Adding Delivery support to MPEG-Pro, an Authoring System for MPEG-4

Frederic Bouilhaguet, Cyril Concolato, Souhila Boughoufalah, Jean-Claude Dufourd,
ENST Paris Dept ComElec, 46, rue Barrault, 75013 Paris

E-mail: { bouilhaguet, concolat, souhila, dufourd }@ enst.fr

Abstract

Authoring MPEG-4 content is quite a challenge. Far from the past one-video-plus-one-audio-streams of its predecessors MPEG-1 and MPEG-2, MPEG-4 allows the content creator to compose together spatially and temporally large numbers of objects of different types (rectangular or shaped video, still image, natural audio, synthesized audio, 2D graphics and 3D). From this context, MPEG-Pro, an MPEG-4 authoring software prototype has been designed to integrate all main authoring capabilities from audio/visual encoding to the final delivery of a complete MPEG-4 application. The storage format, named MP4, has been defined for MPEG-4, but adapted techniques for specific delivery scenarios that are compliant with this format are still in course of proposition and definition. We introduce a solution to author and store MPEG-4 content with a better data accessibility for the streaming over RTP/IP scenario. By respecting scrupulously the MP4 constraints, this content remain reusable by any MPEG-4 compliant tool. We describe how its implementation is integrated in MPEG-Pro whose modular architecture allows a large scope of extensibility.

1 Introduction

MPEG-Pro is a software dedicated to authoring 2D multimedia content compliant with the latest MPEG-4 Systems specifications. Its architecture is voluntarily open. Different components have been integrated since its creation. Some of them implement tools of MPEG-4 Systems, but some others add enhancement techniques to face some limitations of the MPEG-4 standard in the context of multimedia authoring [1]. This time we have implemented and integrated a new component to interleave data of the authored *MPEG-4 presentation*. The main advantage of this module is to optimize the MPEG-4 content for some delivery scenarios. Content producers of interactive TV programs are used to maintain two versions of the same MPEG-2 content [2], stored in two different formats. The first one, optimized for authoring tools, contains no information for transport and keep elementary streams separate to make their editing easier. The second one is optimized for final delivery (i.e. interleaved and packetized), but no longer accessible by the authoring tools due to the complexity induced by the interleaving and packetizing. The MP4 file format is specified to store MPEG-4 content that enables at the same time its edition for authoring, its preparation for streaming over networks and its local playability. It has strict rules to ensure the interchangeability of multimedia contents. But it also offers a high level of flexibility to optimize storage dedicated to different scenarios of either interactive functionality or delivery. In this paper, we present the latest evolution of MPEG-Pro taking into account this flexibility. This evolution consists in giving the user the possibility to interleave the elementary streams of the MP4 file in flexible ways. We expose our own algorithm for interleaving content and its implementation. Then we show how our method is concerned with the specific multiplexer tool of MPEG-4 Systems, named the FlexMux. A solution is proposed to connect the FlexMux tool to our interleaver to optimize the MPEG-4 delivery over RTP/IP.

An overview of MPEG-4 Systems is first presented to understand the requirements of MPEG-4 authoring. Then, the architecture of MPEG-Pro is presented with a special focus on its new *interleaver* component. Finally, we show some advantages of interleaving MPEG-4 content for several delivery scenarios.

2 Overview of MPEG-4 Systems

The MPEG-4 architecture is distributed along the definition of three adjacent layers, as illustrated by the *Figure 1* : the Compression Layer, the Synchronization Layer, and the Delivery Layer [3]. Elementary Streams (ES) are the basic abstraction for any streaming data source. An MPEG-4 terminal plays an object-based MPEG-4 presentations by receiving, decoding and rendering all ESs that contain the coded representations of either audio-visual data (*media* ESs) or session and composition description (*control* ESs).

2.1 Compression Layer

An MPEG-4 content is coded and decoded in this layer by being organized in Elementary Streams (ESs). Individual elementary streams are processed with their specific encoders (on the sender side) and decoders (on the receiver terminal side). The MPEG-4 standard specifies MPEG-4 compliant streams with their decoders. The

Compression Layer organizes the ESs in access units (AU), the smallest elements that can be attributed to individual time stamps. The MPEG-4 Visual and Audio groups enhanced the coding techniques for audio and video ESs (*media* ESs). But the main enhancement in MPEG-4 compared to its predecessors MPEG-1 and 2 is the definition of *control* ESs by the MPEG-4 Systems group. These *control* ESs are the *object descriptors* (OD) stream and the *scene description* (BIFS) stream.

a) OD ES for the session description

The location and properties of elementary streams in a presentation are described by a dynamic set of Object Descriptors (ODs). An OD contains one or more ES Descriptors referring to an audio-visual object present in the scene. Similar to any audio-visual content, ODs are transported in a dedicated elementary stream, called the OD stream. Within the OD stream, it is possible through object descriptor commands (OD commands) to dynamically convey, update and remove complete ODs or their ES Descriptors. Updates are time stamped to indicate the instant in time they take effect. An MPEG-4 session description is provided through the complete set of OD AUs present in the OD stream. The binary coding of the OD stream is specified by MPEG-4 Systems.

b) BIFS ES for the scene description

The session description provided by the ODs is accompanied by a dynamic scene description, again conveyed through one or more ESs. This scene description, initially based on VRML, uses a parametric approach (BIFS - Binary Format for Scenes). Content is identified in terms of audio-visual objects. The spatial and temporal location of each object is defined by BIFS. The description consists of an encoded tree of nodes with attributes and other information (including event sources and targets). Leaf nodes in this tree correspond to elementary data of audio-visual objects, whereas intermediate nodes perform grouping, transformation, and other such operations on audio-visual objects. The scene description can evolve over time by using scene description updates (BIFSUpdates commands). The BIFSUpdates commands are time stamped and contained in BIFS AUs. Interactivity mechanisms are integrated in BIFS. They are defined by linked events from sources to targets nodes, called routes, and special nodes, called sensors, that can trigger events on specific conditions.

2.2 Synchronisation Layer

Elementary streams, just after being encoded or just before being decoded, are conveyed as sync layer-packetized (SL-packetized) streams. This packetized representation additionally provides timing and synchronization information. The sync layer (SL) extracts this timing information to enable synchronized decoding and, subsequently, composition of the elementary stream data.

2.3 Delivery Layer

The term delivery layer is used as a generic abstraction of any existing transport protocol stack that may be used to transmit MPEG-4 content. As a wide variety of delivery mechanisms exist inside this layer, MPEG-4 Systems defines just a unique interface (DAI = DMIF Application Interface) to access this layer.

2.4 FlexMux

For applications where the desired transport facility does not fully address the needs of a service according to the number of transport streams, a multiplexing technique, called FlexMultiplexing, has been defined. The FlexMux tool is a flexible multiplexer that accommodates interleaving of SL-packetized streams.

2.5 MP4 File format

The MP4 file format, initially based on *QuickTime* [4], is designed to contain the media information of an MPEG-4 presentation in a flexible, extensible format that facilitates interchange, management, editing, and presentation of the media. The media-data in MP4 is not encapsulated in frames with description headers. The meta-data is used to describe the media data characteristics (media type, times stamps, size …) by reference, not by inclusion. The meta-data is stored in structure, called ‘moov’, while media-data is stored by chunks in a container structure called ‘mdat’. The ‘moov’ meta-data structure is subdivided in substructures named atoms. The offset position of an AU in MP4 is obtained by a computing mechanism using the ‘stsc’ (*Sample to Chunk*), ‘stco’ (*Chunk Offset*), and ‘stsz’ (Sample Size) atoms.

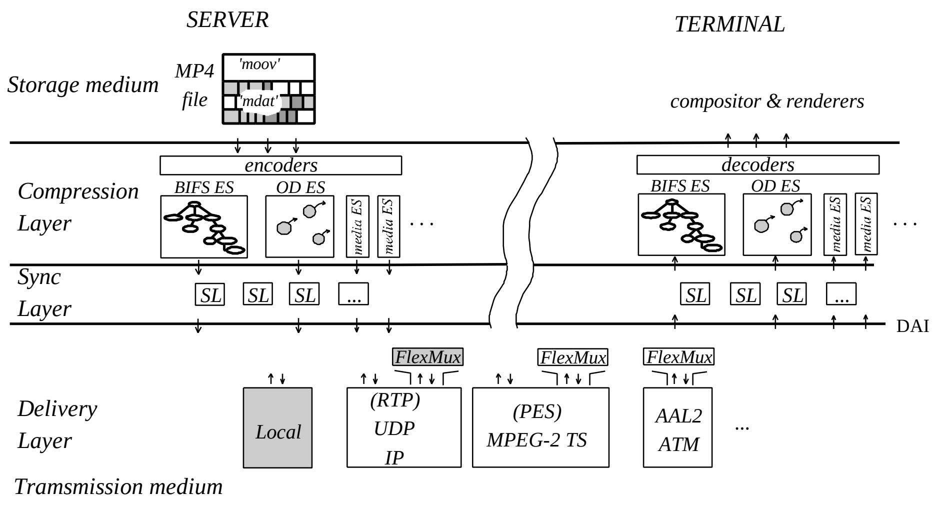

Figure 1: the MPEG-4 Systems architecture

3 MPEG-Pro

3.1 Overview

MPEG-Pro is an authoring software to help authors creating MPEG-4 contents from the end-user interface specification phase to the cross-platform MP4 file. It is based on timeline editing. All content creation principles used in this application match closely MPEG-4 Systems architecture. Every author action is time-stamped and interpreted as BIFSUpdates. Previewing a scene means executing all the update commands sorted by time. The current created MPEG-4 presentation may include more than one scene. Each scene is defined by an ID and its own context (nodes graph, routes list, BIFSUpdates list, OD updates list). First the MPEG-Pro development was mainly concentrated on the authoring of *control* ESs in the Compression Layer and basic storage functionality in MP4. We present here how the implementation of adapted interleaving techniques in MP4 allow the content to be optimised for its use in the Delivery Layer.

3.2 Architecture

a) The Core/Extensions model

The architecture is based on two main modules, named Core and Extensions, as illustrated in the *Figure 2*. The Core module is composed of components that communicate together and with the Extensions. An Extension can communicate only with Core components. If an Extension A needs to communicate with an other one B, it passes through the “actions manager” Core component by defining a new “action”. B is then declared as a listener of this “action”. Each time A needs to communicate to B through this “action”, it will notify the “action” listeners and B will react to the “action” as any listener. This makes possible to add or remove extensions with no damage on the whole application. The user interface components are examples of extensions, so that the functionality of the software is configurable.

At the center of the Core module, the *time manager* is in charge of reading time and reactivating all the components of the whole application that need to be updated at the right time. For example, when the author decides to preview his content by playing a part of it, an object in the *time manager* will force the components that are listeners of *currentSceneChanged* (Scene Editor, BIFS tree viewer) to refresh their visual aspect at each BIFSUpdate command..

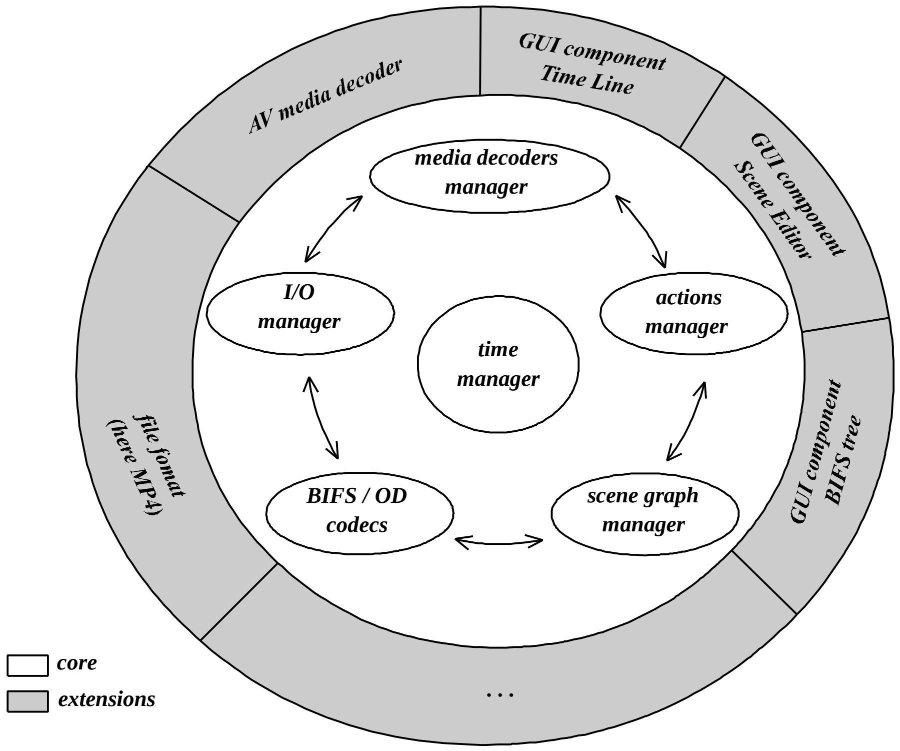

Figure 2 : MPEG-Pro architecture

3.3 Scene description and OD framework

The Core contains functions to author the *control* ESs specified by *MPEG-4 Systems* (OD and BIFS streams). It consists in encoding and decoding BIFS in the OD/BIFS codecs component and maintaining an internal form for BIFS and OD for the scene and updates, exposing a scene modification API (scene graph manager) to the rest of the application. For each scene, a scene controller maintains its BIFSUpdates list and get the ODs of its media nodes from the OD controller. The OD controller maintains a list of the ODUpdates commands. Each author modification is registered as one or several update commands, and inserted after consistency checking in the BIFS and OD controllers.

3.4 Integration of MP4 reader/writer

The *MP4 file format* extension reads MP4 files into memory, provides an API to access the meta-information and the ESs, imports raw streams by creating the needed meta-information, and writes the edited scenes into MP4 files. When reading MP4 content, it provides stored AUs to the *media* (audio/video) and *control* (BIFS/OD) decoders. The *media* decoding is asynchronous and fills the composition buffers used by the compositor and the renderer of the *scene editor*.

4 A method to interleave content in MP4 using MPEG-Pro

4.1 Interleaving principles in MP4

One of the issues to play back an MP4 file in real time is the number of file seeks that must be performed. It is possible to arrange the data in an MP4 file to minimize any seeks during the course of normal playback. This is accomplished by interleaving the ES data in an MP4 file interleaving tool**.** The AUs of an ES are stored in the 'mdat' container MP4 by chunks, while its description is stored in the 'moov' meta-data structure. When the ESs of an MPEG-4 presentation are separate, all AUs of each ES are stored in only one chunk. Interleaving ESs consists in splitting the chunks and mixing them in the 'mdat' container, and reporting the modifications in the 'moov' structure. No method for interleaving is specified by MPEG-4 Systems. Any proposed method must respect the constraints of the MP4 structure.

4.2 Our method

a) Grouping the ESs

Since an MPEG-4 presentation may involve a large number of objects, interleaving all the corresponding ESs may require strategies depending on the interactive scene scenario. So we decided to enable the author to group ES with similar interleaving requirements. From the *Streams List*, the author can create ES groups and add any ES to them by simply dragging and dropping its icon ,as illustrated in *Figure 3*. By default, a newly created ES is first included in the *main* ES group related to the *Streams List.* For each ES group, a specific parameter quantify the degree of interleaving. This parameter, called *MAX_chunkDuration,* fixes a duration limit of the *MP4 chunks* for the *MP4 Tracks* which are associated to the ESs. We define the duration of an *MP4 chunk* as the difference between the DTS of the last and first AU inside the chunk :

$$chunkDuration \ ? \ DTS(lastAU) \ ? \ DTS(firstAU)$$

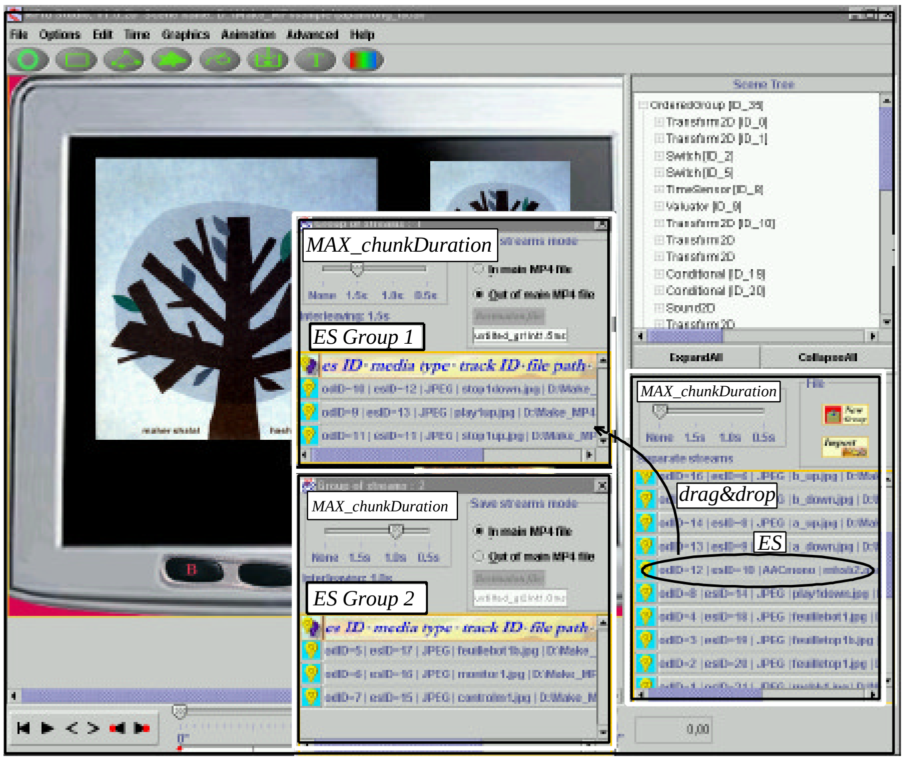
MAX_chunkDuration
ES Group 1
es ID · media type · track ID · file path
MAX_chunkDuration
ES Group 2
es ID · media type · track ID · file path
MAX_chunkDuration
drag&drop
ES
Scene Tree
ExpandAll
CollapseAll
File Options Edit Time Graphics Animation Advanced Help

Figure 3: user interface in MPEG-Pro for grouping ESs

b) Interleaving the ESs in by group

The AUs are interleaved by circularly scanning all ESs that belong to the same group. The ESs are scanned in the order of their appearance in the ES group window. The user can change this order by moving the ESs icons in the list. A weighted round-robin scheme has been implemented to achieve the selection of the AUs from the different ESs. At each cycle, the weight associated to an ES is the number of consecutive AUs to be selected and written in an 'mdat' atom of the MP4 output file. A temporal window whose width is set by the interleaving parameter *MAX_chunkDuration* allows to determine this number. If *cycleNum* is the number of the cycle, an AU is selected if it fulfils the following conditions :

$$(cycleNum ? 1) ? MAX\_chunkDuration ? DTS(AU) ? cycleNum ? MAX\_chunkDuration$$

The *Figure 4* illustrates the round-robin scheme that we use.

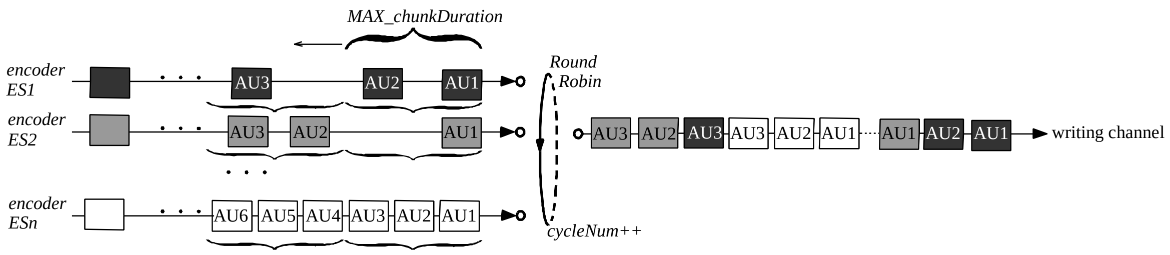

Figure 4 : interleaving scheme for an ES group

c) Storing in MP4

Having grouped and interleaved the ESs, the associated ‘moov’ structure must be calculated. As MP4 gives the possibility to store the data in external files and reference them in the meta-data, the author can choose in MPEG-Pro to store any ES group in a specific file, or he can keep it in the same file as the meta-data. The storing procedure ends by writing separate ES groups in destination files, as illustrated in *Figure 5*, and synchronising the meta-data atoms concerned by the *access units* positions.

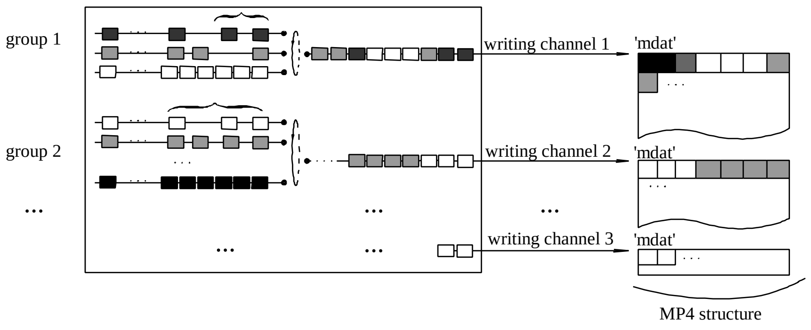

Figure 5 :writing all the groups in a MP4 structure

5 Benefits of interleaving in MP4

5.1 Local scenario

Viewing an MPEG-4 presentation in a local context (i.e. directly from the file and not over streamed SL packets) is an important application. It will be used as soon as MPEG-4 contents will be available on CD or DVD ROM. On such a storage medium where files are never fragmented, interleaving is important because seeking over hundreds of megabytes may slow the reading process and cause player freeze.

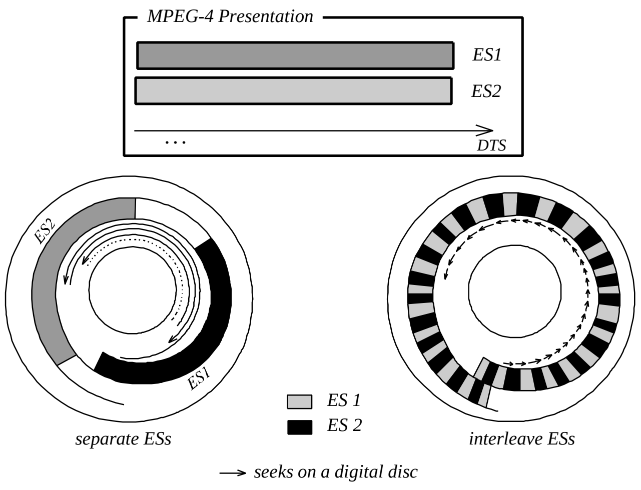

Figure 6 :effect on the seeks

If the content is read from an hard disk which works under the control of an operating system not scrupulous about the files fragmentation, interleaving can loose a bit of its positive effect. But as soon as the hard disk is defragmented, seeks during the course of normal playback will be eliminated. This is illustrated on *Figure 6*. Random access and other kinds of interactivity in the scene require seeks, but some ESs will always be played back at the same time. So grouping and interleaving can minimise the number of seeks.

5.2 HTTP "streaming"

Given the number of HTTP server active over the Internet, it is interesting to try to make use of these to achieve simple streaming without any QoS requirement. To be streamable by HTTP, the file must be constructed in playing order so that the client does not need to download the whole file before starting to play it. When interleaving data, MPEG-Pro puts the MP4 meta-data ('moov' atom) at the top of the file and order the AUs in increasing DTS order. If there are many ES groups, each group is placed in a separate file. If bandwith is sufficient, the terminal may start playing as soon as the a few seconds of buffer are ready for each ES group. The terminal needs to be HTTP aware and to wait for rebuffering if one of the ES groups is "late".

5.3 Flexmuxed stream over RTP

a) Flexmux for RTP

If we examine the existing multimedia streaming solutions that use RTP [5] [6] [7], the presentations do not require a lot of RTP sessions. Usually, they are made of two streams: one for the audio content and one for the video content. MPEG-4 is about to change this situation. Within an MPEG-4 presentation you could possibly have a lot of ESs: one per object of the presentation, plus one for the Scene Description and one for the Object Description. If the same streaming method is kept, i.e. one media stream correspond to one RTP session, this leads to a high-level of complexity when handling all the corresponding RTP sessions. Therefore, multiplexing the ESs into one RTP stream using the FlexMux tool has been proposed by MPEG-4.

The FlexMux tool provides identification of SL packets originating from different elementary streams by means of FlexMux Channel numbers. Each SL-packetized stream is mapped into one FlexMux Channel. FlexMux packets with data from different SL-packetized streams can therefore be arbitrarily interleaved. The sequence of FlexMux packets that are interleaved into one stream are called a FlexMux Stream.

Two different modes of operation of the FlexMux providing different features and complexity are defined. They are called Simple Mode and MuxCode Mode. A FlexMux Stream may contain an arbitrary mixture of FlexMux packets using either Simple Mode or MuxCode Mode. In the simple mode one SL packet is encapsulated in one FlexMux packet. In the MuxCode mode one or more SL packets are encapsulated in one FlexMux packet. The configuration information that defines the FlexMux payload is stored in the MuxCode Table, while the Stream Map Table stores the association between the FlexMux channels and the SL-packetized streams. [3; section 12]. The MuxCode mode is only efficient when the AU within a stream have a constant size, otherwise the simple mode should be used.

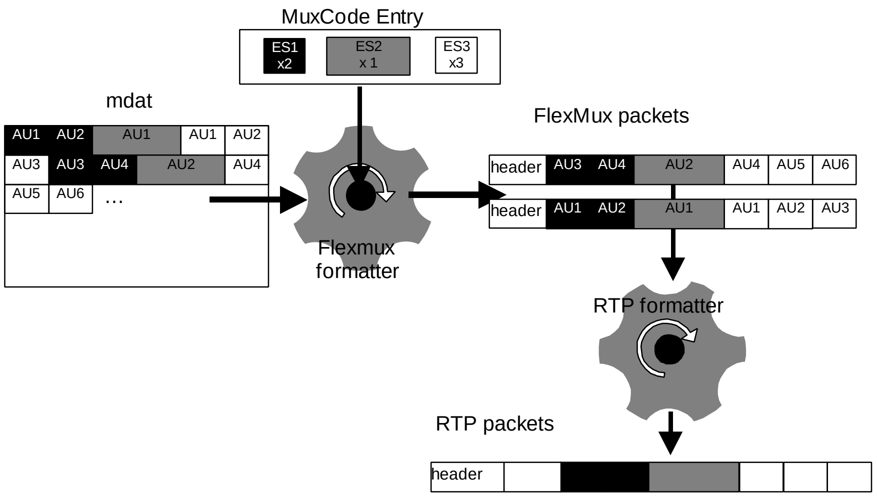

Figure 7: Construction of an RTP packet using the interleaver and the FlexMux tool

Figure 7 illustrates the fact that if the AUs within an ES have constant length and if the MuxCode Table is built in such a way that it matches the way the AUs are interleaved in the MP4 file, then building RTP packets to stream all the ESs using the FlexMux is really easy. The advantage is that, since the AUs of an RTP packet are neighbouring each others, the construction of this latter consists in copying a block of contiguous data according to the MuxCode entrie

b) MP4 for FlexMux

In order to achieve efficiency when creating flexmultiplexed packets, we propose an algorithm to build the MuxCode Table while interleaving the AUs in the MP4 file. This algorithm makes two assumptions. Firstly, the number of AUs of a particular ES chosen during the interleaving phase is piece-wise constant when *cycleNum* increases. Secondly, when this number of AUs is constant, the AUs also have piece-wise constant size. Thus, the algorithm is the following.

For each cycle:

- Determine how many AUs will be chosen for each ES in the group during this phase, using *MAX_chunkDuration*;
- Determine the MuxCode Table entry that should be used to create the corresponding flexmuxed packet, i.e. evaluating the number of ES in this group, the number of AUs per ES, and finally the (constant) size of the AUs of each ES;
- Check if such an entry already exists, i.e. if an entry contains the same ES Ids, with the same number of AU per ES, and the same (constant) AU length for each ES;
- If so, then associate it with this packet;
- If not, create a new MuxCode Table entry and associate it with this packet.

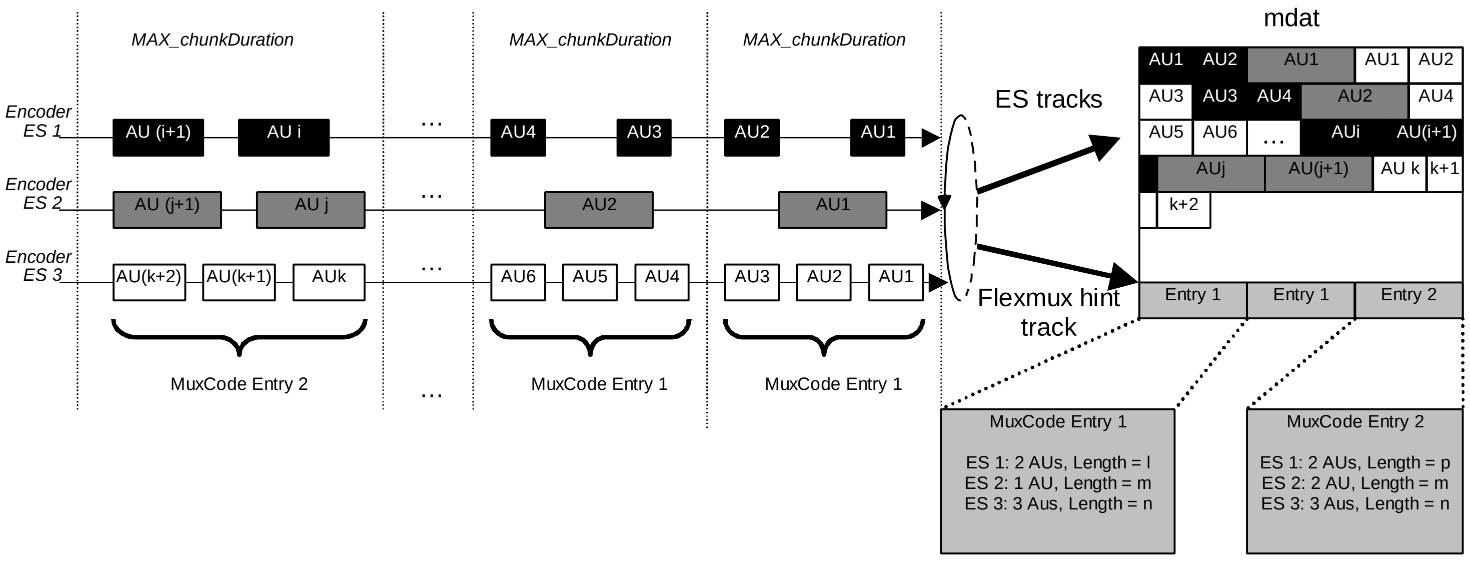

Figure 8: Interleaving an ES group and creating its associated FM hint track

Figure 8 describes how a hint track, defined by MP4 file format, could be used to store the association between the FlexMux packet and its MuxCode Table entry. The hint sample corresponding to a FlexMux packet should contain the index of the MuxCode Entry in the MuxCode table, while the MuxCode table itself should be stored in the 'moov' structure, eventually in the sub-atom 'stsd'.

6 Conclusion

In this document, we presented how a new interleaving tool has been added to MPEG-Pro, our MPEG-4 authoring tool in progress. We explained the advantages of such a tool in three delivery scenarios: local playback, HTTP "streaming" and streaming of Flexmuxed content over RTP/IP, to avoid the overhead of having many RTP streams. Furthermore exploiting the MPEG-4 architecture, we began to study the hinting process to store the Flexmux information. Implementation of the dedicated hinter is pending on the definition of the RTP payload format for Flexmuxed streams which is in progress between the IETF and MPEG.

Bibliography

[1] Boughoufalah, Dufourd, Bouilhaguet (ENST Paris) "MPEG-Pro, an Authorong System for MPEG-4 with Temporal Constraints and Template Guided Editing", Proceedings of the ICME 2000 International Conference, New York, Aug 2000.

[2] Bouilhaguet, Dufourd, Boughoufalah (ENST Paris) “Interactive Broadcast Digital Television- The OpenTV Platform versus the MPEG-4 Standard Framework », Proceedings of the ISCAS 2000 International Symposium, Geneva, May 2000

[3] ISO/IEC 14496-1:2000 MPEG-4 Systems October 2000

[4] Apple Computer Inc. “QuickTime File Format – Publication Draft », June 2000 [5] Microsoft, online http://www.microsoft.com/windows/windowsmedia

[6] Apple Computer, online http://www.publicsource.apple.com/projects/streaming/

[7] Realnetworks, online http://www.real.com/

[5] Schulzrinne, Casner, Frederick, Jacobson RTP: A

Transport Protocol for Real Time Applications RFC 1889,

Internet Engineering Task Force, January 1996.