

Faculty of Engineering of University of Porto



FEUP

**Serviços Web para o Processamento e Gestão de
Conteúdo A/V em Ambientes Profissionais de TV**

**Web services for processing and managing A/V content in
professional TV environments**

Pedro Alexandre Pacheco Cipriano

Thesis submitted under the

Master in Electrical and Computers Engineering

Telecommunications Major

Supervised by: Prof. Dr. Maria Teresa Magalhães da Silva Pinto de Andrade

July, 2008

A Dissertação intitulada

**“Serviços Web para o processamento e gestão de conteúdos A/V em
ambientes profissionais de TV”**

foi aprovada em provas realizadas 18/Julho/2008

o júri

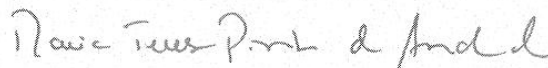


Presidente Professor Doutor Eurico Manuel Elias Morais Carrapatoso
Professor Auxiliar da Faculdade de Engenharia da Universidade do Porto



Professor Doutor José Manuel de Castro Torres
Professor Auxiliar da Faculdade de Ciência e Tecnologia da Universidade Fernando Pessoa

Professora Doutora Maria Teresa Magalhães da Silva Pinto de Andrade
Professora Auxiliar da Faculdade de Engenharia da Universidade do Porto



O autor declara que a presente dissertação (ou relatório de projecto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extractos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são correctamente citados.

Autor - Pedro Alexandre Pacheco Cipriano



Abstract

The work developed within this thesis addresses the increased need of TV stations and post-production houses to reduce the amount of time spent during the ingest process of multimedia material. This problem is specially noticed in places where great amount of multimedia material is produced every day (e.g. Television Networks). By searching and detecting repetitive and common tasks in the various workflows existent all over the world we identify opportunities to develop new tools to improve the speed and efficiency of the process.

The thesis work was carried out as an internship at MOG Solutions, S.A., through the Engineering Faculty of Porto University.

The goal of this internship was to develop tools to facilitate and automate the ingest process of A/V content into video editors. To develop such tools first, a study was made of the APIs provided by the different video editors, gathering at the same time knowledge about how non-linear editors work.

The technologies involved in the development process of these applications implicate subjects such as essence unwrapping of MXF, SOAP communication protocol, XML related technologies and essence encoding details of various types of video and audio essences. The results achieved with the tools developed in this work allowed the automatic ingest of audiovisual content for two non-linear video editors.

Keywords: Television, video, MXF, video editors, workflow automation

Resumo

Este trabalho provém de uma necessidade cada vez maior por parte das estações de TV e casas de pós-produção em reduzir a quantidade de tempo gasto no processo de ingestão de conteúdos multimédia. Este problema identifica-se em especial em locais onde grandes quantidades de conteúdos multimédia são produzidas todos os dias (ex. Estações de TV). Ao procurar pontos repetitivos e/ou comuns nos vários fluxos de trabalho existentes por todo o mundo é nos permitido identificar oportunidades para desenvolver novas ferramentas de forma a melhorar a velocidade e eficiência dos processos envolvidos.

Este documento refere-se ao trabalho feito em estágio na MOG Solutions, S.A., a partir da Faculdade de Engenharia da Universidade do Porto.

O objectivo deste estágio foi, desenvolver ferramentas para facilitar e automatizar o processo de ingestão de conteúdo audiovisual para editores de vídeo. Para desenvolver essas ferramentas, primeiro foi feito um estudo das APIs fornecidas pelos diferentes editores de vídeo reunindo ao mesmo tempo conhecimentos sobre o modo como editores de vídeo não lineares funcionam.

As tecnologias envolvidas no processo de desenvolvimento destas aplicações envolvem assuntos tais como *essence unwrapping* de MXF, protocolo de comunicação SOAP, tecnologias baseadas em XML e detalhes de codificação de vários tipos de essências áudio e vídeo.

Os resultados obtidos com as ferramentas desenvolvidas permitiram a automatização de *ingest* de conteúdos para dois editores de vídeo não lineares.

Palavras-chave: Televisão, vídeo, MXF, editores de vídeo, automatização fluxos de trabalho

Acknowledgments

I would like to thank the company MOG Solutions S.A. for enabling me to undertake this internship and all my coworkers who made a true effort to integrate me as a part of the team and who helped me with this project.

To Prof. Maria Teresa Andrade for helping me every time I had difficulties and for being patient.

To Eng. Orlando Ribeiro for giving me advice and support every time I had a mental block and could not solve a problem and for being patient.

To Eng. Luis Barral and Eng. Paulo Costa for being of great assistance every time I needed technical help and suggestions for the problems encountered during my thesis.

To Eng. Luis Nunes and Eng. Ricardo Oliveira for guidance given and help.

To Eng. Vítor Teixeira for helping me in my internship and also for his help with technical subjects.

And to all the other co-workers a big thanks for all your support.

I would like to thank all my family for the support and strength they gave me to help overcome the various problems and decisions I faced throughout all the internship.

Thank you.

Chapter Contents

Chapter 1 presents the viewer with a general overview of digital television state of the art. This chapter discusses the objectives for this work along with the motivations that led to its execution. A general view of all the steps involved in digital video workflows is discussed, giving an idea of how digital video workflows work.

Chapter 2 gives the reader a general view of state of the art technologies used today in managing, editing and production of A/V content in the professional area. A presentation of some encoding algorithms and techniques is given i.e. the ones necessary for the development of this thesis and others that are used in the industry despite not being important for the thesis. Details about the MXF format are given but only the really necessary topics are described as this is a vast subject.

An introduction to the topic Web Services is made giving an insight about this subject, its applications and the technology used by it. Following Web Services, an introduction to XML is also given.

A review of the QuickTime file format is given, but like MXF, only the necessary details are discussed.

Chapter 3 presents an overview of the first part of the thesis. An introduction explaining its function in the context of digital video workflows is provided. The module developed is dissected by talking about technologies used to develop it and by navigating through its architecture.

Chapter 4 explains and details the second part of the thesis. An introduction to the module is made a description of the various blocks constituting the module is presented. A discussion about difficulties encountered is also provided. An introduction to the XML Interchange format is made and in the end a digital video workflow using this module is illustrated.

Chapter 5 presents the main conclusions of this work. In this chapter I also suggest some future topics of work in this area.

Index

Abstract	V
Resumo	VII
Acknowledgments	VIII
Chapter Contents	X
Figure Index.....	XV
Table Index.....	XVII
Definitions.....	XVIII
1. Acronyms and abbreviations	XVIII
2. Glossary	XX
Chapter 1 – Introduction.....	1
1.1 Motivation and Objectives	1
1.2 Television Digitalization	2
1.3 Digital Video Workflow Overview.....	3
1.3.1 Prepare.....	3
1.3.2 Capture	4
1.3.3 Ingest	5
1.3.4 Management	6
1.3.5 Editing.....	6
1.3.6 Publish.....	7
1.3.7 Deliver.....	8
Chapter 2 - Background	9
2.1 Web Services	9
2.1.1 SOAP.....	10
2.1.2 WSDL	11
2.2 eXtended Markup Language (XML).....	13
2.3 Encoding Technologies	14

2.3.1	Chroma sampling	15
2.3.2	DV – DVC Pro and DVCAM	18
2.3.3	Long GOP MPEG-2 (HDV)	20
2.3.4	AVC- Intra	22
2.3.5	JPEG 2000	24
2.3.6	Interlaced Video	25
2.4	Containers	26
2.4.1	MXF File Format	26
2.4.2	Advanced Authoring Format (AAF)	35
2.4.3	QuickTime File Format	37
2.5	Professional Products	43
2.5.1	Sony XDCAM	43
2.5.2	Panasonic P2 HD	46
2.5.3	Thomson Grass Valley	49
2.6	Digital Asset Management	50
Chapter 3 - Automatic ingest Workflow using Avid Interplay		51
3.1	Workflow Overview	52
3.2	Toboggan EditorBridge Overview	55
3.3	EditorBridge Architecture	57
3.3.1	Create folder	59
3.3.2	CheckIn	60
3.3.3	CheckInAAF	61
3.3.4	Duplicate	61
3.3.5	Move	62
3.3.6	SetAttributes	62
3.4	Summary	62
Chapter 4 – Wrapper from MXF to QuickTime Movie file format		63
4.1	Introduction	63
4.2	Module Overview	64

4.3	Module Architecture	66
4.3.1	MXFToQT Operation	67
4.3.2	MOV Wrapper	68
4.3.3	MXFHandler.....	69
4.3.4	AddEditUnit2QT.....	71
4.3.5	EssenceVideoAudioDescriptors.....	73
4.4	Challenges faced.....	74
4.4.1	Image Description Structure	74
4.4.2	Sound Description Structure.....	75
4.4.3	Closed GOP or Open GOP	75
4.5	XML Interchange Format.....	76
4.6	Workflow Example	79
4.7	Summary	80
Chapter 5 – Conclusions and Future Work		81
5.1	Final Remarks.....	81
5.2	Future Work.....	82
References and Bibliography.....		85
Appendix A - Video Recording Formats.....		89

Figure Index

Figure 2.1 - Web Services Architecture [13].	10
Figure 2.2 - Interfaces and bindings [12].	12
Figure 2.3 – XML document example.	13
Figure 2.4 - Tree structure of a XML document.	14
Figure 2.5 – Evolution of compression technology [15].	15
Figure 2.6 – Pixel components [16].	16
Figure 2.7 – Chroma sampling in 4:2:2 [16].	17
Figure 2.8 - Chroma sampling in 4:1:1 [16].	17
Figure 2.9 - Chroma sampling in 4:2:0 [16].	18
Figure 2.10 - DCT-based encoder simplified diagram [18].	19
Figure 2.11 - Data structure of one video frame for 50 Mb/s structure [19].	20
Figure 2.12 – Presentation order and storage order.	21
Figure 2.13 - Relative picture quality of various video compression algorithms [20].	22
Figure 2.14 – Examples of spatial intra prediction on H.264 [15].	23
Figure 2.15 – JPEG 2000 block diagram [21].	24
Figure 2.16 – Frame composition.	25
Figure 2.17 - Overall data structure of a simple MXF file [6].	27
Figure 2.18 - Overall data structure of an MXF file with optional components [6].	27
Figure 2.19 - KLV Encoding [1].	28
Figure 2.20 – Operational Patterns [6].	30
Figure 2.21 – Frame Wrapping example [24].	31
Figure 2.22 – Frame wrapping with other elements contiguously attached [24].	31
Figure 2.23 – Video stream clip wrapping example [24].	31
Figure 2.24 – Clip wrapping with other elements [24].	32
Figure 2.25 - Detail of the package class inheritance of MXF [6].	33
Figure 2.26 - References between packages [6].	34
Figure 2.27 – Relationships between metadata frameworks and the content of MXF file body [25].	35
Figure 2.28 - AAF Object Model and Storage Layer.	35
Figure 2.29 – QuickTime movie structure and time measuring [27].	38
Figure 2.30 – QuickTime movie with multiple tracks [27].	39
Figure 2.31 – Example of a media mapping from a track [27].	40
Figure 2.32 – Media referencing to a storage device [27].	40

Figure 2.33 – The two types of QuickTime movie files [28].	41
Figure 2.34 - Some commonly used tool sets and components [28].	42
Figure 2.35 - Workflow example using Premiere and XDCAM [33].	46
Figure 2.36 – Content structure on the P2 Card [15].	47
Figure 2.37 – Workflow example using Premiere and P2 [38].	49
Figure 3.1 – Avid Interplay overview.	52
Figure 3.2 – NBC workflow overview.	53
Figure 3.3 – Close up view of the workflow using Toboggan EditorBridge.	55
Figure 3.4 – Module overview.	56
Figure 3.5 – Diagram workflow for client application development [41].	57
Figure 3.6 – Architecture diagram of Toboggan EditorBridge.	58
Figure 3.7 – Parsing XML input message with MOG tools.	58
Figure 3.8 – Interplay Access application.	59
Figure 3.9 – Folder structure.	60
Figure 4.1 – Final Cut Pro import menu.	64
Figure 4.2 – Overview of the MXFToQT module.	65
Figure 4.3 – Simplified rewrapping process.	66
Figure 4.4 – MXFToQT architecture overview.	67
Figure 4.5 - Block diagram of MXFToQTOperation.	68
Figure 4.6 - Block diagram of MOV Wrapper.	69
Figure 4.7- Block diagram of MXFHandler.	70
Figure 4.8 - Typical KLV coding in an essence container [6].	70
Figure 4.9 - Block diagram of AddEditUnit2QT.	72
Figure 4.10 – QuickTime function to add samples to a track.	72
Figure 4.11 – MXFToQT using XML Interchange Format.	76
Figure 4.12 – Encoded Clip in XML Interchange format.	78
Figure 4.13 – Workflow example using MXFToQT.	79

Table Index

Table 2.1 - Differences between IEC 61834-2 and SMPTE 314M [19].	19
Table 2.2 – Comparison between Intra-only and Long GOP (Inter-frame) [15].	21
Table 2.3 – Comparison between Entropy Coding Methods [15].	23
Table 2.4 – General differences between AAF and MXF.	36
Table 2.5 - XDCAM HD recording specifications [32].	44
Table 2.6 - Infinity video compression specifications [39].	50
Table 3.1 – Examples of name-value metadata pairs.	60

Definitions

1. Acronyms and abbreviations

AAF: Advanced Authoring Format.

API: Application programming interface.

CABAC: Context Adaptive Binary Arithmetic Coding.

CAVLC: Context Adaptive Variable Length Coding.

CBR: Constant bit rate.

CCD: Charged Coupled Device.

CIE: *Commission internationale de l'éclairage*.

CMOS: Complementary Metal Oxide Semiconductor.

COM: Component object model.

CRT: Cathode ray tube.

DAM: Digital asset management.

DCT: Discrete cosine transform.

DIF: Digital Interface.

DOM: Document Object Model.

DRM: Digital rights management.

EDL: Edit Decision List.

FCP: Final Cut Pro.

GOP: Group of Pictures.

HD: High Definition.

ISO: International Organization for Standardization.

KAG: KLV Alignment grid.

KLV: Key Length Value.

MTOM: Message Transmission Optimization Mechanism.

MPEG: Moving Picture Experts Group.

MXF: Material Exchange Format.

NTSC: National Television System Committee.

PAL: Phase alternate line.

PCM: Pulse-code modulation.

QTFF: QuickTime File Format.

RPC: Remote Procedure Call.

SD: Standard Definition

SDK: Software Development Kit.

SOAP: Simple Object Access Protocol.

UDDI: Universal Description, Discovery and Integration.

UL: Universal Label.

URI: Uniform Resource Identifier.

URL: Uniform Resource Locator.

VBR: Variable bit rate.

WSDL: Web Services Description Language.

XML: eXtended Markup Language.

2. Glossary

Provided below are well defined meanings for some expressions and terms which need to be understood by the reader to allow him/her to follow and completely understand all the subjects mentioned in this document.

Essence: “an abstract term that describes any data or signal necessary to represent any single type of visual, aural or other sensory experience independent of the method of coding. Also identified by the SMPTE/EBU “Task Force for Harmonized Standards for the Exchange of Program Material as Bitstreams” (TFHS) as Video, Audio, and/or Data information. Essence can also be Graphics, Telemetry, Photographs, or other information.”[1].

Metadata: “Generally referred to as “data about data” or “data describing other data”. Metadata is information that is considered ancillary to or otherwise directly complementary to the essence. Also any information considered useful or of value when associated with the essence.”[1].

Workflow: describes a reliably repeatable sequence of operations.

Digital switchover: refers to the migration process from analogue to digital broadcasting[2].

Datacasting: refers to the broadcast of data over a wide area via radio waves or additional information sent by digital TV broadcasters alongside the TV signal.

Tapeless recording: refers to any recording method using digital data recorded to a medium other than tape, normally flash media, hard drive disks or optical media.

Assets: Anything of material value or usefulness that is owned by a person or company[3].

Intra-frame coding: compressing redundant areas within a single video frame[4].

Inter-frame coding: the coding of the differences between frames[4].

Plug-in or plugin: small program that adds some functionality to a bigger program.

FireWire: refers to a serial bus interface standard, for high-speed communications and isochronous real-time data transfer.

Chroma (video chrominance): refers to “the colorimetric difference between any color and a reference color of an equal luminance, the reference color having a specified chromaticity”[5].

Luma (video luminance): refers to “an objective measure of the visible radiant flux, weighted for color by the CIE Photopic Spectral Luminous Efficiency Function”[5].

Best effort: “Best effort metadata is a category of metadata which may not be known at the time of writing a file. Best efforts should be made to complete these values during file creation. Only metadata parameters marked specifically as best effort metadata may be treated this way. If the value of the parameter cannot be completed correctly then its distinguished value shall be used.”[6].

H264: refers to the MPEG specification ISO 15114-10.

HDV: refers to the high definition video recording format developed by Sony and JVC that uses MPEG2 compression.

Modular programming: refers to a software design technique that increases the extent to which software is composed from separate parts, called modules.

FourCC: refers to a four-character code with 32-bit value that is usually best interpreted as four ASCII characters, it is used to uniquely identify data formats.

Edit rate: “A rational number that specifies the units used to specify the duration of components in a track. In a track which describes an essence container, the edit rate is usually chosen to be the number of editable units per second.”[6].

Edit Unit: An edit unit is a period of time equal to 1/edit rate or the smallest unit of information of a MXF file.

Frame or picture: refers to one of the many still images which compose the complete moving picture. “For interlaced video, a frame consists of two fields, a top field and a bottom field.”[7].

Frame rate: “the rate at which a movie is displayed - that is, the number of frames per second that are actually being displayed.”[8].

Field: “For an interlaced video signal, a "field" is the assembly of alternate lines of a frame. Therefore an interlaced frame is composed of two fields, a top field and a bottom field.”[7].

Interleaving: “refers to a technique in which sound and video data are alternated in small pieces, so the data can be read off disk as it is needed.”[8].

Time base: “refers to a set of values that define the time basis for an entity.”[8].

Chapter 1– Introduction

This chapter introduces the scope in which the work developed is inserted. The objectives to be achieved are listed alongside their justification.

An overview of digital video workflows, and an introduction to the tools developed within the objectives proposed for this thesis is given hereinafter.

1.1 Motivation and Objectives

This sub chapter presents the main motivations and objectives behind this work. With the emerging of new technologies in the field of digital A/V media production, business opportunities have arisen to create products to cope with the newly faced challenges. This work is focused mainly in developing technology to assist and automate tasks in most of the workflows existent in digital media production environments. The objective of this thesis is to develop new tools to solve many shortcomings and eliminate repetitive tasks undertaken by the end user, more specifically on the workflow comprising media acquisition to ready-to-edit media on video editors.

With the change from analog to digital video in Television, many business opportunities arose due to the introduction of completely new workflows. A study effort to find out how the digitalization is going to affect the broadcasters and what vulnerabilities they may have was also executed. The thesis' main focus will be around tapeless based workflows (also called file based workflows). This is an emerging trend in the broadcasting post-production area. While most of the audiovisual content in this field is already digital, post-production workflows are still tape based instead of being tapeless.

The work required to prepare this thesis involved a study of the market to see which solutions the competitors had to offer in this field. Some decisions were made in the beginning and during this thesis based on this knowledge and using it to add real value and functionality to

Introduction

the applications created. This allowed a more aggressive competition with other solutions and options offered by others. This paper attempts to identify factors and choices made throughout the development process describing why and how they will improve television operators and post-production houses workflows.

The motivation to this work has been fostered by MOG Solutions with the perspective of new business opportunities, and by combining their knowhow and experience in this area with the results proposed for this thesis. Being in permanent contact with television operators, hardware manufacturers, system integrators and software producers from all over the world, allowed the company to acquire valuable knowledge to successfully understand the true needs of the customers and always stay up to date. A key advantage when faced against the competitors is their stable and award winning MXF SDK.

Two new modules were developed which will add new functionality to the MOG Solutions Toboggan product, which met the objectives of this thesis. These additions will allow a complete automation between media acquisition to ready-to-edit media on non-linear video editors.

The first module analyzed in this thesis allows the automation of the ingest process of audiovisual media into the Interplay family of non-linear video editors. This module ingests the A/V media into the Interplay data base, making it rapidly accessible from an Avid video editor.

The second module developed enables the MOG Solutions Toboggan product to support the non-linear video editor Final Cut Pro. The module does so by creating QuickTime files from MXF files, maintaining their essence intact (no transcoding is done throughout the process). At the same time metadata is read from the MXF files and is appended to the new QuickTime files. This module is important because Final Cut Pro does not support MXF files natively, and future plans for adding support for it have not yet been announced. So it becomes a necessity to have something that creates files that are supported by the application.

The next subsection provides some insights into the present situation of the television content production chain.

1.2 Television Digitalization

The inevitable change to digital television is one of the main reasons why this thesis exists. Sooner or later every country will only broadcast in digital form. For example “on

February 17, 2009 all full-power broadcast television stations in the United States will stop broadcasting on analog airwaves and begin broadcasting only in digital”[9].

The advantages gained with this change come from the way digital signals are formed: as they only have a finite number of discrete values, they are more robust to noisy conditions.

By being a digital signal, better compression algorithms can be used, obtaining much higher compression ratios than their analog counterpart thus, reducing the amount of bandwidth necessary to represent the same content. “In the case of Television broadcasters this will permit the release of several hundreds megahertz (MHz) in the VHF and UHF frequency bands, which could be reallocated to various uses, for instance convergent services combining features of mobile telephony and terrestrial broadcasting, such as mobile 'datacasting'.”[2].

The digitalization process offers other additional benefits apart from the ones indicated above such as the possibility of end user interactivity.

Due to this change in television all other stages followed and became digital as well. As a result all the content is now preserved in digital form thus creating repositories of digital media. Because of this, solutions for digital media management had to be created. For further explanation of media management see 2.6.

1.3 Digital Video Workflow Overview

In this sub chapter a general and brief explanation of all the steps involved in digital video workflows will be presented. The workflow overview presented here tries to be as much agnostic as it can be, providing the reader with an objective and non specific explanation of all the tasks involved in the process. While this tries to be as general as possible some steps can exist only in some real case scenarios while others will not exist in all cases. The different tasks will be divided in groups related to their position in the digital video process chain.

1.3.1 Prepare

“Capturing video clips is often thought to be the first step in the video creation process.”[10]. If professional results are expected from any shooting sequence then for sure the first step is not making the actual shoot but is essentially planning all the details necessary to start shooting. The preparation is subjective and is dependent on what artistic preferences the

producer has. A good example of this preparation happens for example in big sports events such as football matches, a good amount of time is actually spent preparing the cameras and shooting angles and sequences to provide the viewer with a better experience.

In some situations media capturing happens automatically or is triggered by events and in this case no human intervention is needed (e.g. surveillance cameras).

Cinema is a good example of how important this step is as normally the pre-production phase takes quite some time. Although in cinema other things are taken into consideration, making the pre-production phase longer.

1.3.2 Capture

This is maybe the most important and unforgiving phase in any video production. Any mistake made throughout this stage will be of difficult fix or even impossible to undertake. “In this phase the need for skill and talent is not going to become obsolete.”[10]. Any kind of video capture equipment will consist of an imaging sensor, means to save the data and a logic component to control the capturing process. See below a brief explanation of these elements.

1.3.2.1 Storage

In the past all video material made was recorded on magnetic tape, although later on the recording was already digital, ingesting of the material produced was still time consuming and required a lot of resources. Nowadays as prices go down the trend is to move to tapeless storage. On tape based digital recording we have nowadays various formats being used such as DV, DVCAM, and DVCPRO. In the tapeless professional recording market we already have solutions based on optical disks (e.g. Sony XDCAM products use blue ray disks to save recorded material) and solutions based on flash media (e.g. Sony XDCAM EX, Panasonic P2) . To see in depth detailed description about XDCAM and P2 see Chapter 2.

It is expected that in the future all the different storage mediums will converge to some form of non-volatile computer memory (e.g. flash memory). Their price is going down rapidly and they offer advantages against other types of media, such as faster read access times, faster transfer rates, better resistance to shocks and a good ability to work in rough environments.

1.3.2.2 Sensors

This component is essentially an analog to digital converter, transforming rays of light into digital values that can be then interpreted and processed. The most common types of sensors in the market are CCD and CMOS. “Currently the most advanced sensors can reach 4k

and 8k which translates to 4329 or 7680 pixels lines in the pictures”[10], more than enough to record in HD resolutions (1080p or 720p).

1.3.2.3 Logic

The logic part of a camera consists of all the interface the equipment provides the user with to enable him/her to control its functions, and the internal logic used to process the signal received. The controlling interface can be physical (e.g. buttons on the equipment) or software based.

Physical control interfaces are in vast majority very simple and only allow basic control of the capture process.

Interfaces controlled via software offer much more variety, options like motion detecting software and automatic focus on predefined places on the image the equipment is capturing, are just some examples. The equipment can also provide functionality to automatically improve the image using algorithms for pixel correction and motion compensation. Functions like autofocus and digital image stabilization are also available nowadays in a great variety of equipments.

It is expected that more advanced controls will be provided and that more advanced algorithms will be used in the future as technology prices lowers.

1.3.3 Ingest

Ingest is the process by which the captured material is fed to some storage system to become available for the next step of the workflow. This process is relevant because most of the system’s components involved in the digital video workflow are distributed, be it locally distributed (e.g. same machine) or distributed in different remote machines connected by a network of some sort.

Most steps involved in the making of captured raw material into the final product require some sort of ingest process. It can only be a transcoding operation or just an update of the metadata associated with the material.

With recent network developments most of the transfers between systems is now made a number of times faster than playback speed.

It is important that metadata is preserved in all tasks involving ingest operations. This metadata can be transferred using the protocol on top of the transport layer or simply embedded

the metadata onto the files. MXF format offers good solutions to these problems as it has the possibility to embed a lot of metadata information on the file itself and supports a wide variety of differently encoded essences. For example if various ingest operations need to be done in one workflow then using MXF in all of them assures that at least you know that even if the essence is changed, the file structure will always be the same.

Finally, the ingest process has to be well thought because some changes in the process can be irreversible therefore making it impossible to recover the initial state of the content unless previous data backup had been made.

1.3.4 Management

“Managing video is just storing the captured data in an organized manner.”[10]. “Content Management System (CMS) is a term coined for software components that help us handle the jumble created by accumulating digital assets.”[10].

Content management is important because it provides the necessary functionality to efficiently catalog material and find it in an easy way. Organizing the content chronologically is one of the most used ways to do it. While this may suit some of the demands, it is clearly not adequate to perform more complex searches. Instead of chronologic organization, it is preferred to use metadata tags associated with assets. If the metadata associated with some video/audio is unique then it can be retrieved from the database with little effort rapidly and unequivocally. These metadata tags can also be used to add descriptive information to assets, so that when searching for specific assets one can use descriptive terms as a search item. Some non-linear editors already provide solutions which attempt to integrate DAM directly into editing tools, see Chapter 2 for an overview of some solutions available in the market.

1.3.5 Editing

Editing of audiovisual content is the process of “transforming raw video footage by manipulating the captured data.”[10]. Adding effects, encoding, combining new clips or making cuts to the video material is all part of this stage.

Hardware dedicated solutions have been for a long time the only practicable solution to assist and speed up these operations to acceptable time periods. Nowadays, with the power offered by mainstream processors, dedicated hardware is not a necessity anymore. Therefore,

making life easier for newcomers and giving individuals the opportunity to try and work with this process with little or no investment at all.

There are various software solutions on the market which offer a good variety of options to edit the content captured. This thesis will focus on the three major vendors of non-linear editors, Final Cut Pro from Apple, MediaComposer/NewsCutter from Avid and Premiere from Adobe.

1.3.6 Publish

“Publishing is the art of selecting the right content for the accessible delivery medium.”[10]. This is the stage where the decision of what to publish is taken and it requires a big resourceful effort to filter out bad quality material from good one. While some automatic mechanisms already exist to help in the daunting process of selecting what is of good quality, the process is still in the vast majority made by hand. This is usually done by having a group of people reviewing all the material one by one to approve or disapprove which ones fulfill the requirements for publishing.

Publishing is greatly influenced by the audience it targets and by numbers, after all the most important end result is the number of people who see the published content. At the present time the internet is changing the way publishing is done. With the possibility of publishing any material with reduced costs, the variety is much bigger and with the current search tools the internet provides the end user with the possibility to find what he/she wants in a matter of seconds, and from a huge amount of content. This means that control is now being passed on from the big companies to the end user, especially because the user now has the possibility to see only what he/she wants instead of being “forced” to see what companies want.

Publishing is also influenced according to the media where it is going to be delivered. For example if the delivery is in physical form (e.g. DVDs, CDs) the limitation is only in the capacity provided by the storage media. On the other hand if delivery is made through a mobile media device, the bandwidth constraints imposed will cause the content to be reduced as much as possible, therefore preserving only the important information and discarding the superfluous information.

Another item pertinent to this stage is how to protect the content as everyone has access to it and can reproduce it easily. DRM seems to be the solution but the future is still unpredictable with respect to the outcome of this solution.

1.3.7 Deliver

Delivery is also an important part of the digital video workflow as this is where the decision is made on how the content will be delivered and in which format. Nowadays delivery can be done either by broadcasting, web, p2p, laser discs, hard disks and memory cards which will inevitably have strengths and weaknesses.

The expenses in delivering video content are falling rapidly and many companies are already providing content in a multi channel delivery system, thus trying to appeal to a broader audience. Nevertheless, the future will be on-demand delivery and the winners will be the ones who manage to master this delivery system.

Chapter 2- Background

This chapter presents the reader with a background to the technologies used throughout the development process. Some concepts which had to be understood in order to allow the development of the work done in this thesis are also given.

2.1 Web Services

An important part of this thesis concerns the use of Web Services technologies to integrate new components with already existing ones and distribute the load between different machines.

In this chapter a brief introduction to the subject web services is made. Web services provide default means for interoperability between different software applications running on different platform types and operative systems. Its basic architecture is defined via a group of services exposed on a server, which in turn can be consumed by a remote client application.

Web Services constitute an excellent way to enable inter-operation between applications on the Internet. This is partly due to the fact that Web Services are based on commonly known technologies such as HTTP and XML, as well as on proven and widely adopted approaches from other middleware technologies, such as the case of the services-oriented approach of CORBA. Web Services technology is in fact a compilation of technologies: XML, eXtensible Markup Language ; SOAP, Simple Object Access Protocol; WSDL, Web Services Description Language ; and UDDI, Universal Description, Discovery, and Integration.

Web Services communication is made essentially using the SOAP protocol[11], although other communications' protocols exist, the most used one is undoubtedly SOAP. SOAP protocol encodes its messages in XML format defining explicitly the rules of their construction.

Background

Web Services promote and facilitate the use of distributed systems. If automatic creation of client applications is desired then WSDL [12] is required to be provided alongside the web service.

Another specification of web services is UDDI. The utility of this protocol is to expose metadata about the web services with the purpose of enabling and facilitating its discovery from other applications.

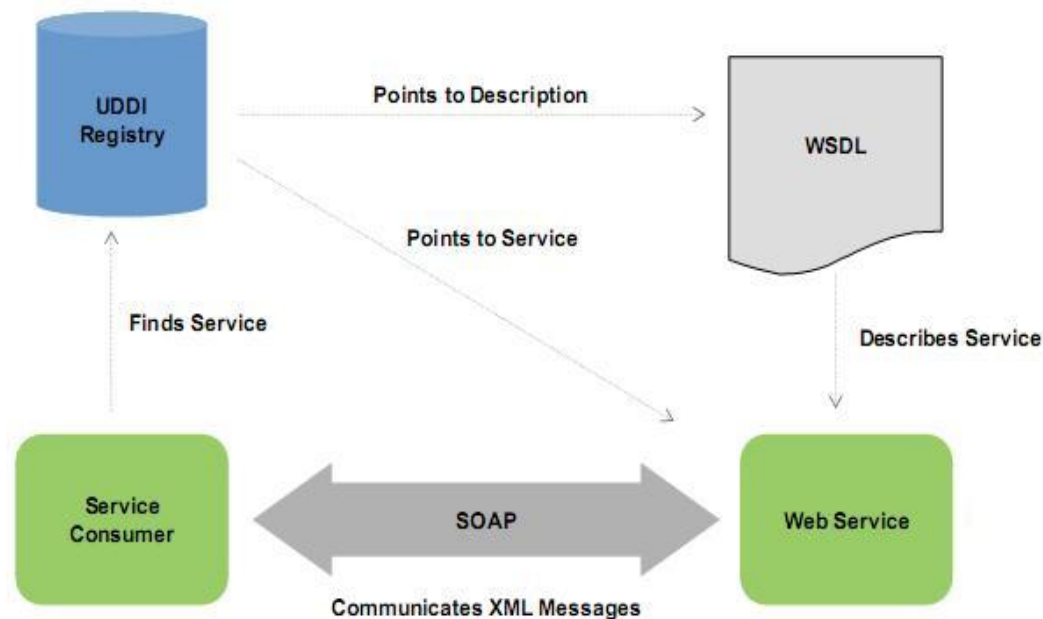


Figure 2.1 - Web Services Architecture[13].

2.1.1 SOAP

“SOAP is an extensible XML messaging protocol that forms the foundation for Web Services.”[13]. Basically it is an information exchange protocol that allows one application to communicate with another, remotely, by using an XML message.

SOAP consists of four parts, one envelope where a description of the message contents and how to process it is, one SOAP body, one SOAP transport binding framework and a SOAP serialization framework. The envelope is constituted by a SOAP header and a SOAP body. The header provides extensible functionality such as authentication information.

The SOAP body contains the important information which is going to be transported. The SOAP transport binding framework defines bindings for specific protocols, and the SOAP serialization framework provides tools for the payload to be mapped or bound directly to data types in the host language.

SOAP has the advantage of being possible to use in a wide variety of transport protocols such as HTTP, TCP amongst others. HTTP is the most used one because it is less problematic when used in conjunction with firewalls.

Because SOAP uses XML to send messages, the protocol is not well suited to send big amounts of data. This problem is attenuated by using the MTOM mechanism to improve the performance of big data transfers. MTOM does so by reducing the overhead introduced with the use of Base64 to encode data.

2.1.2 WSDL

WSDL is a vocabulary based in XML which describes a model for web services. WSDL defines the functionality a web service offers, how the communication is done and where the service is accessible. WSDL provides all the details an application needs to use a web service. Using WSDL provides means to automatically generate a SOAP interface that can be used by an application. Using this concept creating applications that use a web service becomes much easier if the web service provides a WSDL.

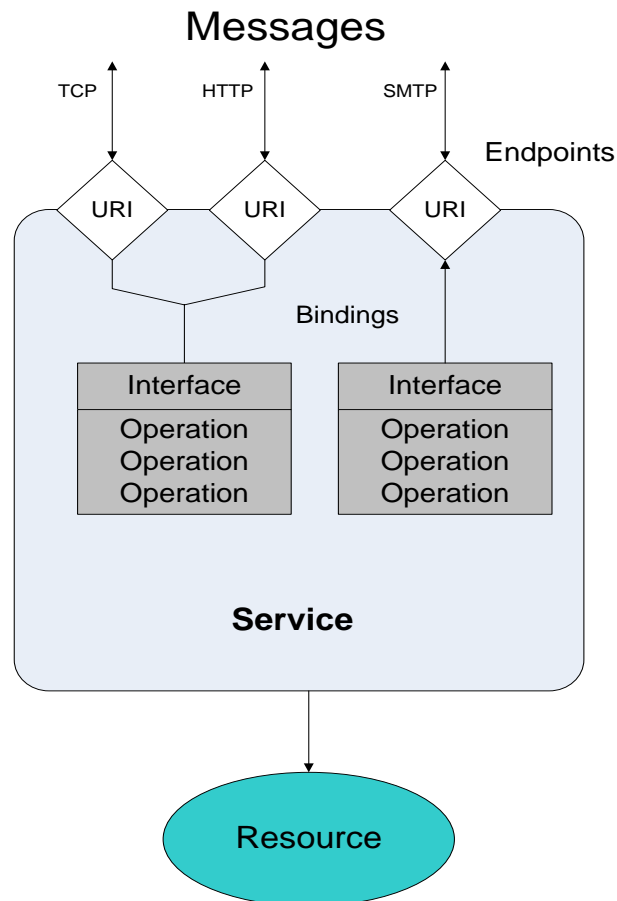


Figure 2.2 - Interfaces and bindings[12].

Brief description of the elements in Figure 2.2:

Endpoints – Specifies one address for one binding.

Bindings – Defines a concrete protocol and data format for a specific portType.

Port Type - Defines a set of operations. Using soap RPC messaging pattern each operation corresponds to a request-response operation.

Service - A collection of related endpoints, where an endpoint is defined as a combination of a binding and an address (URI)[12].

Messages – Abstract definition of the data to be interchanged.

So in short, WSDL defines all the characteristics of a Web Service. WSDL describes the messages being exchanged in the communication, the types of the exchanged information and the protocol being used (i.e., the syntax and semantics of the messages). It also provides the identifier of the Web Service (URI).

2.2 eXtended Markup Language (XML)

XML has been used throughout the thesis in a wide variety of occasions. This chapter introduces the XML language.

XML is a simple and flexible text format that was designed to meet the challenges of large-scale electronic publishing[14]. XML does not do anything, it is simply a text document governed with a set of syntax rules with the purpose of storing and transporting information. XML stores information but does so by attaching a description explaining the context of that information.

Because XML is just plain text, interoperability problems are minor and almost any system can read text files. XML can be used as a way to share information between different systems or applications. It can be used even for storing data which an application needs to read several times (e.g. databases).

The structure of an XML document is quite simple and will be explained hereinafter. Figure 2.3 shows the XML syntax.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<xmeme1 version="3">
  <project>
    <name>Random name</name>
    <children>
      <clip id="Random name">
        <updatebehavior>add</updatebehavior>
        <name>Random name</name>
        <duration>93</duration>
      </clip>
    </children>
  </project>
</xmeme1>
```

Figure 2.3 – XML document example.

XML is formed by elements. Each element has a start tag and an end tag, with a name defined by the user or application. An element can contain other elements within. On Figure 2.3 a start tag is for example <clip> and the corresponding end tag is </clip>. An element can have, optionally, an arbitrary number of attributes. These are indicated in the start tag with a name and a corresponding value enclosed in quotes. For example in Figure 2.3 the start tag <clip id="Random name"> has an attribute "id" followed by its value "Random name".

Background

A generic XML document contains a tree-based data structure where each element has a parent element (excluding the root element) and, optionally, child elements. Figure 2.4 shows the tree structure of the XML document presented in Figure 2.3.

The first element in Figure 2.3 is an optional XML declaration which states what version of XML is in use (e.g. 1.0 on Figure 2.3), the information about character encoding (e.g. UTF-8 on Figure 2.3) and the external dependencies.

The following element is a root element. This element must be present in all XML documents. Following it an arbitrary number of elements can be created depending on the requirements.

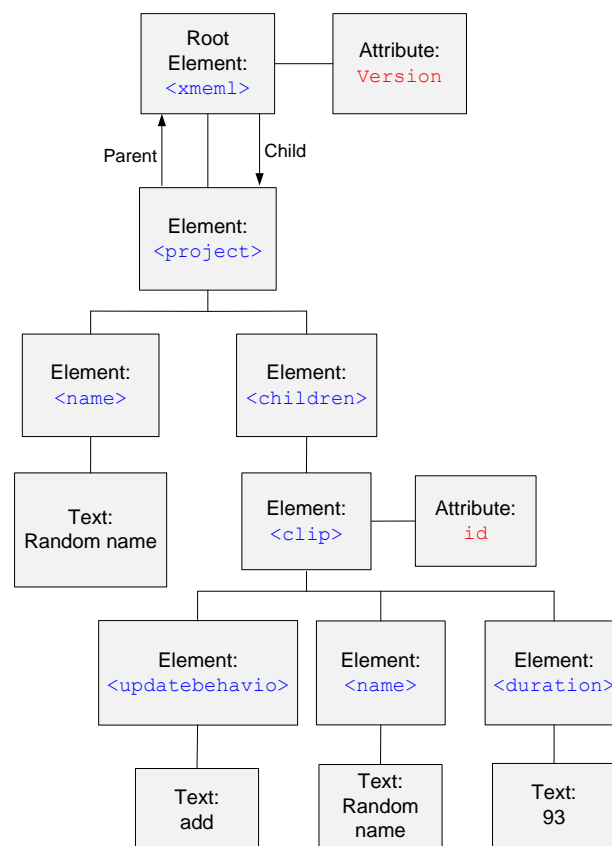


Figure 2.4 - Tree structure of a XML document.

2.3 Encoding Technologies

This chapter presents various compression techniques used by the different digital video production manufactures, focusing on the ones used in the professional market. These techniques were created to solve the problem of big data sizes raw video would need to present

an image with acceptable quality. Without these compression algorithms, digital video production would be unsustainable with requirements well beyond what the latest hardware and network equipment can provide in the present day. See Figure 2.5 for an overview of how compression technology has evolved throughout the years.

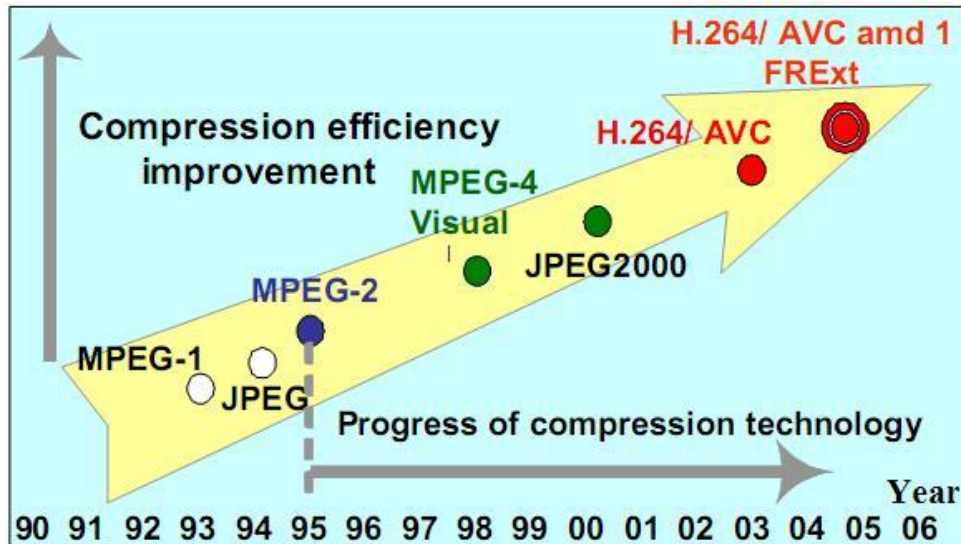


Figure 2.5 – Evolution of compression technology [15].

2.3.1 Chroma sampling

“Component digital video is often referred to as YUV, but this is not an accurate description of component digital video, but instead refers to a set of intermediate qualities used in the formation of analogue composite video. The correct terminology for the components of digital video is Y' Cb Cr. The Y' represents luma, the ' is important, reminding us that it is non-linear (gamma corrected). The Cb and Cr are chroma components that are often wrongly referred to as U and V. The Cb is calculated as $B' - Y'$, and Cr is $R' - Y'$.”[16].

“The CCDs that capture our image inside our video camera capture color by the use of Red, Green, and Blue filters. The phosphors in our CRT monitor, and filters in our LCD monitor and DLP projector also use Red, Green and Blue filters. Separate luma and chroma are used so that the resolution of the chroma can be reduced with respect to the resolution of the luma so that large savings can be made in the amount of data that needs to be transmitted - it is a form of compression. These reductions in the resolution of the chroma components works perceptually as our human vision systems are not as able to see fine details in color as we are in lightness.”[16].

There are various types of chroma sampling. In this section, I will only analyze those relevant to this thesis. The chroma sampling scheme is commonly expressed as a three part

Background

ratio, Chroma sampling ratio (relative to first digit) is defined by the second and third numbers while the luma sampling ratio is defined by the first number.

4:4:4

Full resolution luma is represented by the number 4, and as the chroma components C_b and C_r are also 4, there is no reduction in resolution. 4:4:4 sampling is mostly used for RGB images, although it can be used for Y'CbCr, even though no camera records 4:4:4 Y'CbCr.

So in this case the resolution for each of the three components is equal, meaning that for each color pixel there is one pixel for each of the three components constituting it, see Figure 2.6.

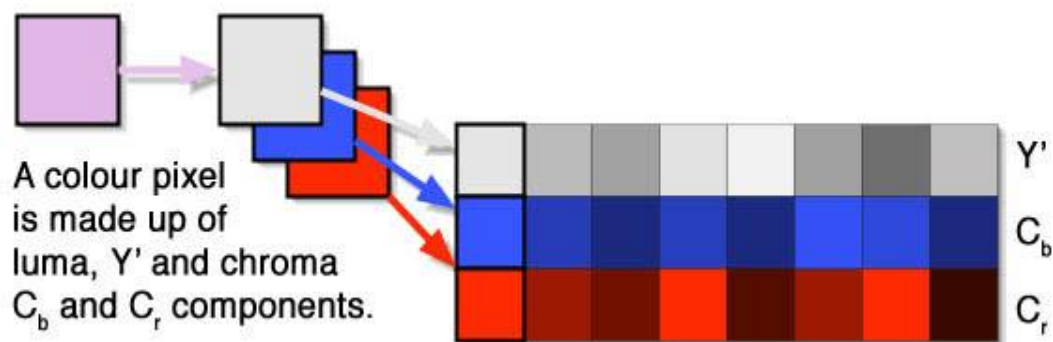


Figure 2.6 – Pixel components [16].

4:2:2

Full resolution luma, and half ($2/4 = 0.5$) resolution horizontally on the chroma components. This is the traditional broadcast standard for chroma sampling and is used by DVCpro50, XDCAM HD422 and P2 AVC-Intra.

In this case two consecutive pixels will have different luma components but, their chroma components will be the same, see Figure 2.7.

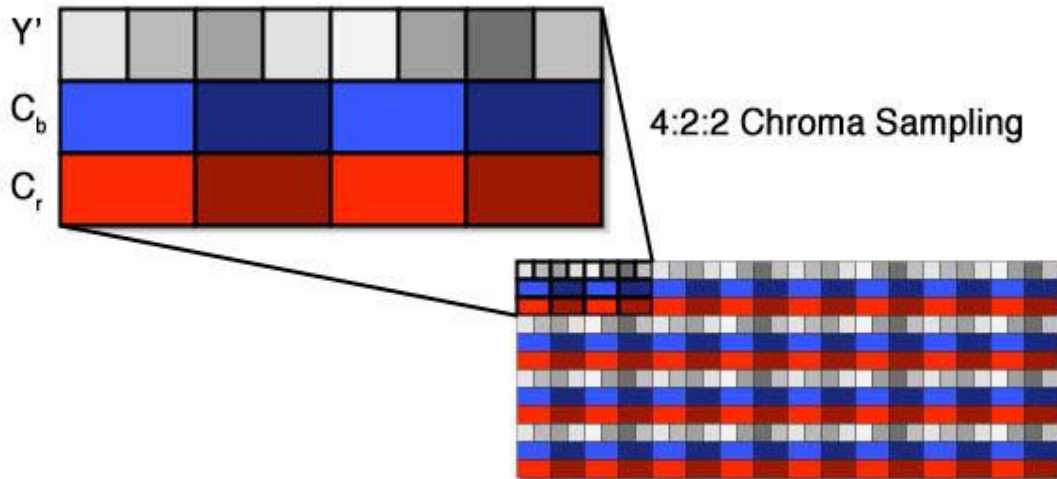


Figure 2.7 – Chroma sampling in 4:2:2 [16].

4:1:1

Uses a full resolution luma and quarter ($1/4 = 0.25$) resolution chroma components. This is the system used by NTSC DV, DVCPRO.

In this case, as seen on Figure 2.8, four consecutive pixels will have the same chroma components.

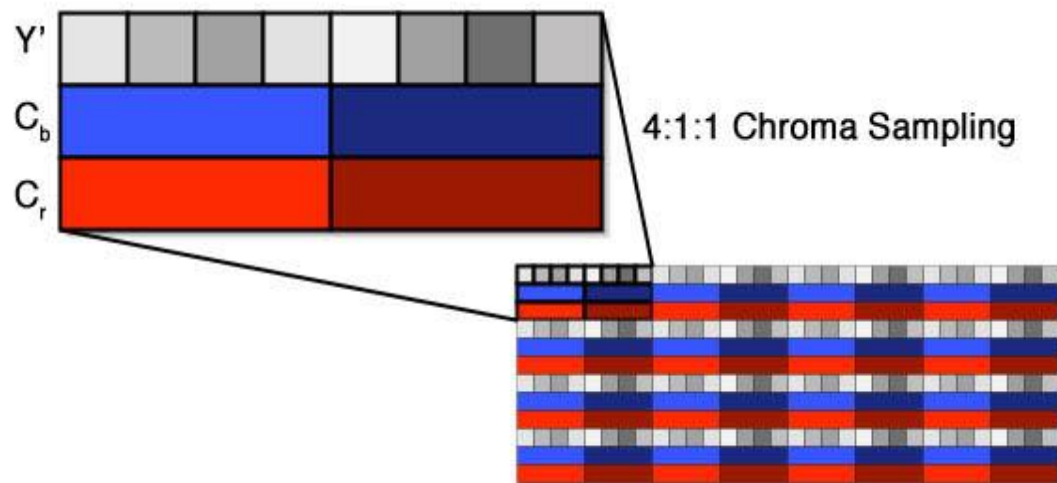


Figure 2.8 - Chroma sampling in 4:1:1 [16].

4:2:0

Full resolution luma and half resolution in the horizontal direction and vertical direction for the chroma components. 4:2:0 is a very complex chroma sampling with many variants depending on whether the video is progressive or interlaced, or if it is being used by PAL DV or MPEG2 (e.g. Sony XDCAM). 4:2:0 compresses the resolution of the color to $1/4$, just like 4:1:1 compresses the resolution of the color. But whereas the compression in 4:1:1 is horizontal only,

Background

the compression in 4:2:0 is horizontal and vertical. The illustration on Figure 2.9 is for PAL DV 4:2:0 chroma sampling[16].

Chroma sampling is a good compression technique providing savings in bandwidth requirements with little deterioration of image quality. For the naked eye a 4:2:2 chroma sampling seems equal to the original picture if no sub sampling was used (4:4:4). The reason why 4:2:2 images are almost identical to 4:4:4 to the human eye, resides on the fact that the human visual system is much more sensitive to variations in brightness than in color.

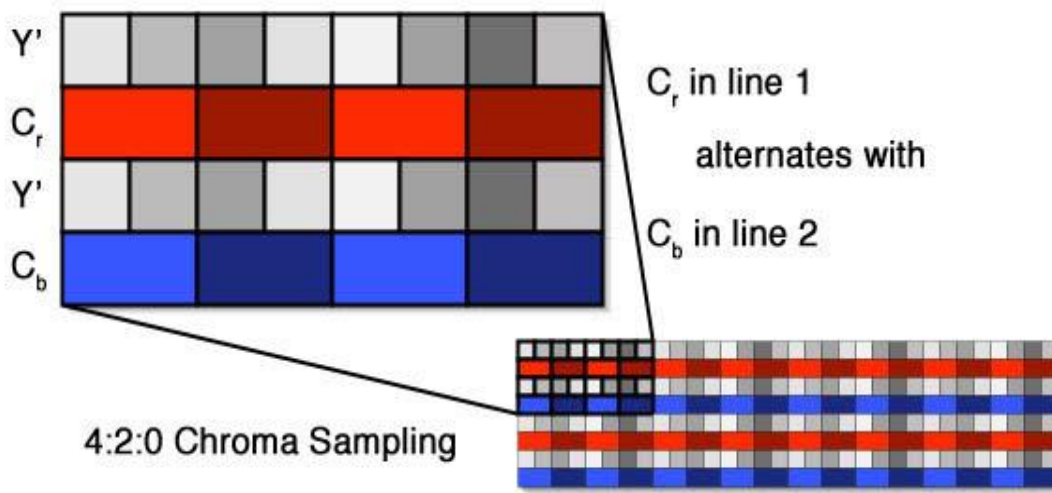


Figure 2.9 - Chroma sampling in 4:2:0 [16].

2.3.2 DV – DVC Pro and DVCAM

In this section the general concept behind the DV family of compressed streams will be explained.

“DV is a DCT block based compression scheme that was originally designed for intra-frame coding with a fixed number of bits per video frame so that on a helically scanned tape was straightforward.”[17].

DV is based on a DCT block diagram like the one illustrated on the Figure 2.10.

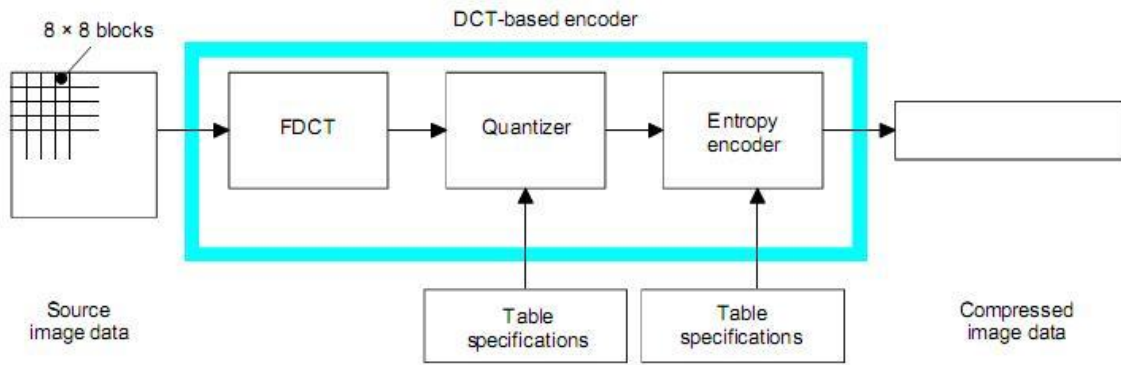


Figure 2.10 - DCT-based encoder simplified diagram [18].

DVCAM and DV are a DV stream compliant with IEC61834-2 while DVC Pro is a DV-Based stream compliant with SMPTE 314M or SMPTE 370M[17]. The main difference between them is the 625 line (PAL) chroma sampling used.

Table 2.1 shows the differences between the different compression schemes. We can see that DV streams use a chroma sampling of 4:2:0 in the 625 line system whereas DV-Based streams use a chroma sampling of 4:1:1. Even though DV streams offer much more audio options than DV-Based streams in professional applications, they are rarely used. The norm being 48 kHz two channel audio.

Table 2.1 - Differences between IEC 61834-2 and SMPTE 314M [19].

		DV (DVCAM) IEC 61834	DV-BASED (DVCPRO) SMPTE 314M	
			25 Mb/s structure	50 Mb/s structure
Data structure		IEC 61834	Same as IEC 61834	See Figure 2.11
Video	Sampling structure	525: 4:1:1	525: 4:1:1	525: 4:2:2
		625: 4:2:0	625: 4:1:1	625: 4:2:2
Audio	Sampling	48 kHz (16 bits, 2ch) 44.1 kHz (16 bits, 2ch) 32 kHz (16 bits, 2ch) 32 kHz (12 bits, 4ch)	48 kHz (16 bits, 2ch)	48 kHz (16 bits, 4ch)
	Locked mode	Locked / unlocked	Locked	Locked

Background

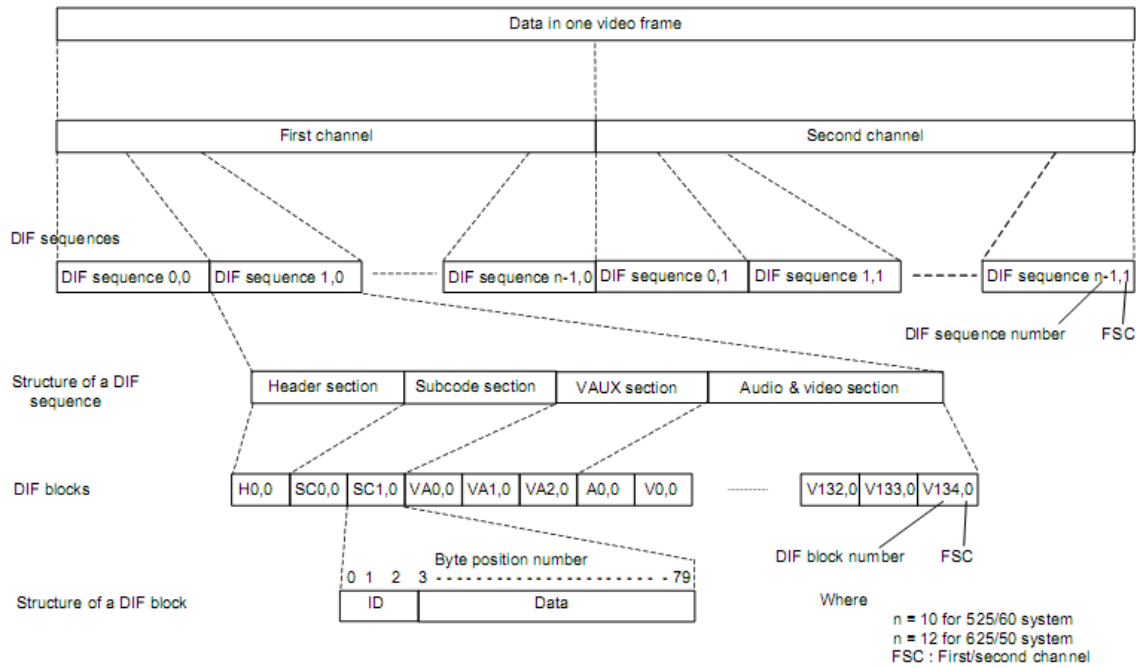


Figure 2.11 - Data structure of one video frame for 50 Mb/s structure [19].

DV also uses adaptive inter-field compression which compresses the two fields that compose a single frame if similarities between the two are detected. In practice static areas of an image will be more accurately represented than areas with motion.

2.3.3 Long GOP MPEG-2 (HDV)

HDV was initially designed for the consumer market which, was later used in the professional market. Its purpose was to offer existing video production environments a cost-conscious upgrade path from SD to HD video. When it was introduced it allowed the recording of HD video on existing recording media (tape media). A description of this encoding technique is given hereinafter.

“Group of pictures describes the predictive form of MPEG encoding.”[17]. In this coding technique there are three types of pictures, “I” pictures, “P” pictures and “B” pictures. “I” pictures are those which have all the information self contained, therefore no need to know information from other pictures to decode them. A “P” picture can be decoded using information from a picture that has already appeared in the MPEG stream (past picture). And finally “B” pictures can be decoded using information from a picture in the past and a picture in the future i.e. pictures that have not appeared in the MPEG stream yet.

To make this easier to decode the pictures are reordered between presentation and storage order. In MXF, pictures are stored using the storage order rather than the presentation order, see Figure 2.12.

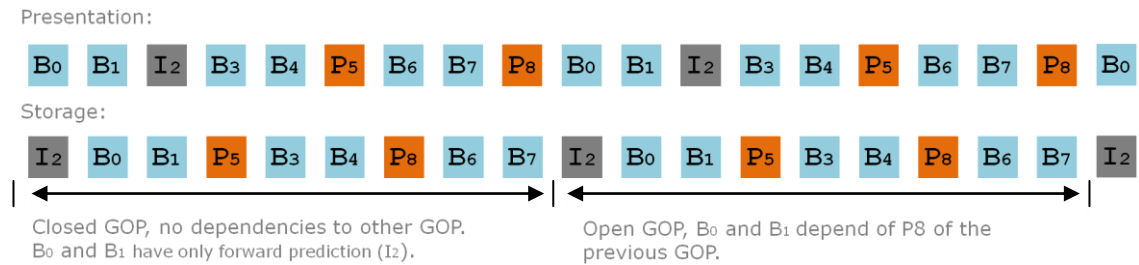


Figure 2.12 – Presentation order and storage order.

The pictures in this compression format may be of Constant Bit Rate (CBR) or Variable Bit Rate (VBR). If VBR is used the compression achieved is greater because some pictures of a video scene may be more complex than others. Using more bits to encode a more complex picture while using fewer bits to compress a less complex one, reduces the amount of bandwidth used without reducing picture quality.

Using motion estimation and compensation across multiple frames (Inter-frame coding) high compression is reachable at the expense of a more complex encoding and decoding, see Table 2.2 to understand and have a general view of the differences between intra-frame and inter-frame compression algorithms. Figure 2.13 presents a comparison between various compression algorithms.

Table 2.2 – Comparison between Intra-only and Long GOP (Inter-frame) [15].

Compression Scheme	Intra-only compression		Long GOP compression	
	Individual frame	Multiple frames (e.g. 15 frames)		
Bit rate saving	Smaller	Use spatial correlation only	Greater	Use spatial and temporal correlations
Processing delay	Smaller	1 frame	Greater	Multiple frames
Editing easiness	Easier	frame by frame	More Difficult	GOP
Multi-generation deterioration	Smaller	Intra structure	Greater	Long GOP structure
Error propagation	Smaller	Max. 1 frame	Greater	Multiple frames
Parallel processing	Easier	Max. 1 frame	More Difficult	Multiple frames

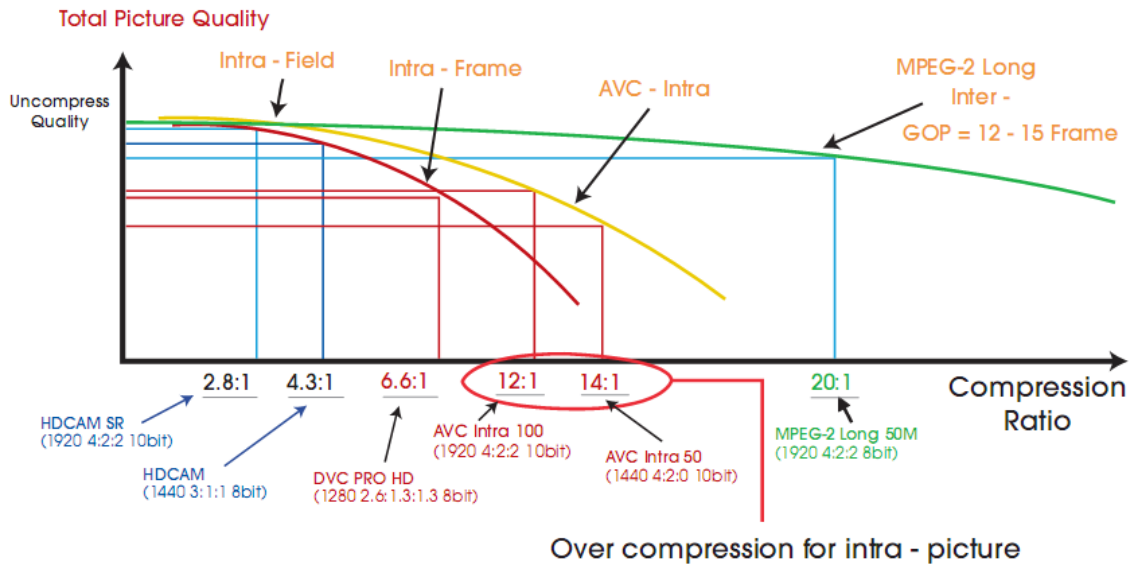


Figure 2.13 - Relative picture quality of various video compression algorithms [20].

Despite the fact that newer and better compression algorithms are being developed and made available, long GOP MPEG-2 has a much more wide support, is more mature and has been integrated in a wide variety of applications. Also important, is the fact that the implementation costs are much lower because the processing power requirements are lower than newer compression algorithms. “With the existing generation of Dual – and Quad-Core CPUs it is now possible to encode/decode multiple streams of HD MPEG-2 Long GOP signals in real-time with software-only tools.”[20].

Another interesting point is that this compression algorithm used in HD pictures with high bit rates (more than 20 Mbps) have almost the same quality as newer algorithms [20].

This compression scheme is used nowadays in Sony XDCAM high definition broadcast products.

2.3.4 AVC- Intra

AVC-Intra is the “term used to represent the implementation of the H.264/AVC Intra-only compression”[15]. By Using intra-frame only compression encoding and decoding complexity is greatly reduced at the expense of less efficiency in compression than long GOP. In pictures where a lot of movement is present, the efficiency between intra-frame only and long GOP is very similar because similarities between frames are reduced in pictures with movement.

While this compression algorithm is constrained by intra only compression, it benefits from newer and better intra compression methods present in H.264, namely spatial intra prediction and new entropy encoding improvements.

In the spatial prediction “each luminance sample in an 8×8 block is predicted from the neighboring constructed reference samples, where 8 different prediction directions and a DC prediction can be selected by the encoder”[15]. “The key to improving coding performance when using spatial intra prediction is to select the proper prediction mode for each block.”[15]. See Figure 2.14 for examples of some different methods used by the intra spatial prediction in the compression algorithm H.264.

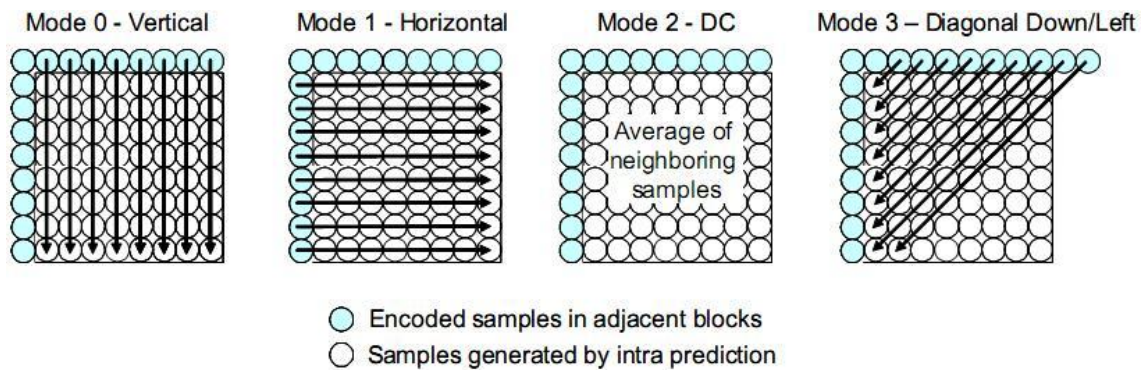


Figure 2.14 – Examples of spatial intra prediction on H.264[15].

Other benefit from H.264 is the improved entropy encoding. H.264 offers two new methods to improve entropy compression: Context Adaptive Binary Arithmetic Coding (CABAC) and Context Adaptive Variable Length Coding (CAVLC). Entropy compression is lossless and the benefits of these two new encodings is their ability to encode, using information “from surrounding areas to achieve high compression efficiency by adjusting the entropy encoding parameters in order to best match the content of the image”[15]. Table 2.3 gives a general view on the different entropy encodings available and where they can be used.

Table 2.3 – Comparison between Entropy Coding Methods[15].

	MPEG-2	H.264/AVC	
	VLC	CAVLC	CABAC
Encoding method	Non-adaptive VLC	Context Adaptive VLC	Context Adaptive Arithmetic Coding
Context Adjustment process	No	Transformed coefficient based	bit by bit
Encoding efficiency	Moderate	Very good	Excellent

While this compression is lossless these new algorithms have a drawback, they are very complex and, when used, the most efficient one (CABAC) requires a lot of processing power. However, since this compression algorithm uses only intra-frame only compression this increase in processing power demands is acceptable.

The use of intra-frame only compression adds another benefit, because all the information to decompress a picture is in itself, the editing is much more accurate and faster because only one picture is needed to be decoded instead of various. For comparison between intra-only compression methods and long GOP methods see Table 2.2.

AVC-Intra in Panasonic is available in two modes, one using 100Mb/s with resolution 1920x1080 or 1280x720 and chroma sampling 4:2:2 with 10 bits. The other uses 50Mb/s with resolution 1440x1080 or 960x720 and chroma sampling 4:2:0 with 10 bits.

2.3.5 JPEG 2000

JPEG 2000 is one format used mostly for long term video preservation because it has the possibility of using lossless video compression. Due to this reason digital cinema is adopting this compression for archiving purposes.

Motion JPEG 2000 file format does not support inter-frame compression, meaning each frame is coded independently using JPEG 2000. Nowadays this format is almost obsolete and has been superseded by other container formats supporting wrapping of JPEG 2000 encoded elements (frames) like, for example, the MXF file format.

JPEG 2000 is an evolution of the old digital image compression JPEG. “Instead of the DCT (Discrete Cosine Transform) used in JPEG baseline, JPEG2000 uses the Discrete Wavelet Transform.”[21].

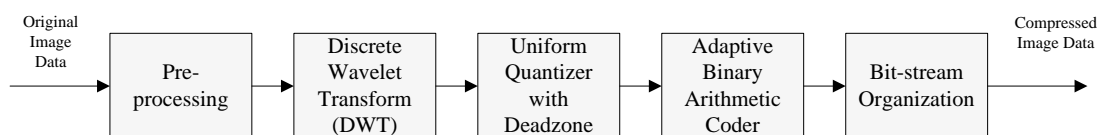


Figure 2.15 – JPEG 2000 block diagram [21].

The use of DPCM (Differential Pulse Code Modulation), which “is a predictive coding used in order to transmit the value of a sample calculated by the means of previous samples

instead of the real value”[21], helps to optimize the performance of the compression algorithm. It is typically used before applying entropy coding and after the wavelet transform.

Using intra-frame compression only, makes it ideal for non-linear editors even though this reduces compression efficiency. All the disadvantages and advantages of such compression scheme were already overviewed in 2.3.3 and 2.3.4. Infinity products from Thomson Grass Valley offer the possibility to record video using this compression algorithm.

2.3.6 Interlaced Video

Interlaced video is a technique to improve video quality, especially in older displays (CRT) without increasing the bandwidth used. CRT TVs do not display the full image at a time, instead they divide the screen in a series of lines, which are then scanned one by one from the top of the screen to the bottom creating a picture, see Figure 2.16.

Interlaced video appeared to solve a very specific problem. When color TVs were introduced, the type of phosphor used to produce the colours did not respond very fast, creating a flickery strobing effect moving down the screen. To solve this it was decided that instead of putting the lines on the screen one at a time consecutively (i.e. lines: 1, 2, 3, 4), they would be scanned on every other line in one pass (i.e. 1, 3, 5, 7) and then in-between the previous lines on the second pass (i.e. lines: 2, 4, 6, 8)[22]. This allowed the phosphors to have more time to recover because a video frame was now being drawn in two very fast scans i.e. twice the speed of a progressive scan.



Figure 2.16 – Frame composition.

The group of odd lines or even lines is called a field (Figure 2.16). One of the disadvantages of interlaced video is the possibility of exhibiting motion artifacts, this happens, for example, if the recorded objects are moving fast enough to be in different positions when each individual field is captured.

On the other hand, in progressive video a frame is captured in the same instant. For progressive video the bandwidth required to obtain the same vertical resolution as in interlaced video is higher. Progressive video has some advantages when compared to interlaced video. For example the absence of motion artifacts, each frame can be used as a still picture, and there is no need to blur the image to reduce eye strain like in interlaced video.

2.4 Containers

2.4.1 MXF File Format

The MXF standard took five years to develop and is composed by the SMPTE 377M and other associated documents, namely: SMPTE 379M, Generic Container; SMPTE 390M, OP-Atom; SMPTE 381M, MPEG; SMPTE 382M, Audio; SMPTE EG41, Engineering Guideline; SMPTE EG42, Descriptive Metadata Engineering Guideline amongst others.

The goal of MXF is to provide a means of exchanging finished or almost finished audiovisual material and metadata among different systems throughout the production stages.

“The MXF specification is designed for an environment where content is exchanged as a file. This allows users to take advantage of non-real-time transfers and designers to package together essence and metadata for effective interchange between servers and between businesses.”[17].

One of the things that make MXF successful and platform-independent is the detailed specification of the file format, the bit-stream syntax and the plug-in architecture of the specifications.

The core of the MXF specification (SMPTE S377M) defines the file format and structural metadata model, the operational patterns, the way essence must be wrapped and metadata schemes can be chosen, making it flexible enough to be used in several ways. It includes the possibility of developing custom metadata models to serve a specific need.

MXF metadata is filled from the acquisition (by the capturing camera) to the post-production stage, making the MXF file format cover all the production life cycle and enabling users to complete or update existing metadata at any stage.

Only quick overviews over some specific aspects of the file format are explained to clarify some basic definitions and make this thesis accessible to the reader.

2.4.1.1 File structure overview

A MXF file is generally divided, as it can be seen in Figure 2.17, in three partitions: header, body and footer.

The header partition must always contain a header partition pack and header metadata. A more complex header may contain a Run In and an Index Table (Figure 2.18).

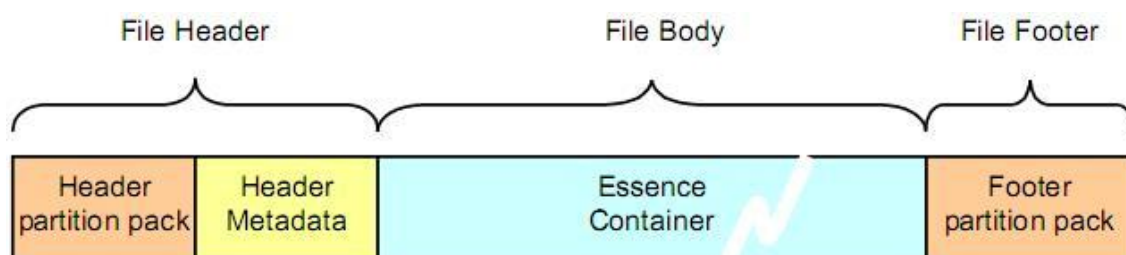


Figure 2.17 - Overall data structure of a simple MXF file [6].

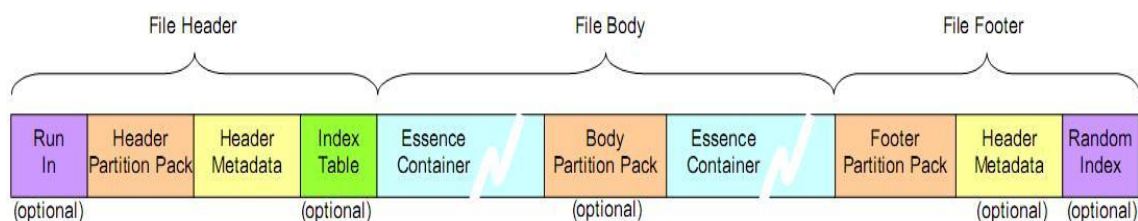


Figure 2.18 - Overall data structure of an MXF file with optional components [6].

The body partition provides a mechanism for embedding essence data within MXF files, if present, must contain a body partition pack and optionally one or more essence containers.

The footer partition is mandatory and it contains a footer partition pack and may contain some metadata and an index table. After the footer partition a Random Index Pack (RIP) may appear.

Index tables improve random access within an MXF file doing the translation between time values and byte offsets. It is also responsible to cope with files that are temporally reordered on the disk compared to their presentation order (e.g. long GOP MPEG-2).

The definition of some optional fields is outside the scope of this thesis. For more details on this fields, please see [17].

Every partition has 4 possible combinations of the states open or closed and complete or incomplete. Being open or closed refers to the state of the metadata, signaling if it is a partial or

Background

a final version, respectively. Complete or incomplete means that all the metadata values are complete and correct or that some invalid Best effort values have been used, respectively.

Given these simplified definitions, the footer partition must always be in the closed state for the file to be valid.

2.4.1.2 MXF file encoding overview

A MXF file is sequentially encoded using KLV coding. In short, Key is a unique identifier that allows the identification of each packet, Length is the size of the Value to allow a correct reading and Value is the value of the field, basically is just a sequence of bytes where the end of the data is known using the Length value, see Figure 2.19.

The KLV encoding process must obey several restrictions specified in SMPTE 377M, such as byte order, KAG values and the use of KLV fill items.

KLV encoding supports recursive grouping, which means that the Value of a KLV may contain other KLV's. There are no depth limits imposed by the SMPTE 336M in coded KLV sets, but SMPTE 377M states that a MXF file may only contain packs or sets that contain individual data items, unless it is specified otherwise in a particular essence container or descriptive metadata specification.

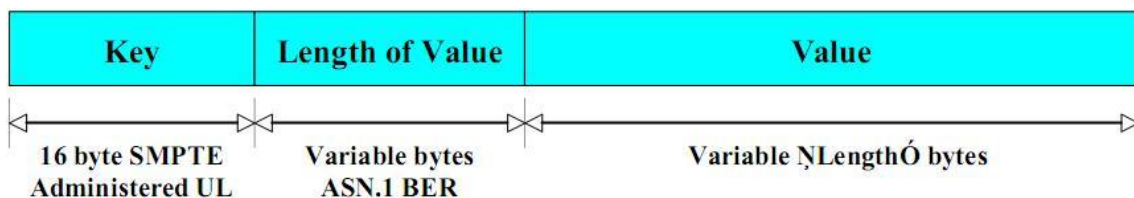


Figure 2.19 - KLV Encoding [1].

The alignment of the KLV's is commonly defined by a KAG, which defines the alignment of the bytes from the recorded data. It is meant to be a technique to optimize disk access performance. Since MXF is a generalized format, that value has to be configured according to one's needs.

The UL in the Key gives enough information for a complete description of the Value defining exactly the content inside the Value.

A decoder that does not recognize a particular key is able to skip over the unknown value and inspect the next key in the line. This allows extra functionality to be added to the MXF specification at a later date, knowing that older decoders will be able to skip over the values. This also decreases problems related with interoperability since other software may not

know a particular UL of one KLV, and still be able to play all the other ULs known to the software application, thus reading the file correctly.

2.4.1.3 Operational Patterns

“Operational patterns are the way of controlling the complexity of MXF files.”[17]. The operational patterns define the structure of the metadata, such as the relationships between source packages and the material package.

The generic operational patterns defined in the MXF standard [6] display three levels of item complexity and package complexity, making a total of nine generic operational patterns available, see Figure 2.20.

Although in the table we only see nine generic operational patterns, there are other specialized operational patterns. Taking into account the scope of the thesis, only the operational pattern OP1a is relevant along with the specialized operational pattern OP-Atom. For more information about this subject see [17] or [6].

OP1a is the simplest generic operational pattern and is the most common one in MXF files today[17]. This operational pattern “contains the audio-visual item as a single playable essence container. This essence container may, for example, contain a single clip or a single item of program material. The essence container shall provide for the continuous decoding of contiguous essence elements with no processing.”[23]. This operational pattern was aimed for tape replacement, being most common as frame wrapped interleaved with audio and video. OP1a states that, a single clip must be played from the beginning to its end without seeking or changing clips during playback.

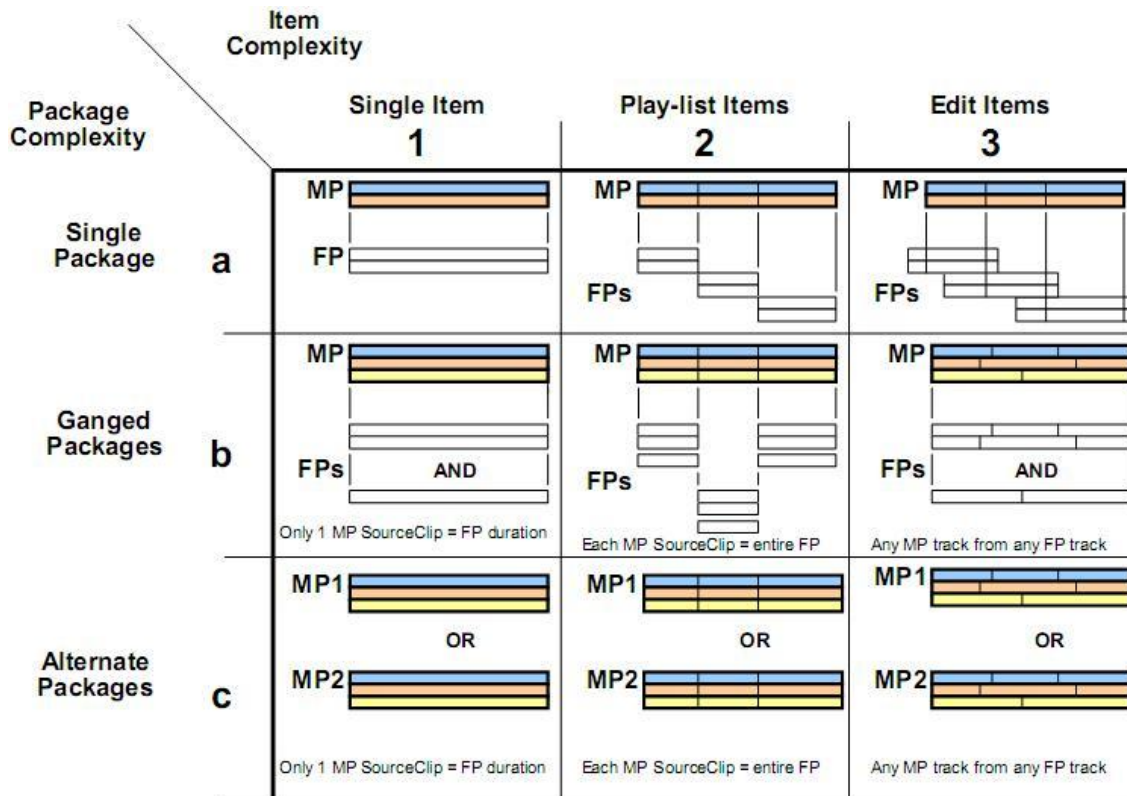


Figure 2.20 – Operational Patterns [6].

OP-Atom has some advantages over generic operational patterns, because it is supposed to define a very simple and tightly constrained metadata structure for a single track of essence, in order to provide a simpler, faster and optimized access.

One of the several constraints of the OP-Atom is that every MXF file must have only one top-level file package that references the essence in the single essence container, making it a mono essence file. OP-Atom is usually used by non-linear editing applications because those need fast access to individual tracks of essence.

2.4.1.4 Frame and clip-based wrapping

Two types of KLV wrapping are available in the MXF format, these are frame wrapping and clip wrapping. While a third wrapping method is possible (custom wrapping) its usage should be preceded by a good reason to use it, as this method may have incompatibilities with some systems. Custom wrapping is outside the scope of this thesis.

Frame wrapping is normally used in long GOP compression schemes. Basically a video frame, a video field, an audio block or any other value that represents the basic unit of the primary essence is wrapped on a KLV. In Figure 2.21 we see the frame wrapping method being used putting in each KLV triplet a value corresponding to an image frame. Applications which process at the KLV level can access frame-by-frame content.

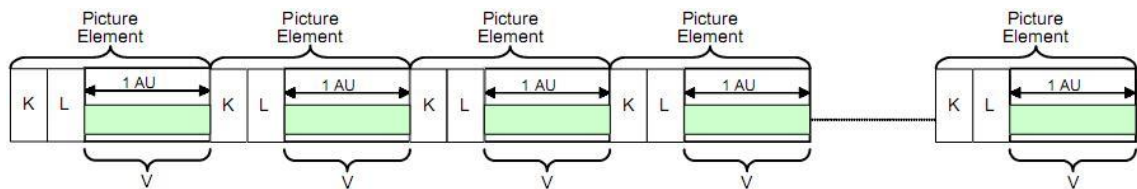


Figure 2.21 – Frame Wrapping example [24].

In frame wrapping, the different channels of sound and data can be KLV wrapped and kept contiguous with the picture KLV creating a group of KLVs with all the different elements attached along a picture frame. By using this method an application can access an individual picture element, and rapidly access the sound associated with that same frame processing it without any further seeking. Figure 2.22 shows an example of content packages (content packages are structures with interleaved elements in a specific order) with other elements such as sound and data elements.

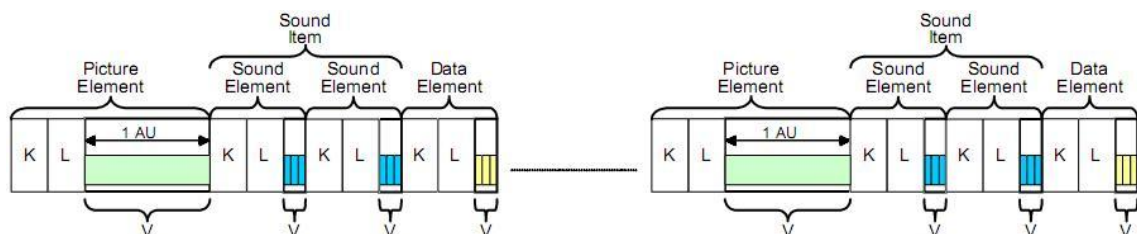


Figure 2.22 – Frame wrapping with other elements contiguously attached [24].

As for clip wrapping the main difference lies in the KLV encoding i.e. the whole of the MXF video stream that may contain a single frame or thousands of frames is now wrapped in the same KLV.

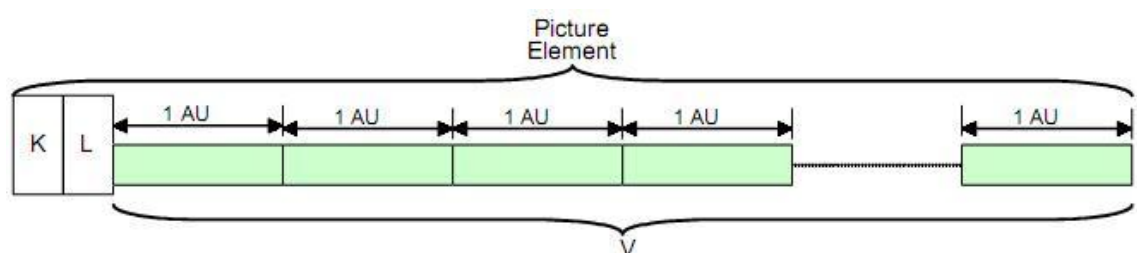


Figure 2.23 – Video stream clip wrapping example [24].

The clip wrapping method can be “very useful in applications such as store and forward servers which process whole files and also in applications where it is desired to use the rich metadata structures of MXF as an annotation to MPEG data”[24].

Clip wrapping can also be made with other elements contiguously attached, making it a content package (like seen above in frame wrapping with various elements) but note that sound and data elements are intended to have the duration of the entire clip, see Figure 2.24.

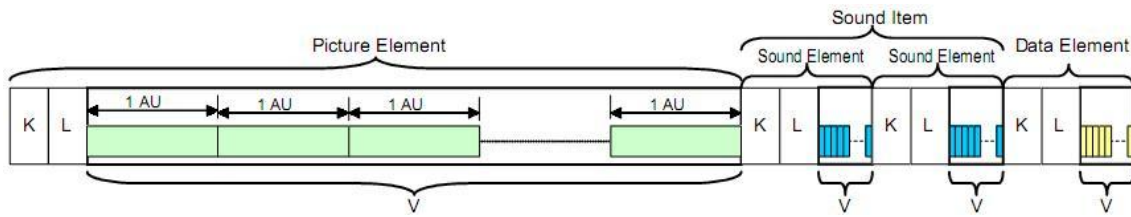


Figure 2.24 – Clip wrapping with other elements [24].

2.4.1.5 Simplified metadata structure overview

Everything inside a MXF file is encoded with KLV, including both metadata and essence.

Metadata is useful for various reasons. For example, with the introduction of some metadata by the capturing camera into the MXF file, it is possible to know where the camera was, how many clips were filmed, the number of takes, the number of scenes, etc. It can also make that metadata travel attached to the file throughout multiple stages of the video workflow.

There is also metadata that is used to aid in synchronization, sequencing, structuring and to reveal the type of essence the MXF file contains. This is called structural metadata.

There are two main metadata structures to have in mind: the Material Package and the Source Package (see Figure 2.25).

A Material Package does not describe the essence itself but the output timeline only, represented by an ordered collection of tracks that will be played.

The Source Package can be seen as an abstract class from which the file package and physical package derive.

A file package describes the actual stored essence and a physical package describes the essence before it was wrapped into an MXF file to make historical annotations.

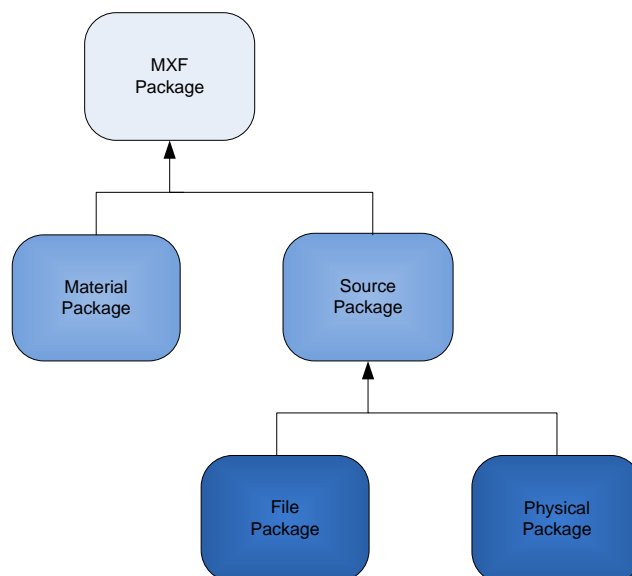


Figure 2.25 - Detail of the package class inheritance of MXF [6].

Source packages that are not directly referenced by the material package are considered lower-level source packages, and are only used to describe the derivation information of the top-level file packages. In a general operational pattern, a top-level file package is always directly referenced by the material package and is the only type of file package that may describe the stored essence. An example of a usual metadata referencing structure can be seen in Figure 2.26.

MXF uses a number of descriptive metadata frameworks¹ collectively called descriptive metadata scheme – 1 (DMS-1).

“The MXF file format specification has a number of structural metadata packages in the header metadata that describe the essence data and essence containers in the file body.”[25].

¹ A collection of related metadata objects (either as sets or properties) using an instance of a defined class hierarchy [17].

Background

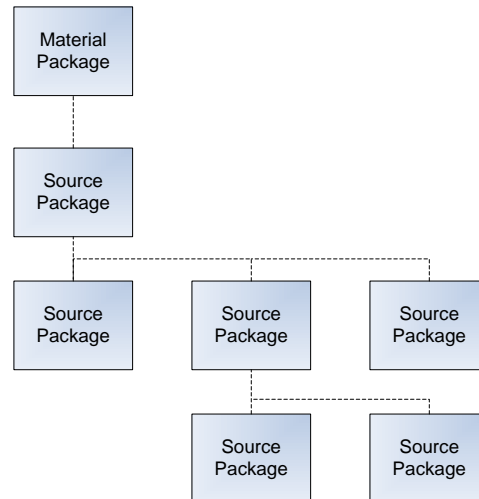


Figure 2.26 - References between packages [6].

“This standard provides a number of descriptive metadata frameworks that may be plugged into the structural metadata packages of the header metadata based on the mechanism defined in the MXF file format specification. These descriptive metadata frameworks are a part of the header metadata and provide additional editorial value to an MXF file.”[25].

Each descriptive metadata framework is grouped using related properties² and sets³ which describe the contents of a MXF file body, as seen on Figure 2.17.

The DMS-1 essentially defines three different frameworks, they are:

- **Production metadata framework:** describes the editorial identifiers that apply to the production as a whole [17].
- **Clip metadata framework:** describes the content as it was created or captured [17].
- **Scene metadata framework:** describes the editorial intent of the content [17].

“Descriptive metadata frameworks give contextual meaning to a metadata set by logically grouping metadata sets used in the same context. For example, a metadata set that describes a location can be used to describe the real location (the actual location of the camera) or the fictional location (where the scene is supposed to be set).”[25]. This means that the same metadata set may have different meanings depending on the framework where it is inserted.

Figure 2.27 gives an idea of the structure of these descriptive metadata frameworks in the header metadata and how they relate to the MXF file body.

² An individual item of metadata [17].

³ A collection of properties that contribute in equal measure to an object whose overall value is greater than the sum of the individual properties [17].

While MXF approaches the metadata problem itself, other types of metadata can be included in a MXF file. This metadata is KLV encoded as an essence. Because this metadata is not part of the MXF specification in order to be read an application must support MXF and the newly introduced metadata. For example subtitles are added to a MXF file using this solution.

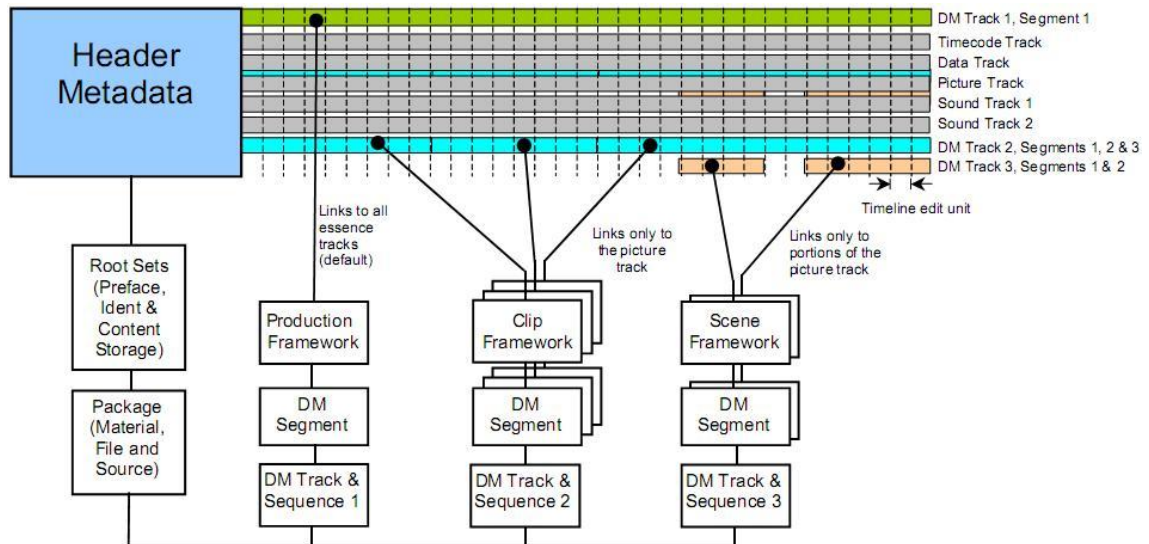


Figure 2.27 – Relationships between metadata frameworks and the content of MXF file body [25].

2.4.2 Advanced Authoring Format (AAF)

The Advanced Authoring Format (AAF) is an interchange toolkit. It exists to get video, audio and metadata from one system to another thus increasing interoperability between file-based products.

As per displayed in Figure 2.28, AAF makes a clear distinction between Object Model and Storage Layer.

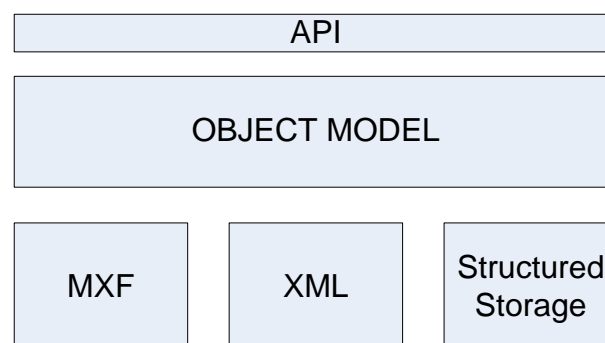


Figure 2.28 - AAF Object Model and Storage Layer.

“The main applications of AAF are transfer of authoring, editing, and media-management metadata, and associated source material.”[17].

Background

AAF arranges its file format in terms of objects. These objects are chosen and defined to reflect the actual processes and content items that go into the authoring process.

MXF uses parts of the AAF model, containing only the sufficient to make it a file interchange format. The development of the MXF format was build reusing all the features that were needed from the AAF data model and if newer ones had to be created they had to be designed in a way that would make possible their inclusion in the AAF data model.

The MXF format is so interconnected with AAF that files created according to the MXF standard may be opened by applications that are designed to read AAF, and this is set as a requirement for the MXF standard. These entire similarities mean that much of the functionality and features overlap between both formats. One can say that MXF is a simplified subset of the AAF data model, build specifically to deal with transfers of completed programs, program segments and source material[17].

So the general idea is that AAF describes a composition that includes content, transitions, effects and metadata that go into making a finished product and MXF is described as a file format to transfer the finished product.

Table 2.4 – General differences between AAF and MXF.

AAF	MXF
Primarily intended for post-production interchange	Primarily intended for store and forward and broadcast play out interchange
External references allowed	External references not encouraged
Ability to have downstream processing (Effects, fades, etc.)	No downstream processing should be required
May include complete “programs” and/or partial clips	Usually contains one complete material sequence
Started as a library (lib) with no file format specification	Started as a SMPTE file format specification
Top-down approach	Bottom-up approach

2.4.3 QuickTime File Format

This sub-chapter reviews some concepts related with QuickTime file format (QTFF).

“The QuickTime file format is a general-purpose multimedia file format that may contain video, audio, still images, animated images (sprites), graphic, text and many other media streams including virtual reality data. It may also contain references to media data stored in other files.”[26].

This said the QTFF is an ideal format for the exchange of digital media between devices, applications, and operating systems because it can be used to describe almost any media structure.

QTFF is as an object oriented structure, much like the MXF format, where each object can be easily expanded and parsed. Unknown objects can be ignored or skipped allowing greater flexibility and extensibility to the format, enabling custom applications to add new objects to the format and still maintain interoperability with other applications.

A great testament to the QTFF extensibility and stability was its use by the International Organization for Standards (ISO) as the basis for the MPEG-4 standard and JPEG-2000 standard.

“QuickTime file format does not define the compression schemes for QuickTime movie files. The QuickTime movie files are served as the containers of various types of video and/or audio data. It is designed such that any third party vendor can add application-specific formats by incorporating their QuickTime components with QuickTime.”[26]. This means QTFF is basically a container supporting a wide variety of media formats.

The next section will give a brief overview of QuickTime Movie structure.

2.4.3.1 QuickTime Movie Structure

QuickTime movies have a time dimension that is defined by a time scale and duration, which are specified by a time coordinate system[27]. A movie always starts at time 0 even though the actual displayed movie may start later in the timeline. The time scale of a movie is the unit of measure for a movie time values. In Figure 2.29 we see an illustration detailing how measuring units connect to QuickTime movies.

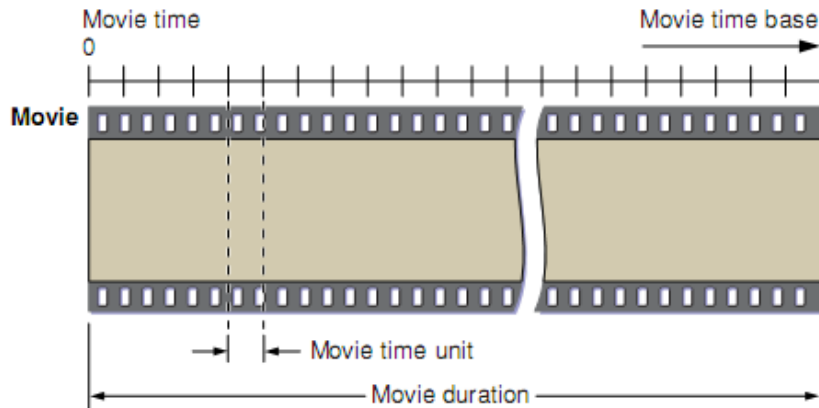


Figure 2.29 – QuickTime movie structure and time measuring [27].

QuickTime movies can contain one or multiple tracks, each of them representing media data which will be interpreted within the movie’s time coordinate system. All tracks start at the beginning of the movie and can have any length. Even though all tracks begin at the start of the movie, their data can be offset to play only at a later time in the movie time base. Tracks that have offset from the movie start contain empty space until the instant where the track actually begins. On video tracks empty space generates a blank screen and in sound tracks it translates into silence.

Tracks will be enabled or disabled while moving through the movie. One or multiple tracks can be enabled at a given time. On Figure 2.30 a movie with more than one track is represented. At the movie’s time value of 6 we see two tracks enabled, one being the Video 1 track and the other being Audio 1 track. In the same time value, one track is being enabled (Video 2 track), because there will be 2 video tracks enabled, and if they overlap spatially, each one will have to be assigned a different layer⁴.

⁴ “A movie can contain one or more layers. Each layer contains one or more tracks that may be related to one another.” [27].

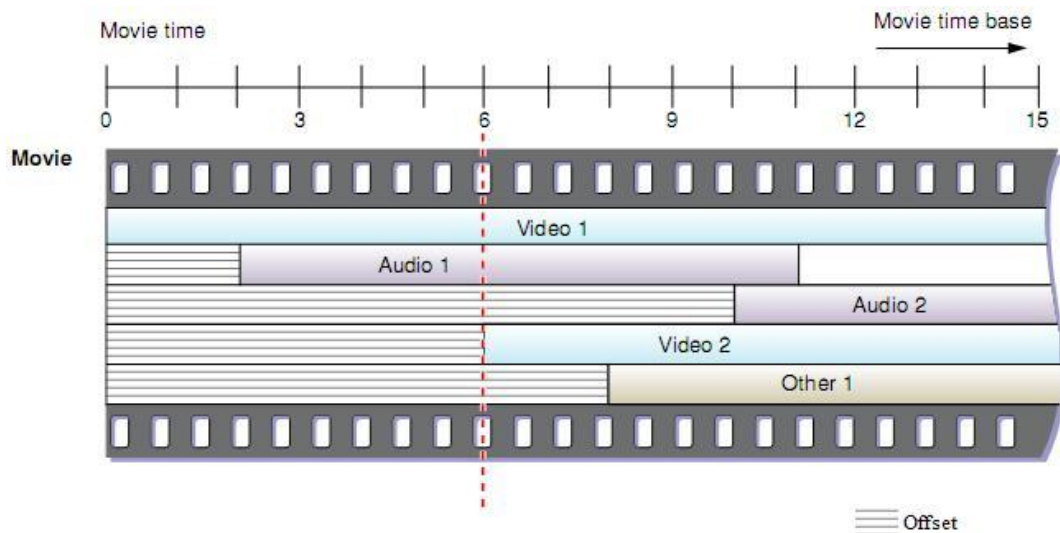


Figure 2.30 – QuickTime movie with multiple tracks [27].

Various audio tracks can be enabled at the same time without any problem, for example two audio tracks enabled at the same time may represent a stereo audio stream, where each one will represent a different audio channel (left, right).

The Audio 2 track and the Other 1 track will only be enabled at a later time value on the movie time base (Figure 2.30).

The duration of a track does not need to correspond to the duration of the movie. The movie duration always equals the duration of the biggest track on the movie.

The track is only associated with one media. It contains a list with references pointing to portions of the respective media. These references are basically an edit list of the media. Figure 2.31 shows how tracks map the media.

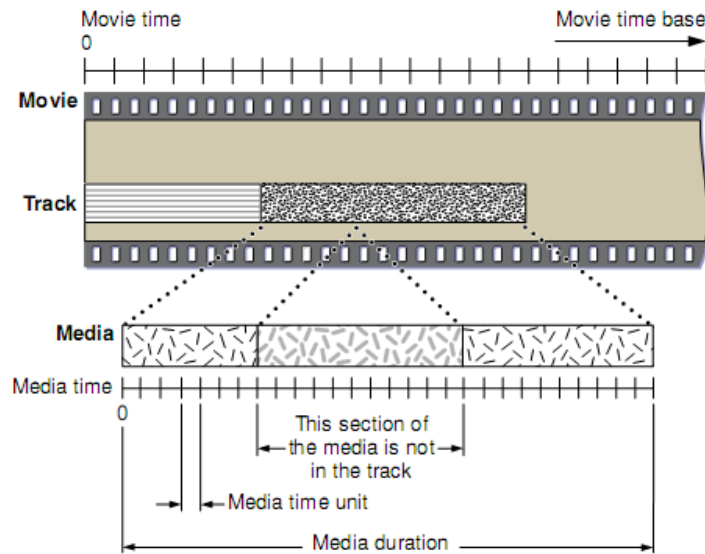


Figure 2.31 – Example of a media mapping from a track [27].

The media describes the data for a track. It does not contain the data, it only references the data, which can be stored in any storage device.

“Each media has its own time coordinate system, which defines the media’s time scale and duration. A media’s time coordinate system always starts at time 0, and it is independent of the time coordinate system of the movie that uses its data. Tracks map data from the movie’s time coordinate system to the media’s time coordinate system.”[27]. Figure 2.32 shows an example of a media referencing the data recorded on a CD-ROM or DVD-ROM.

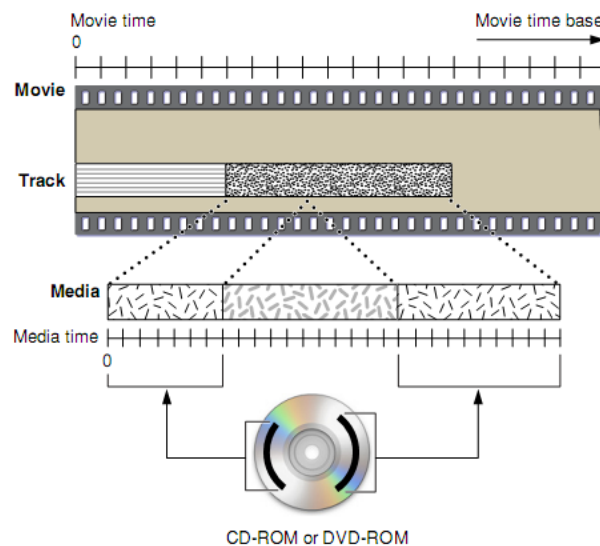


Figure 2.32 – Media referencing to a storage device [27].

2.4.3.2 QuickTime Movie File

It is important to distinguish between a QuickTime movie, the data structure discussed in 2.4.3.1, and a QuickTime movie file.

“A movie file can contain a stored copy of a movie data structure, or it can contain only a reference to such a structure, stored somewhere else.”[28].

If a movie file contains the sample data used by the movie then it is called a self-contained movie file. Basically in a self-contained movie file, all the media data needed for playing the movie is contained in the movie file itself. On the other hand a reference movie does not contain the actual movie data, only a reference pointing to where it is stored. In Figure 2.33 we can see the differences between these two types of movie files.

In this paper, only one type of movie file is relevant i.e. the self-contained type. Reference movie files are outside the scope of this paper because they are not used in any part of the work.

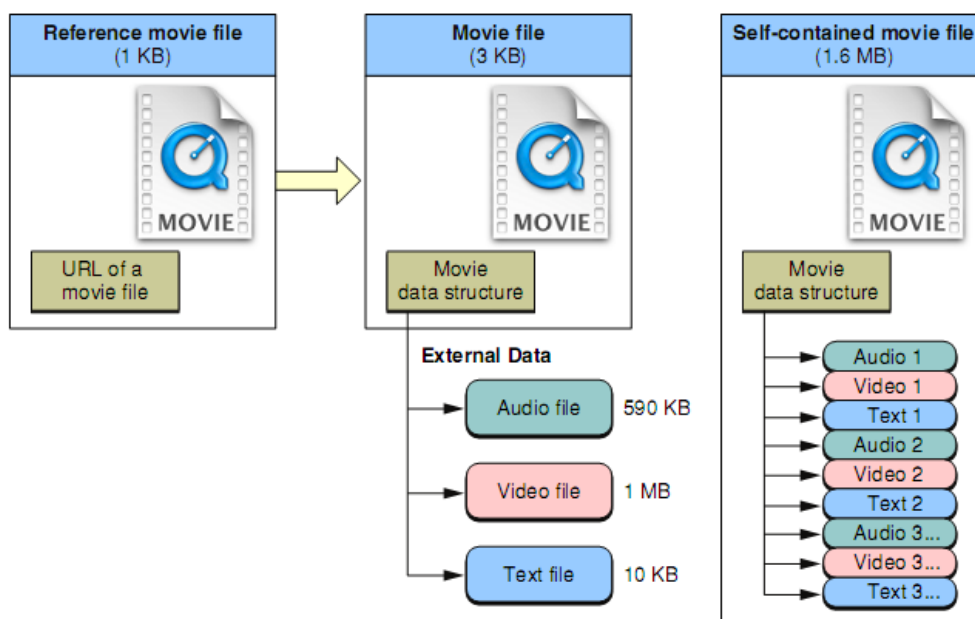


Figure 2.33 – The two types of QuickTime movie files [28].

2.4.3.3 QuickTime Architecture

The QuickTime API is divided in different toolsets each one for particular tasks. The two major toolsets are the Movie Toolbox and the Image Compression Manager. The Movie Toolbox provides the necessary functions to create new movies and manipulate them. The Movie Toolbox API is fairly simple to use and provides a good abstraction layer with high level functions to manipulate QuickTime movies.

Background

The QuickTime architecture makes extensible use of components. “A QuickTime component is a shared code resource with a defined API.”[28]. The components are for example used to add support for different kinds of media to QuickTime. While numerous components are available, QuickTime application only loads the ones that are needed when playing a specific media. QuickTime can even download the corresponding needed component if it is not locally available, as long as an internet connection is provided.

Figure 2.34 shows some of the elements contained in the QuickTime API, from components to toolsets.

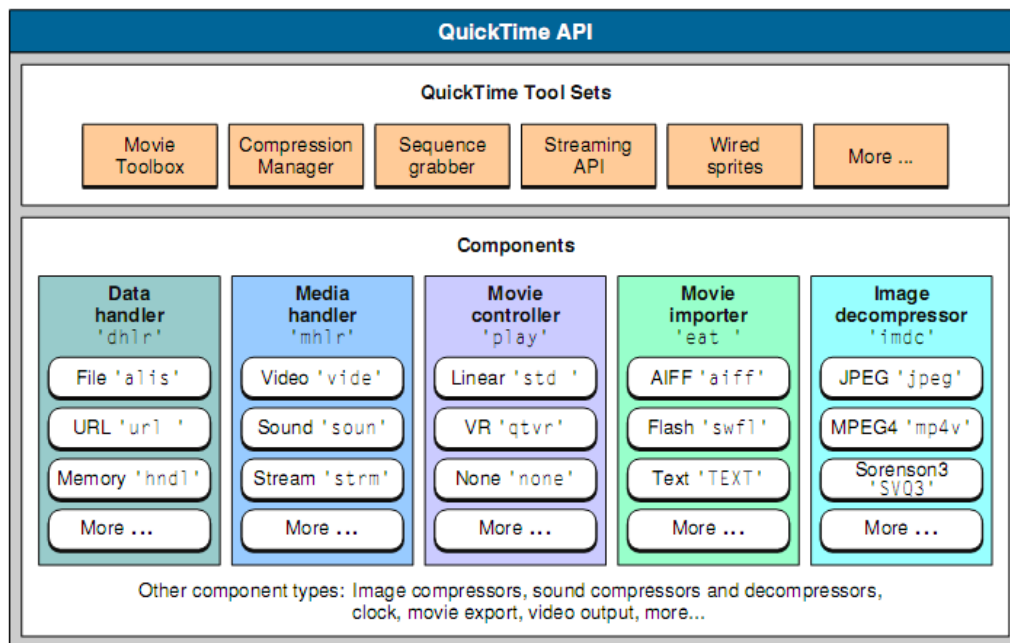


Figure 2.34 - Some commonly used tool sets and components [28].

In order to work with the QuickTime API, knowledge of every type of component or toolset is not necessary, because most of them are used automatically when needed. If needed they all provide an API to work directly with them.

All the components have a type, a subtype, and a manufacturer code, each represented by a four-character code (FourCC).

For example, an image decompressor component has a type of 'imdc' while an image compressor component has a type of 'imco'. QuickTime has many components of the same type, to distinguish between them the subtype and manufacturer code have to be different. If two components have the same type 'imdc', the same subtype 'WMV' corresponding to a Microsoft windows media video, the only difference between them will be their manufacturer code.

2.5 Professional Products

2.5.1 Sony XDCAM

The latest products from Sony in the digital television market are called XDCAM. “Since its introduction in 2004, the Sony XDCAM™ series - made up of SD and HD camcorders and decks, has been offering tremendous benefits to workflows for many types of video productions. The optical disc-based XDCAM system utilizes blue-violet laser technology to achieve high data transfer rates and long recording times for day-to-day operations. Its file-based recording and random access capabilities not only remove the "linear-natured" barriers common to tape media, but also merge the AV and IT worlds by featuring the functions required for both domains.”[29].

Sony XDCAM also offers a small camera, using solid state memory cards as storage media, the XDCAM EX.

XDCAM makes it easier for anyone transitioning to HD because its cameras can record material in SD format using DVCAM compression. DVCAM is very similar with DV, only small differences exist between the two. “The sampling raster of the DVCAM is the same as that of the ITU-R Rec.601. Luminance video signals are sampled at 13.5 MHz, 720 pixels are transmitted per line for both 525-60 and 625-50 systems. In the 525-60 system, each color difference signal (CR/CB) is sampled at 3.375 MHz and 180 pixels are transmitted per line (4:1:1). In the 625-50 system, each color difference signal is sampled line sequentially at 6.75MHz, i.e. 360 pixels of either color difference signal is transmitted per line (4:2:0).”[30].

In DVCAM the sampled video data is reduced by a factor of 5:1 using bit rate reduction, resulting in a transfer rate of 25 Mb/s. “Intra-frame coding which adopts DCT (Discrete Cosine Transform) and VLC (Variable Length Coding) is used.”[30].

XDCAM records HD content using MPEG HD compression (HDV). This compression is based on MPEG-2 Long GOP, which uses inter-frame and intra-frame coding, providing good quality video at lower bitrates than intra-frame only codec's, as seen in 2.3.3. “For HQ and LP mode, a Variable Bit Rate compression method was adopted in order to take advantage of the random access nature of tapeless media.”[31].

See Table 2.5 for the specifications of the compressions, bit rates and video resolutions in XDCAM.

Background

Sony also supports in its XDCAM range of tapeless products the possibility to record proxy data. This data is a faithful copy of the High definition recorded material in much lower bit rates and resolutions. Proxy data is recorded concurrently with the high definition material. This option is completely transparent and non obstructive to the user. “This data can be conveniently used for a variety of applications, such as immediate logging on location, off-line editing, daily rushes of shooting on location, client approvals and more.”[32]. After editing this data the information is then processed to be synchronized with the high quality data.

Table 2.5 - XDCAM HD recording specifications [32].

Video				Proxy Video	Audio				Proxy Audio
MPEG HD420	MPEG HD422	MPEG IMX	DVCAM	MPEG-4	MPEG HD422	MPEG HD420	MPEG IMX	DVCAM	A-law
HQ mode (VBR, maximum bit rate: 35 Mb/s)					Linear PCM	Linear PCM	Linear PCM	Linear PCM	
SP mode (CBR, 25 Mb/s)	(CBR: 50 Mb/s)	(CBR, 50/40/30 Mb/s)	(CBR, 25 Mb/s)	-	8 ch/24 bits/48 kHz	4 ch/16 bits/48 kHz	4 ch/24 bits/48 kHz	4 ch/16 bits/48 kHz	8ch/8 bits/8 kHz
LP mode (VBR, maximum bit rate: 18 Mb/s)							8 ch/16 bits/48 kHz		

This enables users to make edit decisions on the low resolution media (proxy media) much faster and in a broader range of computers, obtaining the same results as if the edit decisions were made directly on high resolution material.

Adding to all of this, all XDCAM products support MXF file format for exchanging or transmission of video, audio and metadata.

2.5.1.1 Workflow examples

This section gives insight on possible workflows using XDCAM tools together with some non-linear editors. Only some examples will be given based within their relevance for this thesis:

- XDCAM with Final Cut
 1. First the XDCAM devices are connected to one terminal giving access to the material recorded in the professional disc;

2. Secondly the material on the professional disc needs to be imported into FCP using the XDCAM Transfer application, all the metadata will be automatically imported and appended to the newly generated files. These new files have to be rewrapped to QuickTime file format because final cut does not support MXF files natively. The Transfer application provides a preview window of the clips or, if existent, sequences created within the devices;
 3. In the end if using the XDCAM Transfer application the edited material can be transferred to the optical disc or to any other media storage devices.
- **XDCAM with Avid**
 1. First the XDCAM devices are connected. The import process is started of HD or proxy media and the media is now ready to be edited. Editing in proxy media requires the user to make a batch import to connect the proxy edited media with the HD media.
 2. Secondly the media can be exported, if needed, back to the optical disc or any other media storage device.

Another possible workflow involves the use of Avid media management products. These solutions provide dedicated hardware to ingest media and, make this operation faster and independent of the terminal where the user is working. This provides a possibility of having multiple persons accessing the same media to work on. These solutions offer many advantages, see 2.6 for an overview of asset management solutions.

- **XDCAM with Adobe Premiere**
 1. The XDCAM devices are first connected and then, the files needed are imported to a project in Adobe Premiere.
 2. Using one of the plug-ins available in the market, the material is imported from the XDCAM device to adobe premiere being now ready to edit.
 3. Finally after editing it is possible to export the edited media back to the professional disc or other media storage device.

At present time, Adobe added native support for XDCAM products, so there is no longer a need to buy a separate plug-in to import XDCAM files to the editor.

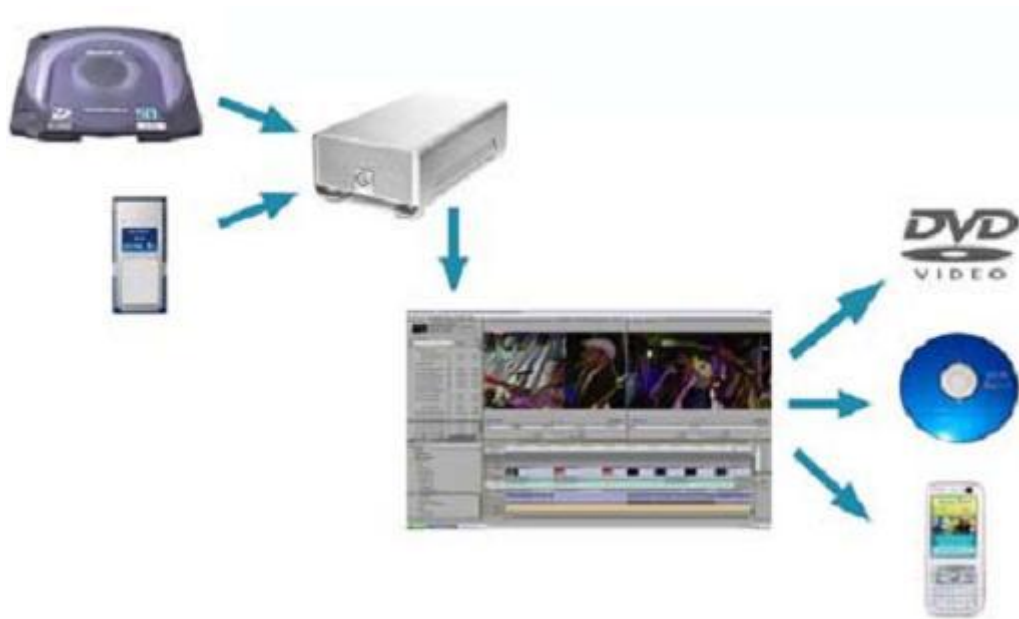


Figure 2.35 - Workflow example using Premiere and XDCAM[33].

As seen above all the workflows are very similar and on other non-linear editors the steps are similar. See [34] for an in depth analysis on this subject.

2.5.2 Panasonic P2 HD

P2 is the name given to the new Panasonic range of tapeless products in the area of digital video recording. Unlike XDCAM, P2 equipment records video and audio to a removable solid state memory card[35].

Using solid state memory cards P2 benefits from the advantages of using this kind of media storage. “P2 HD ensures the highest reliability, especially in challenging conditions of extreme temperature, shock, and vibration.” [35]. The editing can be done on the memory card because it behaves like a personal computer flash drive.

P2 supports both High Definition (HD) recordings and Standard Definition (SD). SD material is captured using DVCPRO or DV compression while HD material uses DVCPRO50, AVC-Intra or DVCPRO HD. Note that AVC-Intra is their latest compression technology it “is a professional intra-frame video codec with bit rates of 50 and 100Mb/s, utilizing the High 10 Intra and High 422 Intra profiles of H.264 respectively.”[36]. H264 is one of the most advanced video compression standard available today on the market, as stated in topic 2.3.

Some P2 products also support the possibility to record proxy data, enabling the option of doing the editing in this low resolution media faster for later synchronization with the associated high definition media.

P2 supports MXF file format on all their products, the MXF files use operational pattern OP-Atom which is a specialized operational pattern that “defines a tightly constrained file structure for a single track of essence”[17] , explanation about operation patterns was given in topic 2.4.1.

In Figure 2.36 we see how P2 organizes the files in the memory card. As we can see we have a main folder CONTENTS, followed by some sub folders each one dividing files by their types, making it easier to search for specific files. The video MXF file for DV/DVCPRO and AVC-Intra is essentially the same, only changing the stored stream type. The MXF in P2 use the not so common mode of clip wrapping (MXF wrapping modes explanation is discussed under topic 2.4.1).

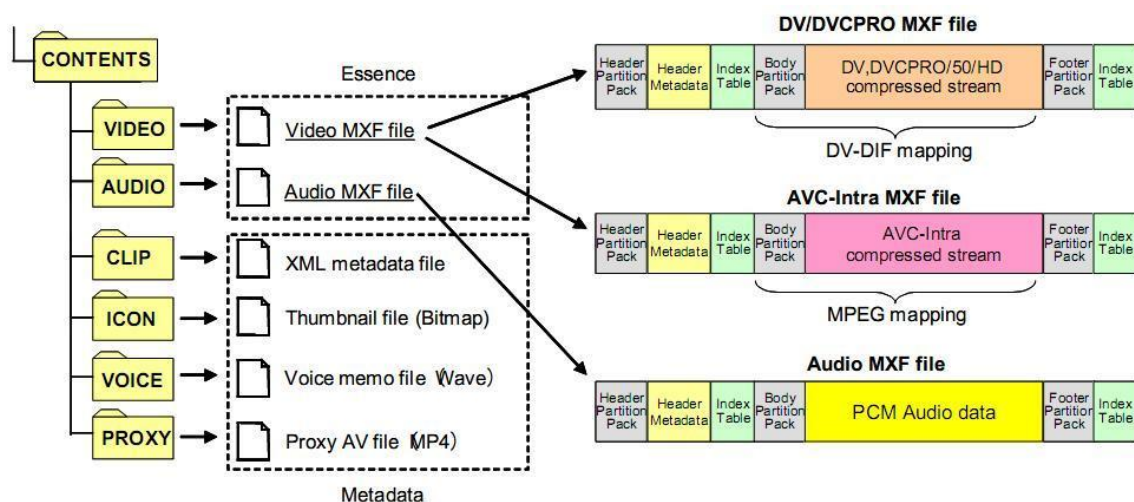


Figure 2.36 – Content structure on the P2 Card[15].

2.5.2.1 Workflow examples

This section gives insight on some possible workflows with P2 products in conjunction with some non-linear editors. Again, only relevant workflows to this thesis will be discussed:

- P2 with Avid
 1. While you can edit the recorded material in the P2 solid state memory card it is not advisable because this media is seen as a temporary medium. This said the files should be transferred first to archival storage media;
 2. The devices are connected via FireWire to a terminal (e.g. personal computer) and after creating a new project in the editor the import process can be started;

3. After editing the files can be outputted, rendering them to any output option the editor provides. It is good practice to save the original media (non-edited) because it may be needed for recovery later. To archive the media for long periods of time Panasonic advise the use of optical media because of its price and durability.

- **P2 with Final Cut**

1. First connect the card to a terminal and before starting the edit process make a backup of the recorded media to another media storage device. Unlike the above case editing cannot happen in the P2 media storage card because final cut does not recognize MXF files;
2. Second stage is to import the media into Final Cut. Final Cut will not make any transcoding operation to the essences being imported. MXF files are only rewrapped to another format (QuickTime file) without P2 metadata. Other software tools available from other companies provide this step with the addition of preserving the P2 metadata on the newly created QuickTime file. One interesting option final cut offers, is the possibility of making sub clips prior to the ingest process thus eliminating the need to import unwanted recorded media in the card[37];
3. Finally after the edit process is finished, the end product needs to be stored in a media storage archive. Panasonic recommends using Blu-Ray media or Digital Linear Tape with data support for MXF for archiving.

- **P2 with Adobe Premiere**

1. In the latest version of Premiere native support of P2 is available. The recorded media is imported from the media card directly to Premiere software. While the user can edit on the P2 media card without transferring it to other storage device, this is not advisable for the same reasons mentioned above. Recommended practice is making some simple editing in the raw content on the memory card to import only the important parts after;
2. Editing is the second stage in this workflow, Premiere supports all P2's compression formats. Premiere maintains all the metadata from the original file "making it easier to find, sort and track content during production"[38].
3. In the end the edited content can be exported in P2 format to a P2 memory card (not advisable) or to another media storage device.

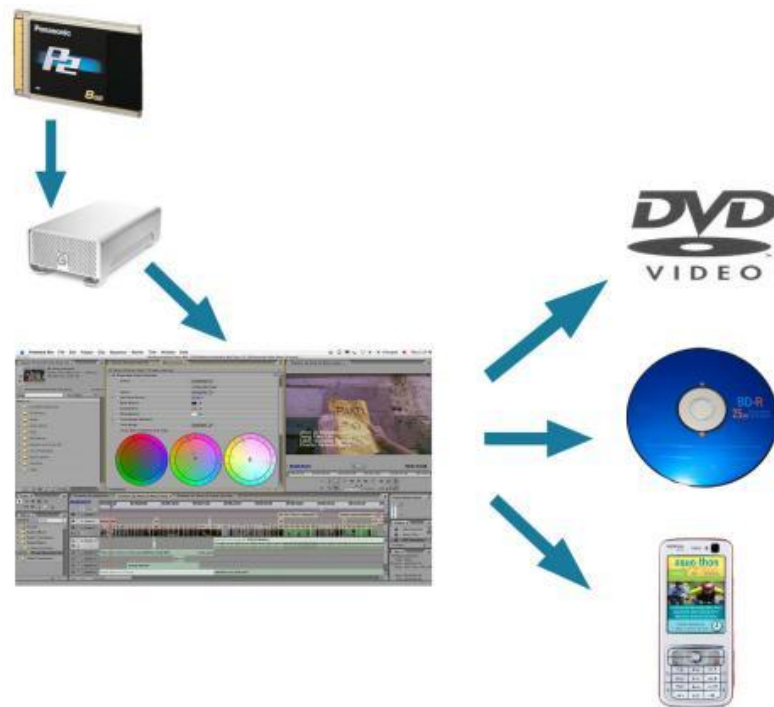


Figure 2.37 – Workflow example using Premiere and P2[38].

Other editors will have similar stages with the workflows overviewed before.

2.5.3 Thomson Grass Valley

Thomson also has a full line of products for production of HD digital media called Infinity. Like P2, recording of video and audio is done on a removable solid state memory card. The benefits of these storage media were already mentioned in sub chapter 2.5.2.

Infinity supports both high definition (HD) recordings and standard definition (SD). SD material is captured using DV compression while HD material uses JPEG 2000 or MPEG-2 intra-frame only compression.

Some Infinity products also support the possibility to record proxy data. Infinity also supports MXF file format on all their products, video, audio and metadata content are written in an open OP1a MXF-based wrapper.

Table 2.6 presents the compression modes available from Infinity, the chroma sampling on each mode and which bit rate is available.

Table 2.6 - Infinity video compression specifications [39].

Compression		
DV25	Both PAL (4:2:0) and NTSC (4:1:1)	
JPEG 2000	SD	HD
	10-bit, 4:2:2	10-bit, 4:2:2
	30-, 40-, 50 Mb/s	50-, 75-, 100 Mb/s
MPEG-2 (requires the DMC 1120 MPEG-2 option board)	SD	HD
	8-bit, 4:2:2	8-bit, 4:2:0
	30-,40-, 50 Mb/s, I-Frame (VBR profile)	60 and 80 Mb/s, I-Frame

2.6 Digital Asset Management

With the transition from tape based to tapeless workflows in the video industry, new challenges have emerged. Issues such as how to manage the increasing libraries of digital media files and metadata so that they can be easily accessed, reused and even redistributed by production teams.

Asset managing is the process of storing, organizing and retrieving electronic digital assets such as photographs, audio visual clips and graphics. The center of most asset managers systems is the database, which references a library of digital assets stored on one or more media storage systems[40].

Asset managers are evolving to an integration of almost all stages present on today’s broadcast and post-production workflows.

Interplay from Avid is an example of one digital asset management product that combines their range of non-linear editors to automate some workflow tasks.

Chapter 3- Automatic ingest Workflow using Avid Interplay

In this chapter I will go through all aspects involved within the work done in the first part of the thesis.

For the development of the module (which was named Toboggan EditorBridge) that allows the automation of the ingest process into the Interplay asset manager, a good knowledge of the Soap protocol and XML was required. The module developed will be integrated in the project “TOBOGGAN for BeijingOlympics2008” and may be used later for future MOG Solutions products/projects.

Avid interplay can be configured to automate routine tasks, track any kind of media, streamline administration, prevent unauthorized access, and accelerate turnaround at every step of the workflow, from pre-production to archive.

Interplay has advanced search tools that give fast access to any asset on the database. Any kind of metadata associated with the assets can be searchable and filtering options give even greater control over the search mechanism.

In Figure 3.1 we can see all the possibilities provided with the usage of the Software Interplay in conjunction with other products.

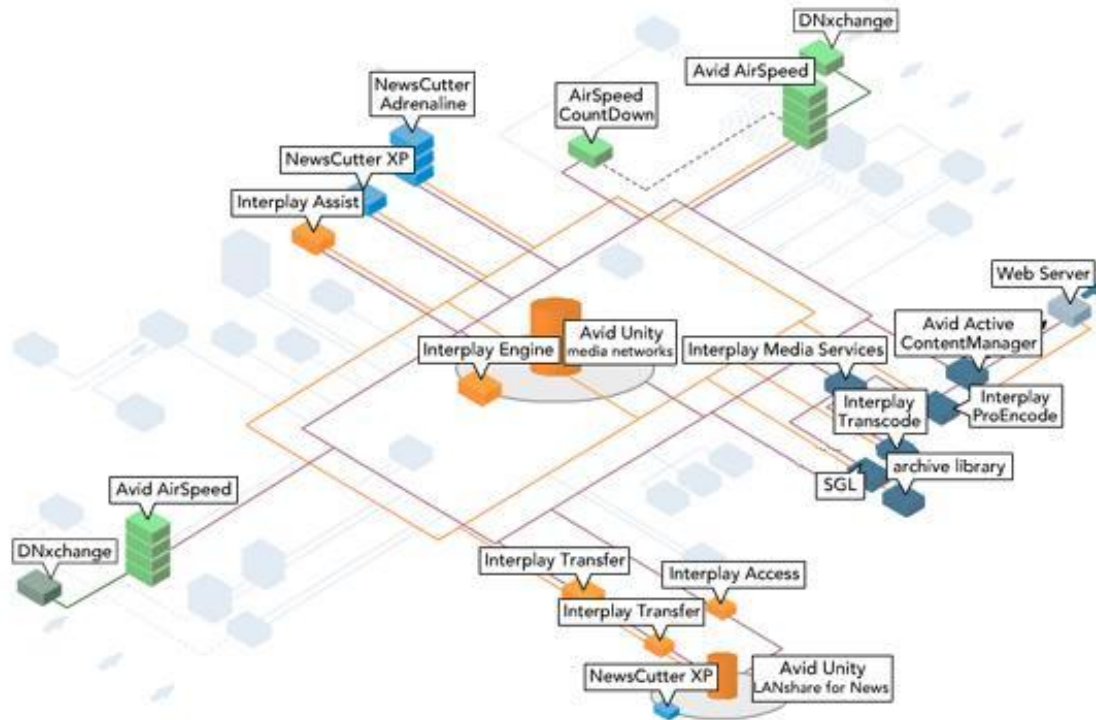


Figure 3.1 – Avid Interplay overview.

Throughout this chapter reference to the application developed will be invoked using the words module and application. Module because it is used in a program built using modular programming.

3.1 Workflow Overview

First to give the reader a better understanding of the usefulness of the application and its place on the complex chain of stages that precedes it, a brief overview of all of these stages will be given hereinafter.

The workflow presented in Figure 3.2 is the one that will be used by The National Broadcasting Company (NBC) in the upcoming Beijing Olympic Games and is a good example of a workflow using the module created (Toboggan EditorBridge). Because most of the parts described in the workflow are outside the scope of this paper they will be described with as much simplicity as possible.

The workflow starts with a user navigating through the assets stored in a server coming from the station control room. This control room is where decisions about which camera goes

live, where management of the sound channels is performed and where additional effects are added to the recorded video media.

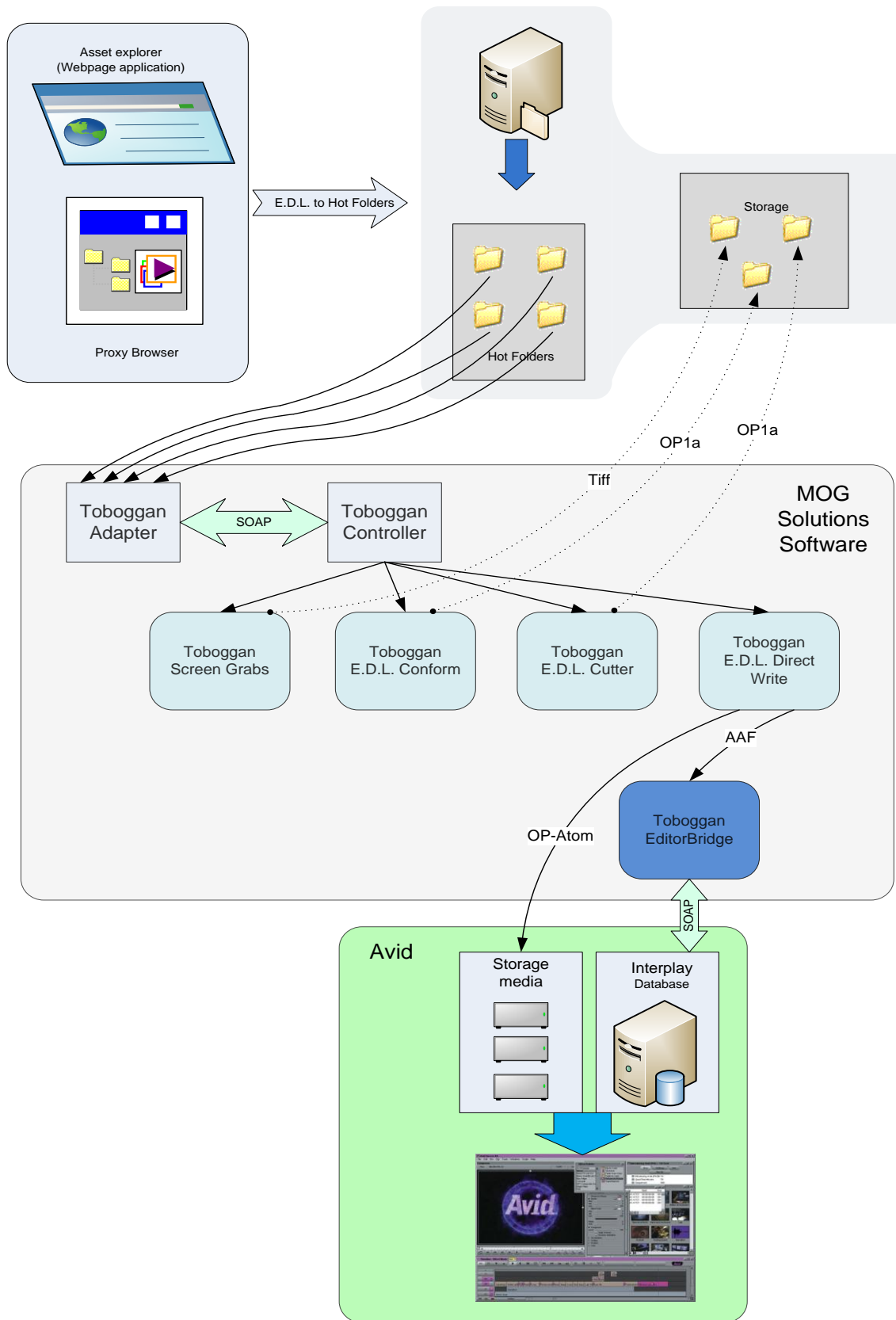


Figure 3.2 – NBC workflow overview.

After selection of the assets wanted, the user goes to a proxy browser application. This application allows the preview of proxy media (proxy media definition already defined in Chapter 2) and the ability to perform simple cuts on the media. This application generates EDLs which is a way of representing a film or video edit, in this case, representing simple editing decisions only. These EDLs will be saved on a server on specific folders (Hot Folders).

The Hot Folders are being watched by the application Toboggan Adapter. Every time new files are created an event is triggered and the application starts to process the file. The application Toboggan Controller will continually request, via SOAP, the Toboggan Adapter if it has any new material to process. If it has, it will distribute them to the correct Toboggan (Toboggan Screen Grabs, Toboggan EDL Conform etc.) depending on the job needed to be done.

These are, in a simple way, the stages preceding the new module in the workflow. If Toboggan EDL Direct Write receives any operation to process, it will use the new module (Toboggan EditorBridge) to automatically ingest the material to the Interplay database, see Figure 3.3.

EDL Direct Write will basically produce OP-Atoms of all the media received at the input, OP-Atoms only have a single track of essence, putting them on a media storage server that is connected with the interplay database. At the input there is the possibility to send instructions to construct sequences using different video essences, so this module will use that information and create an AAF. This means the AAF created can only have a simple sequence dictating to play the video essence from start to the end or have more complex sequences involving different video essences and simple video edits (simple cuts). The AAF created is then sent to Toboggan EditorBridge where it will then be moved to the Interplay database.

This AAF will only reference the media sent (OP-Atoms) to the storage devices using a full path representing the physical location of the media.

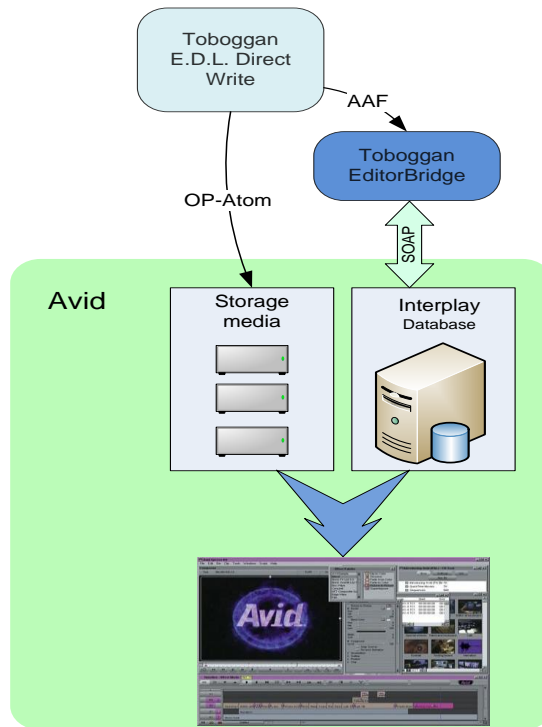


Figure 3.3 – Close up view of the workflow using Toboggan EditorBridge.

In the end after being the media on the database, a user using an Avid video editor can connect to the database search for a specific asset and start editing in just a couple of seconds.

3.2 Toboggan EditorBridge Overview

The module uses the Interplay web services interface which exposes many of the functions supported by the application using SOAP protocol.

A study of the Interplay web services interface was performed before actually starting the development process.

The module had in the beginning an input interface using XML-RPC but due to time constraints that was putted aside and it was used the same input output communication API that is used by all Toboggan MOG Solutions products, Active X.

ActiveX is a component object model (COM) and it was developed with the intent for software interoperability on windows systems.

Figure 3.4 presents the way this module works. In the input we can receive various parameters depending on which function is going to be used from the Interplay web service

API. On the output, indications of the current state of the operation are given and if errors occur a description will also be sent to the output.

This module can be considered as an abstraction layer between MOG Solutions software and the Interplay web service.

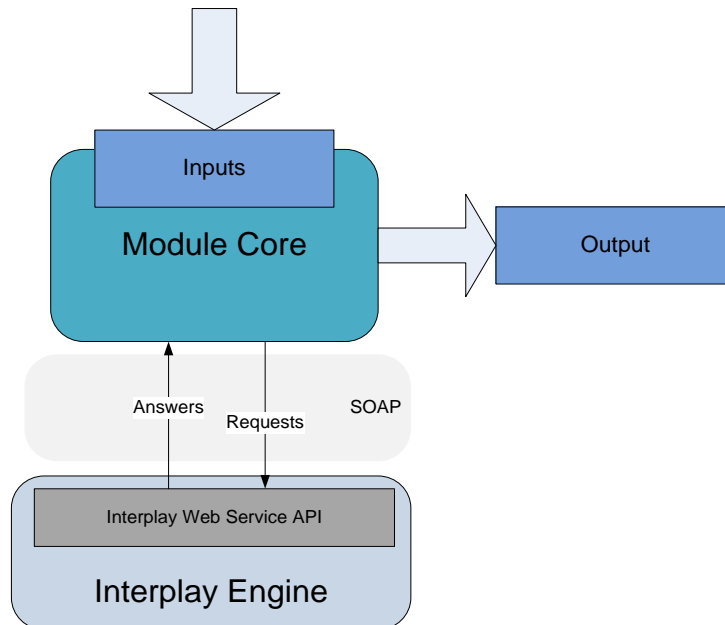


Figure 3.4 – Module overview.

To make the application more future proof and platform agnostic the communication with the web service was built using gSOAP development toolkit which facilitates the development of SOAP/XML Web services in C++.

The use of gSOAP was influenced by the innumerable reviews of different toolkits which stated this was the fastest and the easiest to use, complemented with the fact that is easily ported to different operating systems.

With gSOAP, implementing a client to consume a web service is very fast, provided that the web service has a WSDL. Only two steps are needed to start using the methods exposed by a web service interface. On the first step a tool provided by the toolkit “imports one or more WSDLs and XML schemas to generate a header file with the Web service operations and the C/C++ data types used by the services”[41], in Figure 3.5 this corresponds to the WSDL importer stage. On the second step another tool takes the header file previously generated and generates XML serializers for data types, the client-side stubs, and server-side skeletons. The server-side skeletons are dispensable in this case, because the application developed is a client consuming the remote service. The second step corresponds to the gSOAP Compiler in Figure 3.5.

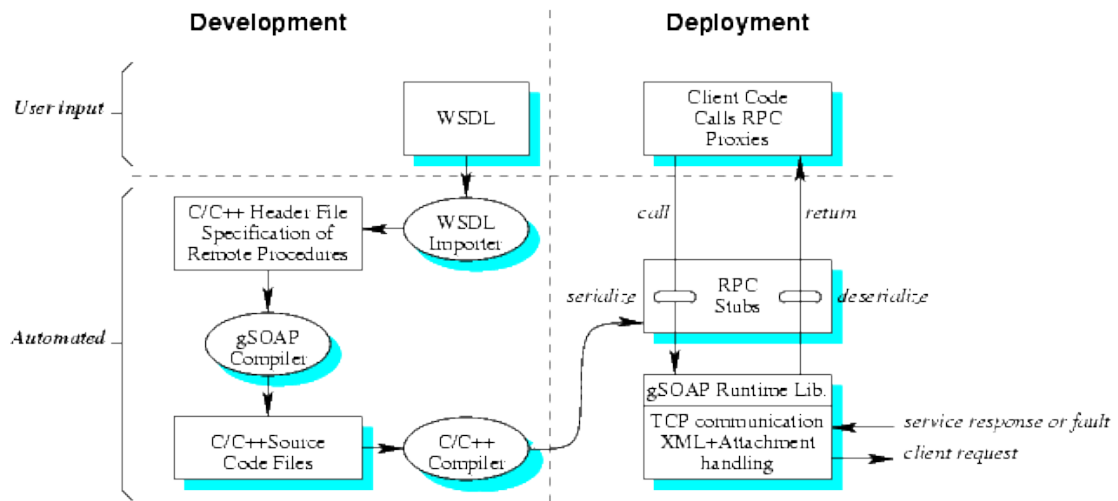


Figure 3.5 – Diagram workflow for client application development [41].

After having the C++ source code files generated by the second tool, they only need to be integrated in the client application project, and access to the web service is now easily done calling the appropriate methods in the code.

The module implements some functions provided by the Interplay web service, while they can all be used right now, for the NBC workflow only one of them was needed. For future projects or products the other functions may be used if needed.

3.3 EditorBridge Architecture

This sub chapter will go through all the functions implemented in this module giving the reader a strong knowledge of the possibilities available using this software.

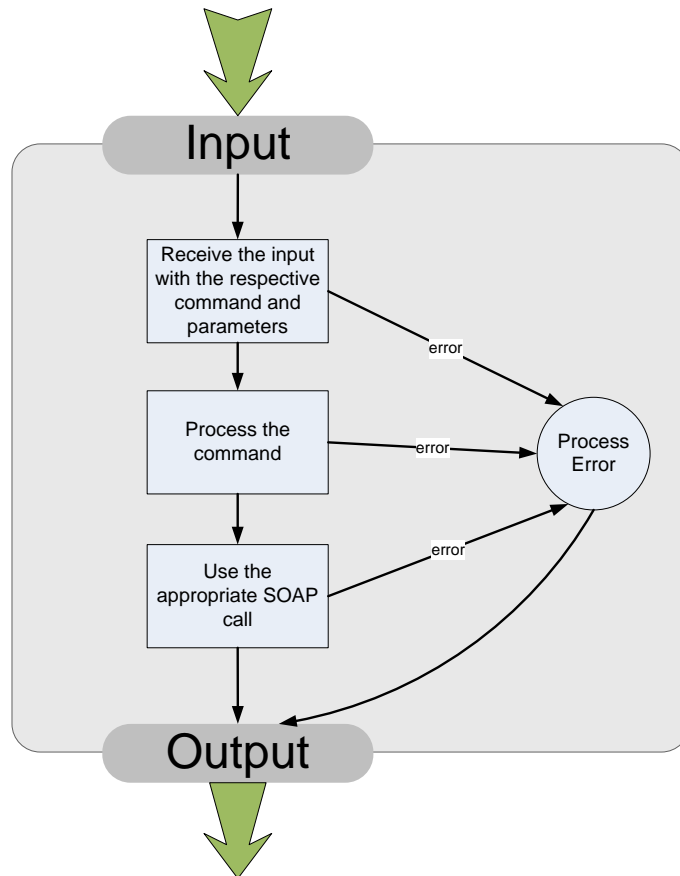


Figure 3.6 – Architecture diagram of Toboggan EditorBridge.

In the first part the application receives something at the input and verifies its syntax if something is incorrect then the application stops, outputting the error description. The input is received as a XML message, so parsing of that message has to be done to retrieve the essential data from it. XML parsing is done using MOG Solutions customized software, see Figure 3.7.

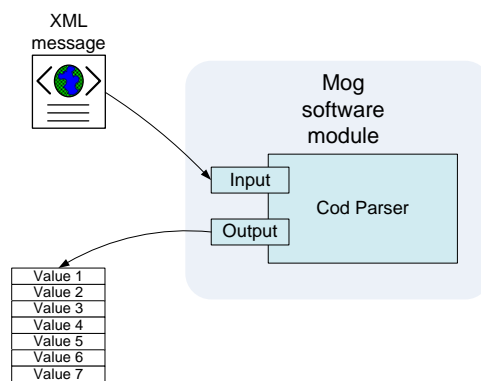


Figure 3.7 – Parsing XML input message with MOG tools.

The module receives as the input the XML message and then provides a variety of functions to retrieve the values from the message.

Secondly, the application processes the data and configures any variable needed, according to the method which is going to be called, using the SOAP protocol.

Thirdly, the application configures the parameters needed to connect to the web service namely the username, the password and the service URL. With these parameters configured the appropriate remote method exposed by Interplay web service is called.

Lastly the application outputs the success or failure of the operation and, eventually the return parameters of the SOAP method call if applicable.

The application supports in total six different functions. Details and usage examples are given hereinafter.

3.3.1 Create folder

This function is used to create a folder or nested path of folders in the asset management database (Interplay). Once created, assets can be check in onto them. Creation of folders with a specific name on them allows the possibility of organizing inter-related content in the same folder, using the name of the folder as a clue for the subject of the content inside it. Organizing the database using a folder structure makes searches across large amounts of media fast and efficient.

In Figure 3.8 it can be seen, on the left, the database content (media, files, and metadata) organized in a folder structure.



Figure 3.8 – Interplay Access application.

To invoke this method only one input parameter is necessary, this being a full path URI using Interplay syntax. Optionally another parameter can be used when calling the remote method, indicating the ownership of the created folder. If the parent folders indicated do not yet exist then they will be created.

As an example calling the method create folder with a URI parameter interplay://folder/folder1/folder2 would create a folder tree structure like the one seen on Figure 3.9.

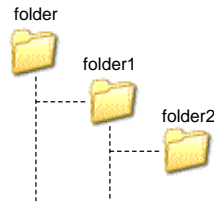


Figure 3.9 – Folder structure.

3.3.2 CheckIn

This function is used to check into Interplay non Avid assets. Non Avid assets can be any file, no constrains are imposed to which type of file can be checked in. If the asset already exists when checking in, then each version will be given a new version number.

When using this function, at the input, a full path pointing to the file that is going to be checked in onto the interplay database has to be given. A parameter indicating where the file will be put in the database has also to be given. Normally this is a full path URI, representing a specific folder location on the database. The file is encoded in Base64 before being sent to the database. The file is then uploaded to the database using the SOAP protocol.

Optionally, metadata can be appended to the file. When calling this remote method from the web service, a list of attributes can be set. The attributes are a pair name-value where the name describes the property. For examples of name-value pairs see Table 3.1.

Table 3.1 – Examples of name-value metadata pairs.

Name	Value
Clip Name	Kid
Take	3 rd Take
Location	Portugal

Some predefined attributes are available to be set. Interplay has also the possibility of defining new custom pairs whenever needed. Interplay distinguishes between SYSTEM attributes and USER attributes, the latter are the only ones that can be given new values using the interplay web service interface. SYSTEM attributes are managed by the database automatically and are read only.

If an attempt is made to set a SYSTEM attribute, the operation will fail with errors.

3.3.3 CheckInAAF

The function `checkInAAF` is used to ingest Avid assets to the Interplay database. Sequences, master clips, sub-clips, and rendered effects are all examples of Avid assets.

Avid assets are usually referenced within Interplay through AAF files. The AAF files only reference the actual media that is normally located on a specific storage server. This is a benefit when using the SOAP protocol to exchange the AAF files, because they only reference the media, they are always relatively small comparing to the sizes of the media (audiovisual media can have up to several gigabytes), making them easy to transport over SOAP (as seen above on 2.1, SOAP protocol is not advisable to be used to transport large amounts of data).

Like in 3.3.2 this function also needs at the input a full path pointing to the file that is going to be checked in onto the database, in this case an AAF file. Attributes can, optionally, also be set and appended with the asset (USER attributes only). Because the AAF file can have metadata referring to the referenced media, when checking in the AAF with new metadata values if any of them was already defined in the AAF file then they are updated with the new value.

The AAF file is also encoded in Base64 before being sent via SOAP. When calling the remote method a URI indicating the location for where the file is going in the database, is necessary.

3.3.4 Duplicate

This function purpose is to duplicate an existing asset, either Avid or File, to another location on the Interplay database. Duplication of folders is not supported.

In the case of an Avid asset, such as a sequence, a new sequence having all the characteristics of the original is created but with its own unique identification. This can enable workflows which require, for example, new sequences to be created from a pre-existing template and, can also be used as a backup methodology keeping always an original copy on the database.

The input parameters needed for calling the remote method correctly are a URI indicating the path of the asset to be copied and a URI indicating a path pointing to a folder where the file will be copied to.

3.3.5 Move

The move function allows, whether it is an Avid asset, a file asset or even an entire folder, to be moved to a different location in the database. This function can be used for workflow management using it to move assets to different folders as they progress in the different workflow stages.

This function has the same input parameters as the ones referenced in 3.3.4.

3.3.6 SetAttributes

The purpose of setAttributes function is to update, create or erase metadata attributes from an asset on the interplay database.

The input parameters are a URI pointing to one asset or a list of URIs pointing each other to a different asset and a list of attributes to set. Only USER attributes can be set leaving SYSTEM ones outside as an option.

The function is very useful because at anytime throughout different workflow stages metadata updates may be necessary. Updating groups of assets is not time consuming using this function because in one iteration an arbitrary amount of assets can be updated at the same time.

3.4 Summary

In this chapter, all the elements that the module referred to were shown - a detailed explanation of each of them was given. The composition of the module was also referred to. An example of a real world implementation was given along with an explanation of its different stages.

A complete module overview of the different technologies used in the construction of the module was written. Special focus was made on how to use gSOAP development toolkit.

In depth analysis of each function supported by the module was given. All the options provided in each function were analyzed and their purpose was explained. The global purpose of each function was also stated.

Chapter 4 – Wrapper from MXF to QuickTime

Movie file format

In this chapter I will go through all the aspects involved in the work done in the second part of the thesis.

The development of this module will allow support from MOG Solutions tools to Final Cut non-linear video editor. The name given to this new module was MXFToQT, throughout this chapter the application developed will be referred to by using its name.

A good understanding of the MXF tools from MOG Solutions was required for the development of the module, since unwrapping of MXF files had to be done in order to gain access to the essences contained on the files.

Like in Chapter 3, application and module will be used both to refer to the program developed in this chapter.

4.1 Introduction

The module purpose arose from the need to support Final Cut Pro non-linear editor using current existent MOG Solutions products and to provide general FCP support to MXF files. Final Cut pro has a peculiarity of not supporting MXF files natively, so rewrapping to a new container format was necessary (QTFF) although the editor supports almost any type of essence existent.

While the market already offers similar solutions they are basically simple MXF converters to QTFF requiring user input (see Figure 4.1) to initiate the process and they don't provide pre-editing options for simple cuts.

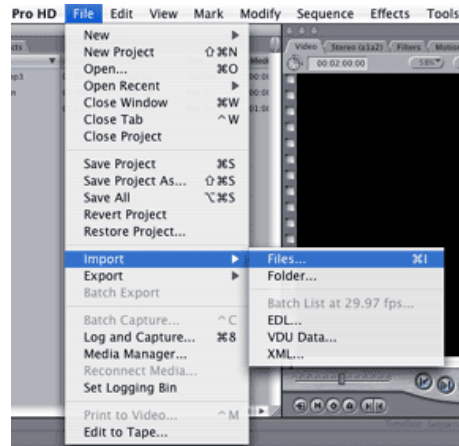


Figure 4.1 – Final Cut Pro import menu.

The module (MXFToQT) supports OP1a and OP-Atom MXF files with possible increased support for other MXF operational patterns files. The types of essences supported are various, video support for HDV, DV, DVC Pro and DVCAM has been achieved. At the current state, audio type supported by the module is PCM, but since most of the recording equipment use PCM for audio streams this does not pose a problem. Motion JPEG 2000 could not be implemented because of lack of time. AVC – Intra support was implemented but due to unavailability of a correct version of Final Cut Pro for testing purposes this functionality cannot be guaranteed.

The module supports both XDCAM and P2. For P2 special attention had to be made because of how P2 organizes the media files.

4.2 Module Overview

MXFToQT uses QuickTime SDK to create new QuickTime movie files. A study of this SDK was made in order to understand how it works and which functions it can provide that are suitable for the development of this module. In Figure 4.2 we can see a simplified diagram exposing the structure of this application.

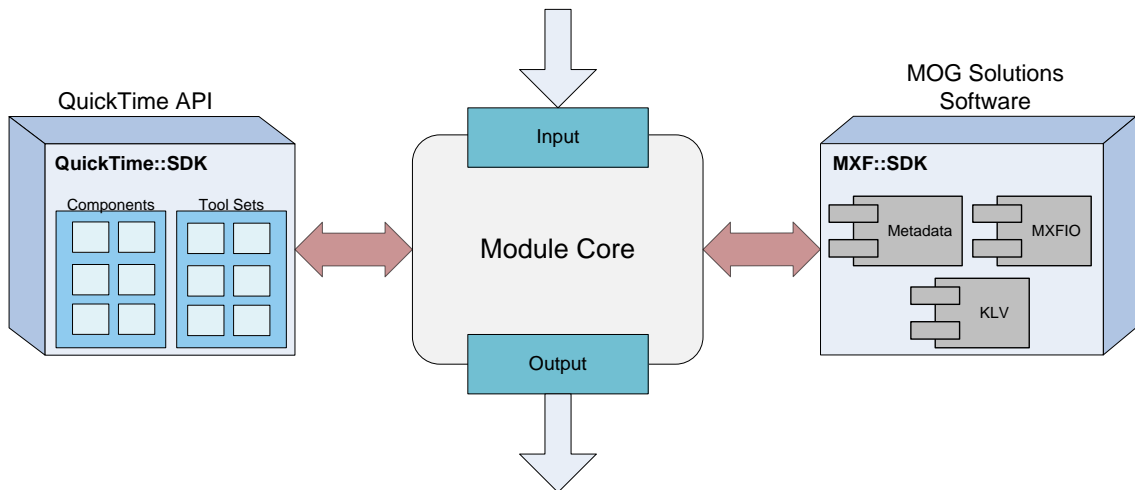


Figure 4.2 – Overview of the MXFToQT module.

Like in Chapter 3 the module exposes its input interface using ActiveX and, also at the input is received a XML message with the parameters needed by the application to function (as seen on Figure 3.7).

At the input, the XML message received should have a list with one or more paths pointing to the physical location of the MXF files that will be converted to the QuickTime format. If files formed by P2 products are on the list, and since for this case multiple MXF files belong to the same clip, the application reads all the files (one video OP-Atom file and multiple OP-Atom sound files) and adds them to a single QuickTime movie file.

Another element that has to be present in the input is the path to where created QuickTime files should be saved. If incorrect input file paths are provided to the application or no output path is provided the application terminates displaying an error.

For reading MXF files and retrieving descriptive information about the essences contained in the files the MXF::SDK from MOG Solutions was used. This provided functions to retrieve the essence data from the files along with the corresponding descriptive metadata. The QuickTime SDK provided the functions to create the QuickTime files. A simplified view of the process is illustrated on Figure 4.3

So with simplicity the module reads the MXF files using MOG software then passes the sample data to the QuickTime API. Before storing the sample data, a description of it has to be given to QuickTime because for an application to be able to read essences it has to first know what type of essence it is.

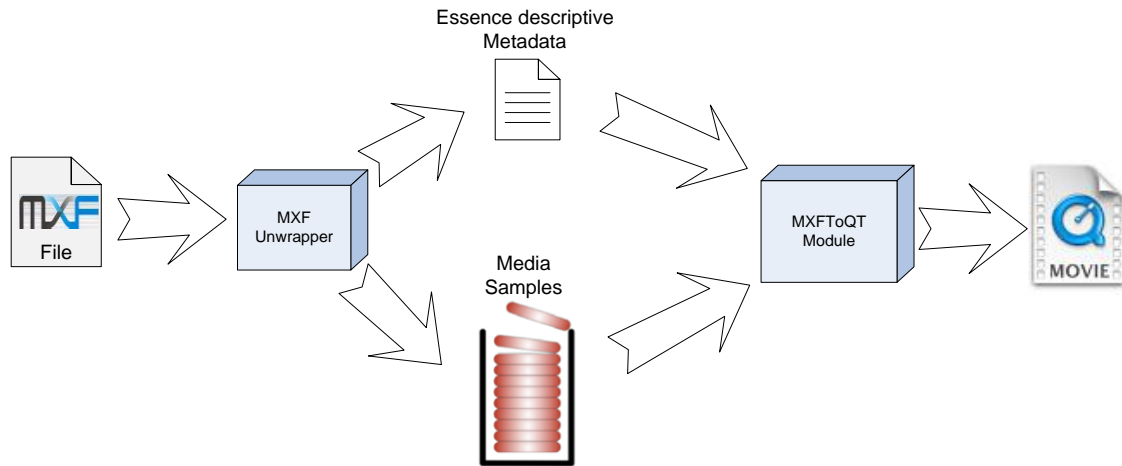


Figure 4.3 – Simplified rewrapping process.

The output of the application is the QuickTime files generated. Also at the output visual information about the current progress and errors, if they occur, is given.

4.3 Module Architecture

This sub chapter will present the architecture of the module. The module was divided into different blocks, each of these grouping a set of inter-related functions.

In Figure 4.4 it can be seen the architecture of the module created. Each of the constituent blocks of the application will be explained in greater detail hereinafter.

MXFToQT starts by receiving at the input one XML message containing the elements to initiate the rewrapping process.

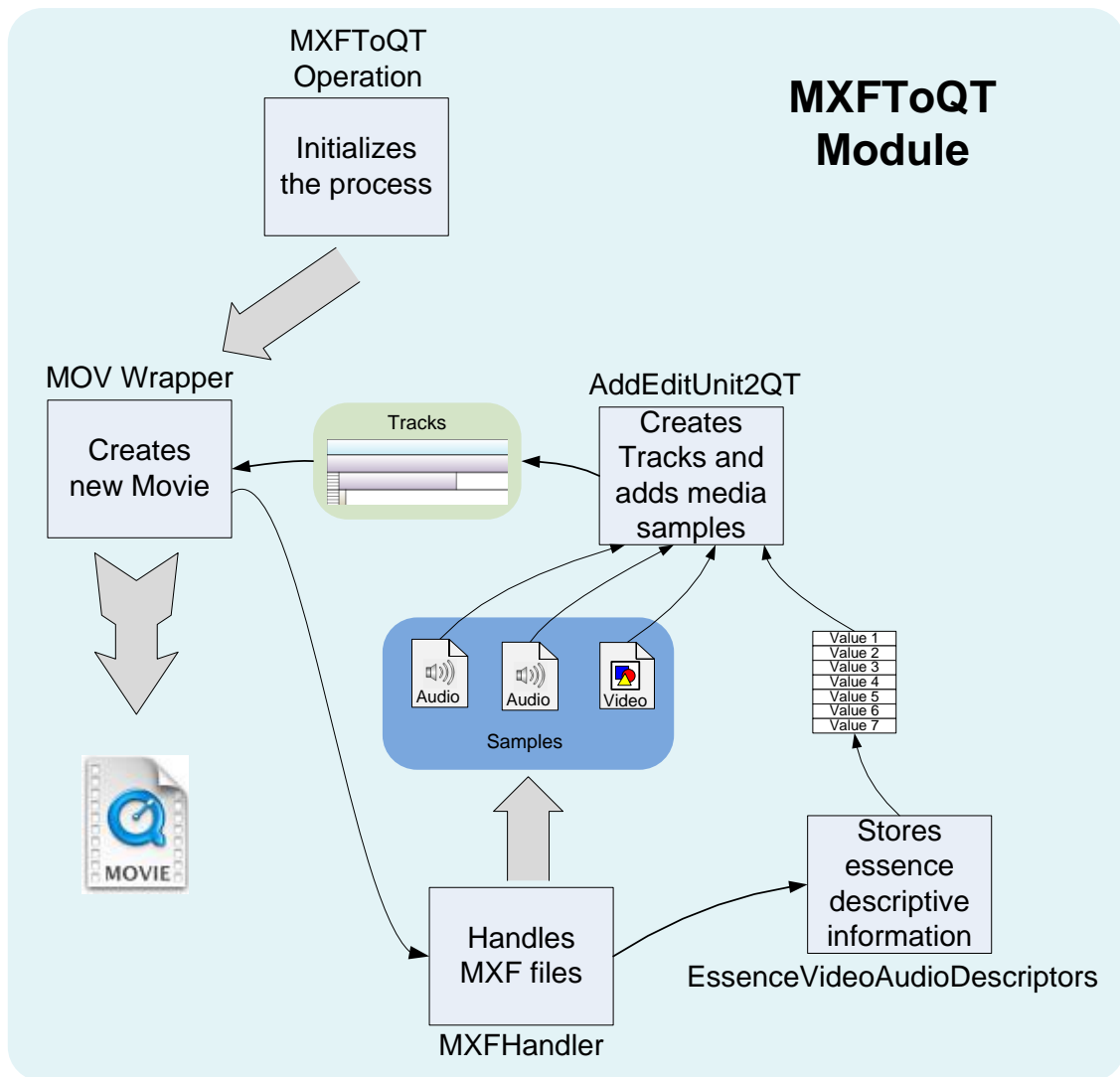


Figure 4.4 – MXFToQT architecture overview.

4.3.1 MXFToQT Operation

In this block all the values incoming through the XML message are stored. Multiple numbers of input paths of MXF files are accepted and stored in a structure. An output path is also stored.

In the next step the application starts the process of the creation of QuickTime movie files. Each MXF file to process will trigger a call to the block MOV Wrapper. In Figure 4.5 we can see the flux of processes happening in this block of the MXFToQT module.

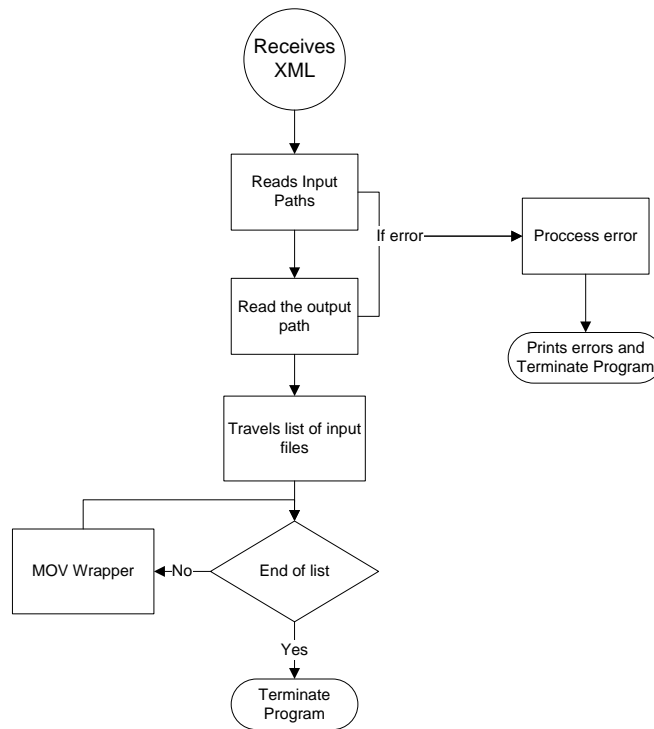


Figure 4.5 - Block diagram of MXFToQTOperation.

When all the list of files is processed the program is terminated. While the program is processing the files, a progress percentage is visually output and if errors occur the program immediately terminates.

4.3.2 MOV Wrapper

This block is where the creation of the QuickTime movie file takes place. In Figure 4.6 is illustrated the diagram for the MOV Wrapper block.

As it can be seen each time the process starts a QuickTime movie is created. After the creation of the movie file the wrapping process starts.

While the complete file has not been wrapped it will continually call the block AddEditUnit2QT. When the end of the wrapping process is reached the block will call a function from AddEditUnit2QT to close the tracks that were created for each of the media elements that were being wrapped. The way QuickTime tracks work was mentioned in 2.4.3.

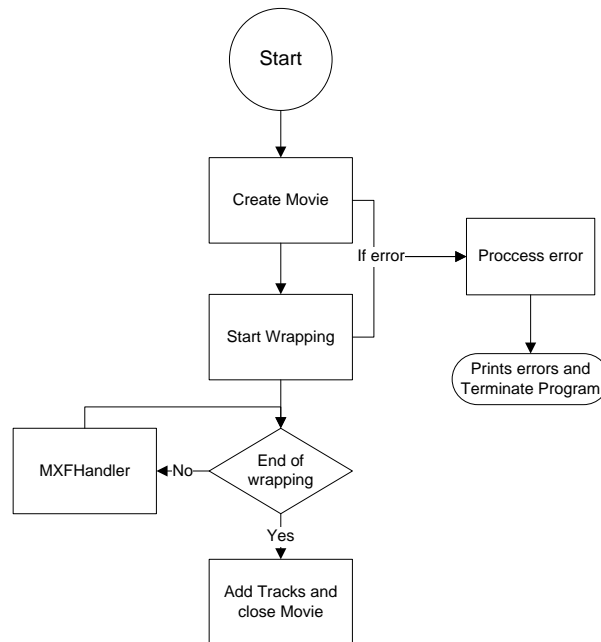


Figure 4.6 - Block diagram of MOV Wrapper.

After closing the tracks they are added to the movie, which is also closed after the tracks are added.

If any error occurs throughout any stage of the block, the program terminates with a description of the corresponding error.

4.3.3 MXFHandler

This block is responsible for all the operations involving MXF files. This is where the actual media is retrieved from the MXF file. This block is also responsible for filling out the image and sound descriptor structures. Figure 4.7 shows a very simplified diagram of the MXFToQT block functioning.

The block starts by opening the MXF file, if any error occurs the application terminates. Afterwards it will prepare the application to start reading the essence.

The next stage is where the application will walk through all the essences elements from the MXF file. The module uses the MOG Solutions essence reader to accomplish this.

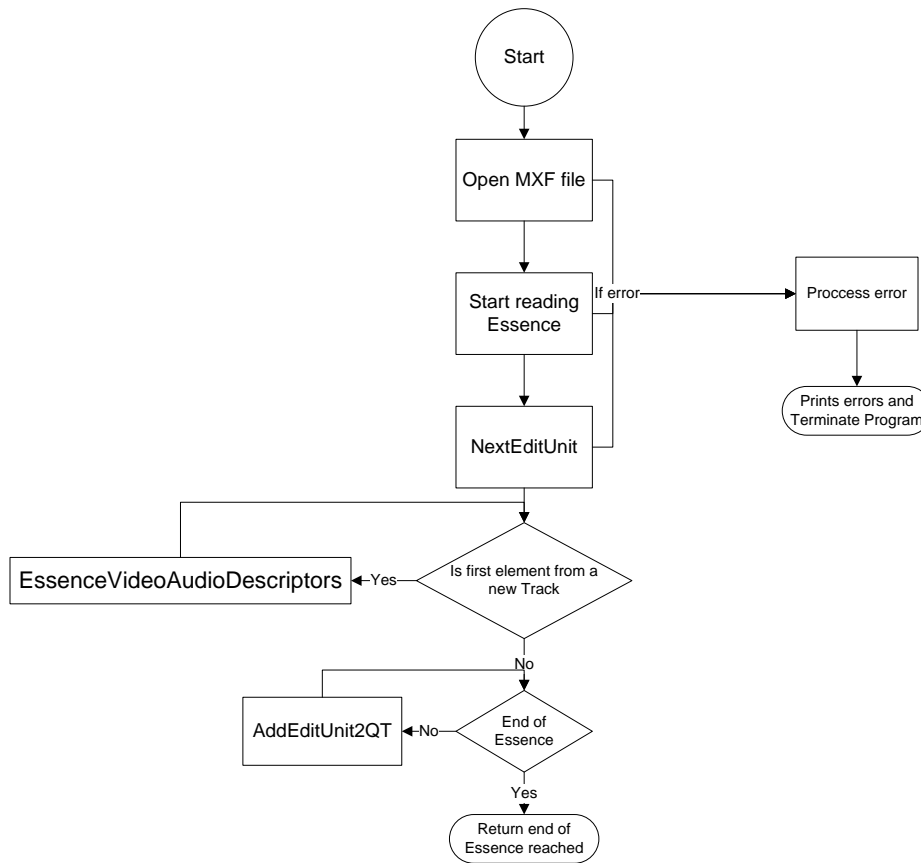


Figure 4.7- Block diagram of MXFHandler.

The essence reader iterates, one by one, each of the Edit Units on the essence container. The essence container is encoded using KLV coding and is normally done like in Figure 4.8 where we see that the data is interleaved. In sub-chapter 2.4.1.4 it was already stated that interleaving elements could be used when storing essence in MXF, at the time a group of elements KLV wrapped was said to be a content package. Edit unit and content package have different meanings, the first meaning one or more pictures that correspond to the minimum editable unit in an MXF file, while the second means a sequence with interleaved elements in a specific order. While they mean different things they very are much related, because all the MXF files used to test this module had a content package equal to an edit unit, they will be used as synonyms in this paper.

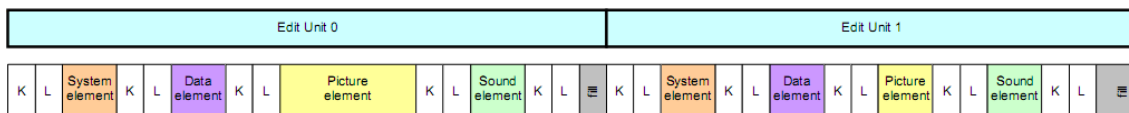


Figure 4.8 - Typical KLV coding in an essence container [6].

So the application goes to the first edit unit and then seeks an element from it. The application filters elements that are not needed, like the system element and the data element. The first time it finds a picture element, before reading its value, it will first fill out the structure

containing the data to describe the very same picture. After reading the picture element it sends its content to the `AddEditUnit2QT` where it will be appended to the new QuickTime file being created.

When finished with an element the application seeks for the next element in the edit unit. For example after reading a picture element if the MXF file has sound essences, it is expected that the next element will be a sound element. Again for sound elements the process is the same as for a picture one. If this is the first time it finds a sound element from a new track it will first fill out the structure containing the data to describe the very same sound. An MXF file can have multiple audio essences (e.g. MXF OP1a file), the structure for describing the sound essences will only be filled out once, it is expected that the sound essences are all of the same type in a MXF OP1a file.

Then the module will send the sound element to the QuickTime movie where it will be added. Because for each type of essence a QuickTime movie needs a different track, the application creates a new track for each track of audio and video on the MXF file.

Special attention was made to prevent that, elements from different MXF tracks were added to the same QuickTime track.

When all the elements from an edit unit are read the essence reader passes to the next edit unit. This process is repeated until the end of the file is reached. When reached the block returns a value indicating that the process has finished.

4.3.4 AddEditUnit2QT

This block is responsible for creating the QuickTime tracks, one track for each of the essence tracks in the MXF file. In the case of Panasonic P2 generated files each QuickTime track will correspond to an MXF OP-Atom file. Figure 4.9 illustrates a diagram showing how this block operates.

QuickTime API provides a set of structures to describe essence and this block takes care of that operation. These structures have to be filled before starting adding any media samples to a QuickTime track.

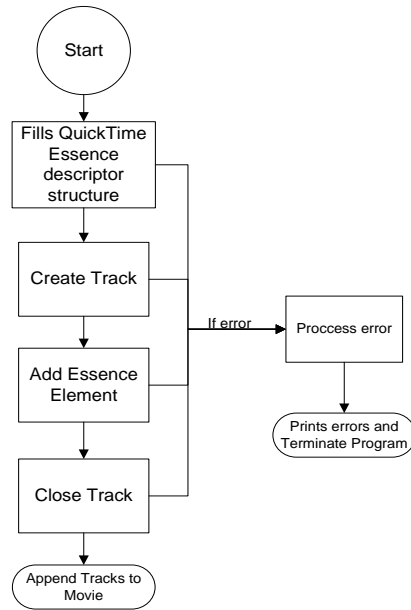


Figure 4.9 - Block diagram of AddEditUnit2QT.

When this block is creating a new track it has to define which type of essence the track will contain. In MXFToQT only two types of tracks are used, video and audio tracks. Definition of the tracks is done using a FourCC code, for video 'vide' and for audio 'soun'. At the creation of a video track the display resolution of the video has to be supplied in conjunction with the video timescale.

For audio tracks no display resolution has to be supplied obviously, although the audio timescale has to be supplied. The QuickTime Movie Toolbox provides a specific function from its API with the solely purpose of adding media samples to a QuickTime track, see Figure 4.10.

```

AddMediaSample2 (
    Media theMedia,
    const UInt8 *dataIn,
    ByteCount size,
    TimeValue64 decodeDurationPerSample,
    TimeValue64 displayOffset,
    SampleDescriptionHandle sampleDescriptionH,
    ItemCount numberOfSamples,
    MediaSampleFlags sampleFlags,
    TimeValue64 *sampleDecodeTimeOut
);
  
```

Figure 4.10 – QuickTime function to add samples to a track.

The function receives as an argument the media associated with a track and which track the samples are going to be added to. **DataIn** argument is the sample data which is being added and the **size** parameter defines its length.

The argument **decodeDurationPerSample** represents the duration of each media sample added specified in media's time scale. The **displayOffset** parameter specifies the offset between the decode time and the display time, this parameter is essential when working with pictures which need reordering between presentation and storage order (as seen in sub-chapter 2.3.3).

Other important parameter is **sampleDescriptionH**, this is where a structure describing the media being added is given. The structure is completely different for audio and video, further details about these structures are given in sub-chapter 4.4.

When adding sample data, this data may comprise various samples in it (e.g. frames), so to indicate how many samples are contained in the data being added to the track, the parameter **numberOfSamples** is used. When adding a sound media sample, the value of **numberOfSamples** is equal to the total length of the sample added (**size** value) divided by bytes per frame. Bytes per frame variable is equal to the number of bytes per channel multiplied by the number of audio channels.

The argument **sampleFlags** is used to give additional information when adding sample media. This parameter was important for the development of this module because with it one can define the type of frame being added ("I", "B" or "P" pictures).

The **sampleDecodeTimeOut** is an output parameter that provides a time value that represents the time where the sample was added. This parameter was only used for debugging purposes throughout the development process so its relevance in this paper is minor.

This function returns an error code if an error occurs. Basically this function is where the wrapping process to QuickTime file format is made.

After adding all the samples to one track this block closes the track and adds it to the QuickTime movie.

4.3.5 EssenceVideoAudioDescriptors

This block is responsible for storing values needed for filling out the structure describing the media being added to a QuickTime track. In this block it is also identified the encoding type of the media being analyzed, for example if a video essence encoded in DV is being read identification of this type of media will be made using a FourCC equal to 'dvc '.

For audio essences, and as mentioned in sub-chapter 4.1, PCM is the only compression supported so it is only needed to verify which type of compression the sound essence has to see if it is supported or not.

4.4 Challenges faced

Throughout the development of the MXFToQT module innumerous difficulties were encountered. In this sub-chapter detailed information about the difficulties encountered when working with the QuickTime SDK and the MOG Solutions software is given.

4.4.1 Image Description Structure

As mentioned before, the image description structure needs to be constructed before adding video media samples to a QuickTime track. An image description structure contains information that defines the characteristics of a compressed image or sequence. Things like compression the used, the number of fields per image, the aspect ratio, the resolution and other properties are defined in this structure.

A FourCC code has to be given to indicate which type of compression the image has. Apple provides a list of all of them along with the ones coming with final cut pro software, followed with a description of each one.

The picture resolution is stored in this structure indicating the width and height of the image. Note that this resolution may be different than the display resolution, for example, in interlaced video which has 2 fields, for each frame the picture resolution of the sample being added is smaller than the display resolution of the video (display height is two times bigger than the picture resolution because a frame is composed by two fields).

The QuickTime API provides functions to add extensions to this image description structure. Extensions for fields and aspect ratio were used for the development of this module.

In the fields extension it was set two properties. The first property states the number of fields of each picture, non-interlaced videos have one field while interlaced have two. The other property says which field is considered to be temporally the first field of an interlaced frame.

The aspect ratio extension sets the pixel aspect ratio of the sample being added to the QuickTime movie track. For television the most used aspect ratios are 4:3 and 16:9.

4.4.2 Sound Description Structure

A sound description structure has to be constructed when adding audio essences. This structure is used to describe the type of sound media being added to a QuickTime track. PCM sound is the only type of audio supported by MXFToQT module so, only the necessary parameters to identify a PCM compressed sound essence are used.

To describe PCM audio, an audio sample rate has to be set. This sampling rate is much bigger than the video sampling rate because of the different characteristics audio has.

A FourCC stating the sound essence type has to be provided. Because only PCM is supported this property is always the same.

The number of audio channels is also set. For now the module only supports mono and stereo channel layouts.

4.4.3 Closed GOP or Open GOP

A group of pictures (GOP) can be closed or open. A GOP is open if it references anything in another GOP and is closed if to decode all the pictures enclosed in the GOP no information is required from other GOPs.

When adding samples compressed with long GOP MPEG encoding, the information stating if the GOP is open or closed has to be given to the QuickTime API.

This information is described on **sampleFlags** argument in the **AddMediaSample2** function, provided by the QuickTime Movie Toolbox. Open-GOP “I” frames need to be marked as partial sync samples while Closed-GOP “I” frames need to be marked as sync samples.

Figure 2.12 illustrates the differences between open and closed GOPs. The purpose of open GOPs is to increase even further the compression rate.

If this information is not correctly described when adding media samples to a QuickTime track then the resulting video will play erratically.

MOG Solutions essence reader does not provide any function to identify if an “I” frame is from an open GOP or from a closed GOP. This functionality was added to their software and, it was accomplished by reading if any “B” frames contained in a GOP have dependencies pointing to any frame outside of the GOP.

4.5 XML Interchange Format

The Final Cut Pro XML Interchange Format provides extensive access to the contents of Final Cut Pro projects, including edits and transitions, effects, layer-compositing information, and organizational structures. The XML interchange format provides the possibility to exchange information from any Final Cut project to other applications or systems for example, asset management systems, database systems and broadcast servers.

The XML interchange format is very similar to AAF. Both of them allow the transfer of authoring, editing, and media-management metadata between different systems. The biggest difference between them is that, one is simply a plain text file format that anyone can read and manipulate, and the other is a binary data file stored in Microsoft Structured Storage which is harder to manipulate[42].

A study of this format was made to integrate into MXFToQT the option of constructing sequences with different media files, and offer the possibility of having them presented in the Final Cut Pro application. Figure 4.11 exemplifies what was pretended when using the XML Interchange format, the application would generate multiple files and then, if wanted, a sequence with one or more files and simple cuts could be constructed using a XML Interchange format file.

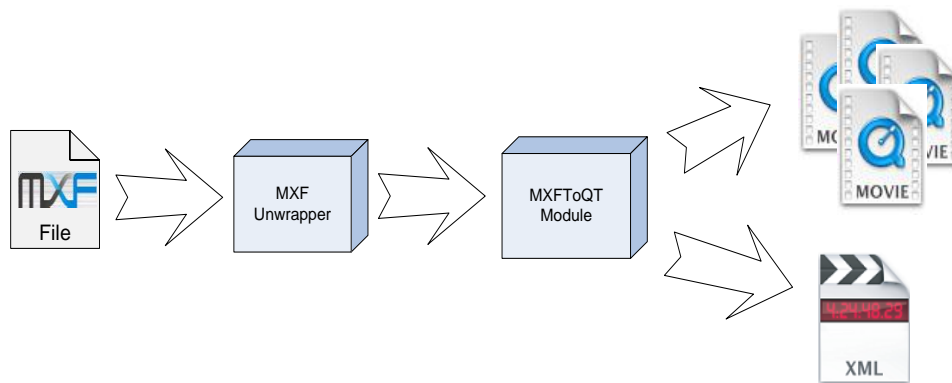


Figure 4.11 – MXFToQT using XML Interchange Format.

To begin the implementation process a XML document creation tool had to be made since, MOG Solutions tools to deal with XML offer only reading and parsing options. The tool relies on Document Object Model (DOM) to navigate through the XML file being created. Actually the file is only physically created in the end, being kept in memory for easier manipulation, while the elements have not been all introduced.

Beyond the base conventions of XML, the Interchange format uses two additional conventions that affect how a document is parsed and translated, inheritance and the id attribute.

The inheritance convention avoids the encoding of shared elements more than once. For example if the duration of one clip is specified in the clip element when associating media to the clip with a file element this new element can inherit the duration information from the clip parent element. This avoids repeating elements throughout the XML document.

The id attribute allows sharing between elements that are not in an element/sub-element relationship with each other. For example if a clip is encoded in the XML file using one id attribute (`<clip id = "Id_Clip">`) after, it can be added to a sequence only referring its id (`Id_Clip`) without the need to copy all the content of the clip element.

In Figure 4.12 a XML file using MOG Solutions XML document creation tool is presented. An examination of what is contained on this XML document is provided hereinafter.

As we can see the first element in the document (excluding the header which specifies the XML version number and the character encodings) "`<xmml version="3">`" has an attribute with the name version. The version attribute specifies which version of the XML Interchange format is being used.

The "`<project>`" followed by "`<name>Random name</name>`" defines a new Final Cut Pro project along with its name.

The element "`<clip id="Random name">`" defines a new Final Cut clip, using an id attribute allows the reference of this element on another parts of the XML document. Following this element other elements describing the clip are encoded in the XML file. Elements like the duration of the clip, the rate and the name of the clip that will be shown on the Final Cut interface, are some of the elements that can be encoded.

A clip needs to be associated with a file element. The file element associates the clip with the physical media file. The child element "`<pathurl>`" states the path pointing to where the media file is located. Some elements like the element "`<duration>`" do not need to be indicated in the XML document, they can be inherited from the parent element "`<clip>`".

Finally the other elements define the characteristics of the different essences inside the file being referenced. Also, descriptive metadata can be added using the XML Interchange format.

The implementation could not be finished in the scheduled time to develop the thesis, thus still missing the creation of sequences. Nevertheless, the foundations for finishing the implementation are set. See sub-chapter 5.2 for more developments on this subject.

```

<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<xmeme1 version="3">
  <project>
    <name>Random name</name>
    <children>
      <clip id="Random name">
        <updatebehavior>add</updatebehavior>
        <name>Random name</name>
        <duration>93</duration>
        <rate>
          <ntsc>FALSE</ntsc>
          <timebase>25</timebase>
        </rate>
        <in>0</in>
        <out>93</out>
        <file id="Random name file">
          <name>Random name</name>
          <pathurl>file://localhost/Video.mov</pathurl>
          <media>
            <video>
              <duration>93</duration>
              <samplecharacteristics>
                <width>720</width>
                <height>576</height>
              </samplecharacteristics>
            </video>
            <audio>
              <samplecharacteristics>
                <samplerate>48000</samplerate>
                <depth>16</depth>
              </samplecharacteristics>
              <channelcount>1</channelcount>
              <layout>mono</layout>
            </audio>
            <audio>
              <samplecharacteristics>
                <samplerate>48000</samplerate>
                <depth>16</depth>
              </samplecharacteristics>
              <channelcount>1</channelcount>
              <layout>mono</layout>
            </audio>
          </media>
        </file>
      </clip>
    </children>
  </project>
</xmeme1>

```

Figure 4.12 – Encoded Clip in XML Interchange format.

4.6 Workflow Example

This sub-chapter presents a workflow example using the module discussed throughout the chapter.

Figure 4.13 illustrates the example that is going to be discussed in this sub-chapter.

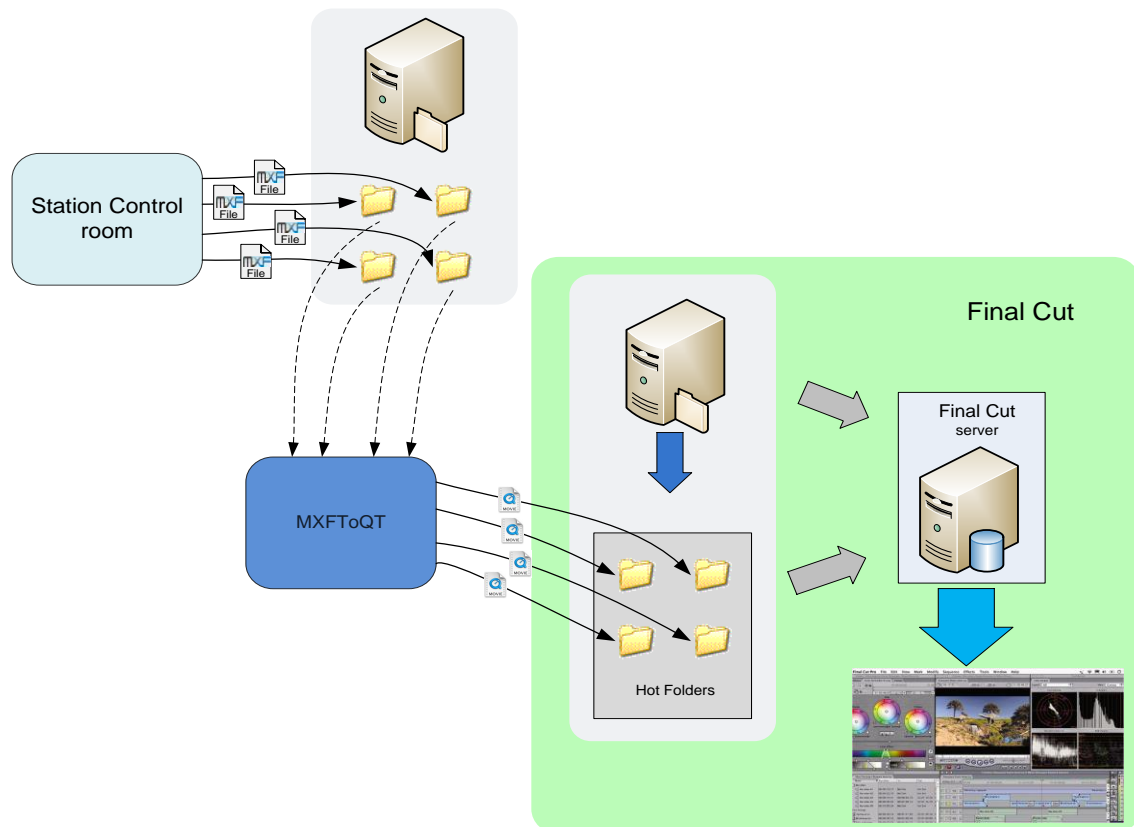


Figure 4.13 – Workflow example using MXFToQT.

This workflow tries to mimic a real situation happening in a television station. The workflow starts with assets being stored in a server coming from the station control room (control room definition already stated in 3.1). The files can then be sent automatically to the MXFToQT application.

The MXFToQT application will convert the MXF files to QuickTime movie files and send them to another server where they are stored.

Using the Final Cut Server application, which is a media asset manager, automation events can be created. For this example a folder watcher could be created to trigger an event every time new files were created in the hot folders. The event could create a response, which

would consist of cataloging the new files in the database, sending afterwards a warning to a user saying that new media had arrived and where it was located.

The user could then open the non-linear video editor Final Cut Pro, navigate to the location where the files are and start editing the media that had just arrived.

4.7 Summary

This chapter shows all the elements that the module referred to entails. An introduction stating in which context the application falls was also undertaken.

A complete module overview of the tools used was presented, stating their function and the reason for their usage. The main tools used and focused on were the QuickTime SDK and the MOG Solutions MXF SDK. An overview of how the module works was also presented.

The module was divided in different blocks each one with a specific function associated. All these blocks were reviewed and discussed with the help of a block diagram illustration.

An introduction to the XML Interchange format was also given followed by one example of where the application could be used.

Chapter 5 – Conclusions and Future Work

This chapter presents the conclusions reached with the development of this thesis, and ideas for future developments that can be done to improve or add completely new functionalities to the modules created.

5.1 Final Remarks

The main objective of developing solutions to automate tasks in a video production workflow was complied. Two applications were developed, each one with very different functionality but both serving the same purpose of automating tasks in digital video tapeless workflows. The conclusions and results obtained with the developed applications are discussed hereinafter.

Throughout the development process, a sense of which decisions could provide a reduction of user input was always kept. The modules needed to be configured initially before being used, and once this was done they functioned in a completely autonomously way with no user intervention.

Detailed information of the developed modules was given with the aid of flowcharts. The modules were first tested inside MOG Solutions and only after approval had been given, they were ready for commercialization.

It was verified that Toboggan EditorBridge has indeed made the automatic ingest of assets into Avid Interplay a reality. Without this module, it was needed to have allocated resources undertaking this specific task. Even though the module has more features supported, by using only the possibility of ingesting automatically media into Interplay it has already proven to be a very valuable tool. Toboggan EditorBridge is already being tested in NBC headquarters, and with good results.

Conclusions and Future Work

The decision of making these modules in C++ has proven to be worth it. Indeed a much faster integration with other MOG Solutions products was possible. Using Web Services on C++ opened the road for a future conversion of all MOG Solutions products which used Web Services. Until now all Web Services products have been developed using C# on the .NET framework so, changing them to C++ enables the possibility of its usage in a wider range of operating systems.

MXFToQT module on the other hand will allow an automation of tasks in a workflow using Final Cut Pro software as the non-linear video editor. The number of applications capable of reading QuickTime movie files is bigger than the applications supporting MXF files, so this is an added benefit of the module developed.

The usage of this module combined with Final Cut Server will offer almost unlimited ways of automating tasks. For example something similar with Toboggan EditorBridge module is possible to be done.

Concluding, using these two tools and joining them with other tools from MOG Solutions makes it possible to automate innumerable tasks on a digital video workflow.

5.2 Future Work

The modules developed have innumerable new functionalities that can be added easily in the future. Thus, some suggestions for these new functionalities are presented hereinafter.

Starting with the module Toboggan EditorBridge, an interesting new functionality would be adding support to a function provided by Interplay Web Services interface that lists the child elements of a folder. After adding support for this, it could be made a function that could retrieve for example the folder structure of the entire Interplay database. This could be useful to provide a visual interface with the Interplay database folder structure to the user, for choosing where the next assets would go to.

Another functionality that can be added to Toboggan EditorBridge is support for sub clip generation. This could provide the possibility of doing simple cuts on the media already inside the interplay database. Metadata could automatically be attached to the sub clips differentiating them from the master clips that generated them.

For the MXFToQT module, finishing up its integration with the Final Cut XML Interchange format will add the possibility of automatically generating sequences with the different QuickTime files being created. The possibility of supporting simple cuts on the media is also added. Lastly the possibility of clicking one file, which opens Final Cut Pro with a new project and multiple files already included on the project, could be provided using the XML Interchange format.

Both of these solutions use an input interface based on Active X. Because this technology only works in windows platforms, a future development could be transforming this interface to a SOAP one thus, enabling multi platform support.

In conclusion the objectives originally proposed have been accomplished and perspectives for future work on this area look promising.

References and Bibliography

- [1] SMPTE., "SMPTE 336M." *Data Encoding Protocol using Key-Length-Value*. [Standard]. 2007.
- [2] European Commission., "Transition from analogue to digital broadcasting." [Online] September 17, 2003. [Cited: May 30, 2008.] <http://lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:52003DC0541:EN:HTML>.
- [3] Dictionary.com., WordNet® 3.0. Princeton University. [Online] [Cited: June 3, 2008.] <http://dictionary.reference.com/browse/assets>.
- [4] Techweb., "TechEncyclopedia." *interframe & intraframe coding*. [Online] [Cited: June 5, 2008.] <http://www.techweb.com/encyclopedia/defineterm.jhtml?term=interframecoding>.
- [5] SMPTE., "SMPTE Engineering Guideline." *Annotated Glossary of Essential Terms for Electronic Production*. [EG 28]. 1993.
- [6] —. "SMPTE 377M." *Material Exchange Format (MXF) - File Format Specification*. [Standard]. 2004.
- [7] International Telecommunication Union., "ITU-T Recommendation H.262." *Information technology – Generic coding of moving pictures and associated audio information: Video*. February 2000.
- [8] Apple Inc., QuickTime. *File Format Specification*. [Online] August 4, 2007. [Cited: June 18, 2008.] <http://developer.apple.com/documentation/QuickTime/QTFF/qtff.pdf>.
- [9] Federal Communications Commission., The digital TV Transition. *DTV*. [Online] [Cited: May 29, 2008.] <http://www.dtv.gov/index.html>.
- [10] Annala, Raino., Fideocam Blog. *Digital Video Workflow*. [Online] November 23, 2006. [Cited: May 30, 2008.] <http://fideocam.wordpress.com/2006/11/23/production-workflow/>.

References and Bibliography

- [11] Payam, Shodjai., Web Services and the Microsoft Platform. *What Are Web Services?* [Online] Microsoft Corporation, June 2006. [Cited: June 25, 2008.] http://msdn.microsoft.com/en-us/library/aa480728.aspx#wmsplat_topic2.
- [12] Microsoft Corporation., Web Services Technical Articles. *Understanding WSDL*. [Online] October 2003. [Cited: June 2, 2008.] <http://msdn.microsoft.com/en-us/library/ms996486.aspx>.
- [13] Systinet Corp., Web Services: A Practical Introduction. [Online] October 16, 2003. [Cited: June 4, 2008.] <http://www.alignjournal.com/whitepapers-pdf/SystinetResPaper.pdf?CFID=191141&CFTOKEN=60763434>.
- [14] World Wide Web Consortium ., About XML. *Extensible Markup Language* . [Online] May 22, 2008. [Cited: June 25, 2008.] <http://www.w3.org/XML/>.
- [15] Panasonic., "White papers." *AVC-Intra (H.264 Intra) Compression*. [Online] September 7, 2007. [Cited: June 8, 2008.] ftp://ftp.panasonic.com/pub/Panasonic/Drivers/PBTS/papers/WP_AVC-Intra.pdf.
- [16] Nattress, Graeme., "Nattress Productions Inc." *Chroma Sampling: An Investigation* . [Online] July 2005. [Cited: June 10, 2008.] http://www.nattress.com/Chroma_Investigation/chromasampling.htm.
- [17] Devlin, Bruce and Wilkinson, Jim., *The MXF Book*. United States of America : Elsevier, 2006.
- [18] International Telecommunication Union., "Recommendation T.81." *Digital compression and coding of continuous-tone still images - Requirements and guidelines*. [Online] September 1992. [Cited: June 8, 2008.] <http://www.w3.org/Graphics/JPEG/itu-t81.pdf>.
- [19] SMPTE., "SMPTE 314M." *Data Structure for DV-Based Audio, Data and Compressed Video - 25 and 50 Mb/s*. [Standard]. 2005.
- [20] Sony., "XDCAM." *Codec Technology for XDCAM Tapeless Products and Systems*. [Online] 2007. [Cited: June 11, 2008.] http://pro.sony.com/bbsccms/assets/files/micro/xdcam/articles/XDCAM_WhitePaper_F.pdf.
- [21] 2KAN., Online guide . *JPEG2000 - F.A.Q.* [Online] [Cited: June 17, 2008.] <http://www.jpeg.org/.demo/FAQJpeg2k/index.htm>.
- [22] NICKY PAGE 2000., NTSC, PAL & Interlace Explained . *Television*. [Online] [Cited: June 27, 2008.] <http://www.digital-digest.com/nickyguides/interlace.htm>.

- [23] SMPTE., "SMPTE 378M." *Material Exchange Format (MXF) - Operational pattern 1A (Single Item, Single Package)*. [Standard]. 2004.
- [24] —. "SMPTE 381M." *Material Exchange Format (MXF) - Mapping MPEG Streams into the MXF Generic Container*. [Standard]. 2005.
- [25] —. "SMPTE 380M." *Material Exchange Format (MXF) - Descriptive Metadata Scheme-1 (Standard, Dynamic)*. [Standard]. 2004.
- [26] Carol Chou., "Action Plan Background." *QuickTime*. [Online] January 24, 2005. [Cited: June 16, 2008.]
http://www.fcla.edu/digitalArchive/pdfs/action_plan_bgrounds/quicktime_bg.pdf.
- [27] Apple Inc., "Movie Creation." *QuickTime Movie Creation Guide*. [Online] January 8, 2007. [Cited: June 18, 2008.]
<http://developer.apple.com/documentation/QuickTime/RM/CreatingMovies/MTCreateMovies/MTCreateMovies.pdf>.
- [28] Apple Inc., QuickTime. *QuickTime Overview*. [Online] August 11, 2005. [Cited: June 20, 2008.]
<http://developer.apple.com/documentation/QuickTime/RM/Fundamentals/QTOverview/QTOverview.pdf>.
- [29] Sony., XDCAM. *XDCAM: Overview*. [Online] [Cited: June 4, 2008.]
<http://pro.sony.com/bbsc/ssr/micro-xdcam/resource.latest.bbsscms-assets-micro-xdcam-latest-xdcamoverview.shtml>.
- [30] Sony Corporation., "DVCAM Format Overview." [Online] 2000. [Cited: June 4, 2008.] <http://www.sony.ca/dvcam/pdfs/dvcam%20format%20overview.pdf>.
- [31] Hugo Gaggioni, Sony Electronics Inc., "XDCAM." *Codec Technology for XDCAM Tapeless Products and Systems*. [Online] 2007. [Cited: June 4, 2008.]
http://pro.sony.com/bbsccms/assets/files/micro/xdcam/articles/XDCAM_WhitePaper_F.pdf.
- [32] Precision Camera Inc., The NEW Way to Shoot. *The Future Shape of Broadcast Production*. [Online] 2007. [Cited: June 4, 2008.] <http://www.pci-canada.com/XDCAM.htm>.
- [33] Adobe Systems Incorporated., "Workflow Guide." *Using Adobe Premiere Pro with Sony XDCAM content*. [Online] April 8, 2008. [Cited: June 8, 2008.]
http://www.adobe.com/products/premiere/pdfs/PremierePro_xdcam_workflowguide.pdf.

References and Bibliography

- [34] Sony Corporation., "XDCAM Professional Disc System." *XDCAM Workflow Guide*. [Online] 2007. [Cited: June 5, 2008.]
<http://www.sony.co.uk/res/attachment/file/95/1133797571595.pdf>.
- [35] Panasonic., "P2 HD." *Frequently asked questions*. [Online] January 24, 2008. [Cited: June 7, 2008.]
ftp://ftp.panasonic.com/pub/Panasonic/Drivers/PBTS/papers/FAQ_P2HD.pdf.
- [36] —. "AVC-Intra." *Frequently asked questions*. [Online] [Cited: June 8, 2008.]
<ftp://ftp.panasonic.com/pub/Panasonic/Drivers/PBTS/papers/AVCIntra%20FAQs.pdf>.
- [37] —. "White Papers." *Using P2 HD with Final Cut Pro*. [Online] [Cited: June 8, 2008.] ftp://ftp.panasonic.com/pub/Panasonic/Drivers/PBTS/papers/WP_P2Workflow_FCP.pdf.
- [38] Adobe Systems Incorporated., "Workflow Guide." *Using Adobe Premiere Pro with Panasonic P2 content*. [Online] October 10, 2007. [Cited: June 8, 2008.]
<http://www.adobe.com/products/premiere/pdfs/p2workflowguide.pdf>.
- [39] Thomson Grass Valley ., Product data sheet. *DMC 1000 digital Media camcorder*. [Online] 2008. [Cited: June 16, 2008.]
<http://thomsongrassvalley.com/docs/DataSheets/infinity/camcorder/CAM-3024D-7.pdf>.
- [40] Claman, Tim., "Digital asset management." *Broadcast Engineering magazine*. April, 2007.
- [41] "gSOAP: C/C++ Web Services and Clients." *Introduction*. [Online] [Cited: June 18, 2008.] <http://www.cs.fsu.edu/~engelen/soap.html>.
- [42] Apple Inc., Final Cut Pro. *XML Interchange Format*. [Online] November 14, 2007. [Cited: June 24, 2008.]
http://developer.apple.com/documentation/AppleApplications/Reference/FinalCutPro_XML/FinalCutPro_XML.pdf.

Appendix A - Video Recording Formats

System		PAL-DV	NTSC-DV	HDV HD1	HDV HD2	HDCAM	HDCAM SR	XDCAM SD		XDCAM HD	XDCAM EX	DVCPRO25	DVCPRO50	DVCPRO HD		
	Manufacturer	various	various	JVC	Canon Sony	Sony		Sony			Panasonic					
Video	Quantization	8 bit	8 bit	8 bit	8 bit	8 bit	10 bit	8 bit		8 bit	8 bit	8 bit	8 bit	8 bit		
	Compression	DCT Intraframe	DCT Intraframe	MPEG2 Interframe	MPEG2 Interframe	DCT Intraframe	MPEG4 SP Intraframe	MPEG2 (IMX) Intraframe	DCT Intraframe	MPEG2 Interframe	MPEG2 Interframe	DCT Intraframe	DCT Intraframe	DCT Intraframe		
	Compression Ratio	5:1	5:1	18:1	18:1	4:1	2,7:1 4,2:1					5:1	2,5:1	6,7:1		
	Color Sampling	4:2:0	4:1:1	4:2:0	4:2:0	3:1:1	4:2:2 4:4:4	4:2:2	4:1:1 4:2:0	4:2:0	4:2:0	4:1:1	4:2:2	4:2:2		
	Luma Resolution	720x576	720x480	1280x720	1440x1080	1440x1080	up to 1920x1080	720x480 720x576		1440x1080	up to 1920x1080	720x480 720x576	720x480 720x576	960x720	1440x1080	1280x1080
	Chroma Resolution	360x288	180x480	640x360	720x540	480x1080	up to 1920x1080	up to 360x576		720x540	up to 960x540	180x576	360x576	480x720	720x1080	640x1080
	Picture Format	576i	480i	720p	1080i	1080i/p	1080i/p	480i 576i		1080p	1080p	480i 576i	480i 576i	720p	1080i/p	1080i/p
	Framerates	25i	30i	25/30p 50/60p ^(*1)	25/30i 25/30p ^(*1)	24/25/30p 25/30i	24/25/30p 25/30i	variable		variable	variable	30i 25i	30i 25i	25/30/50p	25p 25i	30p 30i
	Video Datarate	25 Mbps CBR	25 Mbps CBR	19 Mbps CBR	25 Mbps CBR	112-140 Mbps CBR	440 Mbps CBR	30/40/50 Mbps CBR	25 Mbps CBR	18 Mbps VBR 25 Mbps CBR 35 Mbps VBR 50 Mbps CBR	18 Mbps VBR 25 Mbps CBR 35 Mbps VBR	25 Mbps CBR	50 Mbps CBR	100 Mbps CBR		
Audio	Sampling	32/48 kHz		48 kHz		48 kHz		48 kHz			32/48 kHz		48 kHz			
	Quantization	12/16 bit		16 bit		20 bit	24 bit	16 bit			12/16 bit		16 bit			
	Compression	none		MPEG1 Layer II		none		none			none		none			
	Audio Datarate	1536 kpbs		384 kbps		3840 kbps	13824kbps	3072 kbps			1536 kpbs		3072 kbps		6144 kbps	
Channels	4/2		2 (or 4 ^(*1))		4	12	4			4/2		4	4	8		
Tape	Size	1/4"		1/4"		1/2"		Blu-ray disc		SxS-Card	1/4"		P2-Card			
	Playing Time	80-270 mins		80 mins		49-155 mins	124-155 min	25 GB (single layer) 50 GB (dual layer)		16 GB as of 2007	80 mins	40 mins	32 GB as of 2007			