SMPTE Meeting Presentation

An Open Object-Based Immersive Audio Content Format

Ton Kalker, Jean-Marc Jot

DTS, Inc. {ton.kalker,jean-marc.jot}@dts.com

Written for presentation at the

SMPTE 2013 Annual Technical Conference & Exhibition

Abstract. *We introduce an open and future-proof multi-channel audio format designed for the creation, archiving and distribution of digital media content for the cinema, broadcast and gaming industries. The proposed format extends the existing multi-channel audio formats with the addition of a plurality of audio object channels accompanied with positional rendering metadata. We describe the basic concepts of the sound-field model, describe the associated file and stream format, and report on the status of relevant standardization activities.*

Keywords. *Object-based audio, Immersive audio, Cinema, Broadcast, Gaming*

1 Introduction

With the development of upcoming Ultra High Definition TV standards and the emergence of Immersive Audio systems in movie theaters, audio technology companies and standardization organizations (such as MPEG, ATSC and EBU) have begun developing next-generation multichannel audio technology applicable in the cinema and broadcast industries. Currently, multichannel audio content creation and delivery relies on assumed frontal stereo or horizontal multi-channel surround loudspeaker playback configurations. In this paper, we describe a multi-channel audio content creation model and format which breaks the current constraints tying the creation format to the playback configuration while enabling a more natural listening experience and new level of innovation in entertainment services. This is realized by adapting to linear media creation and delivery the concept of *object-based audio*, previously developed for interactive audio applications such as computer music, virtual reality and gaming systems [5][6][7][8].

In interactive audio applications, the need for object-based audio arises for the fact that the audio scene description parameters are determined at playback time – including the dynamically varying positions of virtual sound sources. These positions are represented by geometrical coordinates in a virtual world, and the soundtrack can be generated for an arbitrary playback configuration by real-time positional audio rendering and superposition of the audio waveforms representing each of the sound sources [9]. A similar rendering operation is performed by the mixing console or digital audio workstation during traditional music or movie soundtrack production to export a recording in standard 2.0, 5.1 or 7.1 multi-channel audio format or in one of the recently proposed 11.1 or 22.2 immersive audio formats [10]. In these *channel-based* formats, however, the geometric description of the audio scene is implicitly and inseparably captured via differences among the output signal channels, valid for a particular assumed loudspeaker playback configuration. Adapting an original channel-based recorded soundtrack to multiple consumer playback configurations requires producing multiple versions of the original recording, or applying heuristic spatial audio format conversion methods that often compromise the original audio quality.

An object-based audio content creation format affords content creators the ability to “create once, play everywhere” and facilitates content interchange, reuse and archiving. For consumers, the advantages of object-based audio include enabling the most faithful spatial audio reproduction possible on the target playback system configuration, and flexibility in the setup of multi-channel home theater sound systems. Additionally, object-based audio formats offer additional flexibility for personalization of the content (such as dialog enhancement) and interactivity (modifying the mix by emphasizing, relocating or replacing sound elements). For broadcast or video-on-demand service providers, an object-based audio format enables single inventory while catering for different delivery channels, languages, hearing or visually impaired consumers, and listening configurations (including enhanced reproduction over headphones, in noisy environments, or at night time).

The remainder of this document is structured as follows. In section 2, we provide a technical overview of MDA (Multi-Dimensional Audio), an audio content authoring and interchange format for linear media in which a soundtrack is represented as a collection of audio waveforms accompanied by a sound scene description specified as a dynamic collection of point and extended sound sources. We review the MDA Core specification, which defines MDA elements and their logical relationships, and the MDA bitstream specification, which embodies a serialization of a MDA sound scene description. In section 3, we conclude this paper with a status update on related development and standardization activities in the cinema and broadcast industries.

2 Format Description

An MDA file or stream is a self-contained object-based sound scene description, referred to as an *MDA program*. It consists of a collection of *audio objects*, each combining an audio waveform with attached metadata. The metadata indicates, for instance, where the object occurs on the program timeline or where it is positioned within the sound scene (see Figure 1). This metadata is used by an MDA renderer to map the audio object waveforms to output loudspeakers at playback time.

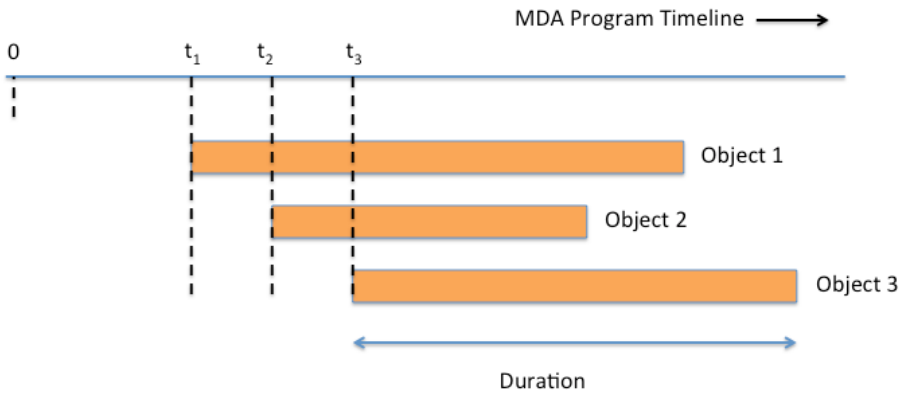

Figure 1. MDA Program as a dynamic collection of objects.

The description of an MDA program relies on two documents, the core specification and the bitstream specification, respectively. The former describes the core MDA concepts and their relationships, whereas the latter specifies how the core concepts are encoded in a bitstream. The former document includes the description of the reference MDA renderer, defining the semantics of an MDA program. The MDA reference render may be used during content creation to monitor the quality of an MDA program with respect to the artistic intent. The reference MDA renderer is based upon the well-known Vector Base Amplitude Panning principle.

In the following three sections we provide a technical overview of MDA: MDA core, MDA bitstream and MDA Reference renderer.

2.1 MDA Core

2.1.1 MDA Program

At top level, an MDA Program consists of a single program *header* and zero or more *entities* (see Figure 2). A *header* (see Figure 3) lists basic metadata for the MDA Program as a whole, including a globally unique MDA Program identifier, sample rate and constraint sets. An MDA *entity* is a hierarchical construct that specifies actual audible objects on the MDA Program timeline.

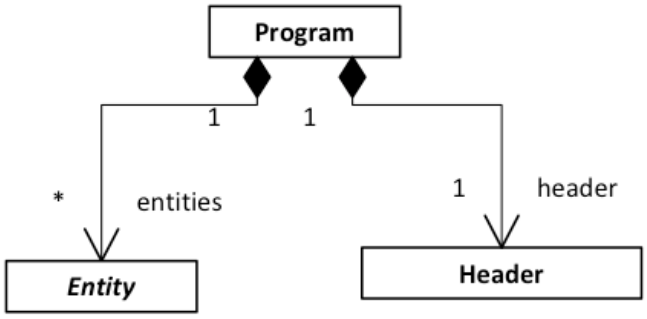

Figure 2. MDA Program.

MDA has adopted the convention that values that need to be understood globally across the MDA ecosystem are provided by means of key-value pairs. In particular, the key part of a key value pair is formatted as a Uniform Resource Identifier (URI) [URI], with the actual value is defined in the appropriate name space. For example, the sample rate of a MDA Program may be given as the URI below:

`http://mda.noname/2013/core/labels/sample-rate/48000Hz`,

where the actual value of 48,000Hz is defined in the MDA namespace. As URIs have a tendency to be rather long and hard to read, MDA allows these URIs to be abbreviated. For example, the MDA constant `AudioSampleRate48000` is an MDA constant that evaluates to the URI above, and that may be used as an alternative for the full URI.

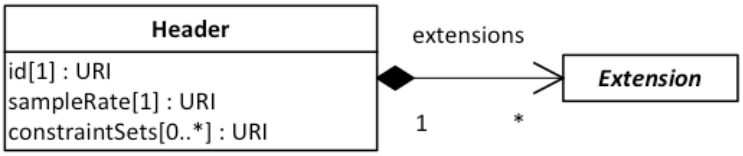

Figure 3. MDA Program Header.

The constraints in a Program header are similarly formulated as a URI and allow a MDA renderer to quickly determine whether or not the MDA Program can be processed.

The MDA Program header, like most other MDA high-level elements, allows a reference to zero or more *extensions*. Extensions may hold additional application specific metadata, but for the purpose of the core MDA specification are considered to be opaque.

2.1.2 MDA Entity

An MDA Entity is a recursive construct that specifies actual audible objects. MDA entities are associated with an (implicit or explicit) offset and duration on the MDA Program timeline. Metadata associated with an Entity are static for the duration of an Entity. Each Entity however has an ID that may be used to link multiple Entities together in the scope of an MDA program, enabling the description of objects with dynamic metadata. It is prohibited for Entities with the same ID to overlap on the MDA Program timeline.

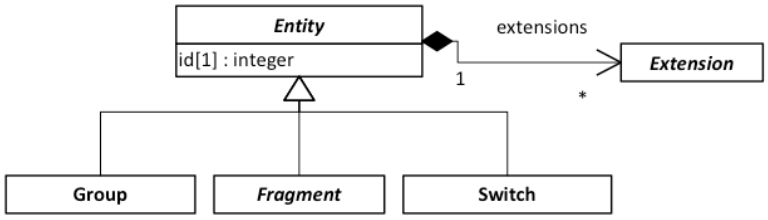

Figure 4. MDA Entity.

An MDA Entity may be either an aggregate construct or an atomic object. The Group construct specifies an ensemble of member Entities, all to be played out simultaneously. The offset of a Group is defined the minimum of all the offset of its members. The duration of a Group is defined as the duration of the minimal time window that encloses its member Entities.

A Switch construct specifies an ensemble of member Entities of which only one will be played out. One of its members is designated as *default*, and will be played out unless otherwise indicated. Offset and duration are defined as for a Group.

2.1.3 MDA Fragment

An MDA Fragment is a base class for an atomic MDA object with explicit offset and duration.

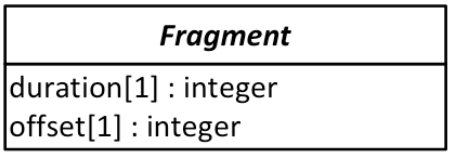

Figure 5. MDA Fragment.

The next level of specificity defines a MonoSourceFragment by associating AudioSamples with a fragment (see Figure 6). In the context of MDA, the type AudioSamples consists of a reference to an actual sequence of audio samples and an offset within that sequence (see Figure 7). In the context of an *MDA bitstream*, this reference may either refer to *assets* embedded in the MDA bitstream or to external assets (in a ZIP file, on a server, or otherwise). This flexibility in specifying the location of audio samples is of relevance in the context MDA for Digital Cinema.

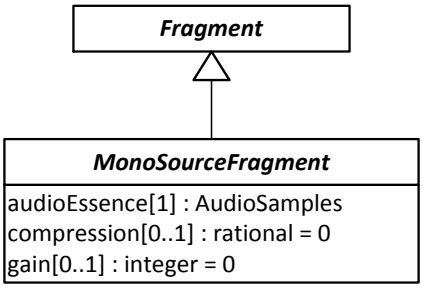

Figure 6. MonoSourceFragment.

Two optional parameters are defined for a MonoSourceFragment. The compression parameter defines the level of Dynamic Range Compression (DRC) and the gain parameter defines an overall gain for the audio samples.

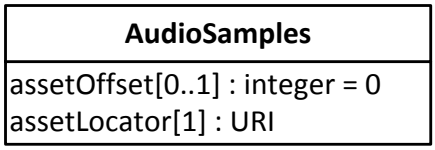

Figure 7. AudioSamples.

2.1.4 MDA ObjectFragment

An MDA ObjectFragment, often referred to as an Audio Object, extends a MonoSourceFragment by adding (static) positional and other metadata (see Figure 8).

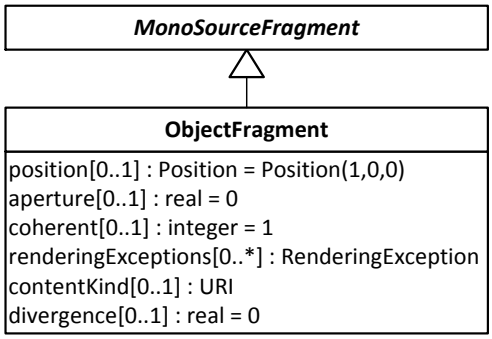

Figure 8. ObjectFragment.

The Position parameter defines the (point) position of the Audio Object in Cartesian coordinates, with the head of listener at the origin O=(0,0,0), positive x-axis to the right, positive y-axis to the front and positive z-axis to the ceiling. For the purpose of the current MDA specification, all Audio Objects are assumed to be at distance 1 from the origin, essentially restricting the Position parameter to a Direction parameter. Position may also be expressed in spherical coordinates (azimuth, elevation and radius), where an azimuth value of 0 corresponds to the positive y-axis and an azimuth value of 90º corresponds to the positive x-axis (see Figure 9).

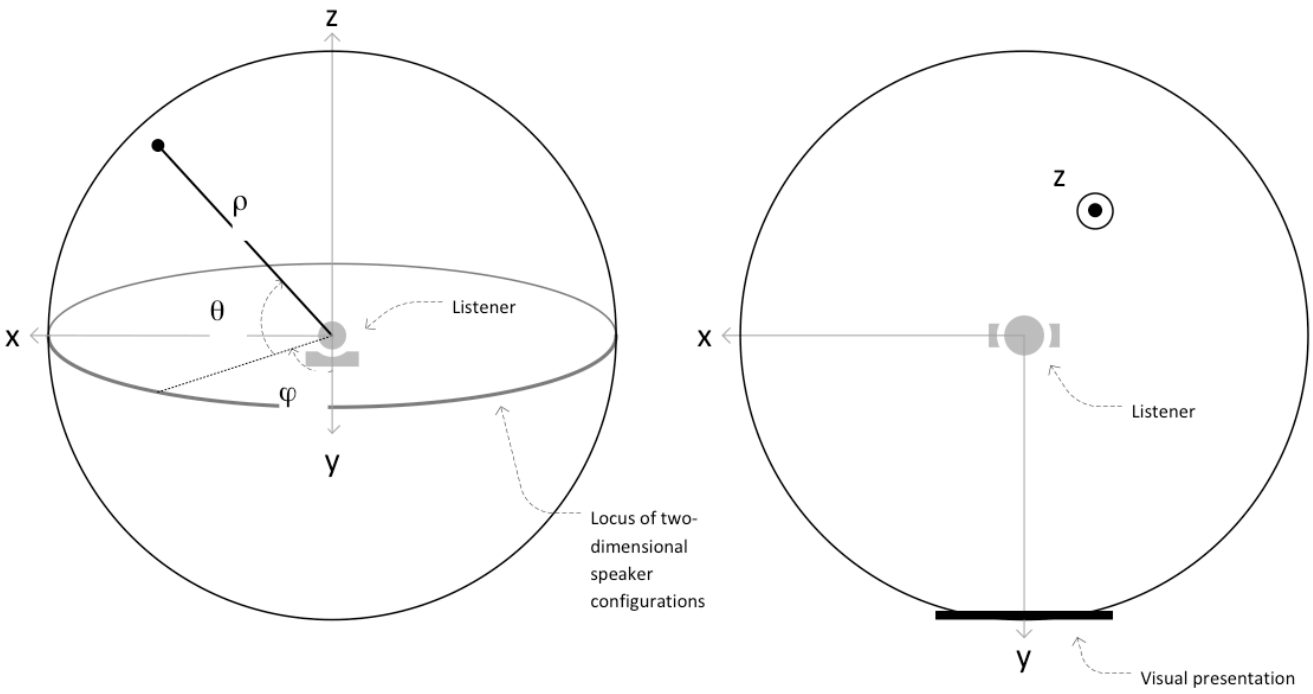

Figure 9. MDA Coordinate System.

Audio Object may also have spatial extent defined, with `divergence` indicating horizontal extent and `aperture` indicating vertical extent. The coherence parameter indicates whether an object is rendered diffusively (`coherent=0`) or coherently (`coherent = 1`). The parameter `contentKind` indicates the kind of content, e.g. dialog (`AudioContentKindDialog`).

Finally, an Audio Object may have Rendering Exception metadata attached, indicating that within the context of a specified target Configuration the reference MDA renderer must be overridden (see Section 2.1.6).

2.1.5 LFEFragment

An LFEFragment is a trivial extension of a MonoSourceFragment and indicates a Low Frequency Sound source.

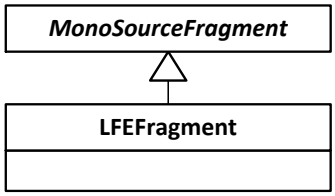

Figure 10. LFEFragment.

2.1.6 Rendering Exceptions

Rendering Exceptions are a powerful tool to better preserve artistic intent when the MDA reference renderer is deemed insufficient. In addition, rendering exceptions may be used to emulate a channel-based mix within an MDA Program (see Section 3.1).

The `targetConfigurations` property (e.g. a 5.1 channel configuration) indicates the rendering configuration to which the `RenderingException` instance applies. An empty `targetConfiguration` property indicates that the `RenderingException` instance applies to any target rendering configuration. The VBAP reference renderer must always be configured to a particular configuration, and is therefore able to decide whether or not a rendering exception applies (see Section 2.3.1).

Two types of rendering exceptions are currently defined, one with respect to position and one with respect to channels. The `PositionRenderingException` allows the author to indicate an alternate position for the object. The `ChannelRenderingException` allows the author to explicitly specify the channels to which the object waveform is routed, listing an explicit gain factor for one or more channels (a channel that is not listed has by convention a gain factor equal to zero).

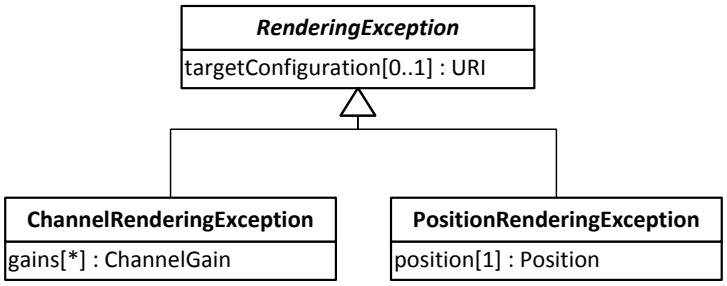

Figure 11. Rendering Exception.

2.2 MDA Bitstream

An MDA bitstream is a serialized representation of an MDA Program as defined in the previous sections. The design of the MDA bitstream closely follows the core specification, with three additional important properties. Firstly, the bitstream is partitioned in independently decodable units, referred to as frames. Frames may be decoded without reference to previous and future frames. This MDA frame structure enables easy random access to any segment of an MDA program. Secondly, and supporting the first property, the assets associated with a frame may be (and are typically) included in a frame as an asset as an asset fragment. Thirdly, a frame is partitioned in fragments, where each bitstream fragment corresponds to a core entity with static metadata. Note however that within a frame metadata may change from fragment to fragment. Also note that a bitstream fragment is not a core concept and that an MDA parser may choose not to (and preferably does not) expose the bitstream fragment structure.

2.2.1 Packets

An MDA Bitstream consists of a recursive sequence of *packets*. As illustrated in Figure 12, each packet consists of a payload preceded by a *packet header*, minimally identifying the nature of the packet (Kind) and the length of the packet (Length). Using the Length property, implementations may ignore and skip packets. In addition packet header may include a synchronization property, enabling random access.

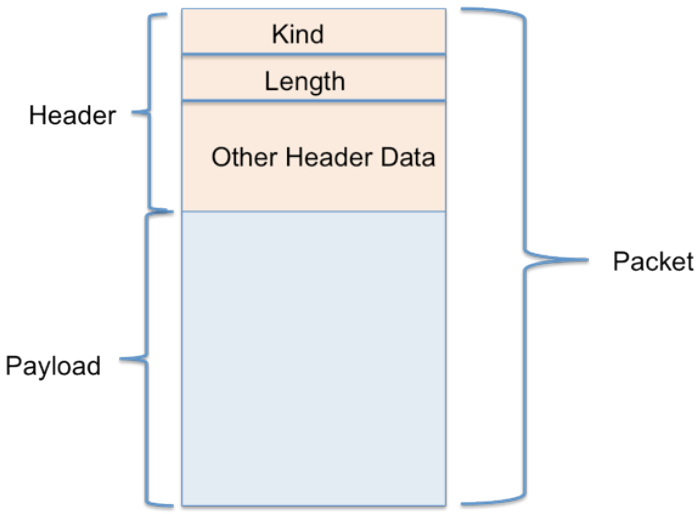

Figure 12. MDA Packet.

2.2.2 Frames

An MDA program uses a single global time line. This timeline is partitioned in frames, which each frame further partitioned into fragments. Correspondingly, as depicted in Figure 13, an MDA program consists of a sequence of *frame structures*, each containing the metadata and (references to) audio samples required for a specified interval (frame) within its timeline. The (references to) audio samples are contained in an *asset structure* that may contain additionally metadata pertaining to pre-processing of the audio samples (e.g. gain, DRC, and others, see Section 2.1.3). Each frame structure contains sufficient information allowing playback to start on any frame structure boundary without requiring access to prior or future frames. An MDA program contains an MDA program header (Section 2.1.1), identifying the MDA program. Each frame in an MDA bitstream represents this header as to enable independent playback.

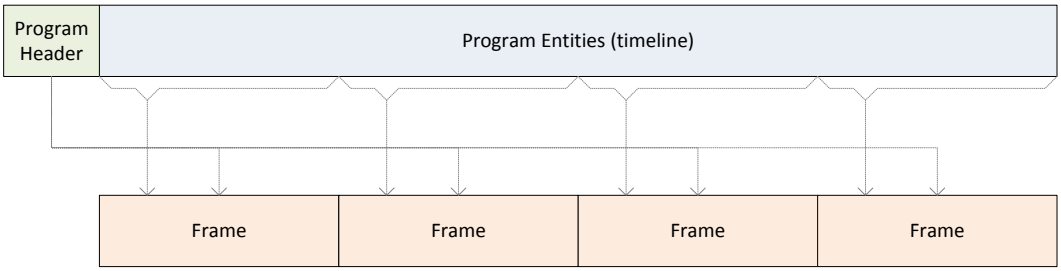

Figure 13. MDA Bitstream Structure.

2.2.3 Fragments

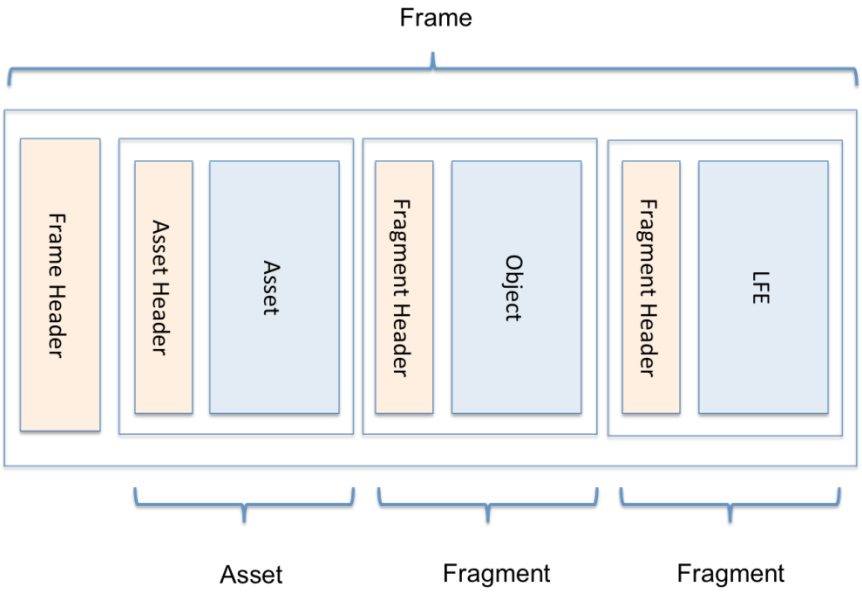

Figure 14. Frame Structure.

As illustrated in Figure 14, frame structures are further segmented into *fragment structures*. Each fragment structure contains a set of MDA entities, where entities correspond to (aggregates of) audible components of the sound field that the MDA program is specifying (Section 2.1.2). Fragment structures differ from frame structures in that they are not independently decodable, relying on metadata that pertain to the whole frame. Fragment structures are further restricted in that its metadata are constant for the duration of the frame, e.g. the position of a point source entity may not change for the duration of a fragment.

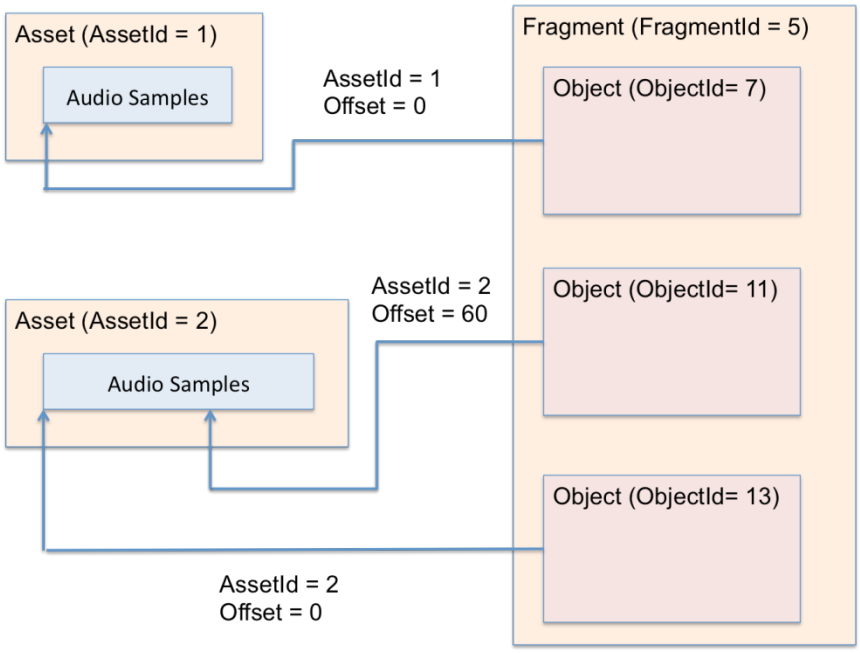

Figure 15. Frames, Fragments and Assets.

Frames may (and typically do) include the audio samples referred to by the entities encoded in a fragment. This allows local retrieval of audio samples without reference to other frames. Note however that the MDA syntax allows the entities to refer to assets that are not contained within a frame (e.g. a ZIP file or a server).

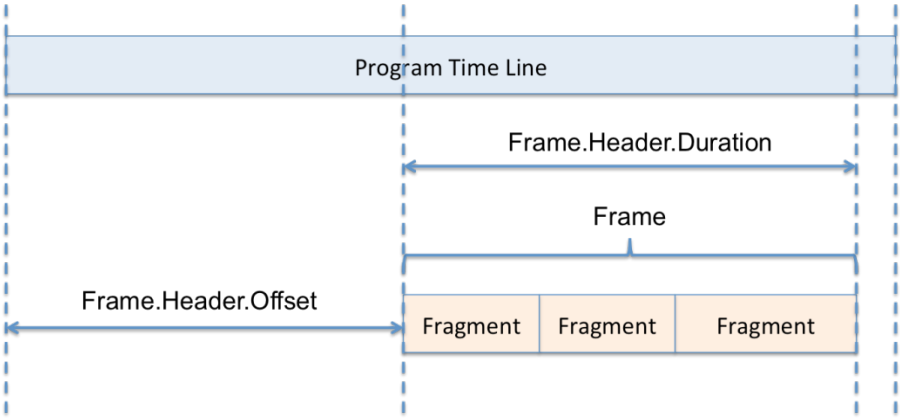

Figure 16. Frame Time Positioning.

Frame structures and fragment structures contain sufficient metadata to position the corresponding frame and fragment intervals on the program timeline (see Figure 16).

2.3 MDA reference Renderer

The MDA Reference renders an MDA Program on set of speakers (channels) according the Vector Base Amplitude Panning principle [VBAP]. The two main ingredients in the MDA reference renderer are configuration and gain calculation. Given a configuration, the MDA reference renderer computes for each MonoSourceFragment a set of channel gain values for its audioEssence that best preserve the artistic intent of the content creator. See Figure 17 for an overview of the MDA reference renderer.

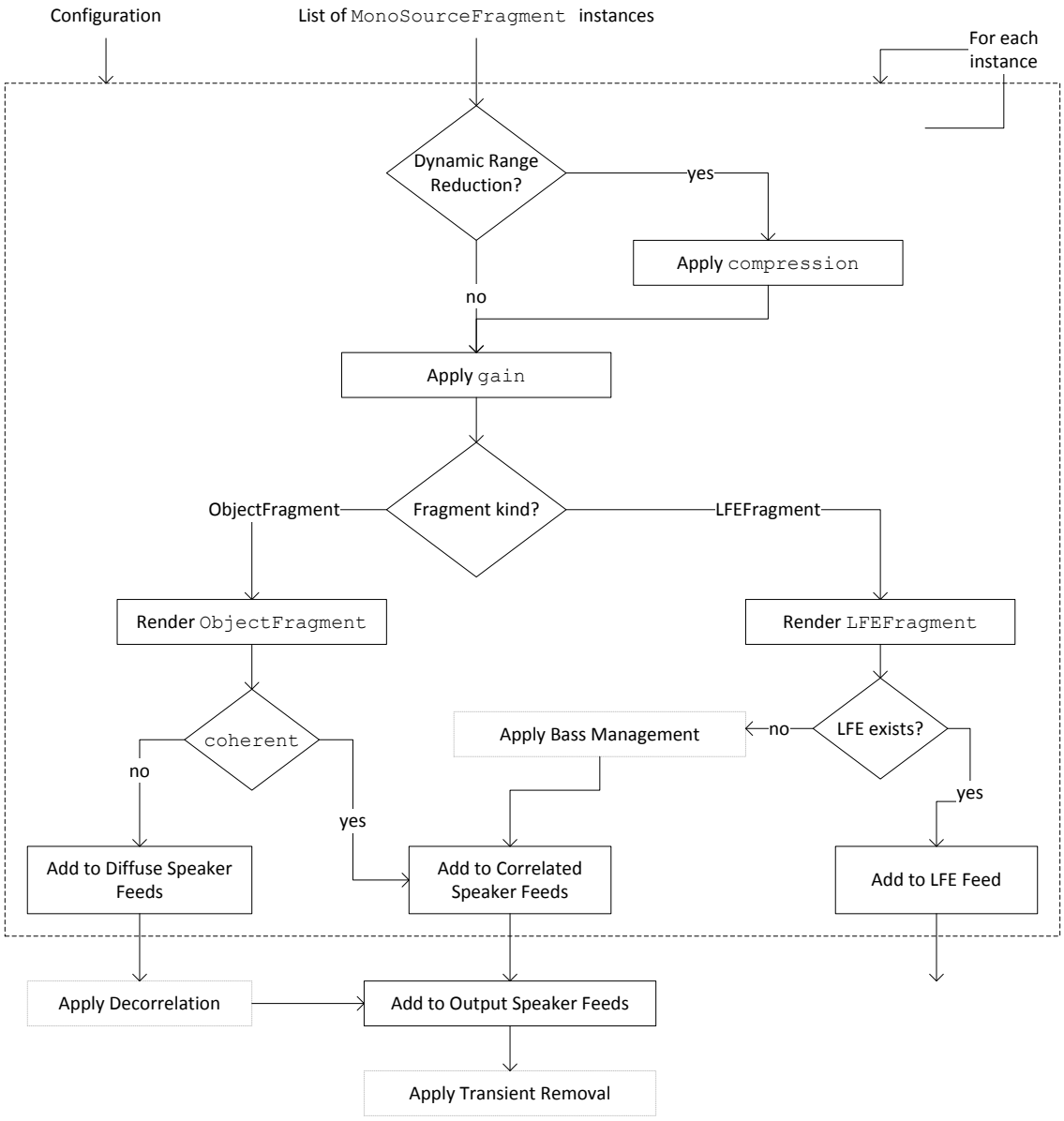

Figure 17. MDA Reference Rendering.

2.3.1 Configuration

The configuration process provides the MDA Reference renderer with the necessary information to render MonoSourceFragments to the loudspeakers. The following information is provided, the first four by means of a configuration file.

1. **Sound Field** – this corresponds to the `targetConfiguration` property of a Rendering Exception, and used by the Reference Renderer to determine whether or not to override the VBAP engine. See Section 2.1.6.
2. **Normal Speakers** – for each normal speaker the following information is provided:
    a. **Name** – the name of the speaker.
    b. **Position** – the position of the speaker on the unit sphere.
    c. **Mix Coefficients** – MDA allows a post-processing step to render speaker output on other speakers. Currently there are only two options allowed: either a speaker

does not render on any other speaker, or a speaker renders completely on other speakers. The latter case is referred to as a *virtual* speaker, whereas the former is referred to as a *real* speaker. The mix coefficients specify the coefficients for rendering on other speakers.

3. **LFE Speakers** – speakers that render low frequency effects.
4. **Patches** – triplets of speakers that define the operation of the VBAP engine. If T is a triplet, let P(T) be the plane defined by T, and let H(T) be the half space bounded by P(T) that contains the origin. Patches are defined as triplets T such that all other speakers (not in the triplet T) lie in H(T).
5. **Virtual Sources** – a (dense) uniform grid of points on the unit sphere. A gain vector is associated with each Virtual Source, specifying the gains with which a Virtual Source will be rendered on the normal speakers. The VBAP point source renderer is used to compute Virtual Sources (using the information in the configuration file).

2.3.2 Point Source Rendering

Point source rendering applies the principles of generalized VBAP rendering as defined in [VBAP]. Given a point source with position S, the render process computes a set of gain values, one gain value per normal speaker. The rendering process consist of the following steps:

1. Find the patch T such that the vector OS intersect T. Let P be the intersection point.
2. Compute the barycentric coordinates {g} of P with respect to T.
3. Normalize {g} such that P is extended to S.
4. Normalize {g} to unit L2 norm.
5. Output the gain values for {g} for the three speakers making up T. Other gain values are set to zero.

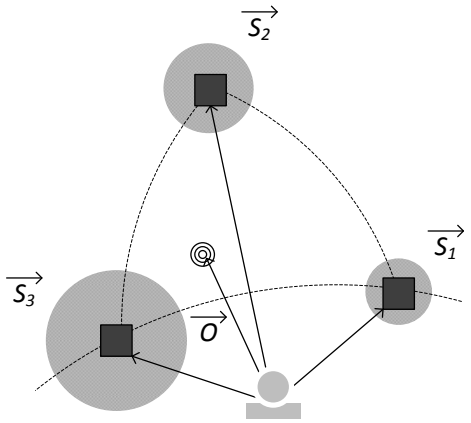

Figure 18. Point Source Rendering.

The rendering process optionally includes post processing for diffuse and coherent rendering as well as for transient smoothing. The specifics of these two post-processing steps are not specified in the MDA Reference Renderer specification (see Figure 17).

2.3.3 Extended Source Rendering

To render an extended source all virtual sources in the extent are enumerated and the associated gain values (per normal speaker) are summed. As a final step the gain values are normalized to unit L2 norm. Post-processing is similar as for point source rendering.

2.4 MDA Example

Figure 19 depicts a sample MDA program. It consists of 4 audio objects: LFE, FX1, FX2 and Dialog. The LFE object starts at Program time t=0, and ends at Program time t = 2 (arbitrary time units). The objects FX1 and FX2 represent two point source objects, starting at t=1 and ending at t=3, each object referring to a single asset (but with different offsets for its two fragments). The two objects are logically grouped, allowing sharing of common metadata. The Dialog object start at time t=0 and ends at time t=2. The Dialog object is marked as dialog using the `contentKind` parameter, potentially allowing special processing (e.g. dialog enhancement).

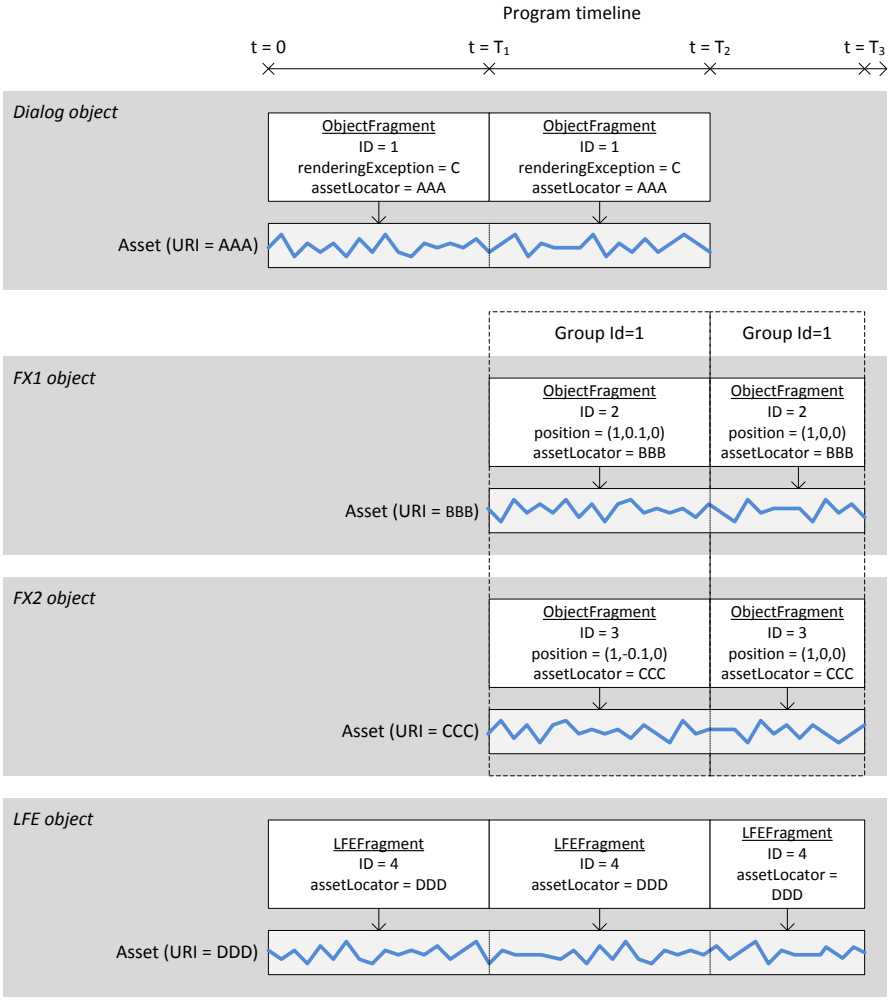

Figure 19. Sample MDA Program.

3 MDA Considerations

In this section we explore some existing and prospective features of MDA.

3.1 Channel-based beds in MDA

It is important that MDA can peacefully coexist with channel-based audio. Fortunately, MDA is easily able to emulate channel-based audio using the MDA concepts of Group and Rendering Exceptions. For example, a 5.1 bed is included as an MDA group consisting of 6 elements. The

LFE channel is the bed is obviously represented by an LFEFragment. The normal channels are each represented by an ObjectFragment. Each of these these ObjectFragments are associated with a a Rendering Exception for a 5.1 targetConfiguration, with each exception being configured for the appropriate channel. For example, an ObjectFragment for the left channel is excepted with a gain value equal to 1 for the left channel and a gain value equal to 0 for all the other channels in the 5.1 configuration.

Note: The emulation of channel-based audio in MDA has resulted in request for additional metadata, enriching the semantics of the MDA. In particular, the addition of a groupKind property to the group construct is currently under consideration.

3.2 MDA for digital cinema

On September 9, 2013, DCI released an DCI Object-Based Audio Addendum [4]. The Addendum introduces a "specification for packaging, distribution and theatrical playback of object-based motion picture D-Cinema audio content that exceeds the delivery capability of the Digital Cinema Package (DCP) audio track file as defined in DCI's Digital Cinema System Specification". The specification defines Object-Based Audio Essence (OBAE) as PCM audio essence enriched with descriptive metadata, including temporal and spatial metadata. An OBAE format compliant with DCI is required to be standardized by SMPTE, interoperable with OBAE rendering systems, and generalically compatible with the Digital Cinema System Specification (DCSS), in particular with respect to packaging and security.

A number of key companies and organizations in the Digital Cinema ecosystem have come together to develop and integrate MDA tools and work flow from creation to delivery. Also, the MDA format has beeb being proposed to SMPTE TC-25CSS for standardization. Because of the work done by this MDA supporters group on tools, work flow and SMPTE standardization, we feel confident that MDA can be offered as in the near future as an OBAE format meets the criteria of the DCI object-based audio addendum.

3.3 MDA tools

The MDA specification is supported by an evolving set of tools, allowing creation, reading and playback of MDA files. This toolset includes source code for packing and parsing of MDA files. The set also includes a sample mixer tool as a plug-in for ProTools (binary) and a simple playback engine (binary). In addition, tools are available for reframing (changing MDA bitstream frame size), and exporting and importing of assets. These latter three tools are mainly being used in the context of digital cinema, testing an end-to-end system from content creation, to cinema server, to cinema processor to B-chain (see Section 3.2).

4 Conclusion

We have presented MDA, a flexible and open format for object audio. We have argued that MDA is at the same time expressive and simple, and is able to serve as standard model for object-based sound scenes.

5 Acknowledgement

MDA is an effort by a large number of people and organizations. MDA, and by implication this paper, would not have been were it not for their continued involvement and support. In particular we would like to thank our colleagues at DTS and the members of the MDA supporters group. In particular we would like to highlight the contributions of Pierre-Anthony Lemieux without whom MDA would not have been possible.

6 References

1. Pulkki, Ville, “Virtual Sound Source Positioning Using Vector Base Amplitude Panning”, *Journal of the AES*, Volume 45, Issue 6, pp. 456-466, June 1997.
2. IETF RFC39886, “Uniform Resource Identifier (URI): Generic Syntax”, January 2005, http://tools.ietf.org/html/rfc3986.
3. ISO/IEC/JTC1/SC29/WG11**, “**Open Object-Based Audio Master Format”, Input Document (information) MPEG2013/M28914, April 2013, Incheon, Korea.
4. DCI, “Digital Cinema Object-Based Audio Addendum”, September 9, 2013.
5. J.-M. Jot, “Real-time Spatial Processing of Sounds for Music, Multimedia and Interactive Human-Computer Interfaces,” ACM Multimedia Systems J. 7(1): 55-69 (1999 Jan.).
6. OpenAL cross-platform 3D audio API, www.openal.org.
7. R. Väänänen and J. Huopaniemi, “Advanced AudioBIFS: Virtual Acoustics Modeling in MPEG-4 Scene Description,” IEEE Trans. Multimedia 6(5): 661-675 (2004 Oct.).
8. G. Potard, *3D-Audio Object-Oriented Coding*, PhD thesis, Univ. of Wollongong (2006).
9. F. Rumsey, *Spatial Audio* (Focal Press, 2001).
10. K. Hamasaki et al., “The 22.2 Multichannel Sound System and Its Application,” 118th Conv. Audio Eng. Soc. (2005 May).