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

ORPHEUS
Object-Based Audio Experience

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

D5.5: Implementation and documentation of final object-based renderers

Version: v1.0

Deliverable type	D (Demonstrator)
Dissemination level	PU (Public)
Due date	28/02/2018
Submission date	08/03/2018
Lead editor	Benjamin Duval (Trinnov)
Authors	Andrew Mason (BBC), Chris Baume (BBC), Michael Meier (IRT), Niels Bogaards (elephantcandy), Benjamin Duval (Trinnov)
Reviewers	Nicolas Epain (b<>com), Olivier Warusfel (IRCAM)
Work package, Task	WP5
Keywords	Renderers, User interfaces

Abstract

This document aims to present an overview of work that has been carried by the various partners in the ORPHEUS consortium related to the renderers for object-based broadcasting clients.

Description of the demos and documentation is provided to illustrate the development related to the rendering in the web browser, in IP studio, in the pre-processor, in the AV receiver and in the iOS mobile app.

[End of abstract]

Document revision history

Version	Date	Description of change	List of contributor(s)
v0.1	19/01/2018	Initial structure, section 4	Benjamin Duval (Trinnov)
v0.2	13/02/2018	Sections 2, 3 and 4	Andrew Mason (BBC), Michael Meier (IRT)
v0.3	16/02/2018	Section 5 Conclusions	Niels Bogaards (elephantcandy) Benjamin Duval (Trinnov)
v0.4	21/02/2018	Introduction, executive summary	Benjamin Duval (Trinnov)
v0.5	26/02/2018	Reviewed	Nicolas Epain (b<>com)
v0.6	27/02/2018	Reviewed	Olivier Warusfel (IRCAM)
v0.7	28/02/2018	Section 2 revised	Chris Baume (BBC)
v0.8	01/03/2018	Updated reference architecture block diagram	Michael Weitnauer (IRT)
v0.9	02/03/2018	Formatting	Werner Bleisteiner (BR), Benjamin Duval (Trinnov)
v0.95	02/03/2018	Language corrections	Nicolas Epain (b<>com), Chris Baume (BBC)
v1.0	05/03/2018	Final editing	Anja Köhler (Eures)

Disclaimer

Copyright notice

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

Executive Summary

This document summarizes the work done by the ORPHEUS partners with regard to object-based audio broadcast rendering on the reception (consumer) side.

Most of the work has consisted of developing, implementing and testing individual components that were integrated into the different clients used during Pilot 1. MPEG-H decoding libraries have been integrated in end devices for end-user object-based audio reception, and tools and components based on the WebAudio API have been developed for rendering inside common web browsers. The IP radio studio, used to produce object-based content, also has a rendering capability, which allows monitoring of the content being produced. The pre-processor is used in any case of object-based rendering.

Most of the work presented in this document consists in actual working prototypes of components of the object-based broadcasting chain. These prototypes have been demonstrated live during the 2nd review of the project, which was held at the BBC premises in December 2017[2].

[2] two videos of this are available on the ORPHEUS website: https://orpheus-audio.eu/videos/

Table of Contents

Executive Summary **3**

Table of Contents **4**

Abbreviations **5**

1 Introduction **8**

2 Rendering in the web browser **9**

3 Rendering in IP Studio **10**

4 Rendering in the Preprocessor **12**

5 MPEG-H reception over a DASH stream with the high-end AV receiver **14**

6 Rendering implementation in the iOS mobile application **17**

7 Conclusions **20**

List of Figures

Figure 1: reception macro-block and monitoring sub-block in the ORPHEUS architecture 8

Figure 2: On-screen pop-up menu for selection of rendering type, in The Mermaid's Tears 9

Figure 3: Main pipeline from IP Studio used for pilot 1, stage C 10

Figure 4: IP studio in use during the Pilot 1 demo at the ORPHEUS 2nd review in December 2017.... 11

Figure 5: Simplified schematic view of the basic preprocessing strategies and signal flow 12

Figure 6: Schematic depiction of reducing four elements of a string ensemble down to two elements by using pre-rendering and Virtual Panning Spots (VPS) 13

Figure 7: Trinnov Altitude32 processor 14

Figure 8: GUI of the Trinnov Altitude32 processor 15

Figure 9: Temporary installation and GUI of the Trinnov Altitude32 processor in the IP studio for the Pilot 1 demo conducted at the ORPHEUS 2nd review in December 2017 16

Figure 10: Screenshots of the Audio Format selection and the Foreground/background balance control 18

Figure 11: Screenshots of interaction on an individual object (left) and 3D configurations (right) 19

List of Tables

Table 1: Audio output formats supported in the ORPHEUS Radio iOS app 17
Table 2: Examples of OBA interaction present in the iOS app 18

Abbreviations

AAC	Advanced Audio Coding
ADM	Audio Definition Model
API	Application Programming Interface
AVR	Audio Video Receiver
DASH	Dynamic Adaptive Streaming over HTTP
HRTF	Head-Related Transfer Function
MADI	Multichannel Audio Digital Interface
MPEG	Moving Picture Experts Group
OBA	Object-Based Audio
RTP	Real-time Transport Protocol
UMCP	Universal Media Composition Protocol
VBAP	Vector Base Amplitude Panning

1 Introduction

This document summarizes the work done by the ORPHEUS partners with regard to object-based audio broadcast rendering on the reception (consumer) side.

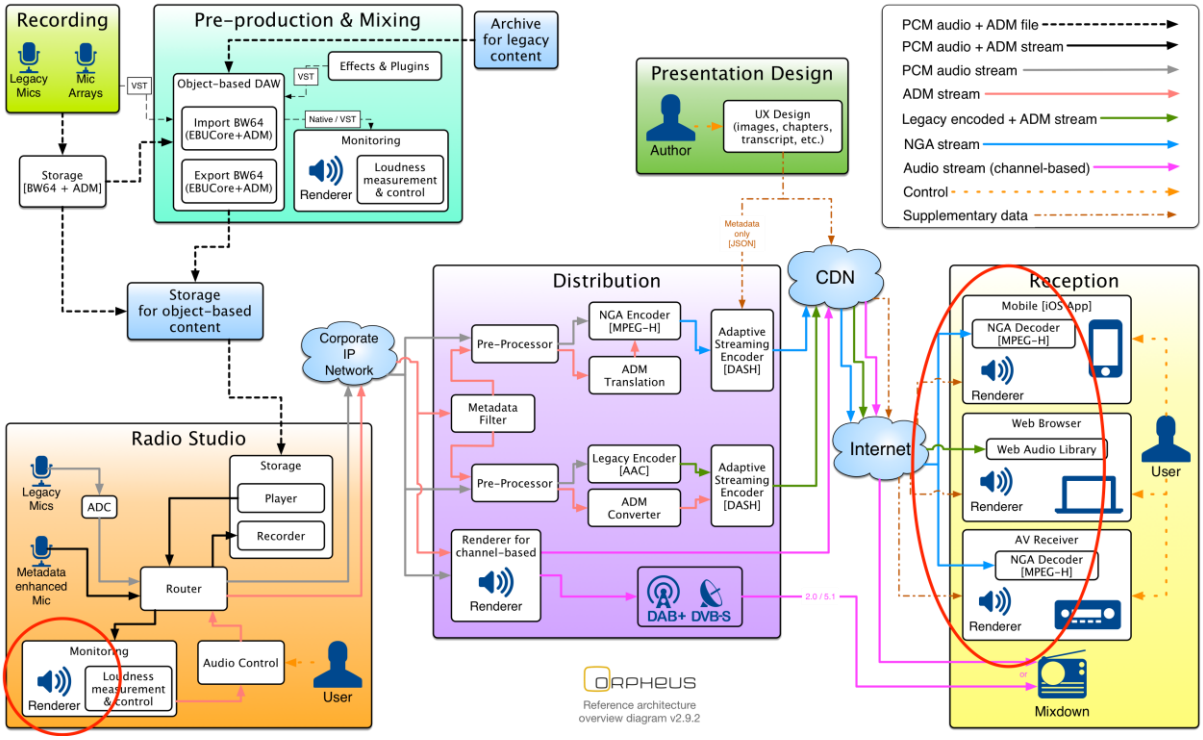

Figure 1: reception macro-block and monitoring sub-block in the ORPHEUS architecture

The ORPHEUS object-based broadcasting system is an extensive system where innovative solutions are required throughout. Work Package 5 (WP5) focuses on the reception side: the systems that consumers will see and use to experience object-based broadcasts.

Two distribution formats were chosen for use during Pilot 1: MPEG-H (for use in the mobile app client and hardware receiver) and AAC+ADM (for common web browsers)

For the integration of MPEG-H into the mobile app and the AV Receiver, FhG IIS has provided custom MPEG-H decoding libraries that can be integrated by the other partners. This integration work has been completed, and was demonstrated in Pilot 1 in December 2017.

Current versions of common web browsers do not support MPEG-H decoding yet, and within ORPHEUS a combination of AAC encoded audio streams and ADM metadata streams is used as an alternative. The emerging WebAudio API standard is used as a client-side rendering platform. Test components have been developed for critical parts of the rendering that were needed in Pilot 1.

2 Rendering in the web browser

Rendering in the web browser of audio transmitted as AAC with ADM-style metadata was implemented for the live transmission of The Mermaid's Tears and later for its evaluation on BBC Taster[3]. The rendering uses JavaScript written by BBC R&D to implement a renderer in combination with the Web Audio API[4].

Two main types of rendering options were presented to the user through a pop-up menu at the bottom left of the browser window, as shown in Figure 2.

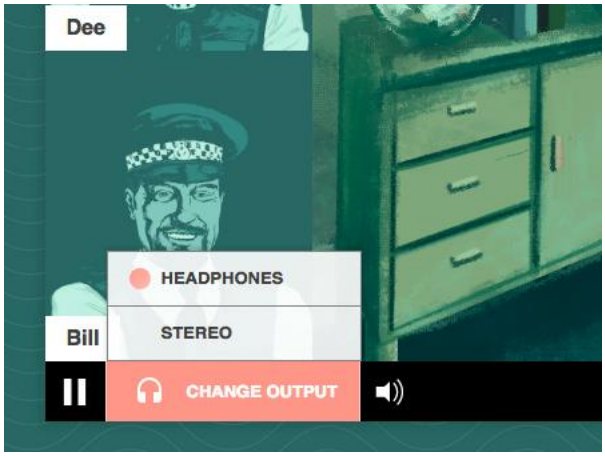

Figure 2: On-screen pop-up menu for selection of rendering type, in The Mermaid's Tears

1. Headphones

   Binaural rendering was used to give the user a 3D immersive audio experience over headphones. This was provided through a custom implementation in the BBC R&D JavaScript library, rather than the binaural rendering in the web browser, as it allowed the use of different HRTFs (although this option was not presented to the user).

2. Stereo/Surround

   Vector-base amplitude panning (VBAP)[5] was used to render to loudspeaker outputs. If the user's sound card had 5 or more output channels, a 5-channel surround option was presented. Otherwise, only stereo output was presented. The VBAP rendering was implemented within the BBC R&D JavaScript library.

[3] BBC Taster trials are time-limited. The Mermaid's Tears can still be replayed at http://mermaidstears.ch.bbc.co.uk

[4] https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API

[5] Pulkki, Ville, "Virtual Sound Source Positioning Using Vector Base Amplitude Panning", JAES Volume 45 Issue 6 pp. 456-466; June 1997

3 Rendering in IP Studio

A renderer is needed in the studio in order to know that the audio that is being created is going to sound as it should when it reaches the audience. For this purpose, a renderer was implemented as a component within IP Studio. For pilot 1, stage C[6], the renderer component was based on code from FhG to implement the renderer from MPEG-H.

An annotated screenshot of the main signal pipeline in IP Studio is shown in Figure 3. 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.

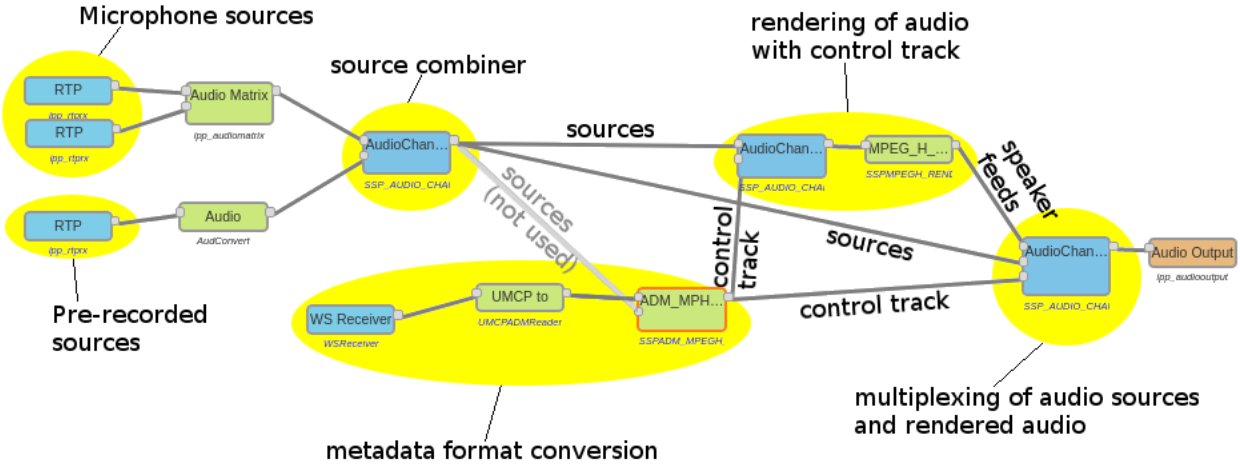

Figure 3: Main pipeline from IP Studio used for pilot 1, stage C

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.

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" (an audio-like PCM signal that carries the metadata). 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 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.

[6] In pilot 1 phase C, a live production was streamed using MPEG-H during the review meeting with a representative of the European Commission and independent assessors. See https://orpheus-audio.eu/videos/ for documentation.

Practical simplifications in this implementation.

In general, the hierarchy of objects that form the output of a broadcast chain is subject to change from time to time. In Pilot 1, stage A, the combination of objects into mutually-exclusive groups ("switch groups") allowed the audience to select from one of three character's perspectives. This was described by metadata in ADM and UMCP, and was relatively complex. For Pilot 1, stage C, a much simpler hierarchy of objects was used.

Rather than implement a fully dynamic meta-data handling system capable of dealing with any possible object hierarchy, knowledge of the nature of production was used to initialise the renderer component when the pipeline was started.

One of the advantages of object-based audio is its adaptability to different listening environments. However, in this instance, the number and position of the loudspeakers in the studio was known in advance, and dynamic reconfiguration was not a required capability. As such, when the pipeline was started, the renderer component was initialised with a static configuration of loudspeakers to which to render.

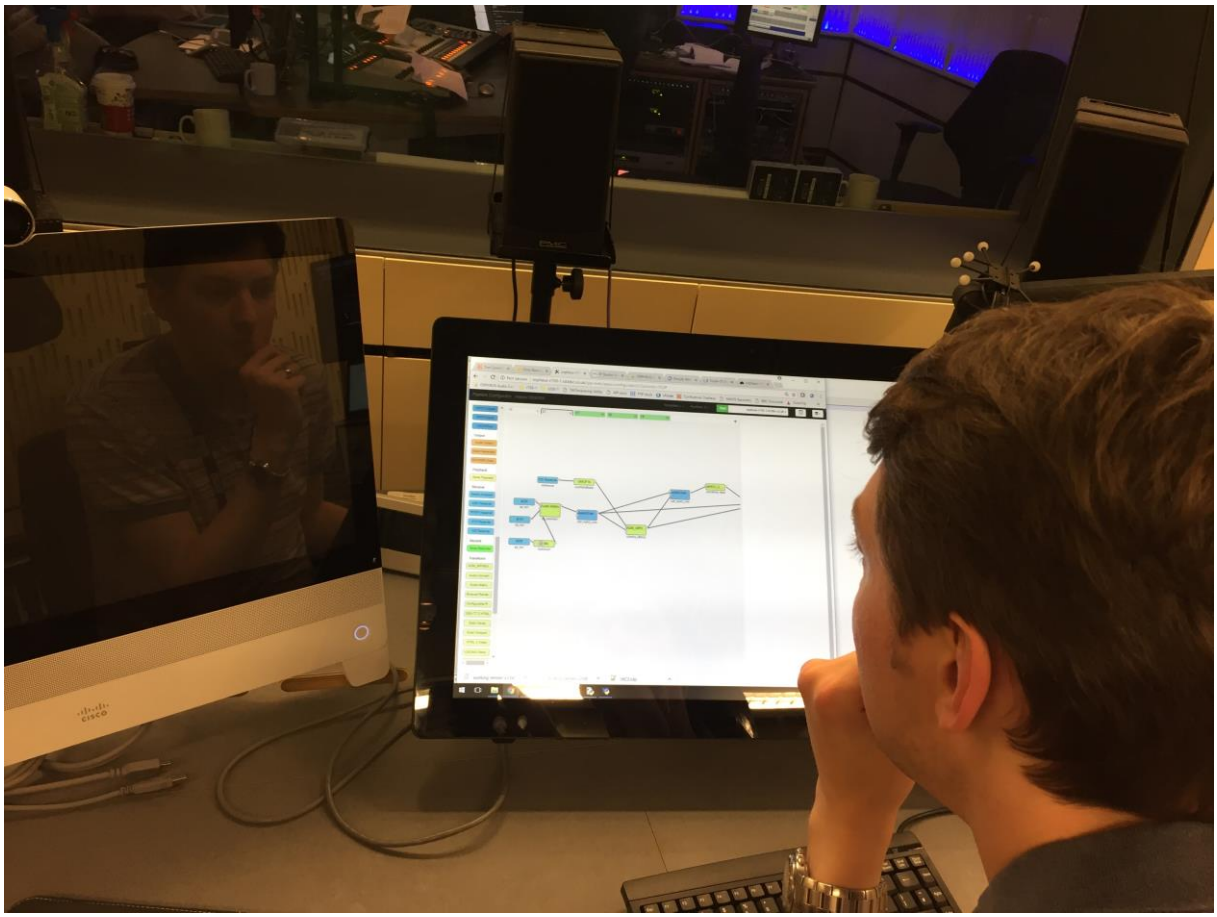

Figure 4: IP studio in use during the Pilot 1 demo at the ORPHEUS 2nd review in December 2017

4 Rendering in the Preprocessor

The main purpose of the *Preprocessor* is to transform one representation of an audio scene into another representation. This is, for example, useful to either reduce the complexity of the audio scene or to increase compatibility with other components in a signal chain.

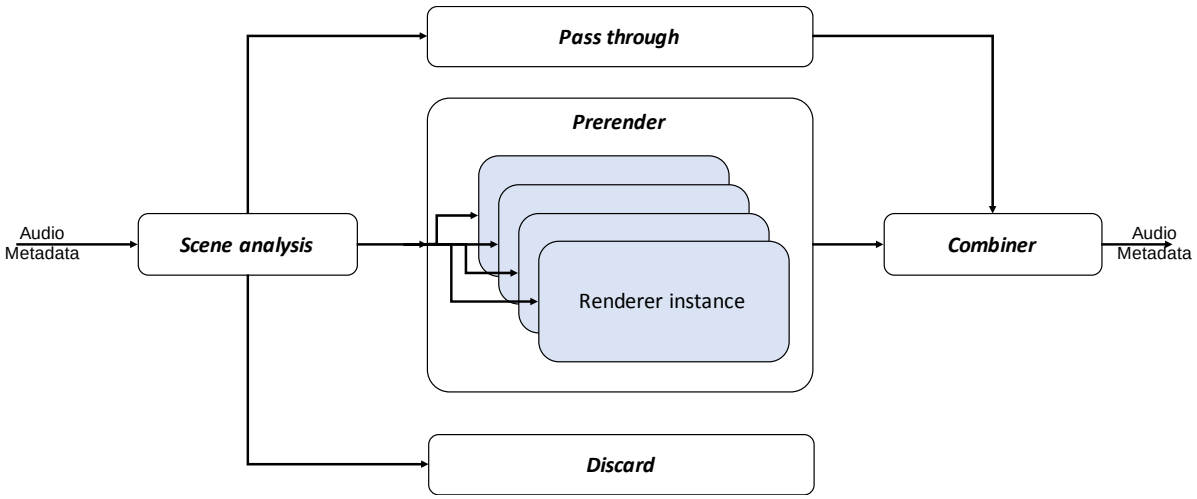

Figure 5: Simplified schematic view of the basic preprocessing strategies and signal flow

Even though it may not be obvious immediately, rendering is an integral part of the preprocessing component. This is because there are in principle three basic strategies that can be applied when transforming an object-based audio scene.

First of all, some elements should be passed through unchanged, for example if they must be kept separated to allow user interaction, or if these elements are very important to the fidelity of the audio scene.

The second and most obvious option to reduce computational complexity is to discard elements. While there are a few valid use cases for this strategy, it is apparent that this is suboptimal and will lead to a degradation of the quality of the audio scene in most cases.

The third strategy is to partially pre-render the audio scene. In its simplest form, a so-called channel bed will be added, which corresponds to a loudspeaker setup suitable for the target application. For headphone playback, the channel bed may also be a binaural signal. But this approach is not limited to using a single channel bed and the bed itself is, in fact, not restricted to predefined speaker setups. For example, multiple elements of an audio scene can be combined and grouped using techniques like Virtual Panning Spots (VPS).

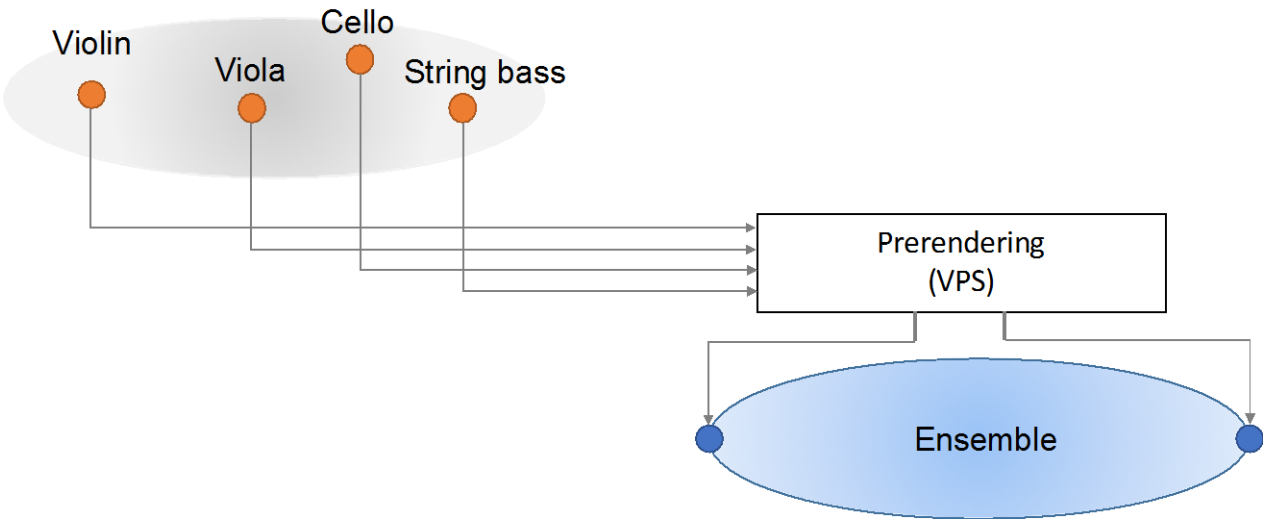

Figure 6: Schematic depiction of reducing four elements of a string ensemble down to two elements by using pre-rendering and Virtual Panning Spots (VPS)

The basic principle with this approach is to pre-render content to virtual speakers within the audio scene. As an example, one could consider a string ensemble represented by four distinct audio elements (see Figure 6). By panning these four elements between two virtual panning spots, the number of elements to render can be reduced while maintaining a nearly equivalent auditory event. The main advantage of this approach is that the position of those virtual speakers can be chosen to be optimal for the content to be preprocessed.

The preprocessing engine can therefore use multiple renderer instances to render parts of the audio scene and use optimized parameters for each subrendering.

The renderer implementation integrated into the preprocessing engine is a software implementation. By default, it uses a VBAP-based three-dimensional panning for rendering to (virtual) loudspeakers. Almost any arbitrary target loudspeaker setup geometries are supported. To render binaural signals, HRTF-based rendering is supported. By default, IRT's HRTF database of a Neumann KU100 dummy head is used, but other HRTF datasets may be used as well.

The number of elements that can be rendered simultaneously is only limited by the computational resources available; there is no algorithmic or architectural restriction.

Furthermore, the software is able to operate both in a file-based and in a real-time mode. The latter is important to process live events, for example, or to be able to use the renderer implementation in production.

5 MPEG-H reception over a DASH stream with the high-end AV receiver

The Trinnov *Altitude32* AV receiver is an AV preamplifier with decoding capabilities for several 3D audio formats, object or (non-object) channel based. It can drive up to 32 loudspeakers, making it the most powerful audio processor on the market. It is used in home theatres to reproduce contents from any physical or network source. Over the course of the ORPHEUS project, Trinnov integrated MPEG-H decoding libraries to the Altitude32.

Figure 7: Trinnov Altitude32 processor

Several steps have been necessary to conduct the integration of the MPEG-H decoder and renderer library in the Trinnov AVR. First, the existing code has been refactored to allow sufficient space for a new decoder. The AVR was already embedding a few decoders, and the cohabitation of yet another one was very difficult due to the software architecture. A new inter-process mechanism has been designed to allow the MPEG-H codec to run on the processor without disturbing the other ones, and to allow the user to switch between contents with different formats without reinitializing the machine.

Then, the computing performance of the AVR has been increased by optimising memory usage and computing efficiency. The smart use of multi-threading combined with efficient signal processing allowed to lower CPU usage, which in turn allowed the MPEG-H decoder to run on the processor.

Once the decoding and rendering library was ready, the integration started with the help of FhG. As the first ever integration of this library in an AVR driving so many loudspeakers, comprehensive tests had to be carried out to ensure the correct behavior. The example files provided were successfully decoded and extensively tested.

In addition, support for MPEG-DASH streaming was added to the AVR. As Trinnov uses a modified version of the free GStreamer media framework, FhG provided a DASH-Receiver as a GStreamer-Source-Plugin to handle this streaming format. This integrated library was tested on local streams first, and, after a few adjustments, on a remote stream. The result was a fluent reception and decoding of the streamed content and a correct rendering over any selected loudspeaker configuration.

A GUI is constructed automatically when the streaming starts, which allows adjusting stream specific parameters in real time. In the first example stream, it was for instance possible to:

- switch between two languages for the dialogue object (German and English),
- change the volume and/or the prominence of the audio objects (the dialogue and a bee that moves around the scene)
- change the position of one of the objects (the bee) within a range authorized by the sound designer

These operations could be done by selecting items in lists or moving sliders. The GUI is using web technology and can be accessed (after authorisation) from any browser that connects to the WiFi-AP of the AVR. Figure 8 shows the Trinnov-AVR Altitude32 next to the automatic GUI running on a tablet.

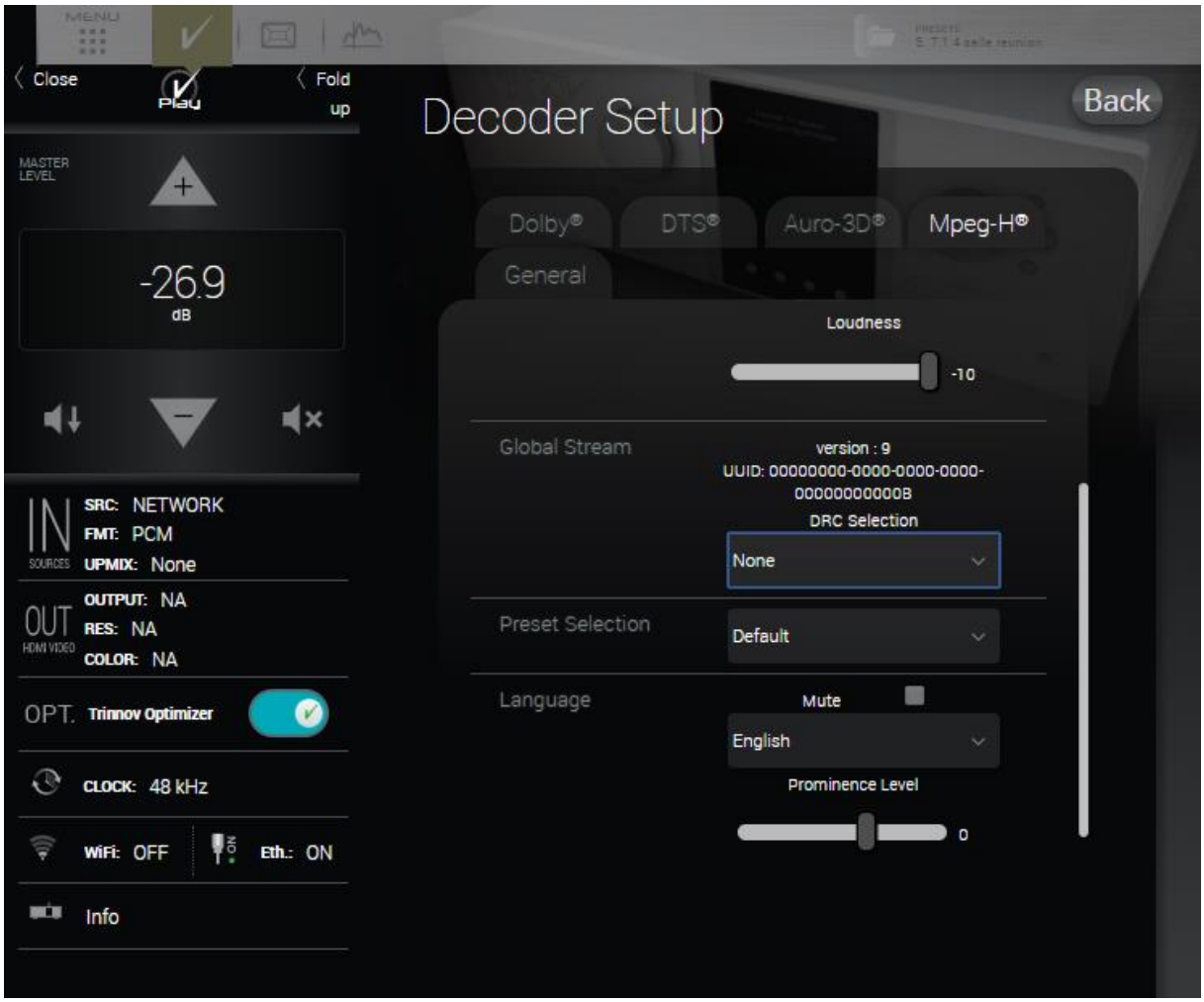

Figure 8: GUI of the Trinnov Altitude32 processor

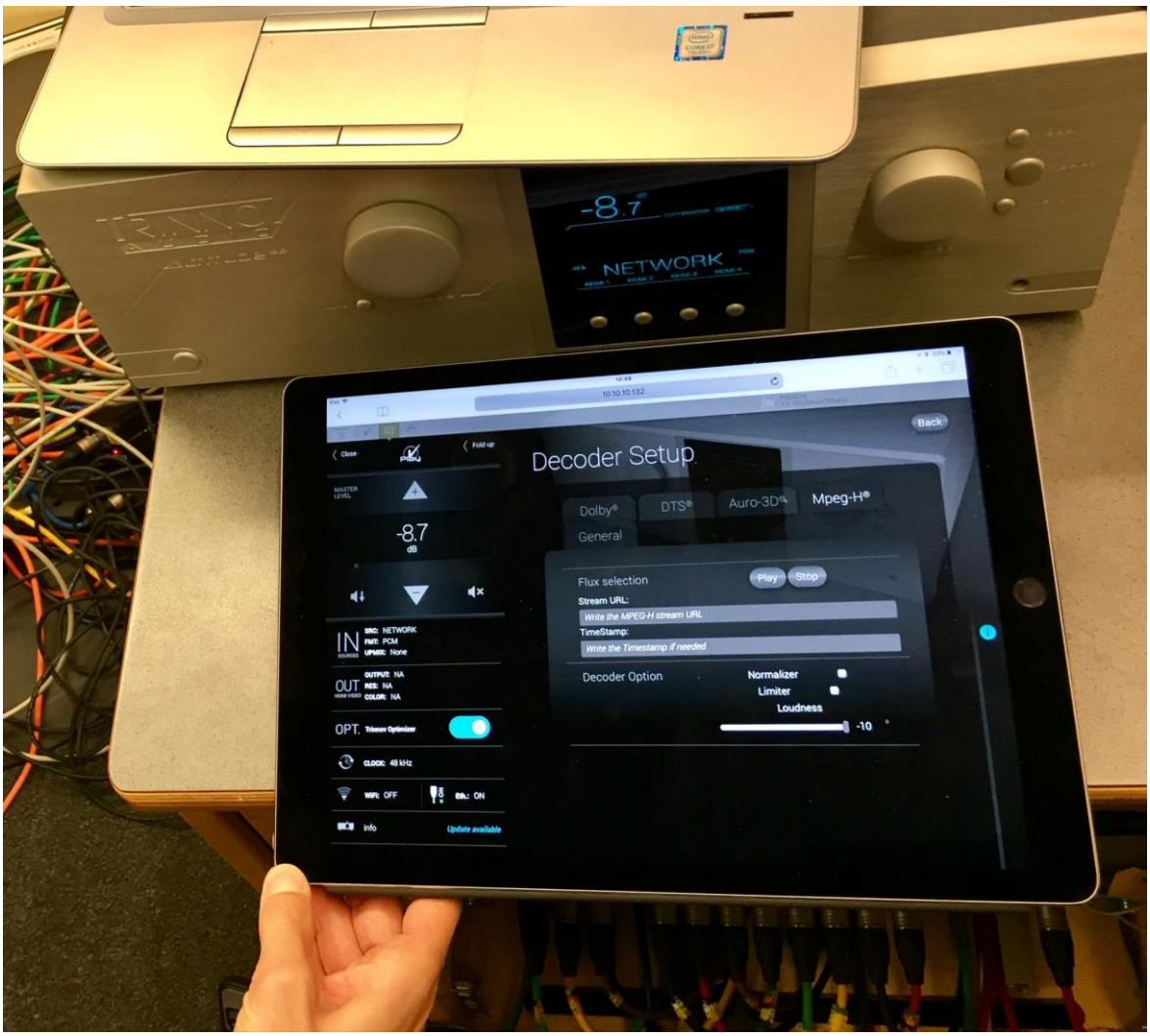

Figure 9: Temporary installation and GUI of the Trinnov Altitude32 processor in the IP studio for the Pilot 1 demo conducted at the ORPHEUS 2nd review in December 2017

6 Rendering implementation in the iOS mobile application

The iOS app's main purpose was to highlight as many of the novel possibilities of object-based broadcasting. A user interface was designed to make to the often very technical options understandable to a broad public. At the same time, the app served as a real-world proof of concept for object-based broadcasting and a complete streaming and rendering infrastructure was designed and implemented to provide a convincing listening experience.

Codec: MPEG-H

The object-based distribution codec chosen for the mobile app was MPEG-H, as it has many object-based audio features, has high bitrate reduction and an entire OBA radio program can be encoded in a single stream. Currently, no mobile operating system ships with an MPEG-H decoder. FHG has developed an MPEG-H decoder library for iOS, but it was only used for internal projects. Elephantcandy successfully integrated the FHG library into the mobile ORPHEUS iOS app.

Streaming: MPEG-DASH

Every ORPHEUS content was streamed to end-users using MPEG-DASH. To receive the MPEG-H file fragments, elephantcandy developed an MPEG-DASH client library compatible with the mp4 fragments containing the MPEG-H data. To allow smooth playback in variable-length playback situations, as well as support offline playback, a smart caching and pre-fetching system was developed, capable of linking MPEG-DASH fragments to the logical time of a radio program.

Output Formats

Mobile radio apps are used in many situations: at home, on the road, during sports activities, etc. For each situation, different reproduction hardware may be connected, from Hi-Fi headphones to wireless speakers or multi-channel surround sound setups. An important feature of object-based audio is that the audio can be rendered optimally for all these different situations. The iOS app supports the following configurations:

audio format	*typical connection*
mono	internal speaker
stereo	headphones, loudspeakers, AirPlay
binaural	headphones
5.1 surround	HDMI

Table 1: Audio output formats supported in the ORPHEUS Radio iOS app

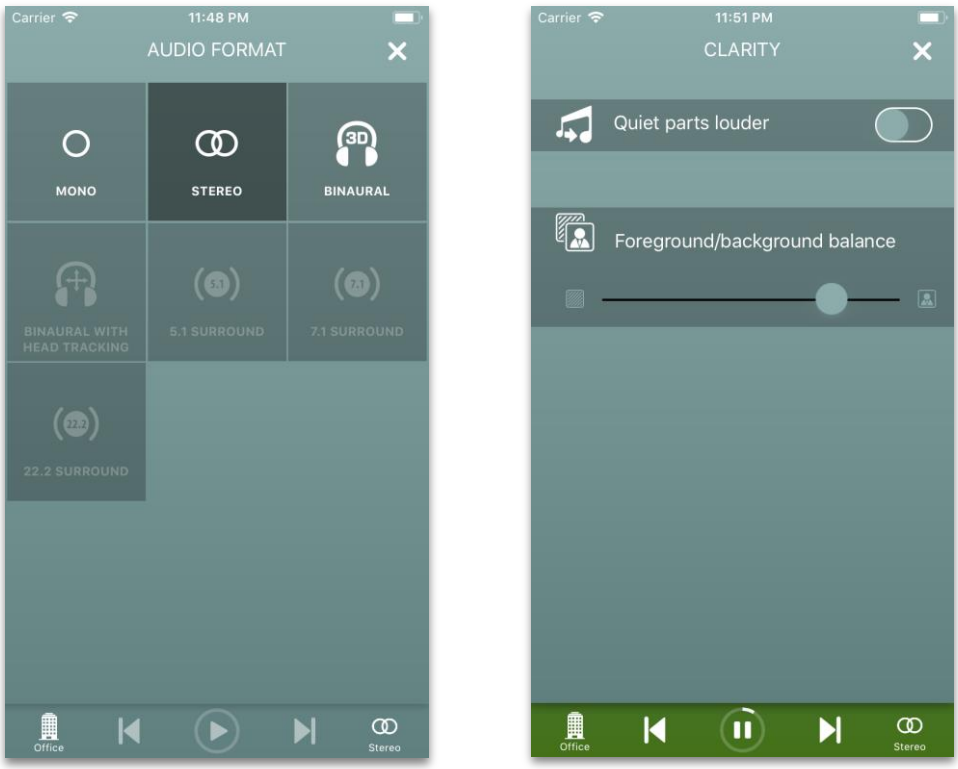

Figure 10: Screenshots of the Audio Format selection and the Foreground/background balance control

Interactive Audio Objects

Using a mobile app, rather than conventional radio reception hardware, it becomes possible to provide much more control over the audio and the program. With OBA, using specialised user interfaces and pre-defined interaction models, a listener may be able to selectively toggle audio objects on and off, adjust their relative prominence, select alternative objects or alter an object's rendered position in 2D or 3D space. The programs available in the iOS app demonstrate many of these features:

Interaction	*Example*
object replacement	language selection (most programs are available in English and German, some also in French)
object prominence	foreground/background adjustment (for instance to adjust commentator vs. stadium noise during a football game)
individual object manipulation	freely movable objects that can be placed anywhere in 3D space, muted, changed in prominence

Table 2: Examples of OBA interaction present in the iOS app

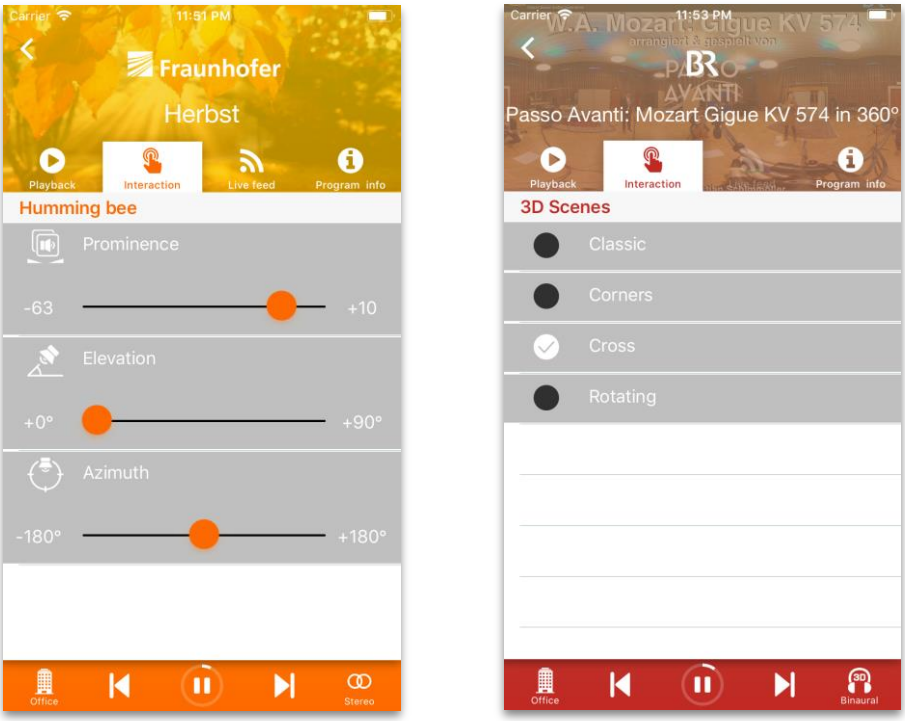

Figure 11: Screenshots of interaction on an individual object (left) and 3D configurations (right)

Non-linear playback

In the ORPHEUS project, object-based audio not only implies the manipulation of audio objects and their rendering to various output formats, but also the interactive selection of which sections in time of the stream to play, based on the listener's preferences and timed metadata. From a rendering point of view in the mobile app, this means that the streamed MPEG-H fragments need to be assembled in dynamic configurations. To this end, a sophisticated caching system was developed for the Pilot Phase 2 version of the ORPHEUS Radio app.

7 Conclusions

This document presented the different implementations of the object-based audio renderers into the demonstrators developed for the ORPHEUS project. This work has been carried within task T5.1, and has been used for Pilot 1 integrated within the WP2.

These object-based renderers allow for several interactions (object selection such as language selection, foreground/background balance, object position manipulation, etc.), that may be specific to a stream. A content-specific user interface is created to let the user perform these controls.

Some of these interactions are similar throughout all the renderer implementations. Others are specific to the typical use of the target platform: a web browser will not offer the exact same capabilities as a high-end AVR; the aim of the IP studio is to create some object-based content, whereas the iOS app is to play it to end users.

For more challenging usage situations, some techniques of bandwidth and/or complexity reduction have been developed in order to ensure the best user experience possible

These demonstrators were presented during the second project review, which took place in London in December 2017, and will be shown again for a larger audience, improved with the latest progress being prepared, at the final ORPHEUS workshop in Munich in May 2018.

[end of document]