adjustments according to the reception conditions and the personal preferences of the recipient.

The new object-based media approach fundamentally changes this principle: A production is now a collection of media assets, so called ‘objects’, along with parameters or ‘metadata’, describing their attributes and relationship. This package is delivered to an end user device, where it gets optimally ‘rendered’ according to the capability of the device, environmental conditions and user preferences.

It seems obvious that this alternative concept is to cause profound changes for existing broadcast system architectures and infrastructures, as well as their related workflows. Yet, it was not the task for the ORPHEUS project to create such new system from scratch.

Therefore, in order to create a universal Reference Architecture for a realistic end-to-end object-based audio broadcast chain, the ORPHEUS consortium took the well-established components and efficient workflows of the existing broadcasting eco-system as initial basis.

By this, we have incorporated a plethora of requirements from real-world broadcast use cases and via multiple pilots. These have been integrated and implemented successfully into our new object-based approach. As demonstrated in these pilots, several different scenarios have been realised, enhancing traditional radio broadcast formats with new object-based audio features and thus adding exiting new user experiences.

The outcome of these pilots was the basis for the ‘practicable’ Reference Architecture, designed and developed in several iterations, based on lessons learned from the various implementations during the project.

So the Reference Architecture presented here can be taken as genuine template for broadcasters and media creators for starting their transition process into object-based audio production and distribution workflows.

References

[1] Scheirer E., Väänänen R. 1999. AudioBIFS: Describing Audio Scenes with the MPEG-4 Multimedia Standard, in IEEE Transactions On Multimedia 1, Nr. 3

[2] Geier M., Spors, S. 2008. ASDF: Audio Scene Description Format. International Computer Music Conference (ICMC), Belfast, UK

[3] Peters N. et al. 2012. SpatDIF: Principles, Specification, and Examples. Proceedings of SMC 2012

[4] Peter Brightwell, Jonathan Rosser, Robert Wadge, Phil Tudor, "The IP Studio", BBC R&D White paper WHP 268, Sep 2013

[5] http://www.bbc.co.uk/rd/projects/ip-studio

[6] ITU-R Recommendation BS.2076, “Audio Definition Model”, https://www.itu.int/rec/R-REC-BS.2076/en

[7] ITU-R Recommendation BS.2094, “Common Definitions for the Audio Definition Model”, https://www.itu.int/rec/R-REC-BS.2094/en

[8] Bleidt, R.L., D. Sen, A. Niedermeier, et al., in “Development of the MPEG-H TV Audio System for ATSC 3.0.” IEEE Transactions on Broadcasting, 2017. 63(1): p.202..236, DOI: https://doi.org/10.1109/TBC.2017.2661258.

[end of document]