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

D4.3: Final report on the work on representation, archiving and provision of Object-based Audio

Version: v1.0

Deliverable type	R (Document, report)
Dissemination level	PU (Public)
Due date	31/01/2018
Submission date	12/02/2018
Lead editor	Andreas Silzle (FHG)
Authors	Niko Färber (FHG), Tilman Herberger (MAGIX), Marius Vopel (Magix), Benjamin Duval (Trinnov), Matt Firth (BBC), David Marston (BBC), Matt Paradis (BBC), Niels Bogaards (ECANDY), Michael Meier (IRT), Olivier Warusfel (IRCAM), Thibaut Carpentier (IRCAM), Markus Noisternig (IRCAM)
Reviewers	Werner Bleisteiner (BR)
Work package, Task	WP 4, T4.1, T4.2, and T4.3
Keywords	Formats, BW64, ADM, MPEG-H, DASH, IP-Studio, Sequoia, iOS app

Abstract

This deliverable describes the representation, archiving and provision of object-based audio. It builds on D4.1, which describes the requirements, and D4.2, which describes the selected and used formats. This deliverable describes the implementation experience gained during the project, especially for the Audio Definition Model and the metadata formats for non-linear reproduction.

This deliverable also serves as a documentation of milestone MS4.3 “Final implementation and documentation of formats for representation, archiving and provision of object-based audio” which has been achieved on 31/01/18.

Document revision history

Version	Date	Description of change	List of contributor(s)
0v1	21/12/2017	Initial version	Andreas Silzle (FHG), Nikolaus Färber (FHG)
0v11	12/01/2018	MAGIX raw text inserted	Tilman Herberger
0v12	19/01/2018	MAGIX text refined	Marius Vopel
0v2	23/01/2018	Additions by BBC	Matt Firth (BBC), David Marston (BBC), Matt Paradis (BBC)
0v3	25/01/2018	Additions by ECANDY	Niels Bogaards (ECANDY)
0v4	29/01/2018	Additions by IRT	Michael Meier (IRT)
0v5	30/01/2018	Additions by IRCAM	Olivier Warusfel (IRCAM), Thibaut Carpentier (IRCAM), Markus Noisternig (IRCAM)
0v6	08/01/2018	Editorial changes	Andreas Silzle (FHG)
V1.0	12/02/2018	Final editing	Uwe Herzog (EURES)

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

Executive Summary

This deliverable describes the representation, archiving and provision of object-based audio. It builds on D4.1, which describes the requirements and D4.2, which describes the selected and used formats. This deliverable describes the implementation experience gained during the project.

The implementation, usage and translation of the Audio Definition Model (ADM) is described for the DAW Sequoia, the real-time production software IP-Studio, the ADMix tools suite, and the MPEG-H format. The formats for the end-user device AV receiver and the end-user software iPhone app and Browser app are explained. It is explained how the Universal Media Composition Protocol (UMCP) format is used in IP-Studio and how it is related to ADM. Also the mapping of ADM to the Sequoia internal data representation is explained and justified.

The use cases found in the ADM expert workshop are listed. The ADM pre-processor, mainly developed to reduce the number of audio objects in a meaningful way, with its reduction mechanism is explained. The issues to translate a general ADM description are described. The solution of defining an ADM Broadcast profile and using ADM template is presented.

The metadata for the non-linear reproduction, importance levels, inserted in the DAW Sequoia, their usage on the production and reproduction side in the iPhone app, are described. With mechanism variable length requirements from the end-user to a program can be implemented in an elegant way.

The usage of DASH and the WebAudio API for the browser implementation in ORPHEUS is explained.

Table of Contents

List of Figures

List of Tables

Abbreviations

AAC	Advanced Audio Coding
ADM	Audio definition model
API	Application programming interface
BBC	British Broadcasting Corporation
BR	Bayerischer Rundfunk
BW64	Broadcast Wave 64Bit
BWF	Broadcast wave
DASH	Dynamic Adaptive Streaming over HTTP
EBUCore	Metadata set definition by the European Broadcasting Union (EBU-UER)
FHG	Fraunhofer Gesellschaft
HOA	Higher Order Ambisonics
IP	Internet Protocol
IP-Studio	BBC's development of an integrated AV technology using IP standard
ITU-R	International Telecommunication Union, Radio communication Sector
JSON	Format for exchanging object data
MHAS	MPEG-H Audio Stream
MPD	Media Presentation Description
MPEG	Moving Picture Experts Group
NGA	Next Generation Audio
NMOS	Networked Media Open Specifications
OB	Outside broadcasting
OSC	Open Sound Control
PCM	Pulse-code modulation
S-ADM	Serial ADM
UMCP	Universal Media Composition Protocol
XML	Extensible Markup Language

1 Introduction

This deliverable is the final report on representation, archiving and provision of object-based audio. It builds on deliverable D4.1 and D4.2.

D4.1 has outlined the basic requirements for representation (a file or stream containing), archiving (long-term storage) and provisioning (the distribution to end-user, including IP delivery, unicast streams and file downloads) of object-based audio. These formats have to interoperate seamless within the signal chain blocks of the broadcast implementation architecture:

- Recording
- Pre-production and mixing
- Radio studio
- Distribution to the end-user and their reception

For both, file and streaming formats, specific audio related metadata has to be attached in order to enable typical object-based audio features. Here, the ITU-R defined Audio Definition Model (ADM) is the envisioned, most recent standard. It encompasses a full range of requirements; amongst them metadata support, existing standards, file size, compression, and support of advanced audio.

The requirements for the streaming formats were explained: synchronization, other existing standards, unicast and multicast transmission, latency, bitrate reduction, quality of service and specific requirements for streaming object-based audio metadata. Because there is not a single format capable of fulfilling all requirements and for all use cases, several formats have to be applied simultaneously. The difficulty and requirements of interoperability between these different formats were outlined.

The backward and forward compatibility issue with legacy systems was discussed: how to integrate legacy content formats in new production systems, as well as how to handle new content the best possible way for legacy emission systems.

The above listed information and requirements were the basis for the selection and new definitions of file and streaming formats, selected for the reference architecture and the pilot implementations.

In D4.2 the selected formats for the ORPHEUS project for production have been defined: BW64, ADM, NMOS, UMCP. BW64 has become the most prominent format for audio and metadata. It allows file sizes larger than 4 GByte and includes the metadata in ADM format. The within the ORPHEUS project necessary and selected metadata parameters were nominated and explained, including their respective value range. NMOS and UMCP are described as fundamental formats in BBC's IP Studio production software.

Various ongoing standardisation activities for these formats, in which several partners are involved, were mentioned. The implementation status of the formats in IP Studio and the DAW Sequoia were explained as well as practical issues for ADM to MPEG-H metadata conversions and the requirements for archiving using BW64.

For distribution and provision to the end-user, MPEG-DASH as advanced adaptive streaming technique is used to deliver AV content to HTML5 capable browsers and MPEG-H clients. The used standards and the technical details for MPEG-DASH and the AAC plus ADM Streaming were listed, as well as the streaming solution of MPEG-H over DASH. For AAC plus ADM streaming the necessary steps are encoding, segmentation, transfer to public domain, media reference and playback and rendering. The advanced features of the Next Generation Audio (NGA) codec MPEG-H were mentioned, including the way how packaged into MPEG-DASH. Short syntax examples are given for both cases, as well as several references to the used standards and explaining publications.

D4.3 now describes the implementation experience gained during the project: which features are used, issues regarding the specification and implementation, the application programming interface (API) and performance issues are explained.

2 Experience with the Implementation of Production Formats for Distribution and Archiving

2.1 In IP-Studio

2.1.1 UMCP and the Production Interface

Within IP-Studio, object-based productions are composed and stored using the Universal Media Composition Protocol (UMCP). As explained in section 2.6 of Deliverable 4.2, UMCP is a method to describe the contribution and processing of individual media assets to produce a new piece of media. These sets of instructions are known as UMCP Compositions.

The ORPHEUS Production Interface for IP-Studio uses the UMCP API to write Compositions describing a production. The API provides a Web Socket subscription feature which broadcasts notifications of Composition modifications to clients. This allows an IP-Studio UMCP rendering pipeline to subscribe to the Composition and produce an in-studio monitoring feed. The Composition can also be received by processing and distribution pipelines for broadcast.

UMCP provides a powerful feature that allows a Composition to reference another Composition as a contributing source. This ‘cascading’ or ‘nesting’ ability enables productions to become modular by breaking them down in to more manageable pieces. It also enables subsections of a production to be encapsulated and reused within other productions by simply creating that subsection in a separate Composition. This was the methodology used in producing ‘The Mermaid’s Tears’ pilot.

In ‘The Mermaid’s Tears’, the client was able to experience three different perspectives of the drama by following one of the three main characters as they moved between locations. Since each audio source could contribute to one or more experiences, this modular ability of UMCP was particularly effective.

In producing the drama, each audio source (e.g., actor microphone or sound effect) was encapsulated in its own Composition, which were termed ‘Elements’. These Elements contributed to parent ‘Room’ Compositions, which built the soundscapes of the various locations in the drama. The contribution of Elements to Room Compositions could be dynamically controlled, allowing for characters to effectively move between rooms. Above these Compositions were three ‘Story’ Compositions, which were the three experiences offered to the client. These Compositions tracked each one of the main characters as they moved between Rooms by switching between contributing Room Compositions. This ensured that the client experienced the entire soundscape at the characters location, which could include the voices of other characters.

The relationship of these three types of Compositions is shown in Figure 1, where the connections between the Compositions change as each of the stories evolve.

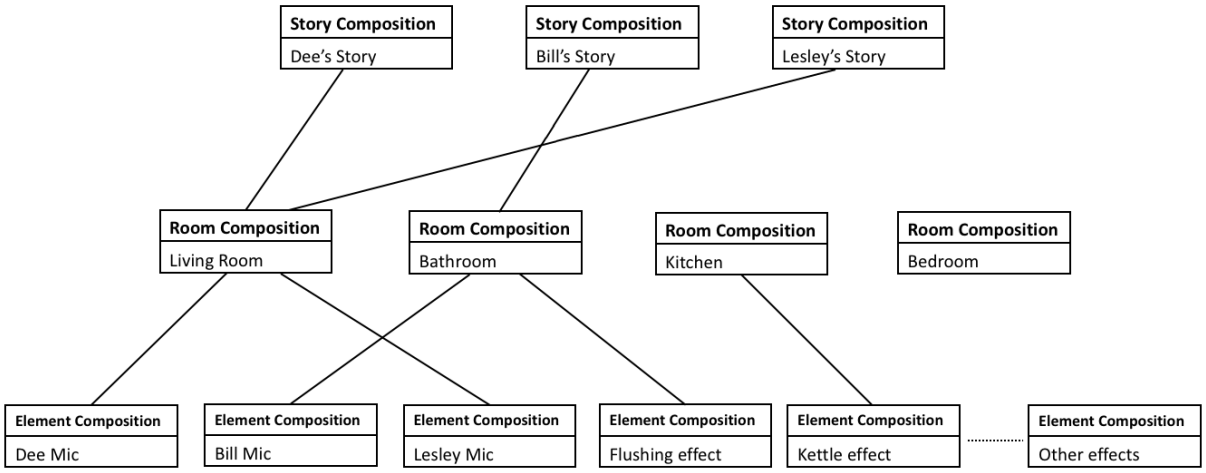

Figure 1: UMCP Compositions for Mermaid's Tears

The flexibility of UMCP enables generic metadata to be inserted in to Compositions as well as data needed to reproduce the object-based production. Using this capability, the production interface was able to insert image triggers in to the UMCP stream, which were subsequently used to dynamically present images to the client to compliment the drama via the web app.

UMCP began development in 2016 at BBC R&D. As a young and evolving technology, it was inevitable that there would be some shortcomings and limitations in UMCP. The ORPHEUS project provided an excellent use case and test implementation for the technology which highlighted some issues.

The most limiting factor encountered during development of the production interface was that the UMCP API was capable of processing only a few parameter updates each second, making smooth fade or pan changes impossible. In addition, the latency of notifications to subscribed clients was over a second. This issue has been addressed to some extent, with the API now able to process approximately 20 updates a second with a latency of around 10ms when loaded with 150 connected clients. There are additional planned improvements which are expected to further increase performance. This includes an active area of work to develop a distributed implementation of the API which is expected to not only dramatically improve performance, but also reliability.

The long-term goal for UMCP is to open source the specification to encourage adoption within the industry.

2.1.2 How UMCP relates to the ADM

The question of why UMCP is required when there is the ADM, or vice versa, can be asked. While they are both metadata formats of sorts, they have different purposes and scopes. UMCP is more generalised, covering both audio and video and does not specify the media parameters themselves, but rather a structure to carry them and how they are connected. Whereas the ADM only covers audio, and it does specify the audio parameters, so is less abstract. So, in theory, UMCP could be used as carrier for ADM metadata.

Another key difference is indicated in their names – UMCP is a *composition* protocol, but ADM is a *definition* model. So UMCP is used to carry real-time information about the composition from the controls used to produce the media, such as the audio mixing desk gain controls used in the Mermaid's Tears pilot. The ADM is used to carry the definition of the resultant audio mix for subsequent distribution and processing.

Therefore, a typical setup in the production process would be to have an ADM metadata (or more likely Serial ADM in a live set-up) generator on the output of the production chain. The UMCP metadata would then be used within the IP Studio production area as an input to the ADM metadata generator to modify its parameters (and possibly the accompanying audio).

2.2 In Sequoia: Mapping the ADM to the project structure of Sequoia

Due to the structural differences between the ADM standard and Sequoia's internal data organization, the implementation of import and export mappings for the specific object-based metadata introduced a non-trivial task. The ADM hierarchy presents a greater number of levels, which may also be nested. By comparison the data model of Sequoia is much simpler and straightforward (see Figure 2 and Figure 3).

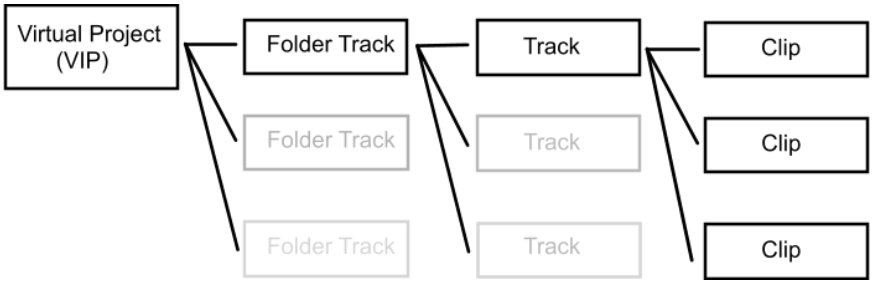

Figure 2: Internal structure of a Sequoia project

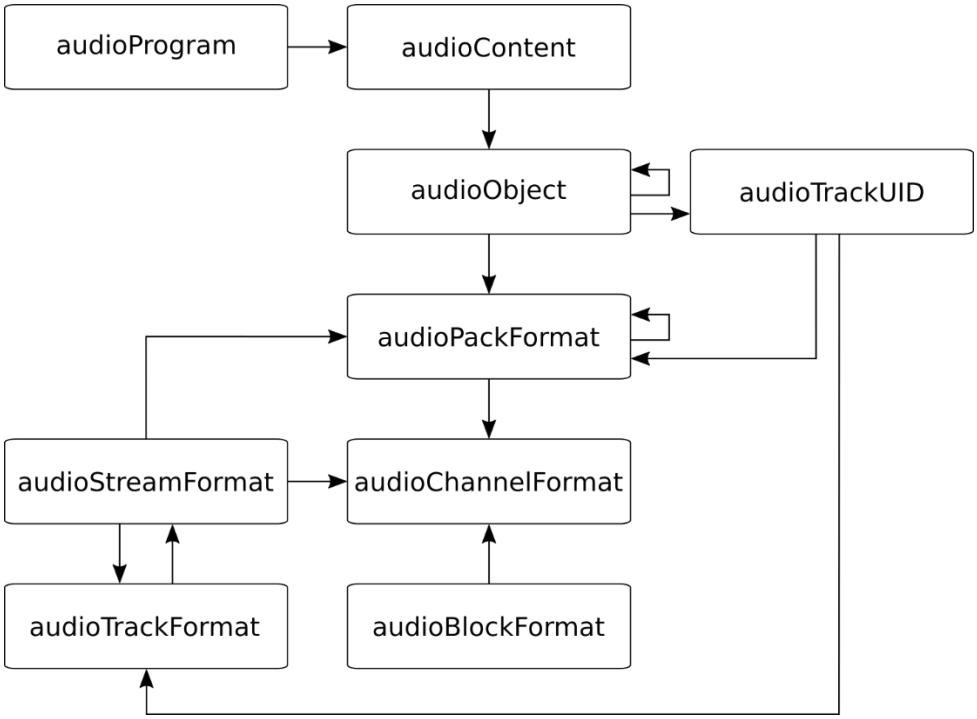

Figure 3: ADM structure

At first it seemed like the obvious solution would be to map ADM objects to Sequoia clips, but after regarding the most important use cases for ADM projects, it was decided to have Sequoia tracks fulfil this role (see Figure 4).

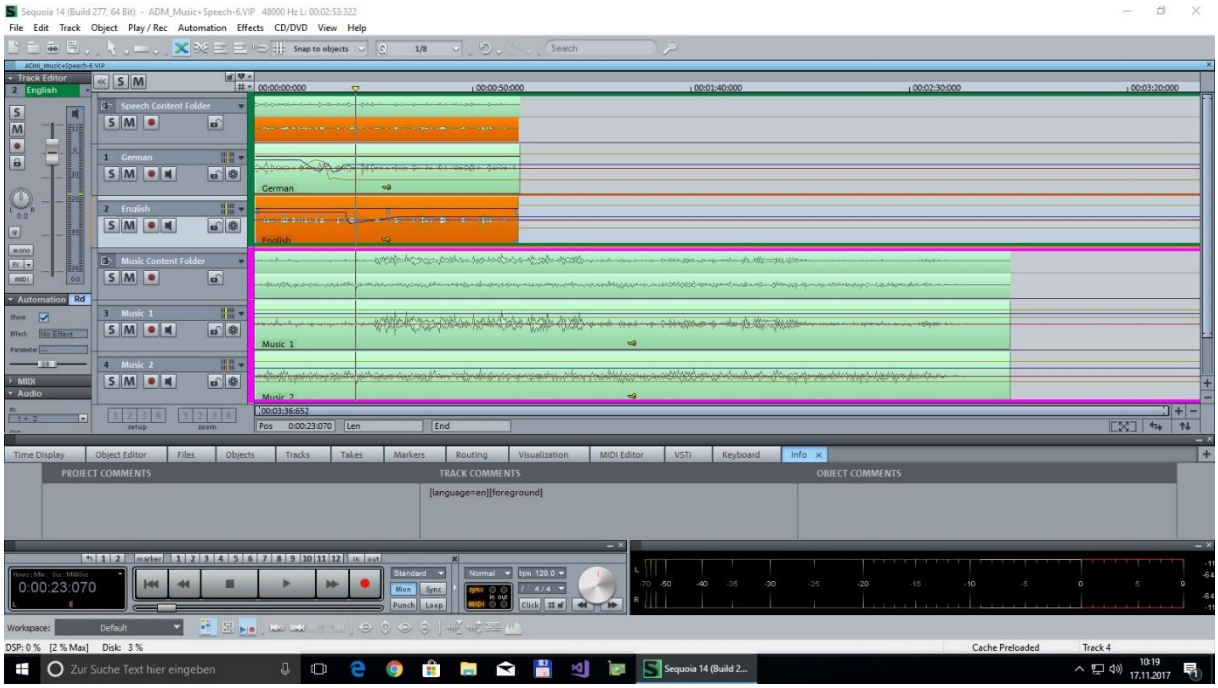

Figure 4: VIP with open ADM project

The following factors brought us to this decision:

User Experience for audio engineers: We wanted to make the ADM integration as transparent and comfortable as possible for audio engineers. They typically use tracks to organize large productions and assign panning information and other metadata to these tracks.

Integration of interactivity features: The ADM allows for the definition of a variety of interactivity features, giving the end client the means of presenting a more individualized experience. Within the ORPHEUS project the focus has been on language switching and foreground/background balancing. In both cases the audio material is typically represented in parallel tracks. Representing ADM objects as clips would likely make the editing of these features rather tedious.

Current technical limitations: A normal track bus in Sequoia as well as any clips placed on it cannot hold more than two audio channels and one set of automation parameters. An ADM object however may contain any number of audio channels with individual pan and gain data. Representing ADM objects as clips would require very extensive and time-consuming modifications to Sequoia's internal structures.

The sum of these factors led to the decision of representing ADM objects using tracks and ADM content elements using folder tracks as a way of grouping objects together.

In Sequoia it is possible for the user to add comments to clips, tracks or whole projects. This functionality was created as a means of easily taking quick notes e.g. about mixing decisions that were made. Having this kind of information saved right within a project file is a convenient way of relaying thoughts or plans to another user in a collaborative setting or keeping this information for when a project is reopened years later.

The track comment field can now be used to access or add metadata to ADM objects by including certain tags as well. For multichannel objects, such tags need to be included on the first associated track only.

Feature	Tagging format	ADM relation
Language association	[language=<desired language>]	This tag sets the language of the ADM audioProgram, and, if multiple different language tags are used, creates additional audioPrograms for alternative language versions with the corresponding objects tied to them.
Foreground/back-ground association	[foreground]	This is represented by generating additional objects labelled 'foreground' and 'background' without any referenced audio. They only serve an additional way of grouping the actual audio objects, which are nested below them. Only foreground objects need to be tagged. All other objects are assigned to the background implicitly. It should be noted that this is not a standardized field in the ADM, but rather a specific way of using it. The feature was added nevertheless, since it was deemed an important use case.
Importance	[importance=<0-10>]	This assigns an importance value to an object. Using this information a renderer might choose to discard an unimportant object e.g. when the load of processing gets too high.
Interactive muting	[onOffInteract]	This tag indicates that a user may mute the associated object interactively. A pre-processing tool might also use this information to render a project to a channel bed, while preserving interactive objects as separate elements.

Table 1: ADM tags supported by Sequoia

The current state of implementation was geared towards the needs of the ORPHEUS project, i.e. only the required features were realized. For a proper release version it is planned to create a dedicated metadata editing module (see Figure 5).

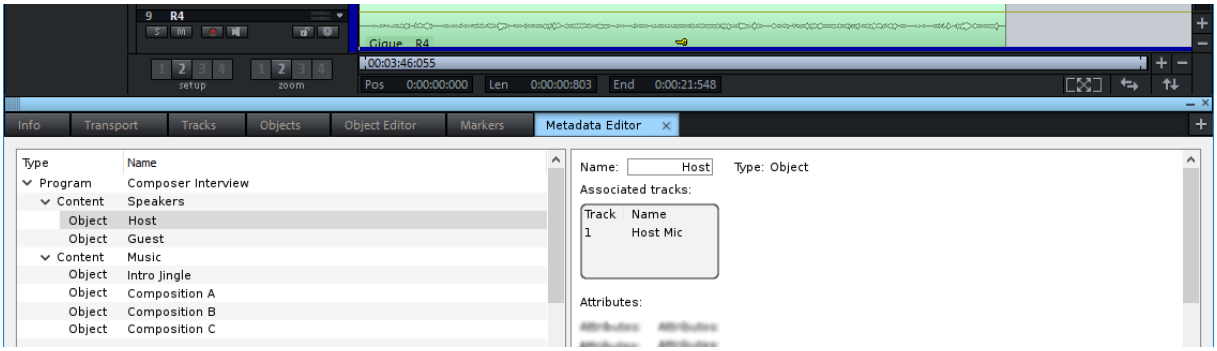

Figure 5: Metadata editor concept

For a more detailed overview see deliverable D3.6.

2.3 In ADMix Tools Suite

The ADMix tools suite is composed of three main components able to read, render and record ADM files, respectively. A fourth component is able to extract the XML data chunk embedded in the ADM file and creates a series of graphs (DOT format) depicting the structure of the metadata. The ADMix tools suite is available for MacOS and MS Windows.

Whereas the ADMix Renderer is a pure standalone application able to read and render an ADM file over any kind of rendering setup, the ADMix Recorder should rather be considered as a “plug-out” that can be interconnected with any Digital Audio Workstation (DAW) using the Open Sound Control (OSC) protocol to transmit the ADM metadata. Although this approach presents some limitations, the advantage is that this application can be used with any existing DAW that does not yet support ADM export/import or that does not yet support all the ADM features.

The audio rendering of the ADMix tools is based on the IRCAM Spat~ real-time software library [1] which supports most of the existing 2D and 3D rendering formats, in particular binaural over headphones, VBAP, and HOA over 2D or 3D loudspeaker layouts. These rendering components are used in the ADMix Renderer as well as in the ADMix Recorder for monitoring the recorded audio signals in real-time.

The different components of the ADMix tools exchange metadata using the Open Sound Control (OSC) protocol as a container. For instance, the Player reads the audio object tracks and associated metadata and uses an ad-hoc OSC syntax to forward these metadata to the Renderer that calculates appropriate signals for real-time reproduction over headphones or loudspeakers. More generally, the ADMix tools can communicate with any external OSC compliant audio device or software application. For instance, the ADMix Recorder can be connected to any Digital Audio Workstation (DAW) using the ToscA [2] plugin that is used to translate ADM metadata to automation. ToscA is a plugin for DAWs that allows the recording of automation tracks for arbitrary parameters that can be sent and received as OSC messages via a network interface.

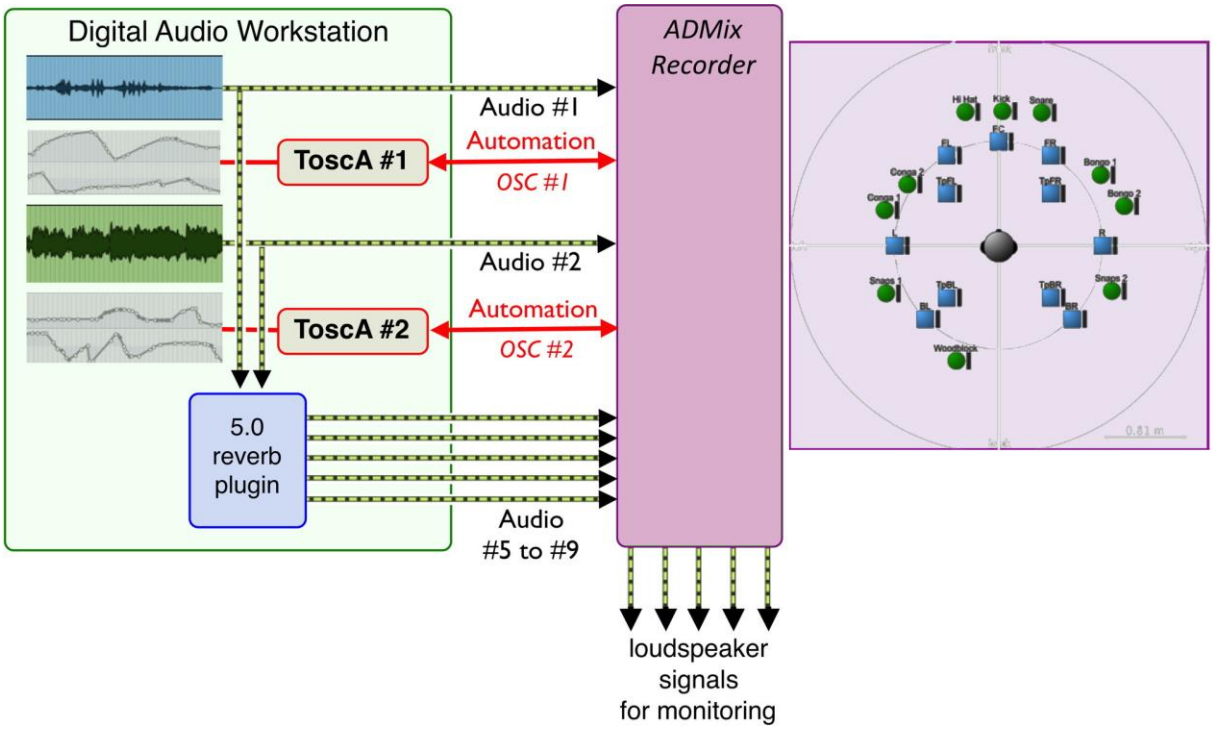

Figure 6: Using the ToscA plugin to connect a DAW with the ADMix Recorder

Figure 6 depicts the usual workflow used to create an ADM file. The *ADMix Recorder* receives the audio signals delivered by the DAW either via hardware connections (for instance if they are run on separate computers) or via virtual sound drivers. These signals may convey for instance single audio object tracks, multichannel stems (e.g. direct-to-speakers or HOA encoded objects) or reverberation beds provided by any reverberation plugin. The *ADMix Recorder* provides a GUI that represents the sound scene and that can be used to manipulate the position of the different objects. An instance of the ToscA plugin has to be inserted into any track of the DAW that is supposed to contain automation data. Each ToscA instance is configured with a channel ID that is used to associate tracks in the DAW with channels defined in the ADM Recorder. As OSC messages are sent via network interface, it follows that the DAW and the *ADMix Recorder* can run on separate computers.

The OSC protocol was chosen for its simplicity, ease of integration and wide usage in the audio community. An ad-hoc (non standardized) syntax was elaborated to convert ADM/XML parameters to an OSC-like grammar. The primary motivation of the ADMix tools was the rapid prototyping of object-based reverberation scenarios; therefore they do not support (yet) the complete ADM specifications, and the OSC grammar only reflects a subset of the most relevant ADM parameters (positions, gains) for these use cases.

A track with an LTC time code signal can be used to synchronize start and stop times between the DAW and the ADM Recorder. However, there is no accurate timing synchronization of messages between the DAW and the ADMix tools. Minor jitter might be noticed due to the OSC transport layer herein used. These timing inaccuracies (typically less than 5 msec) were assumed negligible for the intended scenarios.

2.3.1 Editing of metadata

Static metadata

A GUI interface is provided by the ADMix Recorder to define the structure of the ADM file, i.e. the number and the type of the different objects. On the top of this configuration window (Figure 7), the programme and content name can be specified (1). Below that, a list of "audio packs" (2) and (3) can be set up. Each pack contains one or more "channels". Each channel has a unique channel number that can be used together with the routing matrix on the left side of the window (4). The different inputs of the routing matrix correspond for instance to the audio tracks delivered by the DAW.

In addition, the scene viewer GUI can be used to set the initial position of the different objects.

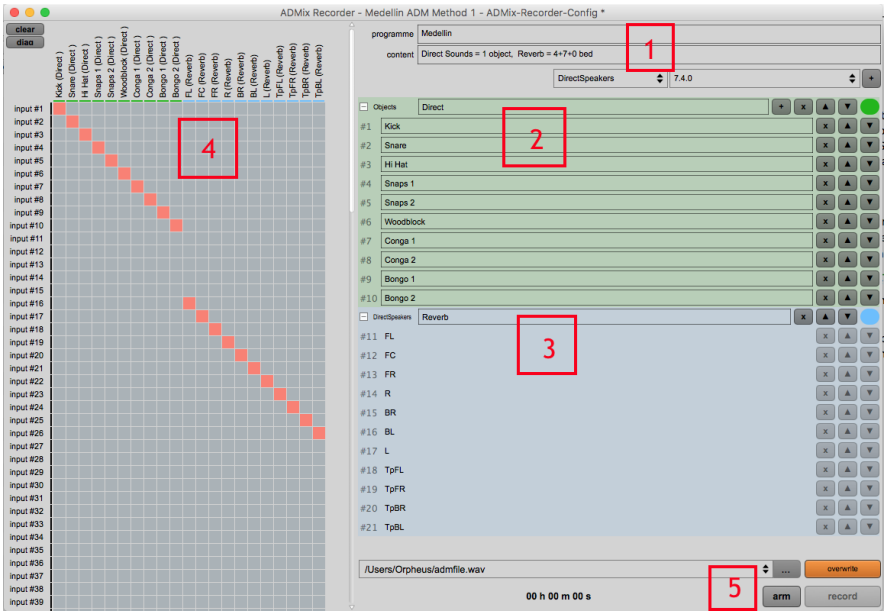

Figure 7: Metadata editor and routing matrix of the ADMix Recorder

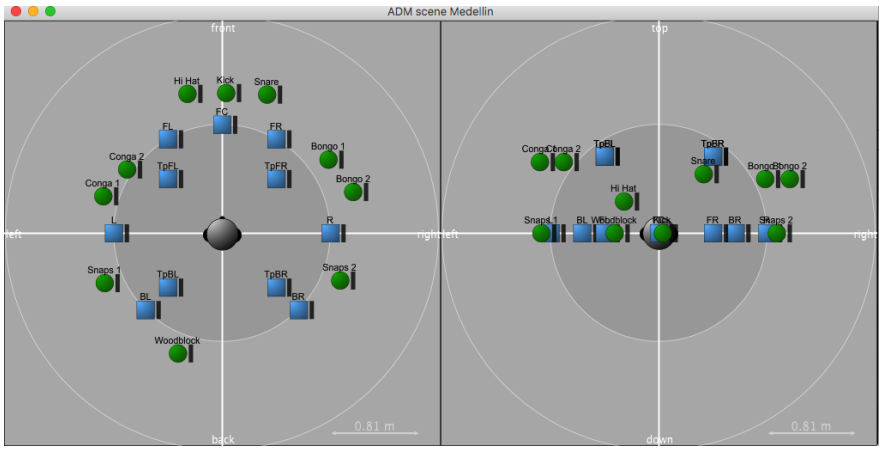

Figure 8: Scene viewer used to set the position of the objects.

Dynamic metadata

The DAW can be used to record and edit any arbitrary parameter using the above-described ToscA plugins. Such parameters can be for instance the position and the gain of the objects. The Upper-left part of Figure 9 depicts the XML configuration file that is used to define the parameters that are transmitted by the ToscA plugin. In this example, the parameters are the position (in Cartesian coordinates) and the gain of the objects. The right part of Figure 9 is a screenshot of the DAW window that shows the list of automation parameters in the track where the ToscA plugin has been instantiated. Once all automation data is written / edited, the DAW can play back the audio tracks and associated automation data that are recorded by the ADMix Recorder to create the ADM file.

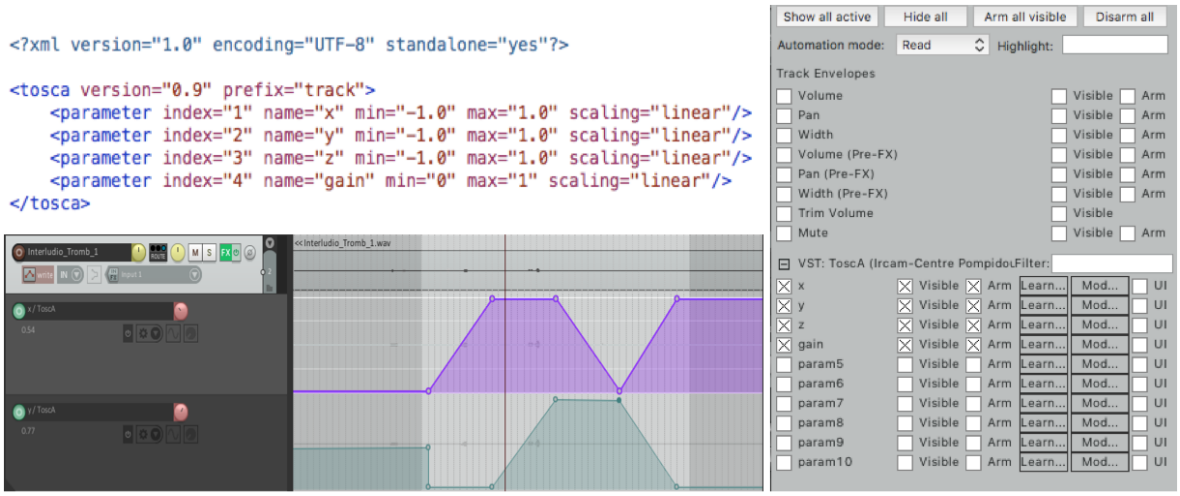

Figure 9: [Upper-left] Configuration file of the ToscA plugin used to define the automation parameters (Cartesian position and the gain of an object). [Right] View of the list of automation parameters in the corresponding track of the DAW. [Lower-left] A view of the x and y automation tracks.

2.3.2 Current Limitations

The modular structure of the ADMix tools has some limitations. In its current state the ADMix Recorder only supports single programme audio content and “flat structures”, i.e. with no nested objects. Moreover, the ADMix Recorder cannot distinguish in between different objects that were recorded as separate audio clips in one single audio track of the DAW. It means that an object is defined for the whole duration of the audio file.

Similarly, the ADMix Recorder cannot be aware of automation envelope breakpoints. The automation data are polled with a given time grain, which will increase the number of AudioBlockFormats recorded in the produced ADM files, compared to a solution where the ADM file would be directly exported from the DAW. For instance, according to the ADM recommendations,

the automation example of Figure 8 would need only four AudioBlockFormats, defining the successive target positions of the source. By default, any compliant ADM renderer will interpolate linearly the position of the source between these different breakpoints. In contrast, using the ADMix recorder, the automation of the source will be polled with a fixed time grain. Post-processing of the produced ADM file could however be envisioned in order to flatten the automation curves.

In its current state, the automation mechanism is only used for the position and gain of the objects. If necessary, other parameters such as the importance level, On/Off interaction flag, start and end of an object, etc. could be easily supported.

The above limitations only apply for the ADMix Recorder. The ADMix Player already supports multiple programmes, nested objects, on/off interaction flags used for interactive muting of the different objects.

2.3.3 Comments on the ADM format

The development of the ADMix tools revealed some needs of clarification or extension of the ADM standard. For instance, the way to deal with object interaction in the case of nested objects was not clearly defined. This motivated the proposition of a clarification in the ADM Usage Guidelines.

In the current state of the ADM recommendations, the default values of some parameters are missing, which could lead to some inconsistencies.

The ADMix tools have been used to experiment on different ways to integrate reverberation beds in an ADM file. This work could lead to some Usage guidelines in order to organise audio packs conveying the room reverberation shared among several sources, with or without separate tracks for first reflexions, etc. Similarly they suggested some extension propositions about HOA and Matrix objects. These points have been discussed in an ADM workshop organised during the project, see section 5.

2.4 In the ADM Pre-Processor

The purpose of the ADM Pre-Processor is to transform one ADM-based representation of an audio scene into another ADM-based representation. A typical use case for this is to reduce the number of elements to render – and therefore the required bandwidth – of an object-based audio scene. Besides reducing complexity, such transformations can also help to foster interoperability. Given the multitude of potential target formats, devices and platforms, it may be a reasonable approach to transform an object-based production into a representation optimized for specific target applications.

The most important aspect in this context is that the creative intent of the content producers is preserved and that the user experience is not degraded. To achieve this, the Audio Definition Model contains a number of features that help to convey this creative intent and can guide automated processes in determining optimal transformations under given constraints.

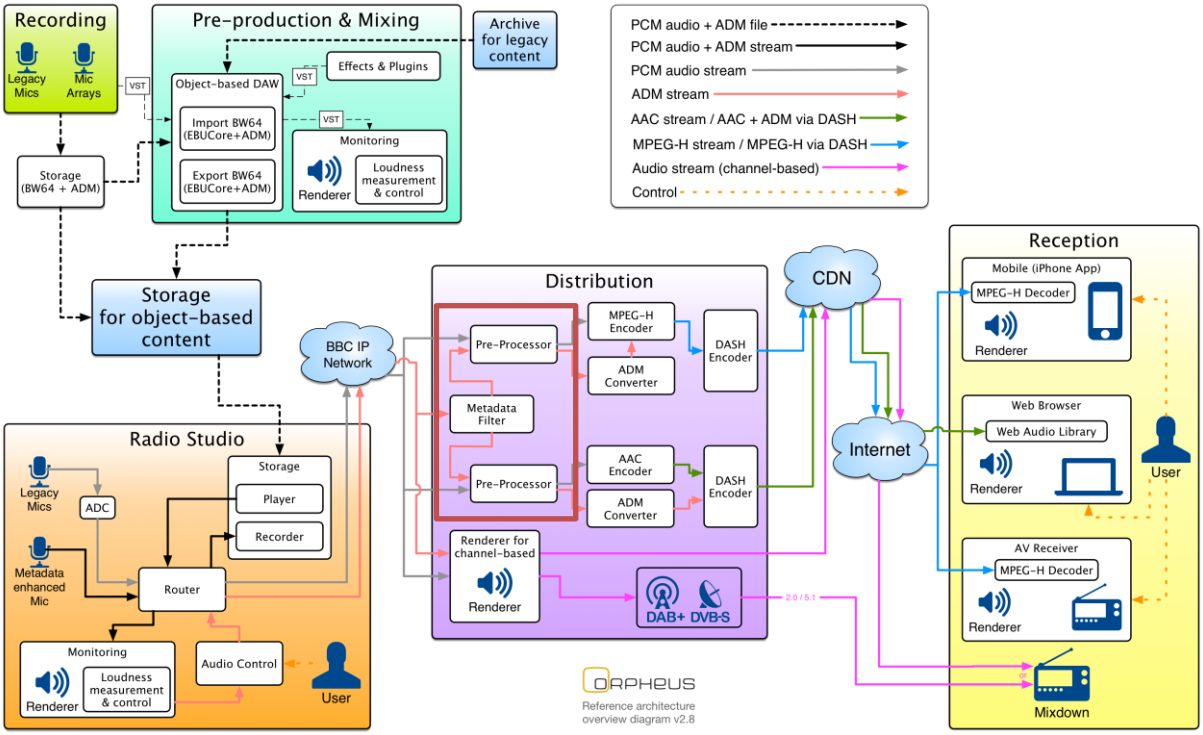

Figure 10: Pre-Processor within the reference architecture

Grouping and nesting of elements in an ADM structure is one of the most powerful features to attach semantic information to an otherwise loose collection of elements. By referencing other ADM elements of the same type, a hierarchy can be created. Especially two elements in the Audio Definition Model can be used to create such a hierarchy: audioObjects and audioPackFormats. audioObjects can be used to group elements that belong to the same content. An example for this can be seen in Figure 11, where direct sound (AudioObject 3) and reverb (AudioObject 4) objects have been grouped by a parent object “Music” (AudioObject 2). Grouping by audioPackFormat elements, on the other hand, is used to combine elements that are related on a technical level. This can be seen in Figure 11 that all parts of an ambience recording that may have been created by multiple microphones, are referenced by the same audioPackFormat (AudioPackFormat 4). This multi-lingual scene features two audioProgrammes with German and English speech. A musical recording is shared by both programs. Furthermore, the speech items (green) have been marked with the highest importance values.

The Audio Definition Model also features metadata that provides a **description of the content itself and its type**. Examples for this include the description of speech, non-speech or mixed content, which may be used by a transformation process to estimate the impact of an operation to determine an optimal solution. This is also closely related to the description of **interactive** elements in an audio scene, as this information has a severe impact on the possible combinations and complexity reductions if interactivity shall still be possible afterwards.

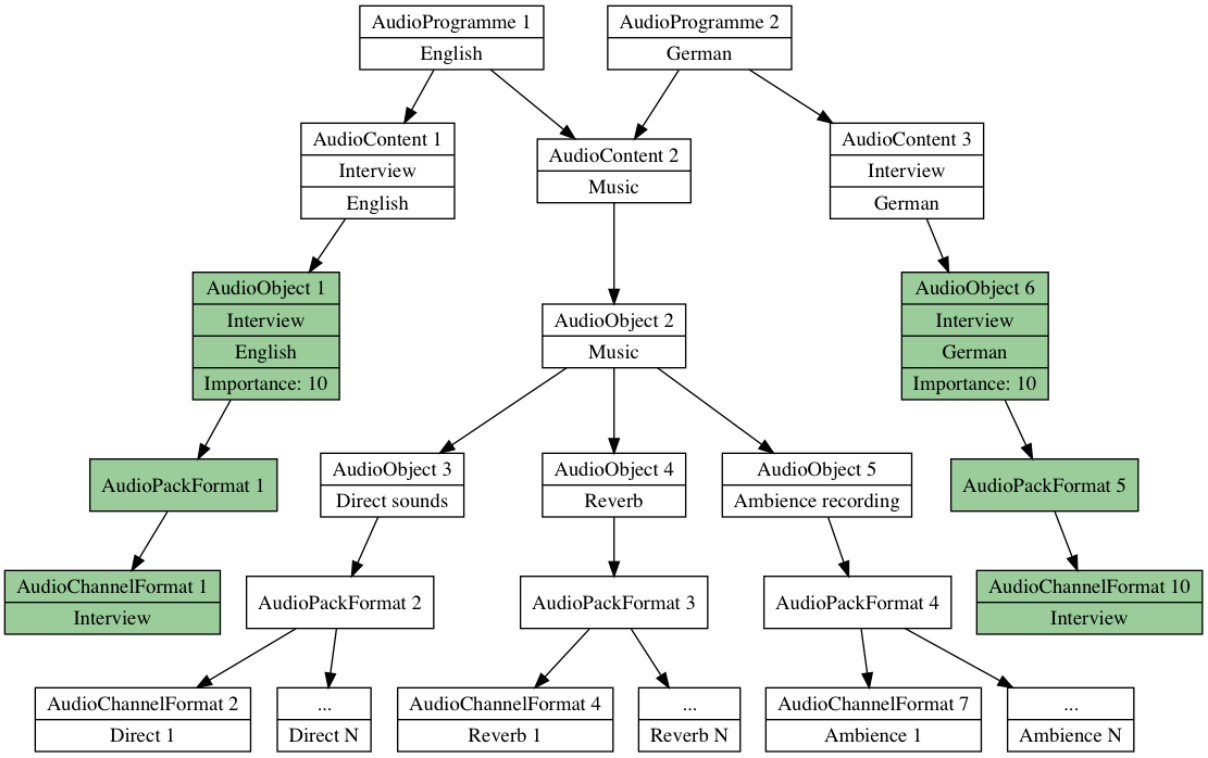

Figure 11: Slightly simplified ADM structure of an audio scene that can be used as input to the pre-processor.

Another very powerful mechanism is the use of **importance values**, which can be set on the audioObject level as well as on the audioPackFormatLevel or even for audioChannelFormat elements of type “object”. The general idea is that elements with a higher importance shall be preferred over elements which are less important. This could, in an extreme case, mean to discard elements beyond a certain importance threshold. In practice, however, it is more useful to use this information to decide, for example, which objects might be combined into a pre-rendered channel bed. Elements above an estimated importance threshold would be left untouched.

While this can only provide a limited overview of the possibilities, it becomes clear that the Audio Definition Model provides many features and additional information that can be used to adapt and optimize content to certain constraints and target applications if applied judiciously.

Yet, the flexibility and rich feature set of the Audio Definition Model comes with the expense that some care must be taken when analysing an audio scene. One particular implementation challenge steams from the fact that elements can be referenced multiple times. This can be due to fact that some content is shared between otherwise independent audioProgrammes, which must be taken into account when pre-processing this scene. Another possibility is that elements, namely audioChannelFormats, can be referenced multiple times through various combinations of audioProgrammes, audioObjects and audioPackFormats, which can lead to the same element being used multiple times with different start times, offsets or group relationships.

It is apparent that supporting all degrees of freedom the Audio Definition Model offers can be quite challenging. Nevertheless, this flexibility enables a rich feature set to encapsulate the creative intent of an audio scene. By employing modern data structures and algorithms, most of the complexity and challenges can be handled efficiently in an implementation. Given the chances and opportunities offered by the Audio Definition Model compared to other currently used formats, the initial learning and implementation challenges have to be considered worthwhile.

3 Experience with the Implementation of Provision Formats for Distribution to the End-user

3.1 Translation of ADM to MPEG-H

Within ORPHEUS there are two main metadata-models: ADM and MPEG-H. ADM is used in production and contribution, whereas MPEG-H is used in distribution to the end-user devices. Though the basic concepts of both metadata-models are similar in nature, the exact syntax and semantic can differ widely and one model may use specific elements which cannot be expressed in the other, which makes translation difficult.

The flexibility of ADM can become a problem for interoperability as ADM-files generated by one tool could be interpreted differently by another one or, in the worst case, not at all. Having recognized this problem, ORPHEUS partners have now started an effort at ITU-R to define more constraint "profiles" of ADM. FHG proposed for example the "Broadcast Profile". ADM files complying with this profile should also be a subset of the MPEG-H metadata model such that conversion is facilitated, see Figure 12. As the ADM Profiles are still under discussion in ITU-R, OPRHEUS has taken the pragmatic approach of "ADM Templates". An ADM Template is a specific example file from which it is known that it can be converted automatically into MPEG-H. If the particular structure and constraints of the ADM template are followed, it can be used as an ingest format to the MPEG-H encoder. The relation of ADM, MPEG-H, Broadcast Profile and Template is illustrated in
Figure 12.

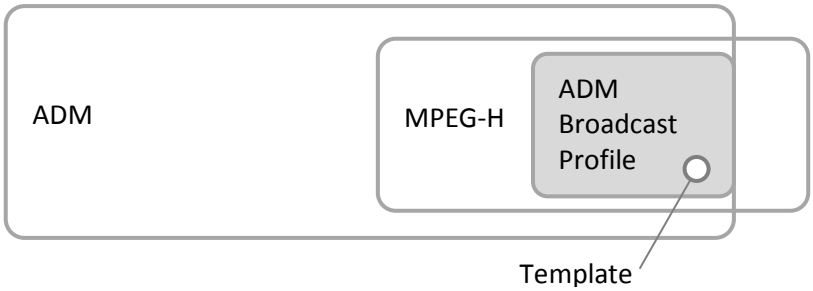

Figure 12: Relation between ADM and MPEG-H metadata

The first ADM Template which has been defined by Orpheus addresses the use case of "Dynamic Objects" in which the audio scene is composed exclusively by dynamic audio objects (no channel bed, no HOA, no interactivity, etc.). More specifically, the content includes two objects, a speaker moving from left to right while his volume is reduced and a "humming bee" which is moving into the opposite direction while increasing its volume. Position and gain are controlled through dynamic object metadata. The content is designed such that correct translation and rendering can be verified by listening to a) the ADM file as directly rendered to stereo, and b) the translated, encoded, and decoded MPEG-H file. The processing chain for the conversion is illustrated in Figure 13. The files highlighted with a blue circle will be published as a test vectors for 3rd party developers and content creators. More ADM Templates for other use cases will follow and help to generate ADM content that can be translated into MPEG-H in a predictable way.

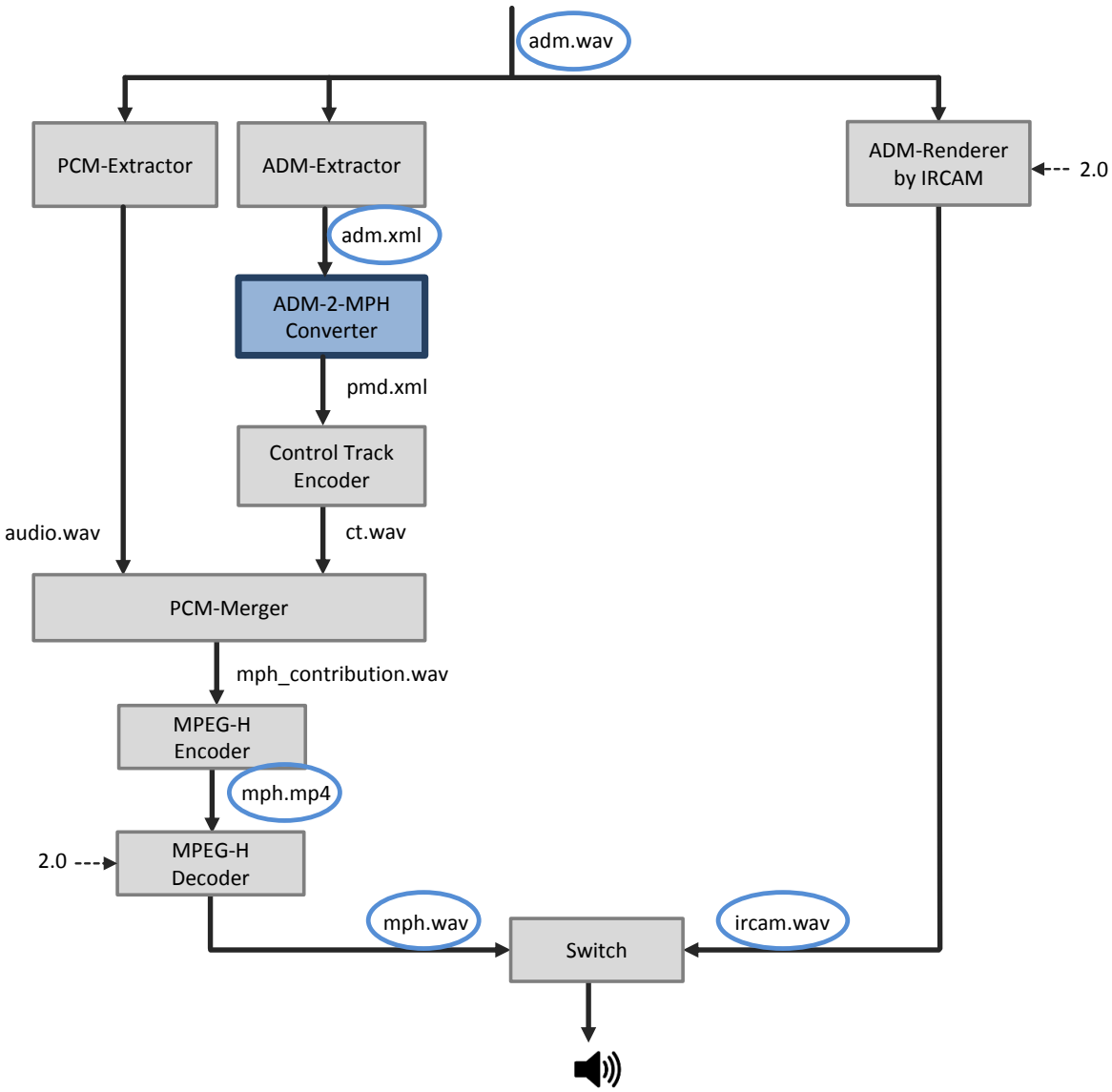

Figure 13: Processing chain for translating ADM to MPEG-H.

3.2 Formats used in iPhone App

The ORPHEUS Radio iOS app uses the MPEG-H codec for all audio and a large part of its metadata. MPEG-H supports multiple audio objects in a single stream or file and uses highly efficient bitrate reduction. The metadata carried over MPEG-H as used in the app are tightly coupled to the actual rendering of the objects: objects can be toggled on/off, positioned in 3D space and their prominence adjusted.

In addition to the MPEG-H stream, a JSON format was developed to contain additional metadata, which is out of scope of the current MPEG-H or ADM standards. This metadata contains information such as live transcript of text, accompanying images, user interface specifications and timed markers. As the goal in ORPHEUS was not to develop a new metadata standard but rather to investigate the usefulness of various metadata in the context of object-based broadcasting, JSON was chosen as a ubiquitous container format and the JSON protocol has been kept as simple as possible.

All streaming in the iOS app happens over MPEG-DASH. Both MPEG-H (as mp4 fragments) and JSON can be easily streamed over DASH and it is possible to jump in time, as required for the app.

3.3 Formats used in AVR

The Trinnov Altitude AVR had to be enhanced with both, the capability of receiving a MPEG-DASH stream and decoding the included MPEG-H audio. The MPEG-H decoder and renderer library was provided by FHG. It was first build for the AVR architecture and tested offline, then integrated into the AVR hardware. The format handling allowed to automatize the building of a UI for each stream, in accordance to the content, and to allow the user to control and interact with it.

For the DASH streaming capability, a modified version of the free GStreamer media framework is used by TRI, while 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 of the targeted loudspeaker configurations.

3.4 Formats used in Browser App

In order to make 'The Mermaids Tears' production available as a public, interactive stream, it was necessary to use a non-standard approach to distribution and rendering. A method has been developed to allow multiple audio objects to be streamed to the browser along with associated metadata in the Serialised ADM format (see 3.4.1). This does not require any 3rd-party plugins or decoders to be present in the user's browser.

The approach uses the Web Audio API and MPEG-DASH to provide data to the browser. JavaScript audio libraries developed by BBC R&D are then used to schedule the playback and render the audio based on the incoming metadata. Figure 14 presents an overview of DASH reception and browser-based rendering showing dashSourceNode to retrieve packets for forwarding to renderers using the WebAudio API, with user interaction via the web browser.

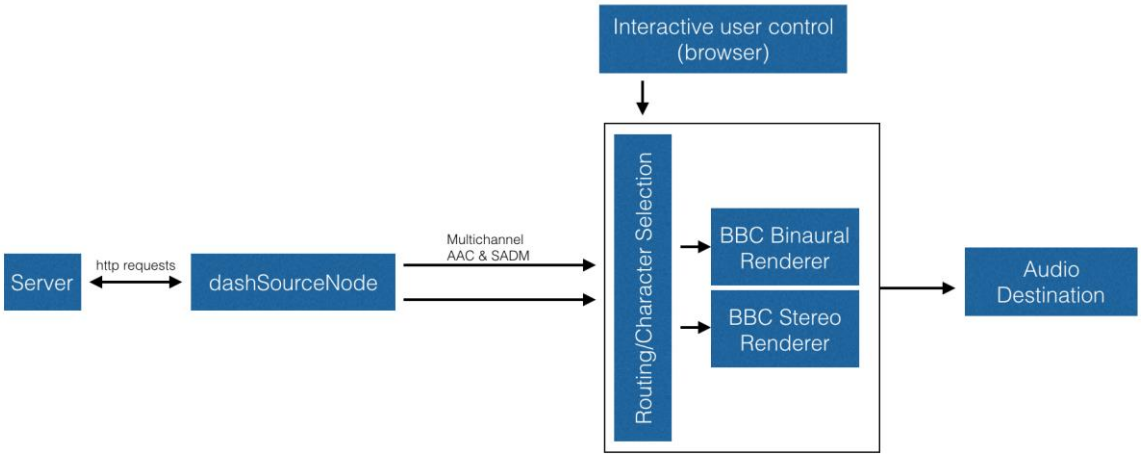

Figure 14: Overview of DASH reception and browser-based rendering

User interactions such as character selection and output format can be used to control the routing of audio objects. Objects can be active or inactive and their position set according to the metadata.

In order to prepare assets for this approach the multichannel PCM audio is split into a set of 5-channel files. These 5-channel subsets are then encoded to AAC.

The compressed files are split into segments to allow them to be requested sequentially by the client. Serialised ADM metadata is also segmented into XML chunks to be requested along-side the audio segments. Once decoded in the browser, audio and metadata segments are scheduled for playback. The audio is routed to a renderer based on the user choice. The location and levels of the

objects are determined by the metadata.

3.4.1 Serial ADM for Playout

As the specification of the S-ADM is still undergoing final refinements in the ITU-R, it was decided to keep the scope of the parameters used to a sensible minimum to ensure the development of the libraries that generate and read the metadata was not over-complicated.

Therefore, the transport part of the S-ADM was kept to the simplest form where each frame is of the 'full' type, so that they carry a complete set of ADM parameters (as opposed to the 'intermediate' type which only carries ADM parameters that have changed since the previous frame). The frame size was set to a duration of 1 second to match that used for the DASH playout frame size.

The ADM metadata was also kept to a very simple profile which only used the 'Objects' type, and the only parameter used in the audioBlockFormat was position (azimuth, elevation, and distance). The audio objects were also static, so there was no added complexity of dealing with dynamic (i.e. moving) objects. To enable the interactivity of choosing one of the three stories, two layers of audioObjects were used. The top layer consists of three story audioObjects, which were complementary to each other (i.e. only one could the played at a time). The second layer consisted of each of the sound effects and microphone recordings from each room. These audioObjects were all time-limited, so were only active at the time they were played.

As a maximum of 15 audio tracks were allowed (to ensure simple interfacing with the MPEG-H encoder profile), audioObjects that did not overlap in time were multiplexed into a single track. While this was done manually for this test, ideally it would be automated.

4 Implementation of Metadata Formats for Non-linear Reproduction

4.1 Implementation in Sequoia

For pilot phase 2 of the project the ORPHEUS team decided to implement and demonstrate an important interactive use case: Programs with a variable length. In practice this means, that, given a certain level of interest, the same program can be listened to at different degrees of content depth and therefore different lengths.

During production the content is to be sectioned and tagged to indicate which sections fit to the following levels of interest:

- Level 1: A very short “headline” version, containing only the core message
- Level 2: A slightly longer version including all key information
- Level 3: An even longer version with additional background information
- Level 4: A more complete version, adding further details
- Level 5: The full version including all details, associations, etc.

These descriptions won’t necessarily fit for any production, but they provide a sense of how the levels are expected to be used.

The task of the DAW is to provide the user with means of editing levels of interest. For Sequoia this was realized in the form of markers on the project timeline, which can be set from the “marker manager”. A marker indicates that all content between its location and the location of the next marker are part of a certain interest level. By using keyboard shortcuts, they can be added easily during playback. For an example with added markers see Figure 15.

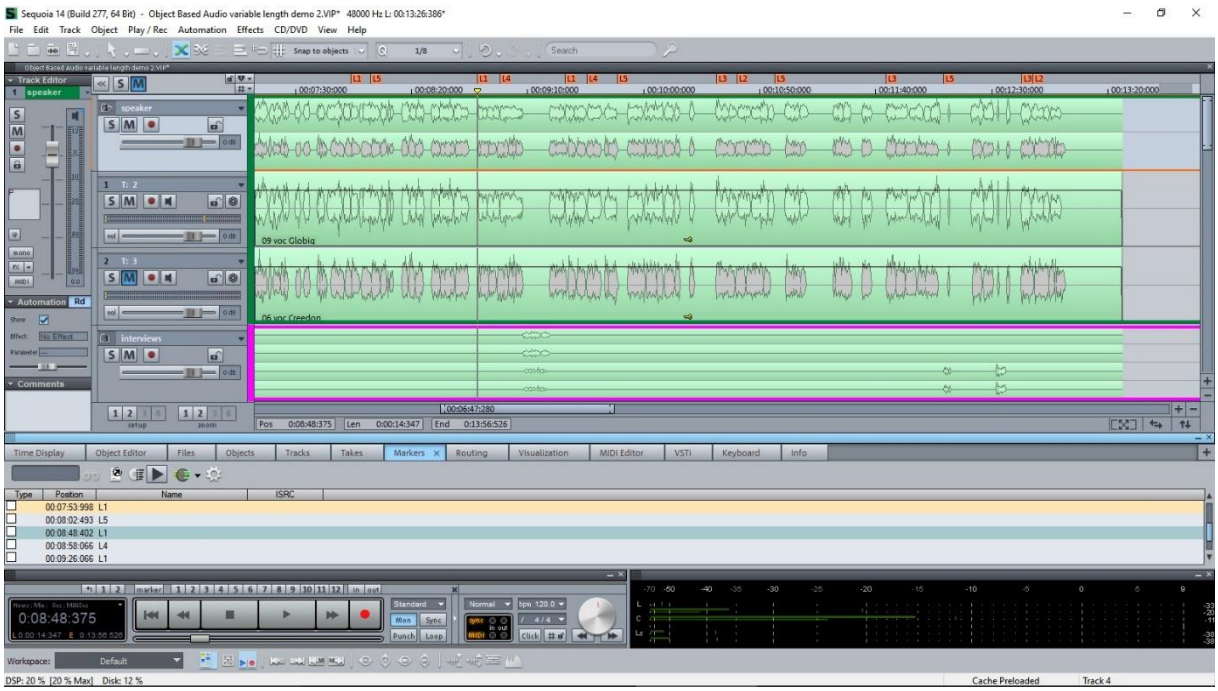

Figure 15: “Experience object-based audio” created by BR with added markers for interest level

In addition, a preview mode has been implemented. By selecting a marker of any interest level, only

the sections of that level and levels below it will be played. Sections marked with a higher interest level will be skipped.

For the ADM export these markers will be stored as time segments (part elements) in the metadata chunk of the BW64 file conformant to the EBUCore definition [3]. This information will then need to be transported further along the broadcast chain to eventually be used on the end-user device.

4.2 Implementation in iPhone App

From an end-user's point of view, non-linear reproduction consists of two concepts: the selection of which programs to play ('on-demand') and the decision of which parts to play from these programs ('variable length'). Object-based broadcasting can be very well suited for non-linear reproduction, when the appropriate metadata is available. In the scope of the ORPHEUS project it was decided to limit non-linear playback to the interaction with a single program, as not enough content was available to meaningfully experiment across multiple programs.

In the ORPHEUS Radio app this metadata takes the form of timed markers that contain information about topics or subjects as well as relative importance levels of a section (clip). This means that sections can be selected for playback based on criteria like "matches the listener's interests" and "matches the listener's desired level of depth". The times markers are simple JSON objects that can be derived from EBUCore metadata, such as produced in Sequoia. This JSON can be streamed over MPEG-DASH in parallel to the MPEG-H audio stream.

5 ADM Expert Workshop

The Audio Definition Model ADM is a very flexible model to describe all kinds of audio content, yet there is not much guidance on how many object-based audio use cases should be handled. Being a fairly new standard, implementers, developers and users are still about to develop a common understanding of how to use the ADM efficiently. The very same use case can potentially be modelled differently using the ADM. During the course of the ORPHEUS project the consortium members gained significant knowledge and experience on applying the Audio Definition Model to the use cases and productions at hand.

It became clear that an extensive exchange not only within the project, but also beyond the consortium with other implementers, end users and adopters, is vital for the advancement of the ADM. A first step to achieve this was to organize an “internal ADM Expert Workshop”.

The workshop was held in parallel with the consortium meeting in Heidelberg from January 16th to 18th 2018. The focus of the workshop was to discuss specific use cases and establish a common understanding on how they are supposed to be modelled. Even though the focus should be on structures, which are realisable right now with the current version of the ADM, the discussion was explicitly open for extensions to the standard. Additionally, issues the partners encountered in the course of working with the ADM, which are not related to any specific use case, should be discussed.

In order to avoid too abstract discussions, participants and interested parties where asked to suggest real-world use cases, questions or problems that have been encountered during their work with object-based audio so far. These included both scenarios that have been already used by the ORPHEUS project and future applications that we’re considered useful and interesting by the participants.

#	Use case	Short description
1	Foreground / Background balance	The user is able to change the balance between foreground and background of the scene.
2	Multiple languages	The user is able to select the target language out of a list of available languages.
3	Binaural content	Use of pre-produced binaural content / dummy head recordings.
4	Multiple Storylines	The user can select different perspectives on the story. An example is the “Mermaid’s Tears” production.
5	Voice over (language + audio description)	The user can activate an additional audio “overlay”, e.g. a translation of the original. Ducking of the original content should be employed to increase speech intelligibility.
6	Hierarchical structure of reverb	Provide a semantic description of the relations between direct sound and reverberation using ADM referencing.
7	Diffuse reverberation	Reduce the amount of required diffuse reverberation channels by applying the ‘diffuse’ attribute to a set of N channels.
8	In-head localization	Intended in-head localisation of some parts of a binaural production can be an important dramatic element.
9	EBU metadata & ADM element connections	The EBUCore metadata set provides a rich set of content metadata, e.g. specifying artist, composer, etc. The possibility to attach this metadata not only to a whole file, but a single element of an ADM scene would be useful. This could be used, for example, to identify a piece of music that is part of the audio scene.

10	Generalized matrix encoding/decoding	Use the “matrix” type of the ADM to express all kinds of matrix encoding/decoding related applications. Examples might include custom down-mix matrices or HOA decoding matrices.
11	Head-locked objects	Within an VR environment, where binaural reproduction with head tracking is used, using elements that are not affected by head tracking can be desirable.
12	Manipulation of HOA content	A global rotation parameter for HOA objects might be useful. This parameter could be open or not to user interaction. Other manipulation such as focus or ‘blur’ could also be interesting.
13	Convolution/filtering	Specifying filtering/convolution to be applied to one or several channels of a pack with an associated set of impulse responses or filter coefficients. Specifying the impulse response would provide some guarantee of the rendering result.

Table 2: List of use cases of the ADM expert workshop with slightly simplified/shortened descriptions

Furthermore, topics that were not related to single use case have been collected before and during the workshop. This includes the interpretation, rendering and usage of “distance”, how to specify a “default” audioProgrammes, start times of nested audioObjects and the usage of importance values.

The key outcomes of this workshop can be roughly divided into two parts.

It is planned to publish the consolidated use cases and the detailed results of the discussions in a separate document. This will include detailed descriptions of the use cases and the proposed solutions, graphical representations and examples in ADM format. The focus of this document will be on solutions that can be implemented with the current version of the ADM standard.

The other part will be feed into the relevant ITU groups and consists of proposals for future extensions and (editorial) clarifications of the ADM recommendation itself as well as the usage guidelines [4].

The very fruitful ADM expert workshop will be repeated also beyond the lifetime of the project.

6 Conclusions

The ORPHEUS project's objective to set up, implement and validate pilot and reference architectures as well conducting pilots for object-based audio broadcasting has led to a comprehensive inventory of long-term established and emerging audio formats and metadata models.

Amongst this plethora we had to analyse, evaluate and select, which of 'ready available' or 'still emerging' solutions fit best not just within the basic conditions of our project, but also in the wide space of practical technical application and content requirements in the broadcasting eco-system.

There, a clear distinction between utmost quality internal professional production formats and excellent but economic public distribution and delivery formats is still necessary and required, and will remain for the foreseeable future.

The choices being taken here within the ORPHEUS project for both sides demonstrate a state-of-the-art solution, being even sort of a 'golden cut' in order to demonstrate best the forward looking perspectives and advantages of object-based audio technology.

References

[1] Carpentier, T., Noisternig, M., Warusfel, O. "Twenty Years of Ircam Spat: Looking Back, Looking Forward", 41st International Computer Music Conference (ICMC), 2015

[2] Carpentier, T. "ToscA: An OSC Communication Plugin for Object-Oriented Spatialization Authoring", 41st International Computer Music Conference (ICMC), 2015

[3] EBUcore Metadata Set: Tech 3293, October 2017, EBU. https://tech.ebu.ch/MetadataEbuCore

[4] Usage Guidelines for the Audio Definition Model and Multichannel Audio Files: https://www.itu.int/pub/R-REP-BS.2388-2-2017

[end of document]