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

D3.4: Implementation and documentation of a live object-based production environment

Version: v1.1

Deliverable type	DEM (Demonstrator, pilot, prototype, plan designs)
Dissemination level	PU (Public)
Due date	30/11/2016
Submission date	01/12/2016 (updated version 1.1 of 20/12/16)
Lead editor	Chris Baume (BBC)
Authors	Chris Baume (BBC), Andrew Mason (BBC), Peter Brightwell (BBC), Max Leonard (BBC), Matt Firth (BBC), Marius Vopel (MAGIX)
Reviewers	Andreas Silzle (FHG), Frank Melchior (BBC)
Work package, Task	WP3, T3.3
Keywords	object-based production environment

Abstract

This report of a demonstrator gives a brief overview on the live object-based production system being developed as part of the Orpheus project. The design of the fundamental technology that drives the system is discussed, and the details of the current implementation are described.

[End of abstract]

Document revision history

Version	Date	Description of change	List of contributor(s)
v0.1	3/11/16	Initial draft	Chris Baume (BBC), Peter Brightwell (BBC), Max Leonard (BBC), Matt Firth (BBC)
V0.2	14/11/16	Added more info on NMOS and explained implementation	Chris Baume (BBC), Andrew Mason (BBC)
V0.3	16/11/16	Changes merged to one document	Chris Baume (BBC)
V0.4	21/11/16	Responding to initial feedback, added contributions from Magix	Chris Baume (BBC), Marius Vopel (MAGIX)
V1.0	30/11/16	Included feedback from Andreas Silzle, Matt Firth and Frank Melchior	Chris Baume (BBC)
V1.1	12/12/16	Made suggested changes from Rob Wadge	Chris Baume (BBC), Robert Wadge (BBC)

Disclaimer

Copyright notice

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

Executive Summary

This report gives a brief overview on the live object-based production system being developed as part of the Orpheus project. The design of the fundamental technology that drives the system is discussed, and the details of the current implementation are described.

Table of Contents

List of Figures

Abbreviations

ADM	Audio Definition Model
AMWA	Advanced Media Workflow Association
API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
JSON	JavaScript Object Notation
JT-NM	Joint Taskforce for Networked Media
NMI	Networked Media Incubator
NMOS	Networked Media Open Specifications
REST	Representational state transfer
RTP	Real-time Transport Protocol
SDP	Session Description Protocol
UMCP	Universal Media Composition Protocol

1 Introduction

The ORPHEUS project aims to deliver an improved audio listening experience using object-based broadcasting. In order to be able to deliver such an experience, we must design and build a system capable of producing object-based audio.

ORPHEUS is divided into two pilots – the first being a live object-based audio broadcast. Live production comes with its own challenges, however many of the tools needed to build such a system are already available.

This document outlines the initial design and implementation of a live object-based radio studio.

1.1 Design pre-requisites

- Based on Internet Protocol (IP), using existing Internet standards where possible
- Build on and co-exist with existing industry specifications
- Use open specifications

1.2 Requirements document

The requirements for the studio are detailed in a previous ORPHEUS deliverable D3.1, entitled "Requirements, designs and workflows of an object-based production environment", available at http://orpheus-audio.eu.

2 Design

This section describes the current technical design of the Orpheus object-based production system. It is important to note that this design is still under development and incomplete, so should not be emulated at this early stage.

We start by discussing methods for creating a system which is interoperable with other similar broadcasting systems before diving into the specifics of how we plan to stream and store object-based audio.

2.1 Interoperability

One of the most important features of a broadcast system is that it should be capable of transmitting and receiving media and metadata using a protocol that makes it interoperable with a variety of equipment from different manufacturers and with other broadcasters. To make this as easy as possible, it should involve as little manual configuration as possible. This should be simple and flexible.

2.1.1 Existing specifications

The Joint Taskforce for Networked Media (JT-NM) is an "open, participatory environment, to help drive development of a packet-based network infrastructure for the professional media industry by bringing together manufacturers, broadcasters and industry organizations (standards bodies and trade associations) with the objective to create, store, transfer and stream professional media".

In September 2015, the JT-NM published its Networked Media Reference Architecture v1.0 . This foundation framework included an approach for interoperability of devices and services, including methods for identity, discovery and timing. The reference architecture document can be found at http://jt-nm.org/RA-1.0/.

The Advanced Media Workflow Association (AMWA) is an "open, community-driven forum, advancing business-driven solutions for Networked Media workflows", whose members include many influential broadcasting organisations. AMWA is also a sponsor of the JT-NM. More information can be found at http://www.amwa.tv/.

AMWA aims to "develops specifications and technologies to facilitate the deployment and operation of efficient media workflows". This is being realised through the Networked Media Incubator (NMI) project, which is creating designs and specifications to implement and test working systems. These specifications are designed to complement the JT-NM reference architecture. AMWA also run interoperability meetings to test that equipment works together as expected in real-world settings.

The Networked Media Open Specifications (NMOS) are a family of specifications, produced by the Networked Media Incubator (NMI) project by the Advanced Media Workflow Association (AMWA), related to networked media for professional applications. At the time of writing, NMOS includes specifications for:

- Stream identification and timing
- Discovery and registration
- Connection management

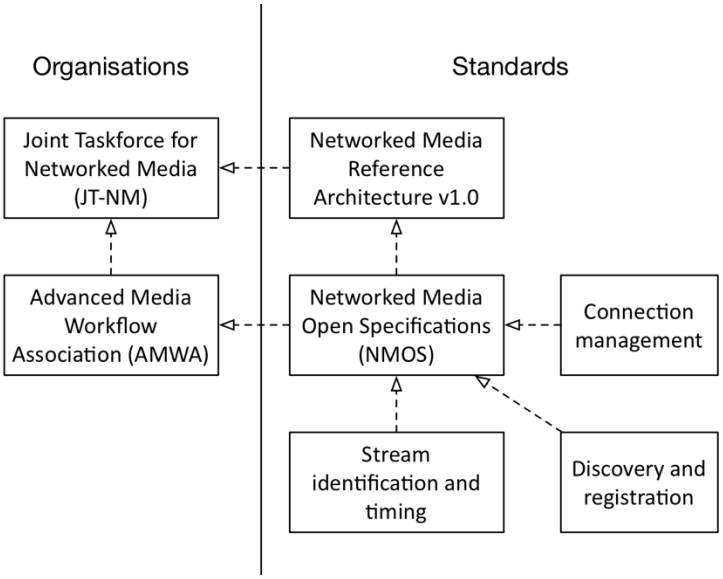

Figure 1: Org chart of relevant organisations and standards for networked media.

2.1.2 NMOS Data Model

NMOS uses a logical data model based on the JT-NM Reference Architecture to add identity, relationships and time-based information to content and broadcast equipment. Hierarchical relationships group related entities, with each entity having its own UUID (ID and UUID are used interchangeably in this document).

The traditional assumption of *"different connectors for different signals"* is replaced using logical interfaces on common network interfaces, exposing video, audio and data inputs and outputs, and control parameters. This makes communication easier and helps with virtualization of broadcast equipment, with multiple devices operating on a shared physical host.

Data modeling -- identifying the important parts of a system and the relationships between them -- has an essential role to play in specifying what goes over the logical interfaces and will be an essential part of the transition of broadcasting and media production infrastructure to make the most of IP. The NMOS data model is shown in Figure 2 and explained in the following sections.

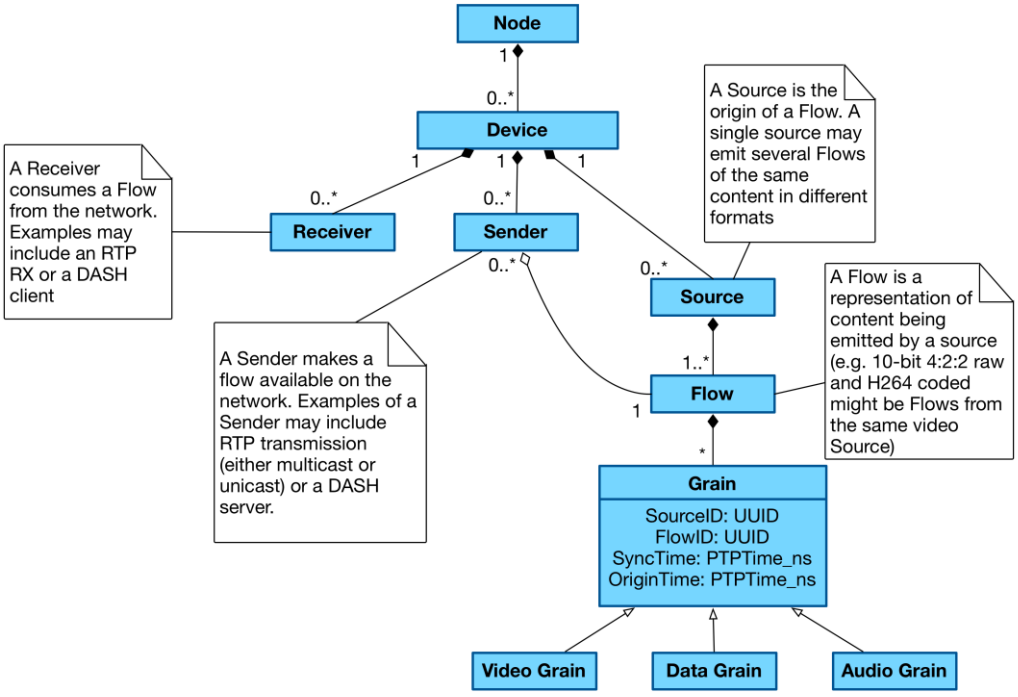

Figure 2: NMOS data model

2.1.2.1 Nodes

Nodes are logical hosts for processing and network operations. They may have a permanent physical presence, or may be created on demand, for example as a virtual machine in a cluster or cloud. Connections between Nodes through which content is transported are created to build a functioning broadcast plant.

Each Node represents the resources in its data model using a RESTful HTTP API called the **Node API**, which forms part of the Discovery and Registration Specification, and may also be used by future NMOS specifications. The Node API specification can be found at https://github.com/AMWA-TV/nmos-discovery-registration/blob/master/APIs/NodeAPI.raml. The API lists sources, flows, devices, senders and receivers, as well as providing a method to connect a receiver to a sender. These concepts are explained below.

2.1.2.2 Devices

Nodes provide **Devices**, which are logical groupings of functionality (such as processing capability, network inputs and outputs). In an audio context, a sound card would be an example of a device.

2.1.2.3 Sources, Flows and Grains

Devices with the capability to originate content must provide an abstract logical point of origin for this, called a **Source**. A Device may have multiple Sources.

Sources are the origin of one or more **Flows**, which are concrete representations of content. Flows contain format, label and source ID and flow ID.

Flows are composed of sequences of **Grains**. A Grain represents an element of Essence or other data associated with a specific time, such as a group of consecutive audio samples. The size of Grains is configurable, so can be adapted to best suit the given application. Typically for audio-visual content, the size of a Grain is configured to store a duration of one video frame (e.g. 1/25 second). For audio-only content, this could be made smaller to reduce the latency.

Grains are a wrapper, don't prescribe what the payload is, but allow tracking within NMOS system of where, when and what it came from.

Grains also contain metadata information that specifies attributes such as the temporal duration of the payload, useful timestamps, originating Source ID and the Flow ID the grain is associated with.

As an example, consider the above concepts in the context of a video camera with an on-board microphone:

- The camera itself is a Node, which provides a single Device with two Sources (one audio Source, one video Source).
- The audio Source provides a single audio Flow.
- The video Source provides (for the purposes of this example) two video Flows, one uncompressed, the other mezzanine encoded.

2.1.2.4 Senders and receivers

Devices transmit and receive Flows over the network using **Senders** and **Receivers.** These can be respectively considered as "virtual output ports" and "virtual input ports" on the Device. Receivers connect to Senders in order to transport content.

Through the Node API, Senders provide an ID, a flow ID and a manifest HREF (provided as a Session Description Protocol, or SDP, file). Receivers provide a list of their current subscriptions, which are

the Senders they are receiving data from.

2.1.3 Identification, synchronization and transport

Each Grain contains Flow and Source UUIDs as well as other attributes such as an **Origin Timestamp** and **Synchronization Timestamp** (see Section 2.1.3.2).

2.1.3.1 Transport

How this data travels between Nodes depends on the transport protocol being used. For each transport, the NMOS data model must be mapped into appropriate places in the protocol. Senders and receivers can support any transport type. A standard means for carrying additional Grain metadata should be defined.

As an example, RTP is typically used for real-time transport. RTP provides customizable **header extensions** into which the NMOS data model fields are inserted. The In-stream Identity and Timing Specification specifies how this is done.

SDP provides standard means of identifying how an RTP stream should be interpreted. SDP provides mappings between header extension IDs and their format. Provides mappings to the payload format and RTP timestamp clock reference (as TR-03, AES67 etc.)

2.1.3.2 Timing model

The NMOS Timing model has been designed to be used with a clock derived from PTP in order to synchronize Devices within a facility. By pairing this clock with a GPS source, absolute timing accuracy between geographically distributed locations can be ensured. As per SMPTE ST 2059-1, an epoch of 1970-01-01T00:00:00TAI is used, with zero phase at the epoch for periodic signals.

The capture of essence such as video frames is governed by this clock, providing frame synchronization between discrete Devices. Each Grain is timestamped using the PTP clock in order to provide the means for synchronization of Flows where required, and unique identity for Grains in time. Two timestamps are used for each Grain:

- **Origin Timestamps** provide the original sampling instant of the media at the edge of the system, uniquely referenceable for all time. When capturing from a live source this should match the Sync Timestamp.
- **Synchronization Timestamps** provide a means to cross-reference between Flows which may have passed through the network fabric via different paths, or passed through processing chains (such as codecs) which impose different levels of delay. Two coincident audio Grains would share the same Sync Timestamp, which remains associated with them as they pass through processing devices.

By pairing these timestamps with Flow identifiers, it is possible to track an audio frame or other essence through its chain of ancestors right back to the Device which originally captured it and the instant in time when that occurred.

2.1.4 Registration and discovery

The Registration and Discovery Specification allows nodes to register themselves and their capabilities, and to discover other available nodes and their capabilities. The specification is designed to use existing Internet standards which allow it to be able to operate at various scales and across network boundaries. This means that the same implementation can be used for a local testbed, or scaled to operate over numerous locations across the world.

Two mechanisms for discovery of Nodes and their resources are described: **peer-to-peer** and **registered**. Although it is intended that only one is used at a time, these two mechanisms

may co-exist if this is operationally useful.

2.1.4.1 Peer-to-Peer Discovery

This mechanism is intended for small or temporary deployments, such as connecting a camera to a display. Peer-to-peer (P2P) discovery requires no additional infrastructure. Nodes make DNS Service Discovery (DNS-SD) announcements regarding the presence of their Node API. Peers browse for appropriate DNS records and then query the Node API for further information.

2.1.4.2 Registered Model

This mechanism can be used at scale by distributing registry across multiple instances, possibly across multiple subnets. It can also be used with HTTP load balancing. Either Multicast DNS or Unicast DNS can be used by the Nodes to announce their presence.

Registered discovery takes place using a **Registration & Discovery System (RDS)**, which is designed to be modular and distributed. An RDS is composed of one or more **Registry & Discovery Instances (RDIs)**. An example configuration of three RDIs is shown in Figure 3. Each RDI provides:

- A **Registration Service**
- A **Query Service**
- A **Registry** storage backend.

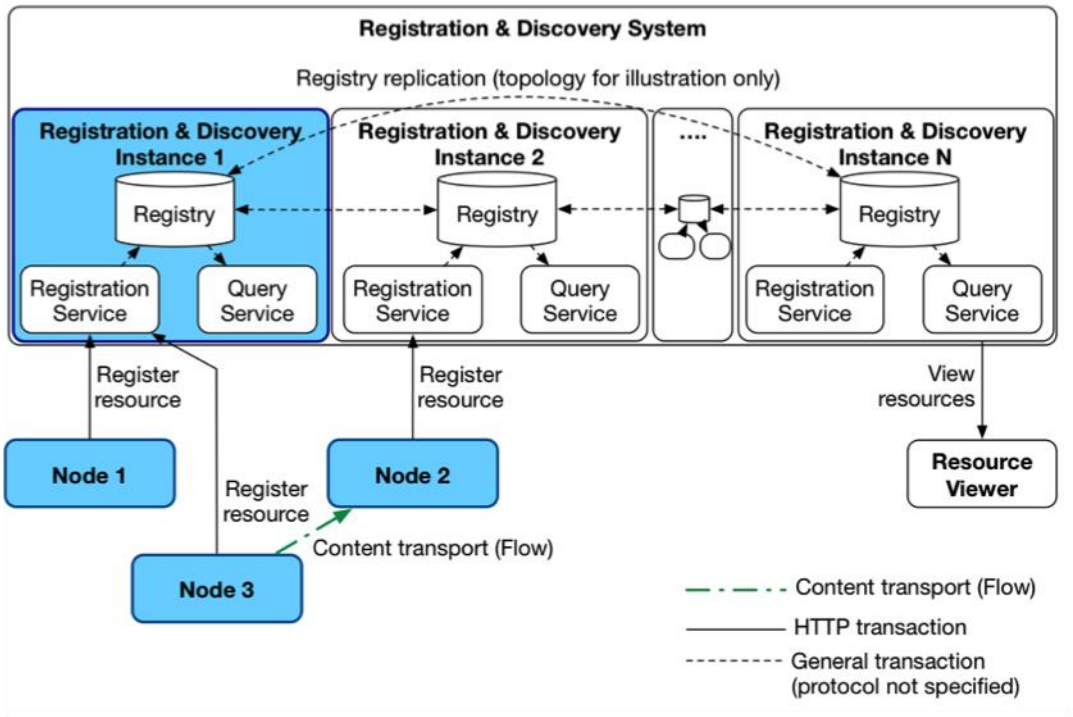

Figure 3: Diagram of three registration and discovery instances configured using the registered model. Nodes can register themselves using any of the instances, and are then visible to all other nodes. Content is transported directly between Nodes.

The Registration Service implements the Registration API of the NMOS Discovery and Registration Specification. Nodes POST to this API to register themselves and their resources. The Registration Service also manages garbage collection of Nodes and their resources by requiring Nodes to send regular keep-alive/heartbeat messages. Nodes that do not provide such an updates are removed.

The Query Service implements the Query API of the NMOS Discovery and Registration Specification. Clients can GET lists of resources from this API. Typical usage examples include:

- Obtaining a list of registered Nodes in order to drive a configuration interface.
- Obtaining a list of Sender resources and a list of Receiver resources in order to provide a connection management interface.

The Query API also provides the ability to generate 'long lived' queries using its Subscription mechanism and WebSockets.

2.1.5 Connection management

Sender and Receiver resources enable connectivity between Nodes. Senders expose a description of what Flow they are sending and an href to access it (the 'manifest').

For RTP, we use **SDP** files as our 'manifest'. Senders expose an SDP file that describes the location of the Sender's RTP stream. *Note: other information in this SDP file is also used by the In-stream Identity and Timing Specification, which includes example SDP files.*

To form a connection between two Nodes, a connection manager PUTs some structured data about a Sender (including a reference to the manifest href) to a Receiver. The Receiver parses the manifest and begins accessing the stream.

Each time a Receiver changes the Sender it is receiving from, the Node resource describing the Receiver must be updated on the Node, and also the Node must notify the Registration & Discovery system. In a registered model set-up, the Query API is used to find the current system state. Receivers must update their current state via the Registration API to the Query API.

2.2 Streaming object-based audio

Audio is characterised as 'object-based' when an audio essence is combined with metadata which describes that audio. Whilst existing broadcast systems need only be concerned about the transport of the audio essence, an object-based broadcast system must carry audio and metadata in parallel, and maintain a permanent link between the two that is pervasive throughout the broadcast chain and delivery to the end-user.

2.2.1 Transport

NMOS does not specify which transport method should be used, however real-time transport protocol (**RTP**) is extensively used throughout the industry for delivering audio and video over IP networks. AMWA have drafted a specification of how to apply the content model using RTP header extensions to carry identity and timing information and signal Grain boundaries (Brightwell, 2016). We have chosen to follow this specification as it falls in line with common industry practice and is compatible with AES67.

The above specification does not define a payload format. Our choices for these are detailed below.

2.2.1.1 Audio payload

In order to maximize audio quality, broadcast audio is transported using linear PCM throughout production, and until as late a stage as possible through distribution and emission. Commonly, a sample rate of 48kHz and bit depth of 16 or 24 bits is used. As such, the payload format for the audio should support these formats.

RFC 3190 is an IETF standard for the transport of linear PCM over RTP packets. We chose to use this as our audio payload format, running at 48kHz and 24-bit.

2.2.1.2 Metadata payload

In an object-based production system, data is equally important to audio and should be transported in a similar fashion. The data payload needs to be capable of handling structured data in a format that is easily readable and widely utilized.

RFC 7159 is an IETF standard which describes **JSON** (JavaScript Object Notation) - a lightweight data-interchange format, which is easy for both humans and machines to read and write. It is programming language independent, but is used extensively in web-based applications. JSON consists of two structures: (a) a collection of name/value pairs, and (b) an ordered list of values.

The RTP payload is made up a JSON string.

2.2.2 Linking audio and metadata

Object-based audio consists of atomized audio content combined with metadata that describes its composition. Our selected transport mechanism allows us to transmit audio and data over an IP network, however the two are sent separately and are not linked in any way. In order to be able to transmit object-based audio, we need to define a protocol that allows us to describe how the individual audio and data streams should be combined together.

2.2.2.1 UMCP

Universal Media Composition Protocol (UMCP) provides the means to describe arrangements of media and processing pipelines with the aim of enabling scalable object based media experiences, described through a common language. The protocol is currently being developed by BBC R&D, with the intention to propose it as an open standard.

UMCP is a method of describing a **composition** that is made up of a series of **sequences**, which contain **events**. These events define interconnected graphs of **processors** and dynamic modification of parameters that change the behavior of those processors.

Each event is described by a block of JSON, containing some mandatory fields. All the sequences in a UMCP composition sit on a coincident timeline, much like in a traditional non-linear editor, with the events on these timelines appearing based on their timestamp. The UMCP makes heavy use of the NMOS timing model and grain structure.

2.2.2.2 Mapping to ADM to UMCP

The Audio Definition Model (ADM, see Section 2.3.1) is a standardized data model which can be used to describe object-based audio. Below we outline how ADM metadata can be mapped to the UMCP.

- A UMCP composition can be directly related to an ADM audioProgramme.
- This UMCP composition could then contain several sequences (representing the ADM audioContents), each with a number of events referencing a source_id that resolves to another UMCP composition, which represent the ADM audioObjects.
- These sub-compositions ADM object representations contain a sequence for the media referred to by the ADM audio pack, consisting of a single event referencing an NMOS Source, and a sequence for every type of processing that is represented in the ADM audio blocks (one for panning, one for gain, one for reverberation etc.), which will have several events, each corresponding to each ADM audio block.

A diagram showing an example of how the ADM is mapped to UMCP is shown in Figure 4. The full algorithm for mapping ADM to UMCP is included in Appendix A.

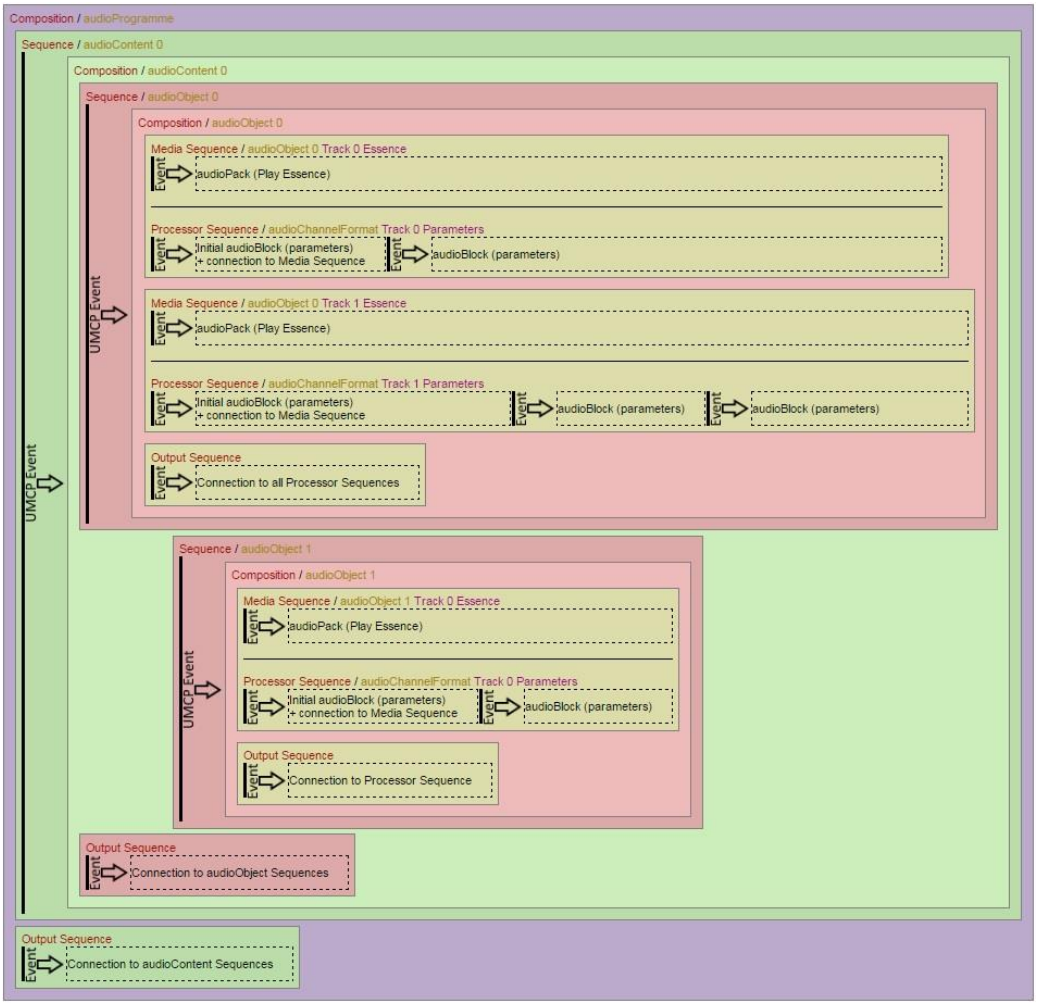

Figure 4: Mapping the Audio Definition Model into the Universal Media Composition Protocol

2.3 Storing object-based audio

Some of the important benefits of object-based audio are the ability to re-use the audio for different purposes, in different re-production environments, and with different personalisations. It is important that the audio is stored in a way that maximises the quality when it is re-used. Previous experience in broadcasting, such as with the AC078 ATLANTIC project (ATLANTIC, 1999) showed that cascaded bit rate reduction has a deleterious effect on audio quality, and it is best avoided where possible. Further, the decreasing costs of storage and network bandwidth mean that it is generally the case in production that bit rate reduction is no longer necessary for audio.

2.3.1 File-based storage

Earlier work in international standardisation, in which one of the Orpheus partners played a leading role, has led to an ITU-R Recommendation of a file format for the storage of object-based audio, together with its metadata. The metadata is described in Recommendation ITU-R BS.2076 – "The Audio Definition Model", abbreviated as "ADM". The file format is described in Recommendation ITU-R BS.2088 – "Long-form file format for the international exchange of audio programme materials with metadata", and is abbreviated "BW64". The data model of the ADM is drawn in Figure 5.

The ADM metadata is typically stored as a "chunk" in the file. This enables the file to contain object-based, scene-based, and channel-based audio, with the contents being described by the metadata. A further document, Recommendation ITU-R BS.2094 – "Common definitions for the audio definition model" simplifies the task of describing some commonly used formats.

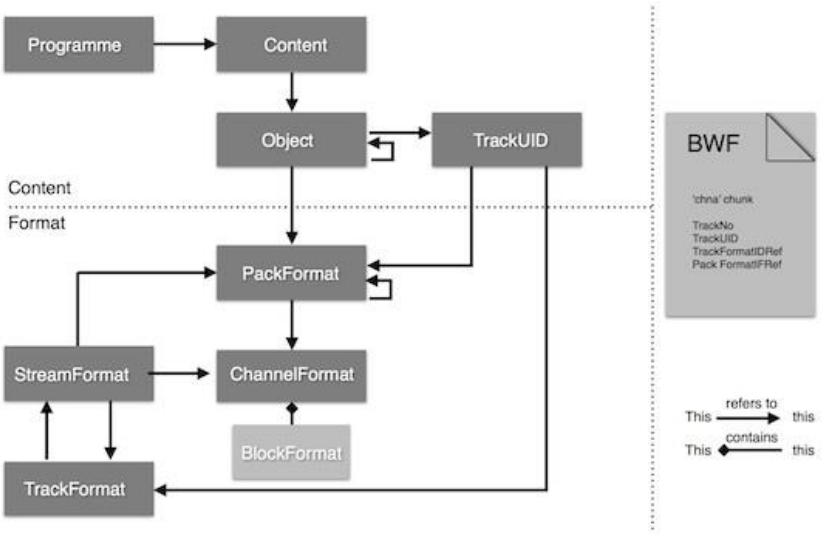

Figure 5: Diagram of the Audio Definition Model

The way that ADM is stored in the file is good for file-based operations: the entire set of metadata can be read and parsed before the audio. It is not ideal for live streaming of audio, or where streams might be spliced.

2.3.2 Stream-based storage

Live streaming of audio implies a need to be able to splice streams, or to start receiving data at an arbitrary time after streaming was started. To allow the interpretation of the audio objects in a timely manner after picking up a stream, it is necessary that the audio objects be accompanied by all the metadata required for that interpretation. This means that some of the set of metadata must be retransmitted periodically.

To facilitate generation of these streams, the audio and the metadata are stored in a "sequence store" as a series of "grains" (see Section 2.1.2.3). Grains are content agnostic, so can contain either audio or metadata. The relationship between grains of metadata and grains of audio are described using UMCP events (see Section 2.2.2.1).

The metadata model used for streaming is the same as the Audio Definition Model (ADM), which is used to describe object-based audio files. However, these object-based streams use JSON to store the ADM metadata, in place of the XML used in BW64. The ADM metadata is represented in UMCP through the mapping detailed in Section 2.2.2.2.

Both the UMCP composition events and the content itself can be accessed through a Media Access API, which has been implemented in IP Studio (see Figure 6). This API includes functionality for storing and retrieving object-based audio to/from the sequence store.

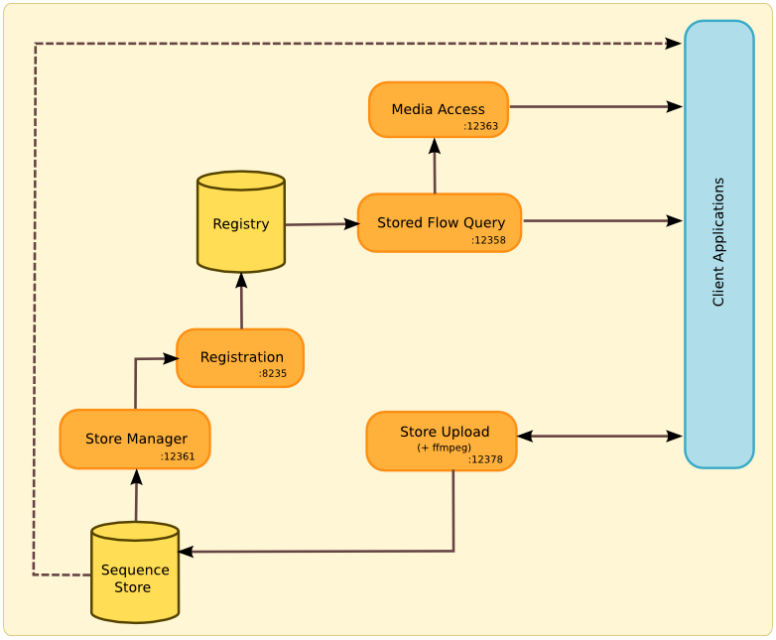

Figure 6: Diagram of the initial implementation of a Media Access API in IP Studio, which can handle import/export of files and reception/transmission of Flows.

3 Implementation

Using BBC's IP Studio, which implements NMOS, includes sequence store and media access API.

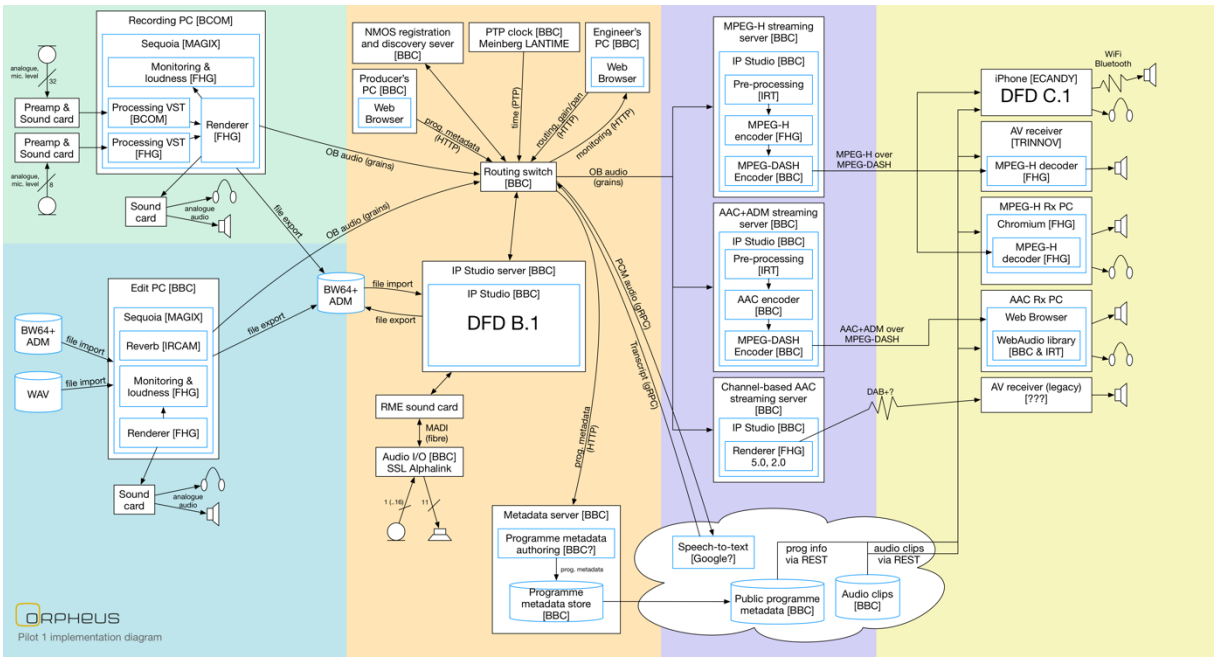

Figure 7: Diagram of proposed implementation - the production environment is shown in orange

3.1 User interfaces

3.1.1 Audio control

The central point of control of the production process is a 27" 10-point touch screen. This can be used to experiment with new types of interaction. It is realised that some tasks might require the use of physical controls with haptic feedback, such that their use does not require vision. Further, consideration is continuously given to the need for people with impaired vision to be able to operate future radio production systems.

A touch-screen is inherently a 2-dimensional surface, and the immersive audio experiences to be offered by Orpheus will require panning of audio signals within a 3-dimensional space. The way in which this will be done is still the subject of investigation. Several interesting possibilities are already known, including the use of gesture-sensing, and of stereoscopic video.

3.1.2 Production

Metadata relating to object-based audio does not only include spatial parameters. During the commissioning and production process, there is a need to capture and modify metadata like title, description, identities of people, segments within a programme, semantically linked data on topics within a programme. A relatively simple system will be created to demonstrate the process: linking into the existing production metadata system used by the BBC would be a complex task, which would have to be done with great care and due process to avoid any risk of data corruption. It has been decided that this linking is not necessary for the purposes of the Orpheus project.

3.1.3 Status monitoring

To ensure that the quality of the broadcast output is maintained, it is necessary to monitor the system continuously. At the simplest level, it will be possible to see which programme is being broadcast, whether the studio is 'on air', if the network load and processing is within capacity. More

advanced quality control measures are to be included later, such as automated loudness compliance monitoring, and automated checks of metadata to detect conditions that are likely to be the result of faults.

3.1.4 Pre-production

Magix have implemented the import and export of ADM files in Sequoia. Tracks are created for the individual objects described in the ADM. Folder tracks are used to represent the audio content level of the ADM hierarchy. During import, gain and panning data is translated into automation curves. Changes to these curves or the audio itself can be exported to a new ADM file.

Creation of ADM files from scratch has also been implemented at a basic level, meaning that certain project setup guidelines need to be followed and only immediately available metadata is written to the file. For additional information items (e.g. content language), separate metadata editing functionality will be implemented.

In the near future, Sequoia will be used to render and write multi-capsule microphone array feeds including corresponding panning information to ADM files. Additional methods for capturing and storing additional metadata are being considered. Work continues to integrate a rendering solution into Sequoia directly, allowing object-based audio to be previewed on the desktop.

We are also investigating the possibilities of streaming object-based audio directly from Sequoia to the production system. This may employ the use of virtual sound card technology (e.g. R3LAY VSC).

3.2 Sound capture

3.2.1 Microphone

An AKG C414 microphone is mounted on a flexible boom in the studio. It is connected to an Axia XNode. The XNode chosen provides a microphone preamplifier built in and its output uses an AES67 interface, connecting easily to the IP Studio platform over IP. Could be extended to act as NMOS device, to implement discovery and registration.

3.2.2 Outside sources

Existing infrastructure in the studio's apparatus room provides access to "outside sources". These are in the form of four AES3 connections to the central audio matrix in Broadcasting House. Selection of which signal from the matrix should be routed to which outside source is controlled using a dedicated touch screen interface (known as "BNCS" broadcast network control system). This allows any signal present in the matrix to be routed into the studio.

3.3 Sound reproduction

3.3.1 Speakers

Recommendation ITU-R BS.2051 – "Advanced sound system for programme production" describes numerous layouts of loudspeakers commonly in use. The recommendation is in the process of being revised to add a layout that is becoming commonly used in immersive audio formats, the so-called "4+7+0" layout. This layout has 7 loudspeakers at head height around the central listening position, and 4 loudspeakers above head height. The "0" refers to floor level loudspeakers, of which there are none in this layout. "4+7+0" represents a reasonable compromise in having enough loudspeakers to provide a realistic immersive experience, but not so many that it is impractical to install them. Some details of precise positions to be used are still under discussion, but the installation in the studio corresponds exactly to one variant currently proposed. The 4+7+0 layout has the added bonus of being compatible with existing 5.1 and 7.1 arrangements.

The loudspeakers are “active” speakers, that is, they have a built in power amplifier. The input to the amplifiers is analogue. Signals to feed the amplifiers come from a multchanel DAC (SSL Alphalink MADI) which is connected to a MADI sound card in the IP Studio PC. Loudspeakers are becoming available that have an AES67 interface. These would present an interesting alternative, but some considerable work was anticipated in making such a large number of them work in the environment of this project.

3.3.2 Headphones

A headphone amplifier has been installed in the studio. Headphones are required for monitoring when the microphone is live, but they could, in general, be fed with binaurally rendered sound for individual monitoring of 3D sound.

3.4 Communication

3.4.1 Talkback

A talkback system from Delec was left in the studio when the rest of the equipment was decommissioned. This is convenient in the short term, as it links several studios, apparatus rooms, and control rooms around Broadcasting House. The intention is to move to an IP-based system with appropriate metadata.

3.4.2 Telecoms

BBC R&D department makes extensive use of an IP telephone system by Cisco. A teleconferencing unit has been installed in the studio. This can be connected to existing radio phone-in software like Phonebox.

Figure 8: Studio 30D in development

4 Conclusions

This document has described the initial design and implementation of an object-based audio production system. We have chosen to follow the Networked Media Open Specifications (NMOS), which describe methods for identity, timing, registration and discovery, and connection management. The key concepts of NMOS are as follows:

- Media networks are made up of Nodes
- Nodes are discoverable, as are the resources they expose
- Data is sent over the network in Grains
- Identity and timing can be tracked end-to-end, no matter what Devices essence passes through

NMOS is agnostic to the transport method. For our system, we have chosen to use:

- RFC 3190 for audio transport (24-bit 48kHz PCM over RTP)
- JSON string sent over RTP for metadata transport

The purpose of our system is to create and transport object-based audio. In order to achieve this, we have selected the following models and protocols to represent the content:

- Audio Definition Model (ADM) for representing object-based metadata
- BW64 with ADM for storing object-based audio in files
- Universal Media Composition Protocol (UMCP) to link audio and metadata in streams

BIBLIOGRAPHY

ATLANTIC. (1999, April). *Project Final Report.* Retrieved from Community Research and Development Information Service: https://cordis.europa.eu/pub/infowin/docs/fr-078.pdf

Brightwell, P. (2016, November 21). *NMOS Mapping of Identity and Timing Information to RTP.* Retrieved from AMWA: https://github.com/AMWA-TV/nmos-in-stream-id-timing/blob/master/specifications/IdentityTimingRTP.txt

Appendix A Converting ADM to UMCP

The process of converting an ADM file to a UMCP composition is as follows. The colours refer to Figure 4.

- The outer audioProgramme ADM element becomes the first composition (purple).
- Within the audioProgramme composition (purple), sequences represent the audioContent(s) of the ADM (darker green).
- For each audioContent sequence (darker green), we create an associated audioContent composition (lighter green) to encapsulate the objects within the 'audioContent'.
- An event within each audioContent sequence (darker green) triggers playback of the audioContent Compositon(lighter green).
- Each audioContent sequence (darker green) feeds the output sequence (green output sequence) of the audioProgramme composition (purple).
- Within each audioContent composition (lighter green), sequences represent the audioObjects contained within (darker pink).
- As with audioContent sequences, for each audioObject sequence (darker pink), we create an associated audioObject composition (lighter pink).
- An event within each audioObject sequence (darker pink)triggers playback of the audioObject Compositon (lighter pink).
- Each audioObject sequence (darker pink) feeds the output sequence (pink output sequence)of the audioContent composition (lighter green).
- During the generation of an audioObject composition (lighter pink), each audioObject channel is extracted from the file and saved as a standard RIFF single-channel WAV file.
- This file is (optionally) uploaded to the store.
- The local file is then (optionally) deleted.
- Each audioObject composition (lighter pink) contains a Media sequence for each channel of that audioObject(tan).
- These media sequences contain an event to trigger playback of the essence (which was uploaded to the store server.)
- For each audioObject channel, there is a Processor sequence relating to the audioChannelFormat ADM element for that channel (tan).
- Events in these sequences correspond to parameter changes stores in audioBlockFormats within the audioChannelFormat.
- The output of the media sequence for an audioObject channel feeds the corresponding processor sequence.
- Each processor sequence feeds the output sequence (tan output sequence) of the audioObject composition(lighter pink)

[end of document]