FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Plataforma de ingest de apoio à curadoria do cinema Português em ambiente escolar

Alexandre Filipe Freitas Gomes de Almeida

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador: Prof[a]. Maria Teresa Andrade

Orientador externo: Alexandre Ulisses

4 de Julho de 2018

Resumo

A evolução das infraestruturas e das tecnologias que suportam a internet veio facilitar a distribuição de conteúdos cinematográficos por esta via ao longo dos últimos anos.

No entanto, apesar do consumo de vídeo *online* ser responsável por quotas cada vez mais significativas do tráfego de dados gerado, não existe ainda a nível nacional uma plataforma de distribuição de conteúdos orientada para o público em ambiente escolar, que envolva produtores, arquivos de cinema e a comunidade educativa, e que fomente a formação e educação cultural destes últimos e a valorização do cinema português enquanto componente artística e património histórico e cultural. Falta uma cadeia integrada que englobe as fases de restauro, digitalização, *ingest*, armazenamento, curadoria e distribuição de obras cinematográficas num ambiente escolar e educativo.

Esta dissertação, realizada em contexto profissional, visou desenvolver uma plataforma de *ingest* e processamento de conteúdos cinematográficos, preparando-os adequadamente para a sua disseminação em contexto educativo.

Primeiramente, tendo por base a arquitectura da cadeia supra-mencionada, foi feito um estudo dos standards de cinema digital e dos formatos mais adequados para o armazenamento e distribuição dos conteúdos.

Posteriormente foi desenvolvido um protótipo da plataforma de *ingest*, com o auxílio de ferramentas de processamento de áudio e vídeo.

Por fim, prodeceu-se à implementação da transcodificação dos ficheiros de cinema digital no sistema profissional de *ingest* da empresa onde decorreu a dissertação.

Abstract

The improvements of the infrastructures and the technologies that support the internet facilitated the distribution of cinematographic contents through this way during the recent years.

However, while online video consumption accounts for increasingly significant portions of the data traffic generated, it does not exist in Portugal a nationwide content-delivery platform aimed at the school environment that involves producers, film archivists and the educational community, promoting the cultural education of the latter and the valorization of cinema as an artistic component and a form of cultural heritage. There is a necessity for an integrated workflow that encompasses restoration, digitization, ingest, storage, curatorship and distribution phases of cinematographic pieces in a school and educational environment.

This dissertation, carried out in a professional context, aimed to develop a platform for ingesting and processing cinematographic concents, preparing them appropriately for their dissemination in an educational context.

Firstly, based on the architecture of the aforementioned workflow, a study was carried regarding digital cinema standards and the most appropriate formats for the storage and distribution of multimedia contents.

Subsequently a prototype of the ingest platform was developed, with the aid of audio and video processing tools.

Finally, the transcoding operation of digital cinema files was deployed in the professional ingest system of the company where the dissertation was held.

Agradecimentos

Gostaria de agradecer primeiramente aos meus pais, pelo apoio sempre dado ao longo do meu percurso académico, e ao meu irmão e à minha irmã, pelo eterno e verdadeiro sentido de irmandade e companheirismo, e pela boa disposição mesmo nos dias mais difíceis.

Ao Pedro Santos pelo enorme conhecimento, ideias transmitidas, e pela orientação dada para a materialização deste projeto, assim como ao Alexandre Ulisses e à Ivone Amorim, pela oportunidade e apoio dados para a ampliação da experiência em ambientes profissionais.

À Professora Maria Teresa Andrade, pelo auxílio e orientação técnica prestados ao longo da realização da dissertação.

Aos meus companheiros de sempre da universidade, Carlos Meneses, César Silva, Francisco Subtil, João Lima, Miguel Alves, Pedro Couto, e Ricardo Gomes, pelo duro percurso que enfrentámos e terminámos juntos.

Aos meus colegas de estágio na MOG Technologies, pelo sentido de camaradagem e partilha demonstrado, e a todos os colaboradores da MOG Technologies, pela simpatia e prestabilidade ao longo do meu período na empresa.

Alexandre Filipe F. G. Almeida

"In order to understand the world,
one has to turn away from it on occasion."

Albert Camus

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

AAC	Advanced Audio Coding
AAF	Advanced Authoring Format
AJAX	Asynchronous Javascript and XML
ATSC	Advanced Television Systems Committee
AVC	Advanced Video Coding
COD	Common Object Descriptor
CPL	Composotion Playlist
CRF	Constant Rate Factor
CSS	Cascading Style Sheets
DCI	Digital Cinema Initiatives
DCP	Digital Cinema Package
DCT	Discrete Cosine Transform
DWT	Discrete Wavelet Transform
GOP	Group of Pictures
HD	High Definition
HEVC	High Efficiency Video Coding
HTML	Hypertext Markup Language
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
ITU	International Telecommunication Union
JPEG	Joint Photographic Experts Group
JSON	JavaScript Object Notation
MKV	Matroska Video
MPEG	Moving Picture Experts Group
MPEG-TS	MPEG Transport Stream
MXF	Material eXchange Format
PCM	Pulse Code Modulation
PKL	Packing List
PNC	Plano Nacional de Cinema
SD	Standard Definition
SMPTE	Society of Motion Picture and Television Engineers
SSIM	Structural Similarity
UUID	Universally Unique Identifier
VBR	Variable Bitrate
XML	Extensible Markup Language

Capítulo 1

Introdução

O recurso a obras cinematográficas como forma de contextualização dos temas que são discutidos em sala de aula começa a ser visto como uma ferramenta com grande potencial para estimular o interesse do público em idade escolar e reforçar capacidade de aprendizagem deste.

É também importante valorizar o cinema, *per se*, enquanto obra de arte e forma de património histórico e cultural, através da sua disseminação alargada em ambiente educativo.

No entanto, e apesar de surgirem iniciativas louváveis de integração das narrativas cinematográficas em disciplinas específicas e do consumo de vídeos *online* ser cada vez maior, ainda não existe uma forma integrada de processar um espólio relevante de cinema, geri-lo e prepará-lo para uma distribuição eficiente.

O programa a nível nacional de distribuição de conteúdos cinematográficos em ambiente escolar assenta ainda num modelo antiquado e consideravelmente burocrático. As múltiplas entidades envolvidas — escolas, órgãos de tutela do sistema educativo, organismos detentores dos direitos de exploração dos conteúdos audiovisuais e arquivos de cinema — comunicam entre si de forma desintegrada, fazendo com o período de tempo que decorre entre o instante em que uma escola requer um determinado conteúdo cinematográfico para exibição nas suas instalações e o instante de tempo em que o conteúdo lhe é entregue se extenda significativamente.

Adicionalmente, o processo de distribuição da cópia do filme requisitado carece de robustez em termos da salvaguarda dos direitos de visualização da obra. As obras solicitadas pelas instituições de ensino são copiadas pelos arquivos de cinema para um disco e entregues às primeiras. Contudo, não há um controlo do local e do número de sessões de exibição de cada obra, chegando mesmo algumas cópias a serem dadas como desaparecidas.

Os fluxos de trabalho dos arquivos históricos de cinema não possuem ainda sistemas integrados de processamento de vídeo que lhes permitam criar uma verdadeira cadeia que englobe as fases de restauro, digitalização, *ingest*, armazenamento, curadoria, gestão e colocação na *cloud* de conteúdos cinematográficos, agilizando o processo acima descrito.

Esta dissertação, realizada num ambiente profissional, visou o desenvolvimento de uma plataforma de *ingest* e de processamento de conteúdos cinematográficos digitalizados, a ser integrada no fluxo de trabalho descrito no parágrafo precedente.

1.1 Enquadramento

A dissertação foi realizada na empresa MOG Technologies, empresa portuguesa fundada em 2007, que atua no desenvolvimento de soluções tecnológicas para a automatização e inovação dos processos de trabalho da indústria audiovisual a nível mundial.

O trabalho desenvolvido no âmbito desta dissertação está inserido num projeto a nível europeu, que para além da MOG Technologies envolve a Cinemateca Portuguesa (instituição tutelada pelo Ministério da Cultura que desempenha uma missão de salvaguarda e divulgação do património cinematográfico português), a empresa YouOn (empresa que opera nas áreas das tecnologias *online* e dos novos media), o INESC TEC (instituição centrada em atividades de investigação científica e desenvolvimento tecnológico) e o ISEP (Instituto Superior de Engenharia do Porto).

1.2 Objetivos

A plataforma desenvolvida deve, com o intuito de contribuir para a solução da problemática descrita no texto que inicia o presente capítulo, ser capaz de tomar na sua entrada os suportes de conteúdos cinematográficos hospedados pela Cinemateca Portuguesa, estruturados e codificados de acordo com standards internacionais de cinema digital, interpretá-los e adaptá-los para formatos apropriados para o seu armazenamento e distribuição a nível escolar através da internet, a ser feita pela empresa YouOn.

Foram definidos os seguintes objetivos para a presente dissertação:

- Realizar um levantamento e análise detalhada das principais soluções e arquiteturas de codificação de vídeo;
- Implementar um sistema capaz de receber conteúdos cinematográficos digitalizados e os disponibilizar em formatos otimizados para o seu armazenamento e exibição em contexto educativo;
- Realizar a integração das aplicações desenvolvidas no sistema de *ingest* profissional da empresa;
- Realizar testes e afinações ao trabalho desenvolvido otimizando-o em termos de rapidez de processamento e disponibilidade de acesso.

1.3 Estrutura da dissertação

Para além do presente capítulo introdutório, esta dissertação contém mais oito capítulos.

Nos capítulos 2 e 3 são expostos os fundamentos teóricos acerca dos formatos de vídeo e das estruturas de cinema digital que a plataforma desenvolvida no âmbito da presente dissertação terá de processar.

No capítulo 4 é apresentada a solução proposta e a respetiva arquitetura para o problema exposto neste capítulo. São igualmente definidos os requisitos funcionais para a plataforma desenvolvida e as tecnologias empregues para o seu desenvolvimento.

No capítulo 5 é feita uma análise às amostras de pacotes de cinema digital a serem processados pela plataforma, fornecidas pela Cinemateca Portuguesa, uma das instituições envolvidas no projeto que abarca a presente dissertação.

Nos capítulos 6 e 7 é detalhado o desenvolvimento de um protótipo da plataforma de *ingest*, e a posterior integração no sistema de transcodificação da empresa que acolheu a realização da dissertação.

No capítulo 8 é feita uma análise dos resultados obtidos e do cumprimento dos requisitos delineados para a plataforma.

Por fim, no capítulo 9, são apresentadas as conclusões do trabalho realizado, e é lançada uma perspetiva sobre o trabalho complementar a ser realizado futuramente no contexto do projeto abordado.

No anexo A é exibido o código do programa desenvolvido de análise de Digital Cinema Packages.

No anexo B constam o código HTML da página web e o código Python do back-end Flask desenvolvidos.

Capítulo 2

Formatos de vídeo

Neste capítulo é feita uma explicação introdutória dos termos associados a um ficheiro de vídeo, seguida de uma análise aos formatos de vídeo mais utilizados para arquivação digital de longo prazo de obras cinematográficas, e dos formatos mais indicados para a distribuição de conteúdos audiovisuais através da internet.

2.1 Codecs e contentores

Um ficheiro de vídeo é normalmente composto por várias faixas de diferentes tipos de dados: uma ou mais faixas de vídeo, uma ou mais faixas de áudio, e possivelmente uma ou mais faixas de legendas.

A informação presente em cada faixa, ou *stream*, que compõe o ficheiro é codificada por um codec. Um codec é um pacote de software ou hardware que codifica uma componente multimédia de uma determinada forma.

Esta codificação, ou *encoding*, pode ser feita havendo ou não compressão da informação inicial, sendo que havendo compressão, esta pode ser feita com perdas ou sem perdas. É usual os conteúdos serem comprimidos, de forma a serem obtidos ficheiros mais leves e por isso mais económicos de armazenar ou distribuir.

Antes de poder reproduzir um determinado conteúdo, um dispositivo tem por isso de executar a descodificação da *bitstream* recebida.

Existem as seguintes formas de compressão:

- **compressão matematicamente sem perdas** (*mathematically lossless compression*): não há de facto nenhuma diferença entre o conjunto de bits inicial e o conjunto de bits que é obtido após a descodificação;
- **compressão visualmente sem perdas** (*visually lossless compression*): há algumas perdas, mas estas não são notórias ao sistema visual humano;
- **compressão com perdas** (*lossy compression*): há uma perda efetiva de informação entre o conjunto de bits original e o conjunto de bits obtido após descodificação.

As várias streams obtidas são encapsuladas num contentor (ou *container* ou *wrapper*). Por principio não devem ser feitas alterações ao conjunto de bits que compõe cada stream, de modo a facilitar a conversão entre formatos (*transcoding*) [9].

Há frequentemente uma ambiguidade entre a utilização dos termos codec e *container*. O *container* não diz respeito à forma como os bits foram codificados, mas sim à forma como as *streams* são agrupadas e interpretadas pelo computador. Em literatura sobre esta temática é também utilizado o termo "formato", que pode referir-se tanto ao codec como ao *container* de um conteúdo multimédia.

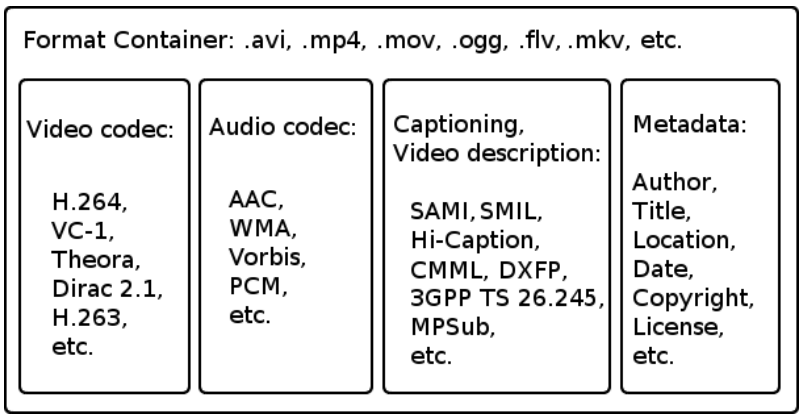

Figura 2.1: Estrutura habitual de um ficheiro de vídeo, com exemplos de *containers* e codecs de vídeo, áudio e legendas. [1]

O foco ao longo deste documento recairá na generalidade sobre os formatos de vídeo, em detrimento dos formatos de áudio. A componente de vídeo requer uma quantidade de bits massivamente maior que a de áudio para ser armazenada e transportada, sendo por isso alvo de maior atenção.

2.2 Formatos de arquivo de longo prazo de obras cinematográficas

Numa primeira fase do fluxo de trabalho (descrito de forma pormenorizada no capítulo 4) em que será integrada a plataforma a ser desenvolvida pela presente dissertação os conteúdos cinematográficos serão digitalizados e armazenados num ou mais ficheiros.

Nos suportes analógicos em que existe atualmente, a informação está a detriorar-se à medida que o tempo que passa, e corre o risco de se tornar obsoleta, pelo que a operação de digitalização se reveste de extrema importância. É necessário converter adequadamente os conteúdos, com a melhor qualidade possível, de forma a preservar a longo prazo o valor intrínseco das obras, armazenando-as de forma acessível e funcional, mesmo para possíveis migrações para novos suportes no futuro. [10]

Não há no entanto uma escolha óbvia no que toca ao formato a utilizar. Ainda não há um formato perfeito para efeitos de arquivo, e no final trata-se se pesar os prós e contras de cada codec e *container* para chegar à escolha de um formato de arquivo.

2.2.1 Critérios de escolha

Apesar da escolha de um formato para arquivo digital de longo prazo de obras cinematográficas não fazer parte do âmbito da presente dissertação (esta decisão foi tomada pela Cinemateca Portuguesa), é feita nesta secção uma breve exposição dos critérios normalmente tidos em consideração.

É fundamental ter presente que a Cinemateca Portuguesa é uma instituição com fins equiparáveis aos de um museu. Cabe-lhe conservar cuidadosamente as obras, pelo que é mais importante a qualidade e a autenticidade dos arquivos do que a rapidez ou a facilidade em editar, cortar peças e fazer transmissões para todo o mundo de forma célere, como acontece nas estações de televisão.

Estes são os critérios para a escolha de um formato de vídeo (codec e *container*) usualmente seguidos pela comunidade audiovisual [11] [12]:

- **Qualidade:** *the quality of the file should be high enough to anticipate future use and to avoid risk of quality loss over time*;
- **Abertura:** *there should be no restriction on the use and reuse of the file such as licences that can constitute a threat to the adoption and support of the format*;
- **Aceitação:** *the format should be widely used by archives or different domains and have sufficient support in existing tools*;
- **Transparência:** *the file format should be easy to analyse*;
- **Durabilidade:** *the format shouldn't be expected to become obsolete or need transcoding too quickly. Backward compatibility in the short term should be ensured*;
- **Funcionalidades:** *the file should be able to deal with complex objects*;
- **Facilidade de utilização:** *the file should be handled easily, efficiently and without (or very limited) risk of error and threat to workflows*.

Algumas escolhas comuns entre projetos e instituições de conservação e divulgação do património cultural audiovisual são:

- codec JPEG 2000 e *container* MXF [10]
- codec Apple ProRes e *container* MOV [13]
- codec FFV1 e *container* MKV [9]

A Cinemateca Portuguesa optou por utilizar o codec Apple ProRes com o *container* MOV para filmes convertidos de suportes analógicos. Do arquivo da Cinemateca Portuguesa constam também filmes já num suporte digital de cinema, o Digital Cinema Package (estrutura descrita no capítulo 3), que utiliza o codec de vídeo JPEG2000 e o *container* MXF.

Segue-se uma breve análise destes quatro codecs e *containers*.

2.2.2 MOV/QuickTime

O MOV ou QuickTime é um *container* de elementos multimédia, desenvolvido pela Apple. Suporta múltiplas faixas de conteúdos, como vídeo, áudio, texto, imagens estáticas ou *timecode*.

Cada faixa pode conter o conteúdo multimédia propriamente dito ou então uma referência para uma *stream* localizada noutro ficheiro. Adicionalmente, o ficheiro mantém informação sobre a reprodução de cada faixa que contém (*time offsets* e *track edit lists*), o que o torna adequado para a edição dos conteúdos e a transcodificação entre formatos.

O QuickTime foi utilizado como base para a standardização do formato MPEG-4, e é largamente aceite e utilizado tanto em ambientes profissionais de pós-produção de vídeo como ao nível do consumidor final. [13]

2.2.3 ProRes

Codec desenvolvido pela Apple, bastante utilizado nos ambientes profissionais de pós-produção de vídeo [14]. É um formato visualmente sem perdas (se codificado com um determinado perfil), possibilitando uma elevada qualidade de imagem com *bitrates* significativamente menores que os de vídeo não-comprimido. Dado ser um *Intra-frame only codec* (cada *frame* é codificada e descodificada de forma independente em relação a outras frames do *vídeo*), a sua descodificação é bastante mais rápida do que a de codecs como o H.264/AVC, proporcionando por isso uma boa performance e flexibilidade de edição em tempo real. Possui múltiplos perfis de *encoding* possíveis, com diferentes níveis de qualidade.

Principais características:

- VBR (Variable Bitrate) encoding;
- Suporte de resolução de imagem até 8k;
- *Intra-frame only codec*;
- 4:4:4 ou 4:2:2 *chroma sub-sampling*;
- 12 ou 10 bits de profundidade por píxel;

2.2.4 MXF

O MXF (Material eXchange Format) [15] é um *container* de vídeo e áudio, criado por membros da indústria audiovisual com o objetivo de solucionar os problemas de inter-operabilidade que surgiram no meio profissional de produção e distribuição de vídeo aquando da transição dos sistemas de armazenamento *tape-based* para um sistema *file-based*. Os variados formatos de compressão de vídeo que passaram a ser utilizados, com diversos perfis de compressão para cada formato, aliados à necessidade da parte dos membros da indústria em incluir *metadata* na transação dos conteúdos audiovisuais foram as principais motivações para o desenvolvimento deste formato.

O MXF funciona de forma independente em relação a qualquer formato de vídeo ou áudio que possa conter, denominando-se por isso de um *essence-agnostic container*.

As características e funcionalidades do MXF assentam nos seguintes requisitos-chave:

- O formato deve transportar *metadata* descritiva sobre os conteúdos, assim como as componentes de vídeo e áudio;
- Deve ser possível ver ou editar os conteúdos à medida que a transferência do ficheiro decorre (particularmente importante durante a transação de ficheiros de tamanho elevado sobre redes de velocidade baixa);
- O formato deve providenciar mecanismos para a extração de informação do ficheiro mesmo que determinadas partes do ficheiro estejam em falta (por exemplo num cenário de transferência de dados via satélite, em que uma receção começa a meio do decorrer processo de envio do ficheiro);
- O formato deve ser aberto, standardizado e independente dos formatos de compressão de vídeo e áudio;
- O formato deve visar a transação de conteúdos ou segmentos de conteúdos audiovisuais profissionais;
- O formato deve ser simples o suficiente para possibilitar a sua utilização em sistemas em tempo real.

O MXF desenvolveu-se a partir do AAF (Advanced Authoring Format), sendo o AAF direcionado para a fase de produção e pós-produção dos conteúdos, e o MXF orientado para transação de conteúdos ou segmentos de conteúdos finalizados.

2.2.5 JPEG 2000

O JPEG 2000 é um formato de compressão de imagem, criado pelo Joint Photographic Experts Group no ano 2000 [15]. É baseado no formato largamente utilizado JPEG. A principal diferença entre estes dois formatos é que o JPEG 2000 utiliza a *discrete wavelet transform* (DWT), em detrimento da *discrete cosine transform* (DCT) utilizada pelo JPEG. É à semelhança do Apple ProRes

um *intra-frame only codec*, quando utilizado para a codificação de vídeo, e bastante utilizado em ambientes profissionais que requerem uma elevada qualidade (indústria do *broadcasting*, cinema digital e arquivos de vídeo por exemplo). A compressão da imagem pode ser feita com ou sem perdas. Suporta qualquer resolução, qualquer *bit depth* e qualquer espaço de cor.

2.3 Formatos de vídeo para distribuição de conteúdos cinematográficos via internet

A escolha de um formato de vídeo para distribuição de conteúdos *on-demand* (esta sendo uma escolha já feita no âmbito desta dissertação) através da internet, seja via *streaming*, seja via *download* e posterior visualização *offline* dos conteúdos, teve de ter em conta uma série de fatores. Desde logo, as limitações inerentes à conexão de internet de cada utilizador, que afetam a cadência com que este poderá receber o conteúdo, e, num segundo patamar, a capacidade do seu dispositivo conseguir descodificar e reproduzir adequadamente o vídeo e áudio rececionados.

Neste cenário não seria exequível por exemplo transmitir um conteúdo tal existe num Digital Cinema Package (estrutura descrita no capítulo 3), com vídeo com 125 Mbps de *bitrate* (um *bitrate* de vídeo comum em formatos de cinema digital).

É por isso necessário re-codificar e comprimir os conteúdos, tornando-os aptos para transmissão através da internet, mantendo no entanto um determinado nível de qualidade de imagem que não interfira com a experiência de visualização de uma obra cinematográfica.

Neste contexto, contrariamente à situação anteriormente abordada, da escolha de um formato de video para armazenamento a longo prazo, não tem tanta preponderância a necessidade em escolher um formato com um elevado prazo de vida. Aliás, é provável que em pouco tempo surjam formatos com melhores eficiências de compressão (usam menos *bitrate* para atingir a mesma qualidade de imagem), e que serão adoptados se assim se justificar. O importante é que os formatos de arquivo sejam passíveis de ser recodificados para esses novos formatos de acesso.

Da perspetiva do distribuidor dos conteúdos, para além dos fatores supra-mencionados, é ainda importante tomar em consideração dois fatores mais: a possível taxa a pagar pela utilização de um determinado formato (*royalties*), e a complexidade/velocidade de *encoding* aquando da adaptação dos conteúdos para preparar a sua distribuição.

Eis então os critérios que foram considerados importantes para a escolha de um formato:

- Eficiência de compressão;
- Compatibilidade com *browsers* dos dispositivos dos utilizadores finais;
- Compatibilidade com *hardware* e *software* dos dispositivos dos utilizadores finais;
- Complexidade de *encoding*;
- Taxas de direitos de utilização (*royalties*).

2.3.1 H.264/AVC, H.265/HEVC, VP9 e AV1

Os 3 codecs de vídeo com perdas (lossy) mais eficientes e mais utilizados atualmente são o H.264 (ou AVC), o H.265 (ou HEVC) e o VP9 [16].

O H.264/AVC [17] é o formato dominante de há alguns anos a esta parte, tanto em distribuições de conteúdo pela internet como em transmissões de televisão e é suportado pela grande maioria dos *browsers* de internet, sistemas de televisão e dispositivos móveis atuais. É utilizado quase pela totalidade dos sistemas de televisão digital terrestre dos diversos países (debaixo da norma DVB-T) e por plataformas como o Youtube, Netflix, ou Vimeo. O *encoder* x264 é o *software open-source* mais utilizado para codificar video em H.264. O standard do H.264/AVC foi publicado em 2003 pelo ITU-T Video Coding Experts Group e pelo ISO/IEC JTC1 Moving Picture Experts Group (MPEG). Conteúdos codificados neste formato são comummente encapsulados num *container* MP4, juntamente com áudio no formato AAC.

O H.265/HEVC [18] é o sucessor do H.264/AVC, apresentado pelo mesmo grupo de trabalho em 2013, numa tentativa de não só conseguir transmitir conteúdos em HD e SD com menos *bitrate*, mas também de agilizar a transmissão de conteúdos em Ultra HD (2K e 4K). O *encoding* pode ser feito com o *software* open-source x265. É habitualmente utilizado num container MPEG-TS ou MKV.

O VP9 [19] é um formato *open-source* e *royalty-free* desenvolvido pela Google, idealizado para competir com os dois formatos supra-citados. A biblioteca de *software open-source* desenvolvida para o encoding chama-se 'libvpx'. É normalmente utilizado com áudio no formato Opus ou Vorbis e com o *container* WebM.

É importante ter em conta que para cada *codec/standard* podem haver diferentes *encoders*, cada um com diferentes performances em termos de tempo de codificação, *bitrate* atingido, entre outros.

Apesar das melhorias que o H.265/HEVC trouxe em relação ao H.264/AVC, a sua adopção e aceitação tem no entanto vindo a decorrer de forma mais lenta que o esperado. Nos serviços de distribuição de TV em Ultra HD via satélite ou cabo o H.265/HEVC é o codec normalmente utilizado, por exemplo em Portugal pelo canal RTP1 4K ou pelo canal SPORT TV 4K UHD, ou no Reino Unido pelo canal BT Sport 4K UHD. E em plataformas *over-the-top* (OTT) a emissora britânica BBC, através do seu *web player*, está a realizar, à data da escrita desta dissertação, transmissões dos jogos do Campeonato do Mundo de Futebol da FIFA 2018 em Ultra HD utilizando também o HEVC [20]. No entanto, as plataformas com mais peso no tráfego de vídeo online, como a Netflix e o YouTube, têm reportado dificuldades em adoptar o H.265/HEVC, invocando as elevadas royalties a pagar pela utilização do codec e elevada complexidade do processo de licenciamento [21], e optaram por seguir o caminho dos codecs *royalty-free* e *open-source*. Por esta altura Netflix e YouTube já distribuem uma grande parte dos seus conteúdos com o codec VP9.

Vale a pena fazer uma menção também ao codec AV1 [22], desenvolvido pela Alliance for Open Media, que engloba empresas como a Adobe, Amazon, Apple, BBC R&D, Facebook, Google, IBM, Intel, Microsoft, Mozilla Foundation, Netflix, Nvidia ou VLC, entre outros. Apesar

de ainda estar na fase final do seu desenvolvimento (já existe um documento da AOM que detalha a estrutura do ficheiro e o processo de decoding, mas ainda se encontra em atualização), o peso das organizações envolvidas no seu desenvolvimento faz antever que este codec poderá dominar o mercado no que à transmissão de vídeo pela internet diz respeito.

Os testes de eficiência de compressão de codecs de video dependem de vários fatores. Há várias métricas de avaliação de qualidade que podem ser calculadas, e cada uma delas terá um resultado diferente em função de definições do *encoding*, *bitrate* do vídeo, estrutura do GOP, resolução do vídeo, entre outros fatores.

Mas de uma forma genérica, num estudo de 2016, De Cock et al, concluíram que o encoder x265 (do codec H.265/HEVC) usa entre 28% e 52% menos *bitrate* que o encoder x264 (do codec H.264/AVC) para obter a mesma qualidade de imagem, enquanto o encoder libvpx (do codec VP9) utiliza entre 21% e 52% menos bitrate [16]. O mesmo estudo conclui também que o x265 tem uma performance superior ao libvpx na maior parte dos casos testados, mas decresce à medida que a resolução do video aumenta, perdendo mesmo para o libvpx em video com resolução 1080p.

Num estudo de Janeiro de 2018, o Graphics & Media Lab Video Group da Universidade Estatal de Moscovo, uma das organizações mais reputadas e referenciadas em termos de análise de eficiência de codecs de vídeo, analisou uma série de *encoders* dos quatro formatos que abordámos nesta secção. Para o codec H.264 foram testados os *encoders* nj264 e x264; para o codec H.265/HEVC fora utilizados os *encoders* Kingsoft HEVC, nj265 e x265; para o codec VP9 foi utilizado o VP9 Video Codec. O *encoder* uAVS2 do codec AVS2 também integra o relatório, apesar de não ser um codec abordado no âmbito desta dissertação.

Os resultados que se seguem são as médias da percentagem de *bitrate* que cada *encoder* usa em relação ao *encoder* de referência x264, para atingir a mesma qualidade de imagem deste (medida através da métrica de qualidade SSIM [23]). Foram analisadas 512.000 sequências de vídeo no total, codificadas a partir de 31 clips de vídeo distintos com a resolução 1920 x 1080.

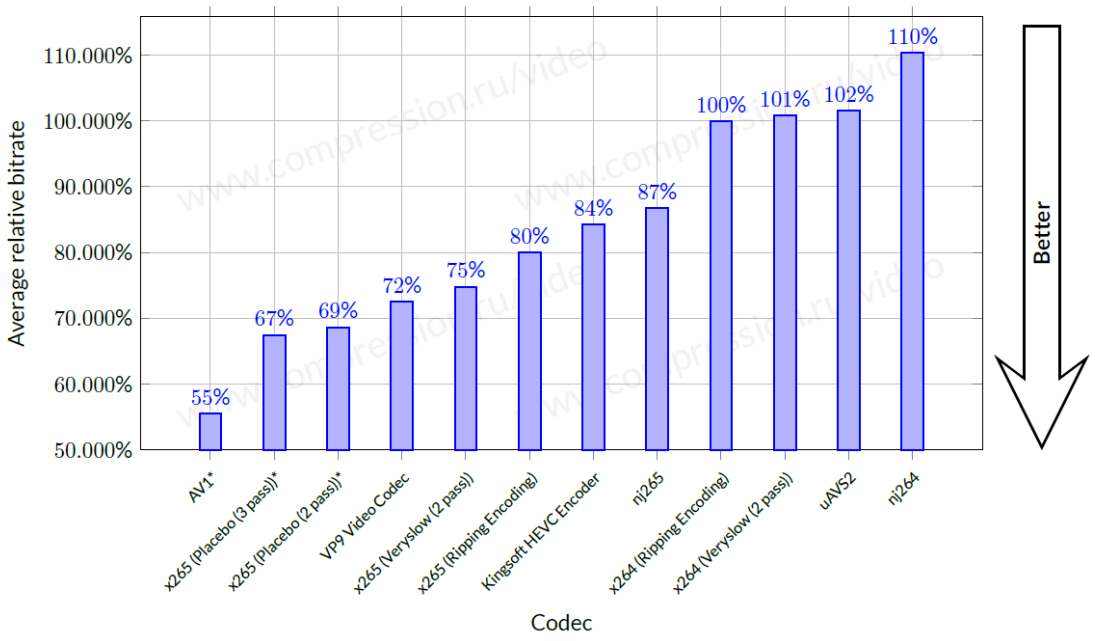

Figura 2.2: Média da percentagem de *bitrate* utilizado por cada *encoder* em relação ao *encoder* de referência x264, para atingir a mesma qualidade de imagem, a partir de diferentes clips de vídeo [2]

Na figura 2.3 são exibidos as velocidades de *encoding* para os *encoders* testados.

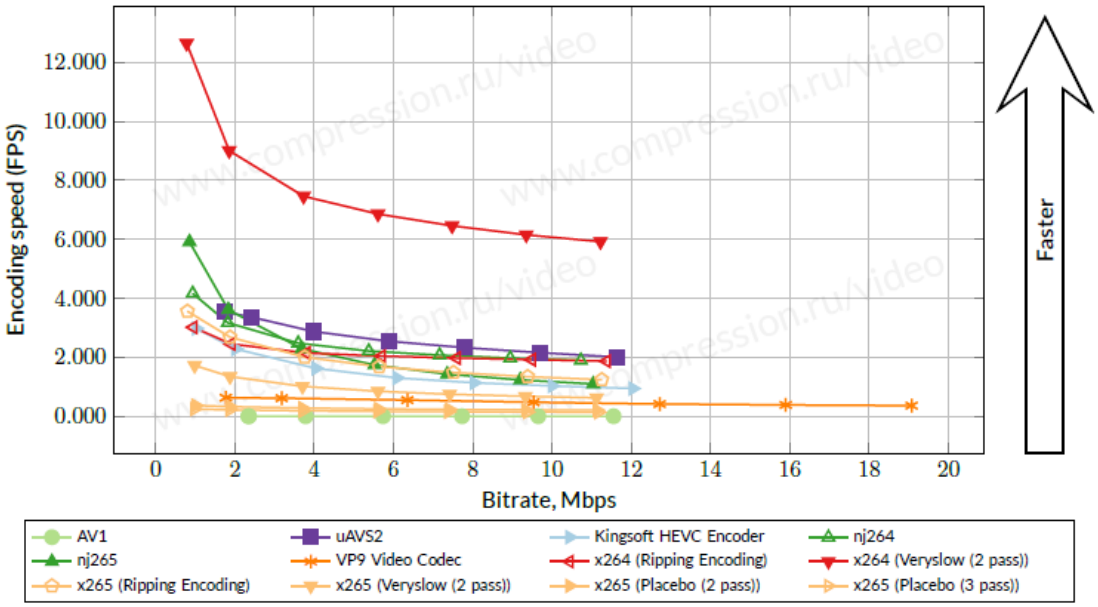

Figura 2.3: Velocidade de *encoding* dos diferentes *encoders*, para um dos clips de vídeo utilizados no estudo [2]

É possível observar que o codec AV1 apresenta resultados substancialmente melhores que os codecs H.265/HEVC e VP9. Estes últimos apresentam resultados na ordem dos já concluídos resultados do estudo de De Cock et al, mas que dependerão bastante dos parâmetros de encoding utilizados.

Em termos de velocidade de codificação, o *enconder* x264 está demarcadamente acima dos demais *encoders* testados, ao passo que o AV1 se encontra no extremo inverso do eixo, com uma velocidade de *encoding* extremamente lenta, fruto da baixa otimização de performance, que deverá melhorar se considerarmos que o codec ainda se encontra em fase de desenvolvimento.

No que diz respeito à compatibilidade com os *browsers* mais utilizados em computadores, exibidas na tabela 2.1, o H.264/AVC tem suporte completo em todos eles, enquanto que o H.265/HEVC e o VP9 estão ainda limitados, de uma ou outra forma.

Tabela 2.1: Compatibilidade dos browsers (versões estáveis, à data da publicação deste documento) para os três codecs abordados (dados de caniuse.com)

	IE	**Edge**	**Firefox**	**Chrome**	**Safari**
H.264/AVC	Sim	Sim	Sim	Sim	Sim
H.265/HEVC	Parcialmente[1]	Parcialmente[1]	Não	Não	Parcialmente[2]
VP9	Parcialmente[3]	Parcialmente[4]	Sim	Sim	Parcialmente[3]

[1] Apenas em dispositivos com suporte de hardware.

[2] Apenas em dispositivos com o macOS High Sierra ou posterior.

[3] Apenas se o codec estiver instalado no computador.

[4] Apenas a partir de Media Source Extensions.

No que diz respeito de *hardware* e *software* para uma eventual reprodução *offline* dos conteúdos, com os devidos codecs instalados, o H.264/AVC levantará em princípio poucos ou nenhuns problemas. O H.265/HEVC e o VP9 estão no entanto dependentes das versões de *players* de *media* mais recentes, e dependendo de determinadas características do vídeo — espaço de cor, profundidade de bits ou *chroma sub-sampling* por exemplo — poderão apenas ser reproduzidos com *hardware*, nomeadamente CPU e GPU, de elevada performance.

Expostas as particularidades de cada codec, e tendo em conta os critérios definidos, assim como tomando em forte consideração as características do projeto sobre o qual esta dissertação incide, optou-se por utilizar, pelo menos nesta fase, o codec H.264/AVC, juntamente com o codec de áudio AAC e o *container* MP4.

A forma de distribuição de conteúdos — *online* via *streaming* ou *offline* via *download* — ainda é neste momento uma decisão em aberto da parte das instâncias na chefia do projeto. O H.264 é um codec flexível para ambos os modos de distribuição, pelo que sua escolha é justificada neste aspeto.

No que diz respeito ao *software* e *hardware* dos computadores existentes em cada escola, encontraremos um cenário enormemente diverso, pelo que optando H.265/HEVC ou pelo VP9 estaríamos possivelmente a pôr em causa a reprodução das obras cinematográficas num número

significativo de escolas. Com o H.264 abrangeremos garantidamente a maior percentagem de público possível, assegurando uma boa qualidade de imagem.

Outros codecs de compressão com perdas, alguns deles versões predecessoras dos codecs que analisamos, existem e ainda são utilizados com alguma quota de relevância em cenários *offline* pelos utilizadores, mas antes de serem transmitidos pelas plataformas de *streaming* são normalmente recodificados num dos três codecs supra-mencionados. Na figura 2.4 podemos ver os codecs dos videos que os utilizadores carregaram para o YouTube entre 2011 e 2015.

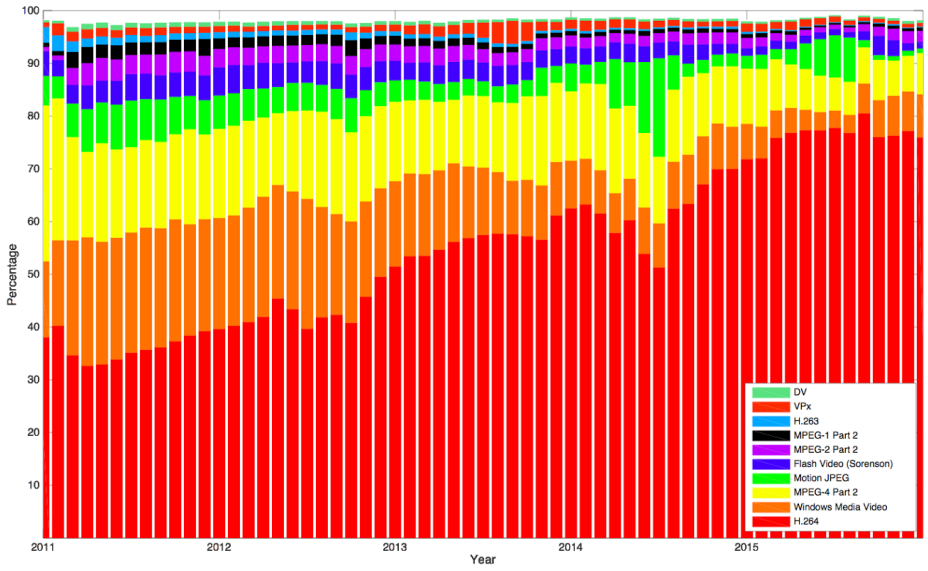

Figura 2.4: Distribuição dos codecs dos videos carregados para o YouTube entre 2011 e 2015 [3]

2.3.2 H.264/AVC, AAC e MP4: análise adicional

Nesta sub-secção é feita uma descrição complementar do codec H.264/AVC, que juntamente com o formato de áudio AAC num *container* MP4 formará o formato de saída dos ficheiros da plataforma desenvolvida no âmbito desta dissertação.

O H.264 tira partido da redundância espacial e redundância temporal da informação presente no vídeo para gerar ficheiros com menos quantidade de informação, possibilitando uma poupança de largura de banda (em cenários de transmissão dos conteúdos) ou de espaço para armazenamento dos ficheiros. [24]

A redundância espacial diz respeito a informação que se repete em cada *frame* do conteúdo (como um bloco de pixels da imagem ser igual ou parecido a um outro bloco da mesma imagem, por exemplo).

A redundância temporal refere-se ao facto de frames consecutivas serem frequentemente visualmente semelhantes, havendo apenas algumas diferenças em certas áreas da imagem, fruto do movimento associado.

Desta forma, ao contrário dos *Intra-frame only codecs*, que codificam sempre uma frame por completo, o H.264 codifica uma frame completa após um determinado número de frames codificadas, sendos as restantes *frames* (*frames* do tipo P (*Predicted* e tipo B (*Bidirectional*)) codificadas como a diferença em relação a uma *frame* completa temporalmente próxima destas.

Por este motivo, vídeo codificado em H.264 obriga a uma maior complexidade no processo de descodificação do conteúdo, já que para a obtenção de uma determinada *frame* pode ser preciso recorrer à informação de *frames* anteriores ou posteriores no tempo.

Este codec possui diferentes perfis de *encoding*, orientados para diferentes finalidades:

- **Baseline Profile:** orientado para serviços conversação em tempo real (vídeo-chamada, vídeo-conferência) e dispositivos com baixa capacidade computacional;
- **Main Profile:** originalmente criado para os serviços televisão digital em Standard Definition (SD), perdeu preponderância com o surgimento dos High Profiles;
- **Extended Profile:** orientado para serviços de *streaming* através da internet, oferece maior capacidade de compressão e robustez à perda de informação na transmissão;
- **High Profiles:** orientado para produção e distribuição de conteúdos com qualidade elevada de imagem, contém sub-perfis distintos apresentando diferentes configurações de *chroma sub-sampling*, profundidade de bits por píxel ou estruturação de *frames* do vídeo, entre outras características.

O Advanced Audio Coding (AAC) é um dos formatos *lossy* de compressão de áudio, standardizado pela ISO/IEC, pela primeira vez em 1997. É um dos formatos de áudio mais utilizados com o formato de vídeo H.264, e será o codec utilizado nos ficheiros de saída da plataforma desenvolvida. Este codec possui uma eficiência de codificação e uma performance superior do que os codecs concorrentes MP3 e AC-3, tanto em conteúdo de elevado *bitrate* como de baixo *bitrate* [25]. Suporta até 48 canais de áudio, com uma taxa de amostragem até 96 kHz.

O MP4 (ou MPEG-4 Part 14) é um *container* de *streams* multimédia, que foi desenvolvido com base no *container* QuickTime. Para além de faixas de vídeo e áudio, possui também suporte para faixas de legendas, imagens estáticas e metadata. Foi standardizado pela ISO/IEC, pela primeira vez em 2003.

2.4 Conclusão

Neste capítulo é feita uma explicação introdutória dos termos associados a um ficheiro de vídeo, seguida de uma análise aos formatos de vídeo mais utilizados para arquivação digital de longo prazo de obras cinematográficas, e dos formatos mais indicados para a distribuição de conteúdos audiovisuais através da internet. Foram descritos os codecs e *containers* que a plataforma desenvolvida irá processar — JPEG 2000 e MXF, ProRes e MOV/QuickTime. Por fim, depois de uma

exposição dos critérios para a escolha de um formato de distribuição de vídeo através da internet, optou-se pelo formato a ser utilizado pela plataforma desenvolvido, o H.264/AVC.

Capítulo 3

Digital Cinema Package

Neste capítulo é feita uma análise pormenorizada às estruturas de cinema digital que a Cinemateca Portuguesa utiliza para armazenar uma quota importante das obras cinematográficas que possui, e que serão processadas pela plataforma desenvolvida por esta dissertação.

Um Digital Cinema Package é um pacote de cinema digital de ficheiros de vídeo, áudio, legendas e *metadata* (informação sobre os conteúdos multimédia). É o suporte digital enviado para as salas de cinema para posterior exibição, via internet, via satélite ou através de um suporte físico de armazenamento. [4]

Começou a ser desenvolvido em 2002 pela Digital Cinema Initiatives, LLC (DCI), um consórcio de 7 estúdios de cinema de Hollywood — Disney, Fox, Metro-Goldwin-Mayer[1], Paramount Pictures, Sony Pictures Entertainment, Universal Studios e Warner Bros. Studios — com vista a uniformizar processos e especificações, garantindo um nível elevado de qualidade e segurança na distribuição e exibição de cinema em formato digital, tendo em mira uma eventual substituição definitiva das películas de 35 mm (o tipo de películas mais comuns na indústria cinematográfica).

No processo de criação de um DCP os ficheiros de vídeo, áudio e legendas nos seus vários formatos são encapsulados em ficheiros MXF e opcionalmente encriptados.

Adicionalmente, são criados vários ficheiros auxiliares de *metadata*, no formato XML, que fornecerão a informação necessária para a localização, verificação de integridade, desencriptação e reprodução dos múltiplos ficheiros audiovisuais que podem existir num DCP.

3.1 Interop vs SMPTE

A DCI completou o documento que contém as linhas-mestras da estrutura de um DCP em Agosto de 2005. No entanto, os standards detalhados sobre a estrutura de cada tipo de ficheiro presente num DCP, da responsabilidade da Society of Motion Picture and Television Engineers (SMPTE), foram apenas completados em 2009.

[1] A Metro-Goldwyn-Mayer abandonou a DCI em Maio de 2005, antes do finalização do documento que detalha as especificações técnicas associadas a um DCP.

Não obstante, em 2006 a indústria cinematográfica agregou o que já existia dos standards da SMPTE e começou imediatamente a incorporar DCP's nos seus *workflows*, chamando-lhes DCP's do tipo Interop.

Os standards de cinema digital mais tarde completados pela SMPTE formaram os chamados DCP's do tipo SMPTE.

Os dois tipos de DCP têm algumas diferenças entre si, e ambos continuam a ser utilizados atualmente, pelo que estas diferenças, quando existirem e forem relevantes, serão detalhadas nas secções subsequentes.

3.2 Vídeo

O vídeo, originalmente composto por múltiplos ficheiros TIFF (um ficheiro para cada frame), com 16 bits por cada componente de cor X'Y'Z'[2], é primeiramente comprimido para um ficheiro no formato JPEG2000, antes do encapsulamento num *container* MXF. Este processo de compressão tira partido da forma como o sistema visual humano perceciona a luz e a cor para obter uma imagem sem perdas visuais. Há assim uma redução do espaço ocupado sem haver um comprometimento da qualidade da imagem.

Características técnicas:

- **Codec:** JPEG2000;
- **Espaço de cor:** X'Y'Z';
- **Profundidade de bits:** 12 bits para cada componente do espaço de cor;
- **Resolução:** 2K (máx. 2048 x 1080) ou 4K (máx. 4096 x 2160);
- ***Aspect Ratios* comuns:** 1.85:1 (1998 x 1080 para 2K, 2048 x 858 para 4K) e 2.39:1 (2048 x 858 para 2K, 4096 x 1716 para 4K);
- ***Frame rates*[3]:** 24, 25, 30, 48, 50, 60, 96, 100 ou 120 fps para a resolução 2K; 24, 25 ou 30 fps para a resolução 4K.

3.3 Áudio

O áudio é encapsulado em MXF sem serem feitas alterções à sua essência. O formato utilizado é o formato sem compressão Pulse Code Modulation (PCM). [26]

Características técnicas:

- **Formato:** PCM;

[2] Uma variante do espaço de cor XYZ, especificada em no standard SMPTE 428-1-2006.

[3] No standard inicial das especificações da imagem SMPTE 428-1-2006 foram apenas previstos os *frame rates* de 24 e 48 fps para a resolução 4K, e 24 fps para a resolução 4K. Os restantes *frame rates* foram adicionados no standard SMPTE ST 428-11:2013.

- **Profundidade de bits:** 24 bits;
- **Taxa de amostragem:** 48 ou 96 kHz;
- **Nº de canais:** Máximo de 16 canais.

3.4 Legendas

Os ficheiros de legendas num DCP podem existir num documento textual ou num ficheiro de imagem PNG. O sistema de projeção encarrega-se de posteriormente fazer o *overlay* das legendas na imagem.

No que a este *asset* de *media* diz respeito há algumas diferenças consideráveis entre os dois tipos de DCP.

Num DCP do tipo Interop as legendas *text-based* (Timed Text) são um ficheiro XML, formatado segundo o standard CineCanvas [27], e são mantidas numa pasta à parte dos demais ficheiros do DCP, juntamente com o tipo de letra a desenhar.

Num DCP do tipo SMPTE as legendas *text-based* são igualmente um documento XML, mas encapsulado num ficheiro MXF, tal como acontece com as essências de áudio e vídeo. Além disso, a estrutura do documento XML segue um standard distinto, o SMPTE ST 428-7 [28]. Adicionalmente, ao contrário de um DCP Interop, num DCP SMPTE os ficheiros de legendas estão no mesmo diretório dos ficheiros de áudio e vídeo.

Também é possível incluir legendas no formato de imagem (PNG), que são posteriormente sobrepostas no vídeo. Esta forma de legendas tem o nome de Subpicture Subtitles.

Ambos os documentos contêm instruções sobre a posição e o instante de tempo em que determinado texto deve surgir na imagem. Contudo, apesar da estrutura dos documentos ser semelhante, existem algumas diferenças que implicarão para o programa que vamos desenvolver formas distintas de processar a informação contida nos ficheiros. Estas diferenças na estrutura interna do documento XML são afloradas com mais detalhe no capítulo 7, em que é documentado a implementação do *parser* de legendas de um DCP no sistema de *ingest* da MOG Technologies.

3.5 Ficheiros auxiliares

Tal como mencionado no começo deste capítulo, para além dos ficheiros de vídeo, áudio e legendas, de um DCP constam também uma série de ficheiros que nos vão ajudar a ler e reproduzir corretamente os conteúdos audiovisuais.

Os ficheiros auxiliares de um DCP estão no formato XML e são o seguintes: Composition Playlist (uma ou mais), Packing List, Asset Map e Volume Index.

3.5.1 Composition Playlist

Uma Composition Playlist (CPL) contém instruções sobre a reprodução dos vários ficheiros de áudio, vídeo e legendas: quais devem ser tocados, que fração deles, e em que ordem. Uma

Composition Playlist indica também dados como o Aspect Ratio, o idioma das legendas, por exemplo.

Existe uma Composition Playlist separada para cada versão do filme, com diferentes idiomas do áudio ou legendas. Exemplificando, um DCP de um filme com faixas de áudio em inglês, espanhol, francês e português teria quatro Composition Playlists, uma para cada faixa de áudio.

As Composition Playlists operam sob o conceito de *reels* (concetualmente um rolo ou bobina de filme). Numa CPL, podem existir um ou mais *reels*, sendo que cada *reel* contém uma única faixa de vídeo e/ou áudio e possivelmente legendas.

O standard da DCI menciona que é prática comum dividir uma longa-metragem em *reels* de 10 a 20 minutos de duração, que são posteriormente reproduzidos de forma contínua.

A figura 3.1 mostra um exemplo de um DCP com três Composition Playlists. A primeira CPL contém instruções para a reprodução da faixa de vídeo da longa-metragem, de uma faixa de áudio em inglês e de uma faixa de legendas em espanhol. A segunda CPL corresponde à reprodução do mesmo da mesma faixa de vídeo da CPL #1, mas com áudio em espanhol e legendas em inglês. A terceira CPL refere-se ao *trailer* do filme, apontando para ficheiros de vídeo e áudio distintos das CPL #1 e #2. De notar que o filme se encontra dividido em dois *reels*, ao passo que o trailer está num único *reel*.

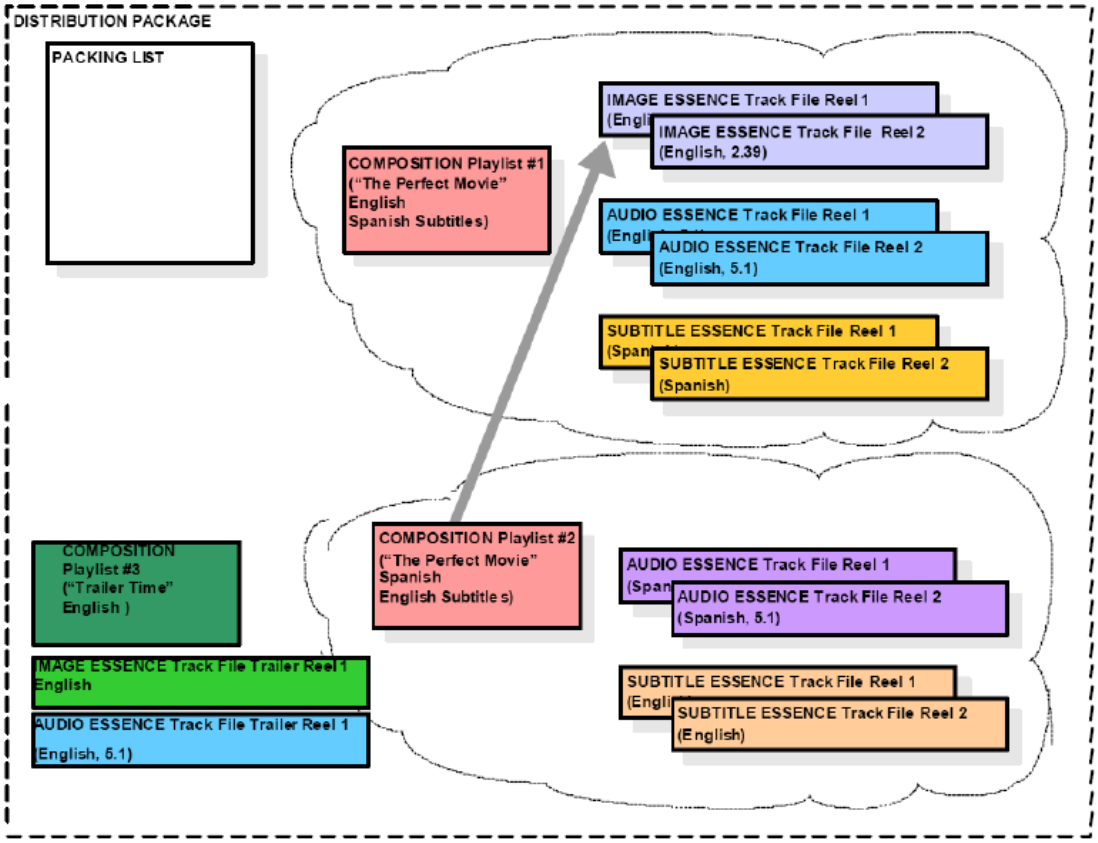

Figura 3.1: Estrutura exemplo de um DCP, com três Composition Playlists [4]

O conceito-chave para compreender o papel de uma Compositon Playlist no contexto de um DCP é ter em mente que um Digital Cinema Package é distribuído, enquanto que uma Composition Playlist é reproduzida. Ou seja, uma das etapas que terá de decorrer no processo de conversão de um DCP para um ficheiro MP4 será perguntar ao utilizador que CPL (de entre várias existentes potencialmente) deseja ver reproduzida no ficheiro de saída.

Informação genérica presente numa Composition Playlist:

- Título do conteúdo
- Tipo de conteúdo (longa-metragem, *trailer*, excerto publicitário, ...);
- Versão do conteúdo;
- Idioma;
- País;
- *Aspect Ratio*;
- Formato da imagem;
- Formato do áudio.

Informação sobre cada ficheiro de media:

- ID único;
- Código de autenticação (opcional);
- *Entry point* (posição do conteúdo em *frames* onde deve começar a reprodução);
- Duração;
- Idioma (para as legendas).

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <CompositionPlaylist xmlns="http://www.smpte-ra.org/schemas/429-7/2006/CPL">
3    <Id>urn:uuid:ba1adbdc-554e-49c8-89d2-cb425d2484f2</Id>
4    <IssueDate>2015-04-10T12:48:46+01:00</IssueDate>
5    <Issuer>Ingreme</Issuer>
6    <Creator>Ingreme</Creator>
7    <ContentTitleText>FILME-X_FTR_F-185_PT-FR_PT_51_2K_20150402_SMPTE_OV</
         ContentTitleText>
8    <ContentKind>feature</ContentKind>
9    <ContentVersion>
10     <Id>urn:uuid:f08a6027-b561-4188-847f-8ba6f710b967</Id>
11     <LabelText>FILME-X_FTR_F-185_PT-FR_PT_51_2K_20150402_SMPTE_OV 2015-04-10
           T12:48:46+01:00</LabelText>
```

```
12    </ContentVersion>
13    <RatingList/>
14    <ReelList>
15      <Reel>
16        <Id>urn:uuid:c634a691-9224-4368-961e-cb55a6d6a819</Id>
17        <AssetList>
18          <MainPicture>
19            <Id>urn:uuid:0f8c4574-1a82-435a-a1c2-c175023d39d3</Id>
20            <AnnotationText>089950.tif</AnnotationText>
21            <EditRate>25 1</EditRate>
22            <IntrinsicDuration>36834</IntrinsicDuration>
23            <EntryPoint>0</EntryPoint>
24            <Duration>36834</Duration>
25            <Hash>FTpU+VH+RkvI4bcJEU6jb5I9a4Q=</Hash>
26            <FrameRate>25 1</FrameRate>
27            <ScreenAspectRatio>1998 1080</ScreenAspectRatio>
28          </MainPicture>
29          <MainSound>
30            <Id>urn:uuid:5c25bed3-1a23-460c-a9c3-1627ab42da73</Id>
31            <AnnotationText>6 channel audio: Filme X Mix 5.1 25fps 31 MAR 24mn33s09fr
                  .L.wav Filme X Mix 5.1 25fps 31 MAR 24mn33s09fr.R.wav Filme X Mix 5.1
                   25fps 31 MAR 24mn33s09fr.C.wav Filme X Mix 5.1 25fps 31 MAR 24
                  mn33s09fr.LFE.w</AnnotationText>
32            <EditRate>25 1</EditRate>
33            <IntrinsicDuration>36834</IntrinsicDuration>
34            <EntryPoint>0</EntryPoint>
35            <Duration>36834</Duration>
36            <Hash>hCyR4dQOcD2Xa/WfzgDIvZEUpD8=</Hash>
37          </MainSound>
38          <MainSubtitle>
39            <Id>urn:uuid:a591734a-bbbd-4a9d-bd15-fca4207dbd96</Id>
40            <AnnotationText>FILME-X_FRANCES_FINAIS_9ABRIL.xml</AnnotationText>
41            <EditRate>25 1</EditRate>
42            <IntrinsicDuration>36834</IntrinsicDuration>
43            <EntryPoint>0</EntryPoint>
44            <Duration>36834</Duration>
45            <Hash>eLEvQi5SeGNPrGSlyUblJsooKrM=</Hash>
46            <Language>fr</Language>
47          </MainSubtitle>
48        </AssetList>
49      </Reel>
50    </ReelList>
51  </CompositionPlaylist>
```

Listing 3.1: Exemplo de uma Composition Playlist de um DCP do tipo SMPTE

3.5.2 Packing List

A Packing List (PKL) contém informação sobre todos os *assets* presentes num DCP, incluindo Composition Playlists e ficheiros de tipo de letra por exemplo, assim como dados para a validação e verificação da integridade dos ficheiros.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<PackingList xmlns="http://www.smpte-ra.org/schemas/429-8/2007/PKL">
  <Id>urn:uuid:933a7363-0960-414b-8d10-54c1f581db7a</Id>
  <AnnotationText>provas</AnnotationText>
  <IssueDate>2015-04-10T12:48:46+01:00</IssueDate>
  <Issuer>Ingreme</Issuer>
  <Creator>Ingreme</Creator>
  <AssetList>
    <Asset>
      <Id>urn:uuid:a591734a-bbbd-4a9d-bd15-fca4207dbd96</Id>
      <AnnotationText>FILME-X_FRANCES_FINAIS_9ABRIL.xml</AnnotationText>
      <Hash>eLEvQi5SeGNPrGSlyUblJsooKrM=</Hash>
      <Size>448490</Size>
      <Type>application/mxf</Type>
      <OriginalFileName>a591734a-bbbd-4a9d-bd15-fca4207dbd96_sub.mxf</
          OriginalFileName>
    </Asset>
    <Asset>
      <Id>urn:uuid:0f8c4574-1a82-435a-a1c2-c175023d39d3</Id>
      <AnnotationText>089950.tif</AnnotationText>
      <Hash>FTpU+VH+RkvI4bcJEU6jb5I9a4Q=</Hash>
      <Size>24416719877</Size>
      <Type>application/mxf</Type>
      <OriginalFileName>0f8c4574-1a82-435a-a1c2-c175023d39d3_j2c.mxf</
          OriginalFileName>
    </Asset>
    <!--RESTANTES ASSETS OCULTADOS-->
  </AssetList>
</PackingList>
```

Listing 3.2: Exemplo de uma Packing List (excerto) de um DCP do tipo SMPTE

3.5.3 Asset Map

As faixas de vídeo, áudio e legendas numa CPL são referidas pelo seu ID único (UUID [29]). A correspondência de cada ID é feita pelo ficheiro Asset Map, que contém uma listagem de todos os *assets* presentes num DCP, e faz o mapeamento entre os ID's dos ficheiros e os respetivos *filenames* no disco rígido em que o DCP está armazenado. O Asset Map tem o nome de ficheiro 'ASSETMAP' nos DCP's do tipo Interop e o nome 'ASSETMAP.xml' nos DCP's do tipo SMPTE.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<am:AssetMap xmlns:am="http://www.smpte-ra.org/schemas/429-9/2007/AM">
  <am:Id>urn:uuid:062e6df7-339b-4541-948f-84c1b7d9b34d</am:Id>
  <am:Creator>Ingreme</am:Creator>
  <am:VolumeCount>1</am:VolumeCount>
  <am:IssueDate>2015-04-10T12:48:46+01:00</am:IssueDate>
  <am:Issuer>Ingreme</am:Issuer>
  <am:AssetList>
    <am:Asset>
      <am:Id>urn:uuid:a591734a-bbbd-4a9d-bd15-fca4207dbd96</am:Id>
      <am:ChunkList>
        <am:Chunk>
          <am:Path>a591734a-bbbd-4a9d-bd15-fca4207dbd96_sub.mxf</am:Path>
          <am:VolumeIndex>1</am:VolumeIndex>
          <am:Offset>0</am:Offset>
          <am:Length>448490</am:Length>
        </am:Chunk>
      </am:ChunkList>
    </am:Asset>
    <am:Asset>
      <am:Id>urn:uuid:735c105d-3342-4675-acc1-1b094b2b36f5</am:Id>
      <am:ChunkList>
        <am:Chunk>
          <am:Path>735c105d-3342-4675-acc1-1b094b2b36f5_sub.mxf</am:Path>
          <am:VolumeIndex>1</am:VolumeIndex>
          <am:Offset>0</am:Offset>
          <am:Length>439473</am:Length>
        </am:Chunk>
      </am:ChunkList>
    </am:Asset>
    <!--RESTANTES ASSETS OCULTADOS-->
  </am:AssetList>
</am:AssetMap>
```

Listing 3.3: Exemplo de um Asset Map (excerto) de um DCP do tipo Interop

3.5.4 Volume Index

Um DCP pode estar potencialmente espalhado por múltiplos discos rígidos. O ficheiro Volume Index faz a identificação dos vários volumes que podem eventualmente compôr um DCP. No entanto, atualmente um DCP estará regra geral contido num único disco, pelo que este ficheiro deixa de ser necessário. Semelhantemente ao Asset Map, o Volume Index tem o nome de ficheiro 'VOLINDEX' nos DCP's do tipo Interop e o nome 'VOLINDEX.xml' nos DCP's do tipo SMPTE.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

```
2 <VolumeIndex xmlns="http://www.digicine.com/PROTO-ASDCP-VL-20040311#" xmlns:xsi="
    http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.
    digicine.com/PROTO-ASDCP-VL-20040311# VolumeLabel.xsd">
3   <Index>1</Index>
4 </VolumeIndex>
```

Listing 3.4: Exemplo de um Volume Index de um DCP do tipo Interop

3.6 Conclusão

Neste capítulo foi clarificada a estrutura do pacotes de ficheiros de cinema digital — tanto dos ficheiros de conteúdo audiovisual propriamente dito como dos ficheiros auxiliares de informação adicional — que serão lidos e processados pela plataforma desenvolvida.

Capítulo 4

Solução proposta

Neste capítulo é apresentada a solução proposta e a respetiva arquitetura para o problema exposto no capítulo 1.

4.1 Arquitectura da solução proposta

A plataforma de *ingest* desenvolvida no âmbito desta dissertação fará parte de uma cadeia de transformação de conteúdos cinematográficos que engloba, em termos de implementação técnica, três organizações: a Cinemateca Portuguesa, a MOG Technologies e a YouOn. Uma apresentação sobre estas entidades foi feita no capítulo introdutório.

Na figura 4.1 são apresentadas de uma forma sequencial as várias fases que compõem a cadeia, assim como a organização responsável pela execução de cada fase.

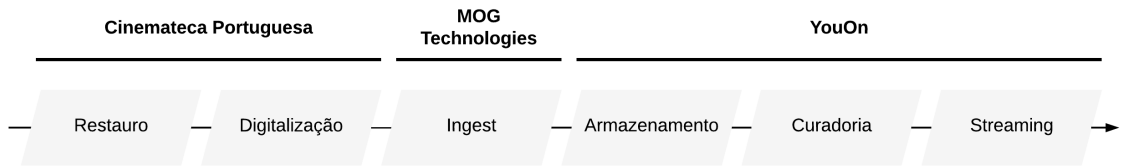

Figura 4.1: Cadeia de processamento e distribuição de obras cinematográficas

A MOG Technologies será responsável pela fase de *Ingest*, que será executada através da plataforma concebida no contexto da presente dissertação. Nesta etapa do *workflow* as obras cinematográficas da Cinemateca Portuguesa selecionadas são transformadas num formato adequado para a distribuição de vídeo sobre a internet, de acordo com alguns parâmetros definidos pelo utilizador.

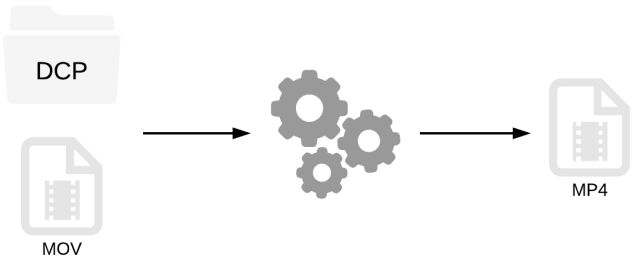

Figura 4.2: *Inputs* e *output* da plataforma de *ingest*

Da parte da Cinemateca Portuguesa darão entrada na plataforma filmes num de dois formatos possíveis:

- Filmes já armazenados originalmente num Digital Cinema Package não sofrerão alterações e seguirão para a etapa de Ingest nesse suporte;
- Filmes em suportes analógicos ou noutros suportes digitais serão convertidos para um ficheiro MOV, com vídeo no formato ProRes e áudio no formato PCM.

O formato de saída do processo de transformação dos conteúdos, conforme o estudo e a posterior decisão descritas no capítulo 2, será um ficheiro MP4, com vídeo no formato H.264/AVC e áudio no formato AAC.

Nesta dissertação o foco recaiu particularmente no *ingest* de conteúdos armazenados num Digital Cinema Package. As ferramentas da MOG Tecnologies já se encontravam devidamente preparadas para o processamento de conteúdos com vídeo ProRes, mas não possuíam ainda os mecanismos necessários para o *ingest* de DCP's.

A figura 4.3 exibe os processos que vão moldar um DCP ao longo da plataforma de ingest da MOG Technologies, terminando na geração de um ficheiro de vídeo MP4. Cada processo tem uma cor distinta de forma a identificar o ator por ele responsável.

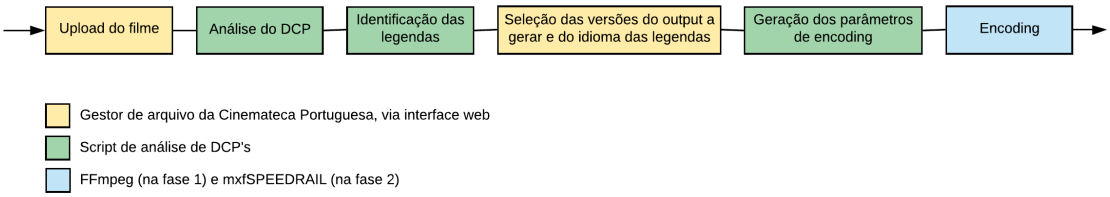

Figura 4.3: Processos associados à plataforma de *ingest*

A etapa de *ingest* começará na Cinemateca Portuguesa, onde o gestor do arquivo cinematográfico da instituição fará a seleção do DCP da obra a converter, através de uma página *web* desenvolvida para o efeito.

Conforme descrito no capítulo 3, um DCP é um conjunto de ficheiros de *media* e ficheiros auxiliares de *metadata*. Inicialmente é necessário realizar uma análise destes ficheiros auxiliares, de forma a ler corretamente os ficheiros de *media* e a identificar as Composition Playlists que existem no DCP (que correspondem a versões distintas do filme, com legendas em idiomas diferentes por exemplo). Essa análise é feita por um *script* de análise de DCP's, desenvolvido na linguagem de programação Python, que identifica também o idioma das legendas (se existirem) em cada Composition Playlist.

Este dados são exibidos na página *web*, onde o gestor de arquivo da Cinemateca pode escolher se deseja incluir legendas no vídeo, e em que idioma para o caso de existirem múltiplos idiomas, assim como as parâmetros do ficheiro de vídeo, como a resolução e a presença de *timecode*, conforme definido nos requisitos do sistema, na secção 4.2 .

Os dados retornam ao *script* de análise de DCP's, que processa as opções do utilizador, e gera os parâmetros necessários para o *transcoder* realizar a conversão do vídeo e áudio, responsável pela conversão dos ficheiros audiovisuais propriamente ditos.

A figura 4.4 apresenta os dois módulos estruturais do processo, e que ficheiros do DCP são tidos em conta para cada módulo.

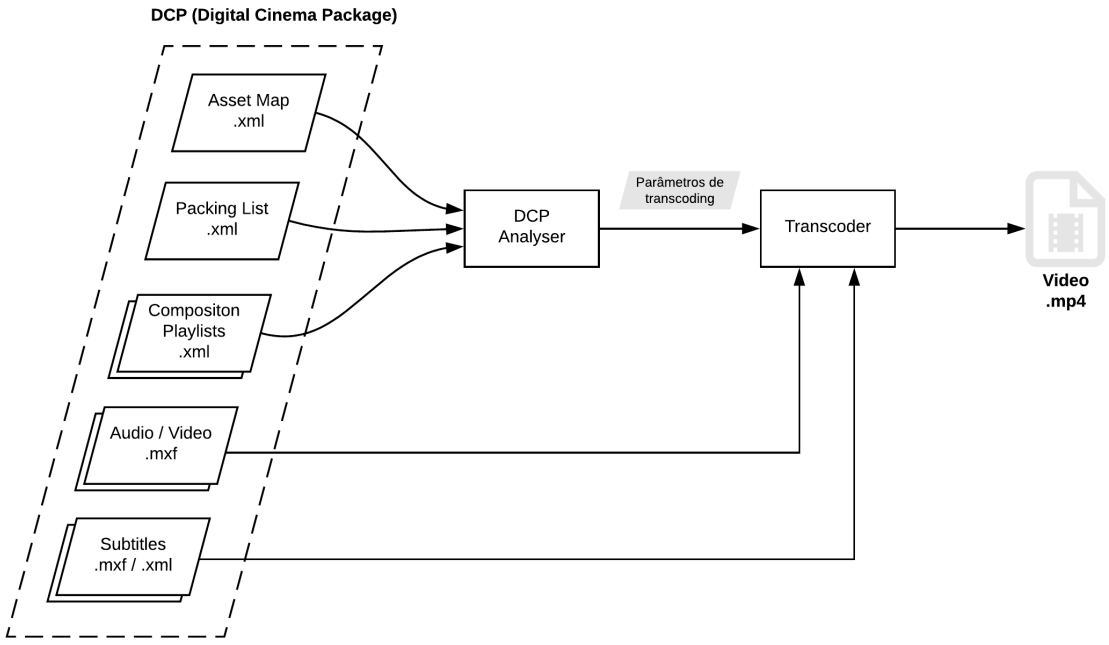

Figura 4.4: Módulos estruturais do processo de ingest

A implementação da arquitetura foi dividida em duas fases:

- Numa primeira fase foi desenvolvido um protótipo completo da plataforma de *ingest*, separado do sistema profissional de *ingest* da empresa (mxfSPEEDRAIL);

- Numa segunda fase foi feita a implementação do módulo de *transcoding* no mxfSPEEDRAIL. A implementação do módulo de análise dos DCP's no sistema da empresa não fez parte dos objetivos da presente dissertação propostos pela empresa.

4.2 Requisitos

Foram definidos pelas várias entidades envolvidas no projeto os seguintes requisitos funcionais para a plataforma de *ingest*:

1. O gestor de arquivo da Cinemateca Portuguesa deve poder gerar cópias de um filme em arquivo de resolução HD para o Plano Nacional de Cinema (PNC) e de resolução SD para visualização interna (*proxy*);

2. O gestor de arquivo da Cinemateca Portuguesa deve poder incluir legendas no vídeo de ambas as versões geráveis, com diferentes idiomas (se existirem) em cada versão;

3. A versão para o PNC deve ter uma resolução de 720p;

4. A versão *proxy* interna deve ter uma resolução de 576p;

5. Ambas as versões devem preservar o *frame rate* do filme original;

6. As legendas devem ser *burned in* na imagem;

7. A versão *proxy* interna deve ter o *timecode burned in*;

8. A versão *proxy* interna deve ter o logótipo da Cinemateca Portuguesa;

9. Ambas as versões devem estar num ficheiro MP4;

10. O vídeo de ambas as versões deve estar no formato H.264;

11. O áudio de ambas as versões deve estar no formato AAC;

12. O áudio de ambas as versões deve estar amostrado a 48 kHz;

13. O áudio de ambas as versões deve ter um máximo de 2 canais.

Na figura 4.5 é exibido um esboço do aspeto visual que as duas versões terão, com os vários *overlays* aplicados nas respetivas versões.

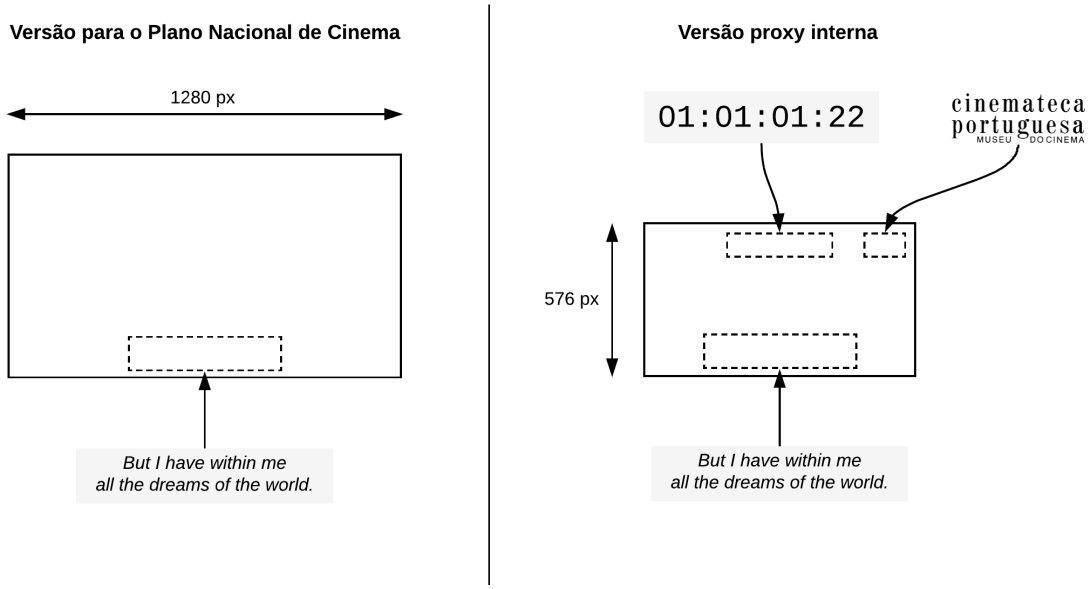

Figura 4.5: Estrutura visual das frames de vídeo das versões de saída da plataforma de *ingest*

Poderão ser portanto geradas duas versões distintas do ficheiro de saída, com diferentes propósitos.

A versão para o Plano Nacional de Cinema (PNC) será a versão encaminhada para as etapas seguintes da plataforma, de distribuição das obras cinematográficas através da *cloud*. Terá uma resolução HD (720p), sendo a largura da imagem fixa em 1280 pixels, e a altura da imagem ajustada de forma a respeitar o *aspect ratio* da imagem original.

A versão *proxy* interna destina-se à visualização *offline* das obras na Cinemateca Portuguesa, tendo uma qualidade de imagem menor e o *timecode* embutido na imagem, para facilitar a visualização, marcação e o corte de cenas do filme. Terá uma resolução 576p, sendo a altura da imagem fixa em 576 pixels e a largura da imagem ajustável também para respeitar o *aspect ratio* original.

A dimensão a fixar segue as resoluções habitualmente utilizadas pela indústria audiovisual para os diferentes formatos e *aspect ratios*, exibidas nas tabelas 4.1 e 4.2 para os formatos de resolução que vamos gerar.

Tabela 4.1: *Aspect Ratios* e respetiva resolução da imagem para o formato 720p [6] [7]. O Pixel Aspect Ratio (PAR) de 1:1 corresponde a pixels quadrados.

Aspect Ratio	**Resolução da imagem (PAR 1:1)**
1.33:1 (4:3)	1280 x 962
1.66:1 (5:3)	1280 x 768
1.77:1 (16:9)	1280 x 720
1.85:1	1280 x 692
2:1	1280 x 640
2.35:1	1280 x 545
2.37:1 (RED Wide)	1280 x 540
2.39:1 (por vezes chamado de 2.40)	1280 x 536
2.40:1	1280 x 533
2.44	1280 x 525

Tabela 4.2: *Aspect Ratios* e respetiva resolução da imagem para o formato 576p (PAL SD) [6]

Aspect Ratio	**Resolução da imagem**		
	PAR 1:1	**PAR 1.46:1 (PAL Widescreen)**	**PAR 1.09:1 (PAL DV)**
1.33:1 (4:3)	768 x 576	-	720 x 576
1.77:1 (16:9)	1024 x 576	720 x 576	-

4.3 Tecnologias utilizadas

Para o desenvolvimento da plataforma foram utilizadas as seguintes ferramentas:

- **FFmpeg:** ferramenta *open-source* de *transcoding* de ficheiros multimédia, com um alargadíssimo suporte de formatos de píxel, codecs e *containers* (interface via linha de comandos);
- **asdcp-unwrap:** ferramenta *open-source* de desencapsulamento de essências de conteúdos multimédia e de extração de metadata de ficheiros MXF de cinema digital (interface via linha de comandos);
- **Subtitle Edit:** ferramenta *open-source* de criação, edição e sincronização de ficheiros de legendas, com suporte para mais de 200 formatos (viar interface gráfica ou via linha de comandos);
- **HTML5, CSS, JavaScript:** linguagens de criação de páginas *web*;
- **Flask:** *framework* de desenvolvimento de serviços web;
- **Python 3:** linguagem de programação;
- **C++11:** linguagem de programação (utilizada pelo *software* da MOG Technologies);

Capítulo 5

Análise dos DCP's de amostra da Cinemateca Portuguesa

De forma a dar a conhecer o tipo e a complexidade dos DCP que dariam entrada na plataforma de *Ingest*, a Cinemateca Portuguesa facultou à MOG Technologies quatro DCP exemplo. Neste capítulo é feita uma análise às características das amostras recebidas, divididas em quatros vertentes: informação genérica, vídeo, áudio e legendas. Esta análise serviu também para despistar eventuais incongruências com as normas definidas pela especificação da DCI sobre DCP.

Por motivos de confidencialidade os nomes dos filmes são substituídos neste documento pelas letras A, B, C e D.

5.1 Informação genérica

Dos quatro DCP's recebidos, três são do tipo SMPTE e um do tipo Interop. A distribuição dos ficheiros auxiliares de *metadata* segue as normas da DCI: um Asset Map, um Volume Index, uma Packing List, e uma ou mais Composition Playlists.

Todos os filmes recebidos estão isentos de encriptação, e as suas Composition Playlists possuem os conteúdos audiovisuais circunscritos a um único *reel*. O desenvolvimento do programa de análise de DCP's foi por isso orientado para esta configuração.

Tabela 5.1: Dados gerais sobre os quatro Digital Cinema Packages enviados pela Cinemateca Portuguesa

	Filme A	**Filme B**	**Filme C**	**Filme D**
Duração	00:23:15	00:19:42	00:24:33	00:13:40
Tamanho	15,7 GB	7,48 GB	23,9 GB	9,95 GB
Tipo de DCP	Interop	SMPTE	SMPTE	SMPTE
Criado por	Fraunhofer IIS easyDCP Creator 2.2.7	Fraunhofer IIS easyDCP Creator 2.0.0	Fraunhofer IIS easyDCP Creator 2.2.7	libdcp 0.93pre
PKL (qnt.)	1	1	1	1
CPL (qnt.)	4	1	3	1
ASSETMAP (qnt.)	1	1	1	1
VOLINDEX (qnt.)	1	1	1	1

5.2 Vídeo

No que aos ficheiros de vídeo diz respeito, nos DCP recebidos estava presente apenas um ficheiro, com o filme completo. Em termos de características da imagem, todas dentro dos parâmetros definidos pela DCI.

Todos os ficheiros MXF de vídeo seguem a norma da SMPTE definida para cinema digital, SMPTE 429-4.

Tabela 5.2: Dados sobre a componente de imagem dos quatro Digital Cinema Packages enviados pela Cinemateca Portuguesa

	Filme A	**Filme B**	**Filme C**	**Filme D**
Package Name	SMPTE 429-4 frame wrapping of JPEG 2000 codestreams	SMPTE 429-4 frame wrapping of JPEG 2000 codestreams	SMPTE 429-4 frame wrapping of JPEG 2000 codestreams	SMPTE 429-4 frame wrapping of JPEG 2000 codestreams
Container	MXF	MXF	MXF	MXF
Codec	JPEG2000	JPEG2000	JPEG2000	JPEG2000
Resolução	2K	2K	2K	2K
Largura	1998	2048	1998	1998
Altura	1080	858	1080	1080
Aspect Ratio	1.85:1	2.39:1	1.85:1	1.85:1
Frame rate	25	24	25	25
Espaço de cor	XYZ	XYZ	XYZ	XYZ
Chroma sub-sampling	4:4:4	4:4:4	4:4:4	4:4:4
Bit depth	12	12	12	12
Bitrate	89,9 Mbps	52,1 Mbps	133 Mbps	102 Mbps
Tamanho	14,6 GB	7,17 GB	22,7 GB	9,73 GB

5.3 Áudio

O áudio está também em conformidade e segue o standard de cinema digital SMPTE 382M. Existe, tal como no vídeo, apenas um ficheiro de áudio em cada DCP. Os filmes B e D possuem 2 canais de áudio, enquanto que os filmes A e C possuem 6 canais, que terão de ser *downmixed* para formar 2 canais à saída.

Tabela 5.3: Dados da componente de áudio dos quatro Digital Cinema Packages enviados pela Cinemateca Portuguesa

	Filme A	**Filme B**	**Filme C**	**Filme D**
Package Name	SMPTE 382M frame wrapping of wave audio	SMPTE 382M frame wrapping of wave audio	SMPTE 382M frame wrapping of wave audio	SMPTE 382M frame wrapping of wave audio
Container	MXF	MXF	MXF	MXF
Codec	PCM	PCM	PCM	PCM
Bitrate	6912 kbps	2304 kbps	6912 kbps	2304 kbps
Sampling rate	48 kHz	48 kHz	48 kHz	48 kHz
Bit depth	24 bits	24 bits	24 bits	24 bits
Nº de canais	6	2	6	2
Tamanho	1,16 GB	325 MB	1,19 GB	225 MB

5.4 Legendas

O filme A tem três legendas, diferentes idiomas (inglês, espanhol e francês). Estando armazenadas num DCP do tipo Interop, as legendas estão num container XML, formatadas de acordo com o standard CineCanvas. A existência de três ficheiros de legendas motiva a presença de quatro Composition Playlists: uma CPL para a reprodução do filme sem qualquer legenda e três CPL's para a reprodução do filme com cada legenda existente.

O filme C possui dois ficheiros de legendas, em inglês e francês, tendo por esse motivo o seu DCP três Composition Playlists. O DCP do filme C é do tipo SMPTE, pelo que as legendas estão formatadas segundo o standard SMPTE 429-5 (confirmar) e estão encapsuladas num ficheiro MXF.

Os filmes B e D não possuem nenhuma ficheiro de legendas.

Tabela 5.4: Dados sobre os ficheiros de legendas dos quatro Digital Cinema Packages enviados pela Cinemateca Portuguesa

	Filme A	**Filme B**	**Filme C**	**Filme D**
Package Name	-	-	SMPTE 429-5 clip wrapping of D-Cinema Timed Text data	-
Idiomas	en, es, fr	-	en, fr	-
Container	XML	-	MXF	-
Formato	CineCanvas	-	SMPTE 428-7	-

Capítulo 6

Desenvolvimento do protótipo

Neste capítulo é documentado o desenvolvimento do protótipo da plataforma de *ingest*.

O módulo de *transcoding* das essências de *media* dos DCP's foi implementado através da ferramenta de processamento de vídeo *open-source* FFmpeg, da ferramenta *open-source* de processamento de legendas SubtitleEdit, e da ferramenta *open-source* de *unwrapping* de ficheiros MXF asdcp-unrwap.

Todas estas ferramentas funcionam através da linha de comandos do computador. Ao longo deste capítulo são apresentados os vários comandos que invocam e encadeiam estas ferramentas, de uma forma progressiva em termos de complexidade.

O módulo de análise de DCP's foi implementado com um programa desenvolvido na linguagem de programação Python.

A página *web* que serve de interface do utilizador com o módulo de análise de DCP's foi desenvolvida recorrendo às ferramentas de criação de páginas web HTML5, CSS e JavaScript.

6.1 Transcodificação das essências de vídeo e áudio

O desenvolvimento da protótipo partiu do processo fundamental da cadeia de conversão: a transformação das essências de áudio e vídeo de um DCP num ficheiro MP4.

Este passo foi concretizado com a ferramenta de processamento de vídeo e áudio FFmpeg.

No comando que se segue, os ficheiros MXF de vídeo e áudio de um DCP, codificados em JPEG 2000 e PCM respetivamente, são desencapsulados e descodificados pelo FFmpeg. O espaço de cor da imagem é convertido de XYZ para YUV (o formato de cor suportado pelo H.264 [30]). As dimensões da imagem são redimensionadas de forma a conservar o *aspect ratio* original do filme. Foi definido nos requisitos funcionais que deviam ser geradas duas versões, uma com resolução 720p, outra com resolução 576p. Neste comando é gerada a versão 720p, que deve ter obrigatoriamente 1280 pixels de largura. A altura é calculada pelo FFMpeg de forma a respeitar o aspect ratio.

O vídeo é depois codificado no formato H.264 pelo *enconder* x264 do FFmpeg, com um Constant Rate Factor (CRF) de 21 (parâmetro de codificação de vídeo H.264 que define a qualidade

da imagem, e que varia entre 0 e 51, correspondendo os valores mais baixos a níveis mais elevados de qualidade). O áudio, originalmente com 6 canais, passa por um processo de *downmixing* para 2 canais, e é codificado no formato AAC. As *streams* de vídeo e áudio obtidas são depois multiplexadas num *container* MP4. Segue o comando que traduz as operações descritas:

```
ffmpeg -i dcpvideo.mxf -i dcpaudio.mxf -c:v libx264 -pix_fmt yuv420p -vf "scale
    =1280:-2" -preset ultrafast -crf 21 -ac 2 output.mp4
```

Listing 6.1: Comando do FFmpeg de transcodificação do vídeo e do áudio

Figura 6.1: Imagem do DCP (acima), no espaço de cor XYZ e imagem do ficheiro de saída no formato H.264(abaixo), no espaço de cor YUV

Durante a execução deste comando, o FFmpeg exibe uma série avisos relativos à descodificação da imagem, no formato JPEG 2000, possivelmente devido à conversão do espaço de cor da imagem de XYZ para YUV. Alguns utilizadores na comunidade de suporte *online* do FFmpeg reportaram ter encontrado o mesmo tipo de avisos e, em alguns casos, o aparecimento de artefactos na imagem do ficheiro de saída. Nas imagens obtidas com este comando (figura 6.1), no entanto, não foram detetados artefactos de qualquer tipo. É distinguível apenas uma diferença na

tonalidade das cores das imagens de entrada e de saída, que advirá das representações cromáticas distintas em cada espaço de cor XYZ e YUV.

Alcançado o objetivo de transformar vídeo e áudio de um DCP num ficheiro MP4 com as características desejadas — vídeo em H.264, áudio em AAC com 2 canais — seguiu-se a incorporação no vídeo das três componentes pedidas nos requisitos: legendas, *timecode* e marca-d'água.

6.2 Aplicação de legendas

O FFmpeg possibilita a inclusão de legendas num ficheiro de video de duas formas:

- Sobrepondo as legendas no vídeo (operação comummente denominada de *subtitles burnin* ou *subtitles hardcoding*);
- Acrescentando uma *stream* de texto ao *container* (um *container* MKV, por exemplo, teria uma *stream* de video H.264, uma *stream* de áudio AAC e uma *stream* de texto).

Confrontados com as duas possibilidades, optou-se pelo *burnin* das legendas no vídeo. A inclusão de legendas sob a forma de uma *stream* adicional multiplexada no container poderia levantar ligeiras dificuldades na visualização dos conteúdos. Ainda não está definido por esta altura se os conteúdos serão disponibilizados através de *streaming* ou através de *download* posterior visualização *offline*. No primeiro cenário não é certo que o *player* da plataforma desenvolvida pelas empresas parceiras do projeto suporte *streams* de legendas. No segundo cenário, de *download*, a correta exibição das legendas estaria dependente do player instalado nos computadores das escolas. A decisão de sobrepor no próprio vídeo as legendas foi a tomada por garantir que as legendas serão certamente e devidamente exibidas.

A operação de *subtitles burnin* no FFmpeg é realizável com um dos seguintes filtros:

- filtro 'subtitle', que aceita ficheiros do tipo SubRip (SRT);
- filtro 'ass', que aceita ficheiros do tipo Advanced SubStation Alpha (ASS);
- filtro 'overlay', que aceita ficheiros de legendas baseados em imagens, como um ficheiro PNG.

Conforme descrito no capítulo 3, as legendas num DCP podem estar sob a forma de texto ou sob a forma de imagens. Os DCP's de amostra que a MOG Technologies recebeu da Cinemateca Portuguesa que continham legendas tinham-nas no formato de texto, pelo que os processos desenvolvidos foram orientados para o tratamento de legendas nesse formato.

As legendas *text-based* de um DCP consistem num documento XML, possivelmente encapsulado num ficheiro MXF (dependendo do tipo de DCP gerado). O FFmpeg não aceita ficheiros XML/MXF à sua entrada como um ficheiro de legendas. Dada a natureza textual do documento XML, foi lógico tentar a sua conversão para um dos formatos *text-based* de legendas que o FFmpeg aceita.

Foi portanto necessário encontrar uma forma de:

1. Desencapsular o ficheiro MXF de legendas de um DCP, quando este é do tipo SMPTE, obtendo um ficheiro XML (esta operação não é necessária quando o DCP é do tipo Interop);

2. Converter um ficheiro XML num ficheiro de um dos tipos supra-mencionados — SRT ou ASS.

Era importante que as ferramentas que encontradas operassem através da linha de comandos, tal como o FFmpeg, e não através de uma interface gráfica (GUI). Dessa forma seria possível encadear os vários passos numa *pipeline*, em que o *output* de um programa é o *input* do programa seguinte, automatizando o processo.

O passo 1, de desencapsulamento, ou *unwrapping*, foi executado com a ferramenta asdcp-unwrap. Esta ferramenta é parte integrante da biblioteca de *software open-source* 'asdcplib', que inclui ferramentas de análise, extração de informação e criação de DCP's.

Existem vários parâmetros opcionais que podem ser passados ao programa, mas para o objetivo a atingir é suficiente passar o nome do ficheiro de entrada e o nome do ficheiro de saída:

```
asdcp-unwrap sub1.mxf sub1.xml
```

Listing 6.2: Comando de desencapsulamento de legendas num ficheiro MXF para um ficheiro XML

Caso o FFmpeg não conseguisse ler os ficheiros MXF de áudio e vídeo dos DCPs que lhe são passados, poderia ter sido usada esta biblioteca para fazer o *unwrapping* também dos ficheiros de áudio e vídeo, injetando posteriormente no FFmpeg as essências em JPEG 2000 e PCM. Tal não foi necessário, dado que o FFmpeg consegue realizar o desencapsulamento das essências de som e imagem.

Para o passo 2 foi utilizada a ferramenta SubtitleEdit. Este programa, também *open-source*, aceita mais de 200 formatos de legendas de input/output, incluindo os desejados formatos SRT e ASS.

Para o formato de saída optou-se, pelo menos nesta fase, pelo formato SRT, dado que este transporta a informação mais importante de um ficheiro de legendas — texto e tempos de entrada e saída — e tem uma estrutura mais simples que o ASS.

A sintaxe do SubtitleEdit é similar à do asdcp-unrwap: são-lhe passados os dados do *input*, *output* e opcionalmente parâmetros de conversão. Uma pequena diferença existe: no output não é indicado o nome do ficheiro, mas apenas o formato que se deseja obter. O ficheiro de output terá o mesmo nome do ficheiro de input, com a devida diferença na extensão.

```
SubtitleEdit /convert sub1.xml srt
```

Listing 6.3: Comando de conversão de um ficheiro de legendas XML para um ficheiro SRT

Com a aplicação destas duas ferramentas é atingido então o objetivo delineado: a conversão das legendas de texto de um DCP para um formato legível pelo FFmpeg.

A passagem de um ficheiro de legendas SRT ao FFmpeg faz-se recorrendo ao filtro 'subtitle'. Os filtros do FFmpeg atuam sobre as *frames* do vídeo, que são obtidas após o *demuxing* e a descodificação do ficheiro de entrada, num processo apresentado na figura 6.2.

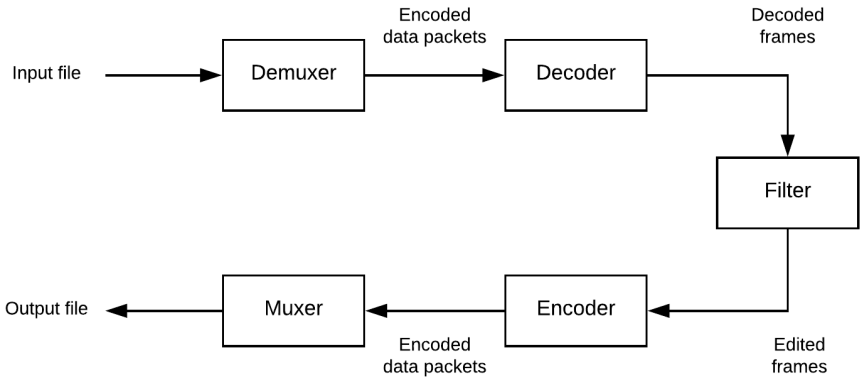

Figura 6.2: Lógica de processamento do FFmpeg

O seguinte comando ilustra um exemplo da utilização do filtro 'subtitles' para a sobreposição de legendas a um vídeo no formato MP4.

```
ffmpeg -i video.mp4 -vf subtitles=subtitle.srt out.mp4
```

Listing 6.4: Comando inserção de legendas no FFmpeg

De notar que o ficheiro de legendas não é passado ao FFmpeg no início do comando, juntamente com os restantes inputs, mas sim na invocação do próprio filtro de legendas.

6.3 Aplicação do *timecode*

A sobreposição do *timecode* no video foi invocada no FFmpeg através do filtro 'drawtext', que aplica na imagem num texto personalizável. Neste caso o texto a aplicar será um *timecode*, que é uma marca temporal utilizada para efeitos de sincronização. São exibidas os campos horas, minutos, segundos e frames, incrementados à medida que o vídeo é reproduzido.

São passados os seguintes parâmetros ao filtro: tipo de letra do texto, campos do *timecode* que vão ser incrementados, framerate, posição na imagem, tamanho do texto, cor do texto, existência de *box* envolvente, e cor da *box* (se existir).

Eis o comando que invoca filtro drawtext do FFmepg para a operação de *timecode burnin*:

```
ffmpeg -i video.mp4 -vf "drawtext=fontfile=/Windows/Fonts/arial.ttf:timecode
    ='00\:00\:00\:00':r=25:x=(w-tw)/2:y=(10):fontsize=60:fontcolor=white:box=1:
    boxcolor=0x00000099" -y output.mp4
```

Listing 6.5: Comando do FFmpeg para a sobreposição do *timecode* no vídeo

O eixo das coordenadas de posicionamento do *timecode* na imagem, 'x' e 'y', têm origem no canto superior esquerdo da imagem. A cada coordenada podem ser passados diretamente valores em pixels, ou então variáveis a que o FFmpeg atribui um significado. 'w' é a largura das frames do vídeo, 'tw' é a largura do texto a ser desenhado, 'h' é a altura das frames do vídeo, 'lh' é a altura de uma linha de texto. No comando acima, 'x=(w-tw)/2' centra o *timecode* horizontalmente, e 'y=10' coloca o *timecode* 10 pixels abaixo do topo da imagem.

6.4 Aplicação da marca-d'água

No FFmpeg a sobreposição de uma marca-d'água no video é conseguida utilizando o filtro 'overlay'. A imagem a ser sobreposta é passada como um dos *inputs* da cadeia de conversão, e, dado que há mais do que um *input* do tipo gráfico (são passados vídeo e logótipo), o filtro é colocado após o parâmetro '-filter_complex', em detrimento do parâmetro '-vf' utilizado nos comandos anteriores.

```
ffmpeg -i video.mp4 -i watermark.png -filter\_complex "overlay=10:10" out.mp4
```

Listing 6.6: Comando do FFmpeg para a aplicação da marca-d'água no vídeo

É passada de forma semelhante ao *timecode burnin* a posição da imagem a sobrepôr, com os valores após o termo 'overlay' como a distância em pixels desde o canto superior esquerdo da *frame* de vídeo, segundo o eixo horizontal e vertical, respetivamente.

6.5 Aplicação simultânea dos filtros visuais

Identificado o comando que converte o vídeo e áudio de um DCP num ficheiro MP4, assim como os filtros visuais que realizam as operações requeridas, foi gerado um comando único do FFmpeg, que aplica em simultâneo os três filtros, e gera na saída as duas versões de distintas resoluções (HD e SD). Seria mais simples realizar as operação com um comando para cada resolução de saída, no entanto há uma poupança de recursos computacionais executando a transcodificação numa única instância do FFmpeg.

Neste comando foi necessário usar a opção 'filtergraph output labelling' e a opção de seleção de *streams* para realizar o encadeamento dos filtros, e a opção de mapeamento de *streams*, através do parâmetro '-map', para gerar os ficheiros de saída.

As linhas de texto corresponde a um único comando do FFmpeg, mas foram separadas em função dos vários parâmetros passados, de forma a facilitar a interpretação do comando.

```
ffmpeg -ss 00:00:00 -i dcpvideo.mxf -ss 00:00:00 -i dcpaudio.mxf -i logo.png

-filter_complex "[0:v] scale=-2:576 [sc1]; [sc1] subtitles=sub1.srt [sub1]; [sub1
    ][2] overlay=10:10 [ovr1]; [ovr1] drawtext=fontfile=/Windows/Fonts/arial.ttf:
    timecode='00\:00\:00\:00':r=25:x=(w-tw)/2:y=(10):fontsize=60:fontcolor=white:
    box=1:boxcolor=0x00000099 [tc1]"

-map "[tc1]" -map 1 -pix_fmt yuv420p -c:v libx264 -preset ultrafast -crf 21 -ac 2
    output576.mp4

-filter_complex "[0:v] scale=1280:-2 [sc2]; [sc2] subtitles=sub1.srt [sub2]"

-map "[sub2]" -map 1 -pix_fmt yuv420p -c:v libx264 -preset ultrafast -crf 21 -ac 2
    output720.mp4
```

Listing 6.7: Comando do FFmpeg para aplicação simultânea dos filtros visuais

Na primeira linha do comando são passados três *inputs*: vídeo, áudio e logótipo da Cinemateca Portuguesa. É adicionalmente incluído um parâmetro aos *inputs* das faixas de vídeo e áudio, que indica em que instante devem estas começar a ser lidas. A inclusão deste parâmetro será útil para proceder à sincronização entre vídeo e áudio, quando for analisada a informação presente na(s) Composition Playlist(s) de um DCP, documentada nas secções subsequentes.

Na segunda linha é desenhada a cadeia de filtros do vídeo do primeiro ficheiro de saída. Os filtros de *scaling* (para 576p nessa linha), *subtitles burnin*, *timecode burnin* e *watermark burnin* são aplicados de uma forma sucessiva, em que o *output* de um primeiro filtro é o *input* do segundo, e assim sucessivamente.

Na terceira linha o vídeo (já processado pelos filtros) e o áudio são mapeados para comporem o ficheiro de saída de resolução SD. Os parâmetros de *encoding* são os já expostos na secção 6.1.

Na quarta linha é desenhada a cadeia de filtros do segundo ficheiro de saída. Para este ficheiro de saída o *timecode* e a marca-d'água não é aplicada, e a imagem é *downscaled* para 720p.

A quinta e última linha, idêntica à terceira linha, faz o mapeamento do vídeo e áudio que compõem o ficheiro de resolução HD.

6.6 Análise de um DCP e automatização do protótipo

A geração de dois ficheiros de vídeo, de resoluções e características distintas, a partir dos ficheiros de vídeo e áudio presentes num DCP é realizada com sucesso segundo o processo documentado neste capítulo até ao presente ponto.

No entanto, não foi tido ainda em consideração para o processo documentado que num DCP podem existir múltiplos ficheiros de vídeo, áudio e legendas, e que estes ficheiros por si só não são

suficientes para saber como exibir o filme. Para isso têm de ser analisadas as várias Composition Playlists que podem existir num DCP, que correspondem a uma versão única e diferenciada de reproduzir o filme. Têm igualmente de ser analisados o Asset Map e a Packing List do DCP, que contêm informação para a correta identificação e leitura dos ficheiros audiovisuais do DCP.

Para realizar a extração de todas estas informações da pasta do DCP foi desenvolvido um programa em Python, que serviu também de simples interface com o utilizador numa fase inicial, através da linha de comandos do computador.

O utilizador digita na linha de comandos a localização da pasta do DCP que deseja converter e em que pasta deseja guardar o ficheiro de saída. O programa, de seguida, analisa primeiramente os ficheiros XML presentes na pasta do DCP e identifica a Packing List e as Composition Playlists presentes.

De seguida, o programa verifica quais das Composition Playlists possuem legendas (e em que idiomas se estas existerem), e imprime na linha de comandos a informação reunida.

O utilizador indica a CPL a utilizar para gerar o filme. A partir deste ponto, o programa extrai automaticamente toda a informação necessária dos ficheiros XML que constam do DCP - a Packing List, o Asset Map e a Composition Playlist selecionada - para proceder ao transcoding, nomeadamente:

- IDs dos *assets* de vídeo, áudio e de legendas (da CPL);
- *Paths* no disco rígido dos ficheiros de vídeo, áudio e de legendas (do Asset Map e da Packing List);
- *Entry points* dos ficheiros de vídeo e áudio (da CPL);
- *Frame rate* do vídeo (da CPL).

Obtidos estes dados, o programa configura os comandos para cada ferramenta — asdcp-unrwap, SubtitleEdit e FFmpeg — substituindo cada espaço no comando pela informação extraída dos ficheiros de metadata auxiliar do DCP, e ordena a execução das ferramentas, iniciando o processo de conversão.

A lógica de encadeamento das ferramentas e do caminho que cada ficheiro do Digital Cinema Package é exibida na figura 6.3.

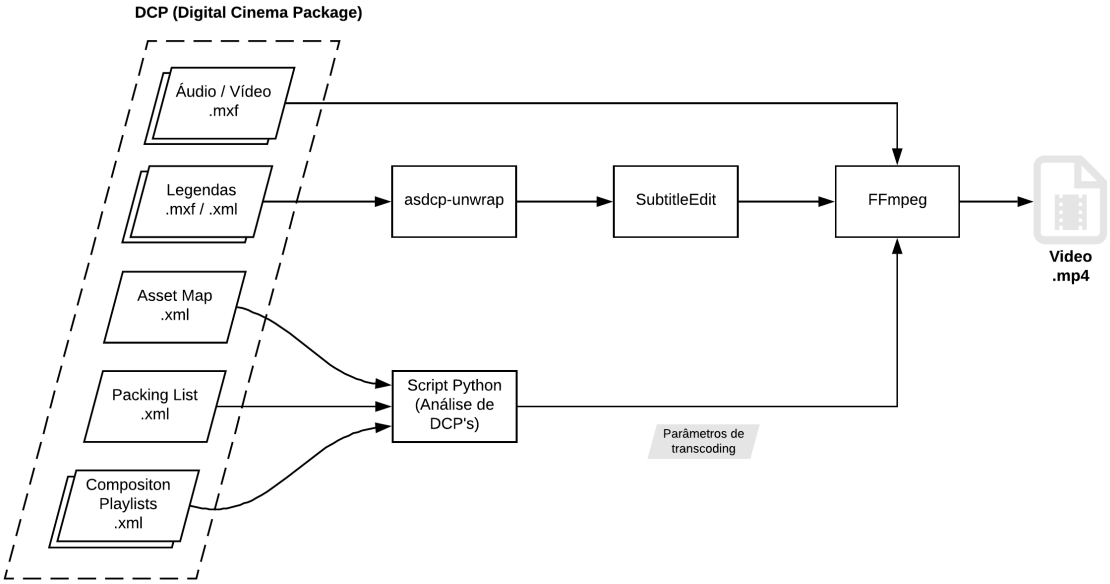

Figura 6.3: Lógica conceptual do processo de análise e conversão de um DCP com as ferramentas *open-source* utilizadas

Claro está que caso o ficheiro de legendas do DCP a processar já se encontre num container XML (caso dos DCP's do tipo Interop), o bloco 'asdcp-unwrap' não é necessário.

6.7 Interface com o utilizador

De modo a tornar a utilização do sistema mais *user-friendly* foi seguidamente desenvolvida uma página *web*, para servir de interface com o utilizador.

No *script* primeiramente desenvolvido eram geradas automaticamente as duas versões possíveis — a versão com resolução 720p para o Plano Nacional de Cinema, e a versão *proxy* 576p para consulta interna — com a mesma legenda.

Nesta fase foi dada ao utilizador a possibilidade de escolher que versões gerar, permitindo também a seleção de legendas distintas para cada versão.

O *front-end* deste protótipo de interface foi desenvolvido utilizando as ferramentas de design de páginas web HTML5, CSS, JavaScript e JQuery. O *back-end* foi implementando recorrendo à micro-framework Flask, que faz a gestão dos pedidos HTTP feitos na página Web. A mensagens entre *front-end* e *back-end* são trocadas no formato JSON (JavaScript Object Notation).

Na página é a princípio exibido ao utilizador uma secção para a seleção/upload da pasta do DCP que deseja converter.

Analogamente ao processo na interface via linha de comandos, é apresentado um menu, exibido na figura 8.1, que permite ao utilizador seleccionar que versões do filme deseja gerar, e com que legendas. Definidas as opções, e clicando no botão 'Iniciar encoding', o processo de transcodificação é despoletado no *back-end*.

A página foi desenvolvida recorrendo à tecnologia AJAX (Asynchronous Javascript and XML), pelo que nova informação é exibida na página à medida que o utilizador executa os vários pedidos sem haver a necessidade de recarregar a página HTML por completo.

O código do programa de análise de DCPs desenvolvido em Python pode ser consultado no Anexo A.

O código HTML da página web e o código Python do back-end Flask podem ser igualmente consultados no Anexo B.

Capítulo 7

Implementação no motor de transcodificação do sistema de *ingest* da MOG Technologies

Esta capítulo descreve a implementação no sistema profissional de *ingest* da MOG Technologies da transcodificação das essências de áudio, vídeo e legendas que figuram num Digital Cinema Package

Durante o desenvolvimento do protótipo, documentado no capítulo 6, o trabalho realizado abrangeu as três componentes-chave da plataforma: interface com o utilizador, análise dos ficheiros estruturais de um Digital Cinema Package, e o *transcoding* dos ficheiros audiovisuais.

Nesta fase o trabalho incidiu mais concretamente na última componente, do *transcoding* dos ficheiros de *media*, preocupando-se em dotar o motor de transcodificação do *software* de *ingest* da empresa, o mxfSPEEDRAIL, das valências necessárias para o suporte e processamento dos tipos de ficheiros multimédia que um Digital Cinema Package compreende — vídeo JPEG 2000 num container MXF, áudio PCM num container MXF, e legendas de cinema digital SMPTE ou CineCanvas num container XML/MXF — e a sua posterior transformação num ficheiro MP4, de acordo com os parâmetros definidos nos requisitos do projeto que abrange esta dissertação (especificados na secção 4.2).

Na figura 7.1 é exibida a lógica conceptual do processo de *transcoding* dos ficheiros audiovisuais de um DCP no motor de transcodificação do mxfSPEEDRAIL.

Alguns dos blocos da figura já estavam devidamente preparados para realizar as operações requeridas, no entanto alguns estavam apenas parcialmente implementados, ou não implementados de todo.

Segue-se uma especificação do nível de conformidade de cada bloco para o processamento de ficheiros audiovisuais de Digital Cinema Packages.

Blocos completamente funcionais :

- Video Decoder;

- Video Encoder;
- Audio Encoder;
- MP4 Wrapper.

Blocos parcialmente funcionais:

- MXF Demuxer;
- Video Resizer;
- Timecode Renderer;
- Audio Mixer.

Blocos não implementados:

- Subtitles Decoder;
- Subtitles Renderer;
- Watermark Renderer.

As secções que se seguem detalham o processo de implementação dos blocos por implementar e o processo de reajuste dos blocos parcialmente funcionais.

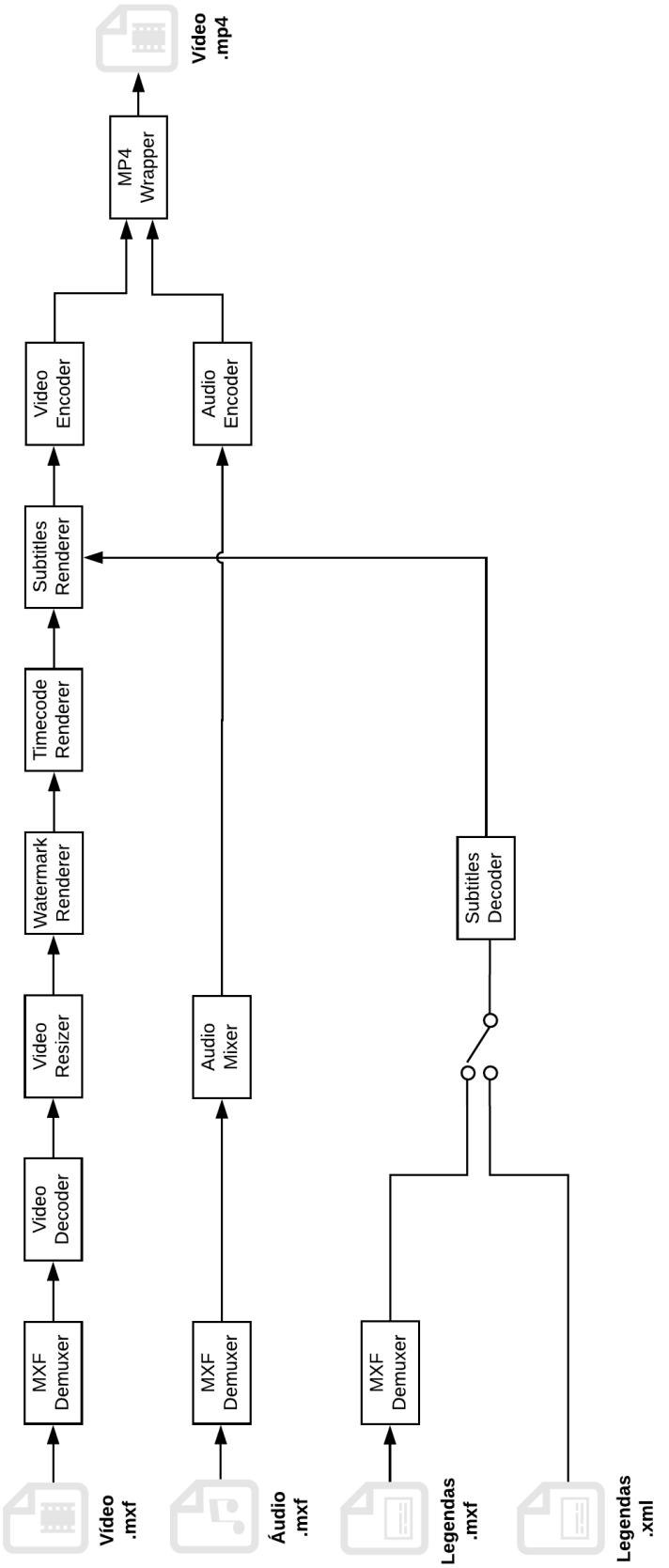

Figura 7.1: Modelo conceptual do processo de *transcoding* de um DCP no motor de transcodificação de *media* do mxfSPEEDRAIL

7.1 Ficheiro de configuração da operação de transcodificação

As configurações para o processo de *transcoding* a decorrer no motor de transcodificação são indicadas num ficheiro XML desenhado pela MOG Technologies, denominado de Common Object Descriptor (COD).

Este ficheiro pode ser visto como o equivalente à passagem de parâmetros que é feita para o FFmpeg nos vários comandos que figuram do capítulo 6.

Eis as informações mais relevantes para o processo que o COD aporta:

- Codificador de vídeo/áudio a utilizar e respetivos perfil, resolução e *bitrate*;
- Instrução para *timecode burnin*;
- Mapeamento dos canais de áudio;
- Ficheiros de áudio/vídeo de entrada e respetivos *entry points*;
- Pasta de saída.

O COD é gerado de uma forma automática nas componentes de *middleware* e de interface gráfica do mxfSPEEDRAIL. Na interface o utilizador seleciona os ficheiros de deseja *processar* e define os parâmetros de *encoding*. A componente de *middleware* recebe esses dados e gera o ficheiro COD, que é depois lido pelo motor de transcodificação.

No decorrer do trabalho realizado no mxfSPEEDRAIL no contexto desta dissertação o COD foi utilizado de forma estática, sendo os parâmetros acima referidos alterados manualmente, dado dos objetivos propostos figurarem apenas intervenções ao nível do motor de transcodificação.

Futuramente as informações necessárias para a realização da operação de sobreposição de legendas e de uma marca-d'água serão igualmente introduzidas no COD pela componente de *middleware*.

Eis alguns dos campos que terão de ser passados:

- Instrução para *burn in* de legendas;
- *Path* do ficheiro de legendas;
- Instrução para o *burn in* de marca-d'água;
- *Path* para o ficheiro de imagem de marca-d'agua.

O código-fonte do motor de *transcoding* é significativamente complexo, no entanto a documentação acerca deste e do ficheiro COD é parca, pelo que durante esta fase foi dispendido bastante tempo em tarefas de *debugging* no código, de modo a compreender como funcionam os vários elementos do sistema.

7.2 Desencapsulamento de vídeo JPEG 2000 e áudio PCM

A primeira retificação efetuada no motor de transcodificação prendeu-se com o *unwrapping* de essências de vídeo codificado no formato JPEG 2000 e de áudio no formato PCM, encapsuladas em ficheiros MXF.

Apesar do *software* estar apto para a leitura de ficheiros MXF na sua entrada, e de conseguir realizar o desencapsulamento de vídeo codificado em formatos profissionais de elevada qualidade como os formatos DNxHD ou ProRes, entre outros, não era ainda capaz de identificar o formato JPEG2000 como um dos formatos passíveis de estarem encapsulados.

Foi necessário fazer o mapeamento entre os identificadores únicos de conteúdo de 16 *bytes* definidos pela SMPTE (*Universal Labels for Unique Identification of Digital Data* [31]) para vídeo JPEG 2000 e áudio PCM e os códigos identificadores dos ficheiros recebidos pelo sistema.

Tabela 7.1: *Universal Labels* de ficheiros MXF com essências de vídeo JPEG 2000 e áudio PCM [8]

Universal Label	***Definition***
060e2b34.04010107.0d010301.020c0000	*Identifiers for MXF-GC* *Mappings using JPEG2000 compressed pictures*
060e2b34.04010101.0d010301.02060100	*Identifier for MXF-GC,* *Frame-wrapped Broadcast Wave audio data*

Adicionalmente, foi também necessário etiquetar de forma interna as características de espaço de cor e de chroma sub-sampling do vídeo JPEG 2000 a processar, XYZ e o 4:4:4 respetivamente.

7.3 *Downmixing* do áudio

Para os testes ao processo de transcodificação do sistema partiu-se de um ficheiro *standard* do COD para a obtenção de um ficheiro MP4 com vídeo H.264/AVC e áudio AAC, de forma a tentar compreender a resposta do sistema aos vários parâmetros do ficheiro de configuração.

À partida, na conversão da imagem do formato JPEG 2000 para o formato H.264/AVC não foram identificados problemas. No que ao aúdio diz respeito, no entanto, e apesar de à saída ser conseguida a obtenção de áudio codificado no requerido formato AAC, não foi obtido o número de canais desejado. Os 6 canais de áudio da faixa PCM original foram convertidos para 6 faixas distintas, com 1 canal cada, sendo o desejado a obtenção de uma única faixa de áudio com 2 canais.

Foram feitos testes com múltiplas combinações entre as configurações de áudio no COD e certos parâmetros no código C++ do motor de transcodificação, que resultaram nos seguintes *outputs*:

- 6 faixas de áudio com 1 canal cada;

- 3 faixas de áudio com 2 canais cada;
- 2 faixas de áudio com 2 canais cada;
- 2 faixas de áudio com 1 canal cada.

Em nenhum cenário foi conseguida a obtenção de uma faixa de áudio com 2 canais. De facto, verificou-se que o sistema descartava sempre os canais de áudio para além dos dois primeiros, pelo que tiveram de ser efetuadas alterações ao código dos filtros de áudio do sistema e ao código do *parser* do COD, nomeadamente ao nível das *flags* que identificam os esquemas de canais de áudio (*sterero*, *5.1 surround*, etc) e dos parâmetros de entrada e saída dos filtro de mistura de áudio.

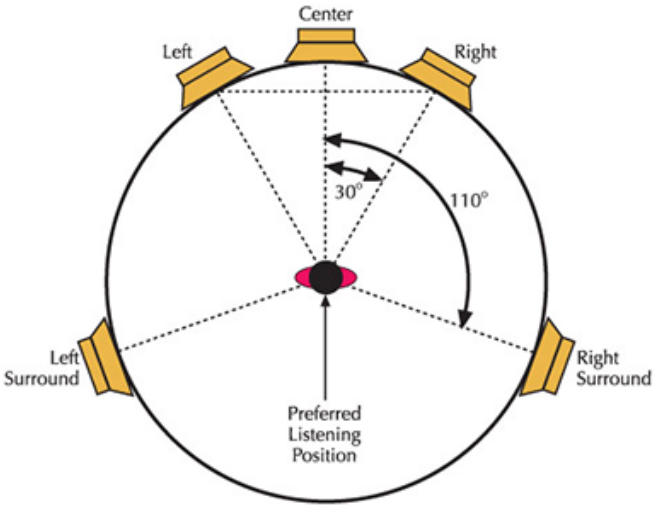

Figura 7.2: Diposição dos 5 canais de áudio do formato *5.1 surround* (o sexto canal é o de *low frequency effects*) [5]

Os canais de áudio originalmente no formato *surround 5.1*, exibidos na figura 7.2, são convertidos para 2 canais (*stereo*) segundo a equação 7.1, definida pelo *standard* de compressão de áudio da Advanced Television Systems Committee (ATSC) [32], em que L é o canal Left, R é o canal Right, C é o canal Center, Ls é o canal Left Surround, Rs é o canal Right Surround, e Lo e Ro são os canais esquerdo e direito da saída *stereo*, respetivamente.

$$\begin{aligned} Lo &= 1.0 \times L + 0.707 \times C + 0.707 \times Ls \\ Ro &= 1.0 \times R + 0.707 \times C + 0.707 \times Rs \end{aligned} \tag{7.1}$$

A verificação da correta mistura dos canais por parte do motor de transcodificação da empresa foi realizada utilizando as ferramentas FFmpeg e Audacity, de acordo com a seguinte metodologia:

- Geração de um ficheiro WAV (formato aceite pelo Audacity) para cada canal de áudio do ficheiro PCM original do DCP (6 ficheiros gerados portanto);

- Isolamento da faixa de áudio do ficheiro de saída do sistema de transcodificação, e posterior conversão para 2 ficheiro WAV (um ficheiro por cada canal *stereo*);

- Importação dos ficheiros WAV para o programa Audacity e respetiva verificação visual da presença dos 6 canais originais nas forma de onda dos 2 canais de saída.

Eventualmente também foi identificada uma anomalia no programa de reprodução de *media* que estava a ser utilizado para reprodução dos ficheiros de entrada e saída (VLC), que não estava a realizar correctamente o *mixing* dos 6 canais de áudio originais para os auscultadores *stereo* utilizados.

Recorreu-se a partir desse ponto aos reprodutores ffplay e MPC-HC, que possibilita a visualização gráfica dos níveis de cada canal de áudio durante a reprodução dos ficheiros.

Esta correção no código demorou bastante tempo a ser resolvida, dada também a já referida falta de documentação sobre o código e o COD, pelo que o tempo despendido, que não se contava que fosse tão elevado, acabou por ter repercussões na realização das implementações e dos reajustes subsequentes.

7.4 Aplicação do *timecode* na imagem

O bloco de *timecode rendering* nas frames do vídeo estava já implementado no motor de transcodificação da empresa. Contudo, os ficheiros de saída obtidos após a ativação da instrução de *timecode burn in* no ficheiro de configuração da transcodificação apresentavam uma distorção na imagem (exibida na figura 7.3).

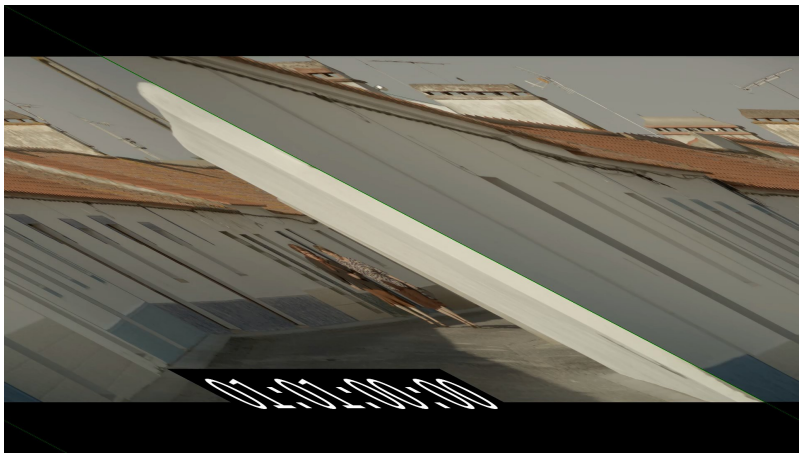

Figura 7.3: Distorção na imagem após aplicação do *timecode*

Foi identificado um sintoma do problema numa secção do código em que os valores do número de pixels por linha horizontal da imagem não estavam a ser corretamente calculados. Contudo a identificação da raíz do problema não foi conseguida. A correta formação das frames de saída

foi obtida com a mudança do *encoder* de H.264 a utilizar pelo motor de transcodificação, de MainConcept (o *encoder* por defeito) para o *encoder* x264.

A correta aplicação do *timecode* é apresentada nas imagens do capitulo de validação dos resultados obtidos (capítulo 8).

7.5 Ajuste da resolução e *aspect ratio* da imagem de saída

Foi detetada também uma incongruência nas dimensões da imagem de saída do processo. Definiu-se no ficheiro de configuração o formato de resolução saída para 720p. Contudo, o motor de transcodificação estava erradamente a produzir imagens 1280 pixels de largura e 720 pixels de altura, independentemente do *aspect ratio* da imagem original, forçando portanto o *aspect ratio* para 16:9.

Conteudos cinematográficos têm usualmente *aspect ratios* distintos do 16:9, que é a norma para as produções de televisão. Entre os *aspect ratios* mais comuns nas produções de cinema figuram o 1.85:1, o 2.35:1 e o 2.40:1. É por isso importantíssimo realizar corretamente o *downscaling* do vídeo original sem induzir distorções nas imagens do ficheiro final.

Para solucionar o problema foi acrescentado à componente do código-fonte responsável pela codificação do vídeo H.264 (agora utilizando o *encoder* x264, conforme explanado na secção anterior) um excerto de código que verifica o *aspect ratio* do vídeo do ficheiro de entrada, e, para o caso de ser diferente de 16:9, reajusta a dimensão da altura, de acordo com a tabela 4.1, de forma a respeitar as proporções das dimensões da imagem original.

(a) Vídeo de saída com resolução 1280 x 720 (*aspect ratio* forçado para 16:9)

(b) Vídeo de saída com resolução corrigida 1280 x 692 (o *aspect ratio* original 1.85:1 é respeitado)

Figura 7.4: Vídeos de saída com diferentes resoluções e *aspect ratios*

7.6 Leitura do ficheiro de legendas e posterior aplicação no vídeo

Os módulos de interpretação dos ficheiros de legendas de um DCP e da sua posterior renderização nas *frames* do vídeo estavam completamente por implementar no sistema de transcodificação da empresa.

Dado que o módulo de aplicação do *timecode* já se encontrava implementado e funcional, e como ambos os processos apresentam algumas semelhanças entre si, decidiu-se partir deste

bloco para implementar o processo de sobreposição das legendas. O Timecode Renderer começa por imprimir um texto de notação temporal (por exemplo '00:00:00:00'), e incrementa os valores apresentados a cada *frame* que decorre. O texto a exibir é lido uma única vez, no ínicio do processo (*timecode* de arranque). Com o Subtitle Renderer pretende-se igualmente imprimir caracteres no vídeo, mas com um determinado conteúdo textual dinãmico (tem de ser lido múltiplas vezes ao longo do tempo), e que entra e sai de cena em certos instantes, conforme definido no ficheiro de legendas do DCP.

Para a operação de leitura do ficheiro de legendas de um DCP, integrou-se no código do motor de transcodificação um trecho de código que faz o *parsing* do ficheiro XML de legendas (assumindo que o ficheiro MXF de legendas, se o DCP é do tipo Interop, é previamente desencapsulado) e extrai as informações mais relevantes para a operação de sobreposição das legendas no vídeo:

- Conteúdo textual da legenda;
- Instante de tempo em que a legenda deve surgir no ecrã;
- Instante de tempo em que a legenda deve desaparecer do ecrã.

Tal como mencionado no capítulo 3 de estudo de um Digital Cinema Package, os ficheiros de legendas dos DCP's do tipo Interop e do tipo SMPTE apresentam algumas diferenças entre si.

Apesar de em termos da estrutura hierárquica dos documentos XML não haver diferenças, existe uma diferença importante ao nível das notações temporais utilizadas para assinalar os instantes de tempo de entrada e saída das linhas de legendas.

Nos ficheiros de legendas de um DCP SMPTE (especificados no standard SMPTE ST 428-7:2010 [28]) as notações temporais seguem o seguinte formato:

- **HH:MM:SS:EE —** Horas, minutos, segundos e *editable units*, respetivamente. Uma *editable unit* corresponde a uma *frame*.

Nos ficheiros de legendas de um DCP Interop (que seguem o formato CineCanvas [27]) as notações temporais podem ser expressas de duas formas possíveis:

- **HH:MM:SS:TTT —** Horas, minutos, segundos e *ticks*, respetivamente. Um *tick* corresponde a 4 milissegundos (ms), pelo que este campo tem uma amplitude de valores entre 0 e 249;
- **HH:MM:SS.sss —** Horas, minutos, segundos e décimas de segundo, respetivamente.

O objetivo passou por converter cada notação temporal lida para o correspondente valor em *frames*. É conveniente operar numa base de *frames* porque:

- Surgirão à entrada do sistema ficheiros de vídeo com diferentes *frame rates*;
- O módulo Timecode Render, que serviu de base ao Subtitle Renderer, também opera à base de *frames*;

- Os *entry points* de áudio e vídeo nas Composition Lists de um DCP são igualmente anotados em *frames*.

A lógica por detrás do processo de leitura e sobreposição de legendas implementado é a que se segue: no início do processo o programa lê os atributos TimeIn e TimeOut e o conteúdo textual do(s) elemento(s) Text do primeiro elemento Subtitle do ficheiro XML de legendas, e converte as notações temporais lidas para o respetivo número em *frames*. De seguida, recebe a uma *frame* do vídeo, e compara o nº da *frame* recebida aos números das frames do elemento de legendas lido:

- Se o nº da *frame* recebida for menor que o nº da frame do atributo TimeIn do elemento de legendas lido, a *frame* é passada para o bloco seguinte do motor de transcodificação sem ser alterada;
- Se o nº da *frame* recebida for maior que o nº da *frame* do atributo TimeIn e menor que o nº da *frame* do atributo TimeOut, sobrepõe na *frame* o conteúdo textual lido;
- Se o nº da *frame* recebida for maior que o nº da *frame* do atributo TimeOut, o programa lê os dados do elemento Subtitle seguinte do documento e repete a verificação descrita, até chegar ao final do ficheiro.

Os termos atributos TimeIn e TimeOut dos elementos Subtitle constam dos excertos de exemplos dos dois tipos de ficheiros de legendas apresentados nos blocos de código que se seguem.

De notar que cada elemento Subtitle pode conter um ou mais elementos Text, correspondentes a múltiplas linhas de texto.

Cada elemento Subtitle contém um ou mais elementos Text, sendo que cada um destes elementos corresponde a uma linha de texto. O conteúdo textual está guardado no valor.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <DCSubtitle Version="1.0">
3    <SubtitleID>3f11260b-f455-4c57-9f2e-04a12cbe929d</SubtitleID>
4    <MovieTitle>FILME X</MovieTitle>
5    <ReelNumber>1</ReelNumber>
6    <Language>es</Language>
7    <LoadFont URI="arial.ttf" Id="arial"/> <Font Id="arial" Color="FFFFFFFF" Effect="
         border" EffectColor="FF000000" Italic="no" Underlined="no" Script="normal"
         Size="42">
8
9    <!-- ... -->
10
11     <Subtitle SpotNumber="13" FadeUpTime="20" FadeDownTime="20" TimeIn="
           00:04:50:239" TimeOut="00:04:53:190">
12       <Text VPosition="14" VAlign="bottom" HAlign="center" Direction="horizontal">
             Pues que se queden ahi,</Text>
13     </Subtitle>
14     <Subtitle SpotNumber="14" FadeUpTime="20" FadeDownTime="20" TimeIn="
           00:04:54:073" TimeOut="00:04:57:107">
```

```
15        <Text VPosition="20" VAlign="bottom" HAlign="center" Direction="horizontal">
              que de mi</Text>
16        <Text VPosition="14" VAlign="bottom" HAlign="center" Direction="horizontal">
              casa no me saca nadie.</Text>
17      </Subtitle>
18
19      <!-- ... -->
```

Listing 7.1: Exemplo de um ficheiro de legendas (excerto) de um DCP do tipo Interop

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <dcst:SubtitleReel xmlns:dcst="http://www.smpte-ra.org/schemas/428-7/2010/DCST"
       xmlns:xs="http://www.w3.org/2001/XMLSchema">
3    <dcst:Id>urn:uuid:9eafaaa8-71c0-4c53-b2b8-271390da3d7b</dcst:Id>
4    <dcst:ContentTitleText>FILME X</dcst:ContentTitleText>
5    <dcst:AnnotationText>This is a subtitle file</dcst:AnnotationText>
6    <dcst:IssueDate>2015-04-10T11:31:19.000-00:00</dcst:IssueDate>
7    <dcst:ReelNumber>1</dcst:ReelNumber>
8    <dcst:Language>fr</dcst:Language>
9    <dcst:EditRate>25 1</dcst:EditRate>
10   <dcst:TimeCodeRate>25</dcst:TimeCodeRate>
11   <dcst:StartTime>00:00:00:00</dcst:StartTime>
12   <dcst:LoadFont ID="theFontId">urn:uuid:c5a3d9cd-60c9-4317-90f8-3873052ce80b</dcst
         :LoadFont>
13   <dcst:SubtitleList>
14     <dcst:Font ID="theFontId" Size="39" Weight="normal" Color="FFFFFFFF" Effect="
           border" EffectColor="FF000000">
15       <dcst:Subtitle SpotNumber="1" FadeUpTime="00:00:00:00" FadeDownTime="
             00:00:00:00" TimeIn="00:00:04:02" TimeOut="00:00:06:20">
16         <dcst:Text Vposition="14" Valign="bottom" Halign="center" Direction="ltr">"
               Cette nature intrigante, travaillant</dcst:Text>
17         <dcst:Text Vposition="9" Valign="bottom" Halign="center" Direction="ltr">
               sans aucun sens apparent,</dcst:Text>
18       </dcst:Subtitle>
19       <dcst:Subtitle SpotNumber="2" FadeUpTime="00:00:00:00" FadeDownTime="
             00:00:00:00" TimeIn="00:00:06:23" TimeOut="00:00:09:09">
20         <dcst:Text Vposition="14" Valign="bottom" Halign="center" Direction="ltr">
               comme le vent,  d apr ès</dcst:Text>
21         <dcst:Text Vposition="9" Valign="bottom" Halign="center" Direction="ltr">
               les étranges ordres lointains,</dcst:Text>
22       </dcst:Subtitle>
23
24     <!-- ... -->
```

Listing 7.2: Exemplo de um ficheiro de legendas (excerto) de um DCP do tipo SMPTE

Capítulo 8

Resultados

Neste capítulo são exibidas as características técnicas (obtidas com o programa MediaInfo) e capturas de imagem dos vídeos gerados pela protótipo da plataforma desenvolvida e pelo intervencionado sistema de transcodificação da MOG Technologies. Para todos os exemplos de ficheiros de saída exibidos foi utilizado como entrada o Digital Cinema Package do Filme A, detalhado no capítulo 5. É igualmente exibida a página principal da interface gráfica desenvolvida para a interação com a plataforma.

Por fim é feita uma verificação do cumprimento dos requisitos para a plataforma, definidos na secção 4.2 do capítulo 4.

8.1 Interface gráfica do protótipo da plataforma desenvolvido

A figura 8.1 exibe o menu principal da interface gráfica, que funciona através de um navegador *web*, desenvolvida para o protótipo da plataforma, e documentada no capítulo 6. Após selecionar a pasta do Digital Cinema Package a ser processado (numa página precedente), o utilizador pode eleger as versões do filme que deseja gerar, e optar por adicionar legendas (nos idiomas disponíveis) para cada versão.

Figura 8.1: Interface da plataforma de *ingest*

O código HTML da página web e o código Python do back-end podem ser igualmente consultados no Anexo B.

8.2 Vídeo de saída do protótipo desenvolvido

A figura 8.2 exibe uma captura de imagem do vídeo de saída do protótipo desenvolvido para a plataforma de *ingest*, da versão gerada para o Plano Nacional de Cinema, com legendas embutidas e resolução no formato 720p. As características técnicas desta versão de saída são listadas no bloco de texto que se segue à imagem.

Figura 8.2: Captura de imagem do vídeo de saída do protótipo desenvolvido – versão para o Plano Nacional de Cinema, com legendas e resolução 720p.

```
General
Complete name                            : D:\output720.mp4
Format                                   : MPEG-4
Format profile                           : Base Media
Codec ID                                 : isom (isom/iso2/avc1/mp41)
File size                                : 34.5 MiB
Duration                                 : 2 min 25 s
Overall bit rate                         : 1 993 kb/s
Writing application                      : Lavf57.83.100

Video
ID                                       : 1
Format                                   : AVC
Format/Info                              : Advanced Video Codec
Format profile                           : Baseline@L3.1
Format settings                          : 1 Ref Frames
Format settings, CABAC                   : No
Format settings, RefFrames               : 1 frame
Codec ID                                 : avc1
Codec ID/Info                            : Advanced Video Coding
```

```
Duration                                 : 2 min 25 s
Bit rate                                 : 1 648 kb/s
Width                                    : 1 280 pixels
Height                                   : 692 pixels
Display aspect ratio                     : 1.85:1
Frame rate mode                          : Constant
Frame rate                               : 25.000 FPS
Color space                              : YUV
Chroma subsampling                       : 4:2:0
Bit depth                                : 8 bits
Scan type                                : Progressive
Bits/(Pixel*Frame)                       : 0.074
Stream size                              : 28.5 MiB (83%)
Writing library                          : x264 core 152 r2851 ba24899
Encoding settings                        : cabac=0 / ref=1 / deblock=0:0:0 /
    analyse=0:0 / me=dia / subme=0 / psy=1 / psy_rd=1.00:0.00 / mixed_ref=0 /
    me_range=16 / chroma_me=1 / trellis=0 / 8x8dct=0 / cqm=0 / deadzone=21,11 /
    fast_pskip=1 / chroma_qp_offset=0 / threads=12 / lookahead_threads=2 /
    sliced_threads=0 / nr=0 / decimate=1 / interlaced=0 / bluray_compat=0 /
    constrained_intra=0 / bframes=0 / weightp=0 / keyint=250 / keyint_min=25 /
    scenecut=0 / intra_refresh=0 / rc=crf / mbtree=0 / crf=21.0 / qcomp=0.60 /
    qpmin=0 / qpmax=69 / qpstep=4 / ip_ratio=1.40 / aq=0

Audio
ID                                       : 2
Format                                   : AAC
Format/Info                              : Advanced Audio Codec
Format profile                           : LC
Codec ID                                 : mp4a-40-2
Duration                                 : 2 min 24 s
Bit rate mode                            : Constant
Bit rate                                 : 341 kb/s
Channel(s)                               : 2 channels
Channel(s)_Original                      : 6 channels
Channel positions                        : Front: L C R, Side: L R, LFE
Sampling rate                            : 48.0 kHz
Frame rate                               : 46.875 FPS (1024 SPF)
Compression mode                         : Lossy
Stream size                              : 5.89 MiB (17%)
Default                                  : Yes
Alternate group                          : 1
```

Listing 8.1: Características técnicas do vídeo de saída do protótipo desenvolvido – versão para o Plano Nacional de Cinema

A figura 8.3 exibe uma captura de imagem do vídeo de saída do protótipo desenvolvido para a plataforma de *ingest*, neste caso da versão para visualização interna, com legendas, *timecode* e marca-d'água embutidos, e resolução no formato 576p. As características técnicas são igualmente

listadas de seguida.

Figura 8.3: Captura de imagem do vídeo de saída do protótipo desenvolvido – versão *proxy* para visualização interna, com legendas, *timecode*, marca-d'água, e resolução 576p.

```
General
Complete name                            : D:\output576.mp4
Format                                   : MPEG-4
Format profile                           : Base Media
Codec ID                                 : isom (isom/iso2/avc1/mp41)
File size                                : 27.1 MiB
Duration                                 : 2 min 25 s
Overall bit rate                         : 1 564 kb/s
Writing application                      : Lavf57.83.100

Video
ID                                       : 1
Format                                   : AVC
Format/Info                              : Advanced Video Codec
Format profile                           : Baseline@L3.1
Format settings                          : 1 Ref Frames
Format settings, CABAC                   : No
Format settings, RefFrames               : 1 frame
Codec ID                                 : avc1
Codec ID/Info                            : Advanced Video Coding
Duration                                 : 2 min 25 s
Bit rate                                 : 1 219 kb/s
Width                                    : 1 066 pixels
Height                                   : 576 pixels
Display aspect ratio                     : 1.85:1
Original display aspect ratio            : 1.85:1
```

```
Frame rate mode                          : Constant
Frame rate                               : 25.000 FPS
Color space                              : YUV
Chroma subsampling                       : 4:2:0
Bit depth                                : 8 bits
Scan type                                : Progressive
Bits/(Pixel*Frame)                       : 0.079
Stream size                              : 21.1 MiB (78%)
Writing library                          : x264 core 152 r2851 ba24899
Encoding settings                        : cabac=0 / ref=1 / deblock=0:0:0 /
    analyse=0:0 / me=dia / subme=0 / psy=1 / psy_rd=1.00:0.00 / mixed_ref=0 /
    me_range=16 / chroma_me=1 / trellis=0 / 8x8dct=0 / cqm=0 / deadzone=21,11 /
    fast_pskip=1 / chroma_qp_offset=0 / threads=12 / lookahead_threads=2 /
    sliced_threads=0 / nr=0 / decimate=1 / interlaced=0 / bluray_compat=0 /
    constrained_intra=0 / bframes=0 / weightp=0 / keyint=250 / keyint_min=25 /
    scenecut=0 / intra_refresh=0 / rc=crf / mbtree=0 / crf=21.0 / qcomp=0.60 /
    qpmin=0 / qpmax=69 / qpstep=4 / ip_ratio=1.40 / aq=0

Audio
ID                                       : 2
Format                                   : AAC
Format/Info                              : Advanced Audio Codec
Format profile                           : LC
Codec ID                                 : mp4a-40-2
Duration                                 : 2 min 24 s
Bit rate mode                            : Constant
Bit rate                                 : 341 kb/s
Channel(s)                               : 2 channels
Channel(s)_Original                      : 6 channels
Channel positions                        : Front: L C R, Side: L R, LFE
Sampling rate                            : 48.0 kHz
Frame rate                               : 46.875 FPS (1024 SPF)
Compression mode                         : Lossy
Stream size                              : 5.89 MiB (22%)
Default                                  : Yes
Alternate group                          : 1
```

Listing 8.2: Características técnicas do vídeo de saída do protótipo desenvolvido – versão para visualização interna

8.3 Vídeo de saída do motor de transcodificação da empresa

As figuras 8.4 e 8.5 exibem capturas de imagem do vídeo gerado pelo motor de transcodificação da empresa, após as alterações efetuadas ao código. São exibidas as aplicações de legendas e de *timecode*, neste caso em vídeo de resolução 720p. As especificações técnicas do vídeo gerado são listadas após as imagens.

Apesar da aplicação da marca-d'água no vídeo ter sido devidamente executada na fase de implementação do protótipo, descrita no capítulo 6, na fase de implementação dos processos no motor de transcodificação da empresa não foi possível fazê-lo a tempo da lógica do processo figurar neste documento.

Figura 8.4: Captura de imagem do vídeo de saída do sistema de transcodificação da empresa – versão para o Plano Nacional de Cinema, com legendas e resolução 720p.

Figura 8.5: Captura de imagem do vídeo de saída do sistema de transcodificação da empresa – versão com *timecode* e resolução 720p.

```
Format profile                           : Base Media
Codec ID                                 : isom (isom/iso2/avc1/mp41)
File size                                : 376 MiB
Duration                                 : 10 min 0 s
Overall bit rate                         : 5 252 kb/s
Writing application                      : Lavf57.71.100

Video
ID                                       : 1
Format                                   : AVC
Format/Info                              : Advanced Video Codec
Format profile                           : High@L3.1
Format settings                          : CABAC / 4 Ref Frames
Format settings, CABAC                   : Yes
Format settings, RefFrames               : 4 frames
Format settings, GOP                     : M=3, N=12
Codec ID                                 : avc1
Codec ID/Info                            : Advanced Video Coding
Duration                                 : 10 min 0 s
Bit rate                                 : 5 000 kb/s
Width                                    : 1 280 pixels
Height                                   : 692 pixels
Display aspect ratio                     : 1.85:1
Frame rate mode                          : Constant
Frame rate                               : 25.000 FPS
Color space                              : YUV
Chroma subsampling                       : 4:2:0
Bit depth                                : 8 bits
Scan type                                : Progressive
Bits/(Pixel*Frame)                       : 0.226
Stream size                              : 361 MiB (96%)
Writing library                          : x264 core 152 r2851 ba24899
Encoding settings                        : cabac=1 / ref=3 / deblock=1:0:0 /
    analyse=0x3:0x113 / me=hex / subme=6 / psy=1 / psy_rd=1.00:0.00 / mixed_ref=1 /
     me_range=16 / chroma_me=1 / trellis=1 / 8x8dct=1 / cqm=0 / deadzone=21,11 /
    fast_pskip=1 / chroma_qp_offset=-2 / threads=8 / lookahead_threads=1 /
    sliced_threads=0 / nr=0 / decimate=1 / interlaced=0 / bluray_compat=0 /
    constrained_intra=0 / bframes=2 / b_pyramid=2 / b_adapt=1 / b_bias=0 / direct=1
     / weightb=1 / open_gop=0 / weightp=1 / keyint=12 / keyint_min=7 / scenecut=40
    / intra_refresh=0 / rc_lookahead=12 / rc=abr / mbtree=1 / bitrate=5000 /
    ratetol=1.0 / qcomp=0.60 / qpmin=0 / qpmax=69 / qpstep=4 / ip_ratio=1.40 / aq
    =1:1.00

Audio
ID                                       : 2
Format                                   : AAC
Format/Info                              : Advanced Audio Codec
Format profile                           : LC
Codec ID                                 : mp4a-40-2
```

```
Duration                                 : 10 min 0 s
Bit rate mode                            : Constant
Bit rate                                 : 192 kb/s
Channel(s)                               : 2 channels
Channel positions                        : Front: L R
Sampling rate                            : 48.0 kHz
Frame rate                               : 46.875 FPS (1024 SPF)
Compression mode                         : Lossy
Stream size                              : 13.8 MiB (4%)
Default                                  : Yes
Alternate group                          : 1
```

Listing 8.3: Características técnicas do vídeo de saída do sistema de transcodificação da empresa

8.4 Verificação do requisitos

As características técnicas do vídeo listadas na secção precedente comprovam que os requisitos definidos para a plataforma foram, no que ao protótipo desenvolvido diz respeito, verificados na sua totalidade.

A implementação nos sistema de transcodificação da empresa que se seguiu ao positivo desenvolvimento do protótipo não possibilitou ainda, devido a limitações de carácter temporal, o cumprimento integral dos requisitos para os vídeos gerados pelo sistema. A geração de vídeo no formato de resolução 576p carece ainda de testes adicionais, e é necessário implementar o módulo de sobreposição de marca-d'água no vídeo.

Tabela 8.1: Verificação dos requisitos nos vídeos gerados

Requisitos	**Protótipo (via FFmpeg)**	**Motor de ingest da MOG Technologies**
1. O gestor de arquivo da Cinemateca Portuguesa deve poder gerar cópias de um filme em arquivo de resolução HD para o Plano Nacional de Cinema (PNC) e de resolução SD para visualização interna (proxy)	Sim	-
2. O gestor de arquivo da Cinemateca Portuguesa deve poder incluir legendas no vídeo de ambas as versões geráveis, com diferentes idiomas (se existirem) em cada versão	Sim	-
3. A versão para o PNC deve ter uma resolução de 720p	Sim	Sim
4. A versão proxy interna deve ter uma resolução de 576p	Sim	Parcialmente
5. Ambas as versões devem preservar o frame rate do filme original	Sim	Sim
6. As legendas devem ser "burned in"na imagem	Sim	Sim
7. A versão proxy interna deve ter o timecode burned in	Sim	Sim
8. A versão proxy interna deve ter o logótipo da Cinemateca Portuguesa	Sim	Não
9. Ambas as versões devem estar num ficheiro MP4	Sim	Sim
10. O vídeo de ambas as versões deve estar no formato H.264	Sim	Sim
11. O áudio de ambas as versões deve estar no formato AAC	Sim	Sim
12. O áudio de ambas as versões deve estar amostrado a 48 kHz	Sim	Sim
13. O áudio de ambas as versões deve ter um máximo de 2 canais	Sim	Sim

Os requisitos 1 e 2 não se aplicam para o caso do moto de transcodificação da empresa, já que este apenas lida diretamente com os ficheiros audiovisuais.

Capítulo 9

Conclusões e Trabalho Futuro

9.1 Conclusões

Findo o período de realização da dissertação, pode afirmar-se que os resultados obtidos foram positivos.

O estudo inicial acerca dos standards de cinema digital assegurou a correta leitura e análise dos Digital Cinema Packages que a Cinemateca Portuguesa inserirá na plataforma de *ingest*, e a escolha de um formato de vídeo e áudio de saída da plataforma, de acordo com os critérios elencados e as alternativas existentes, garante que os conteúdos cinematográficos convertidos estarão devidamente adaptados para a sua distribuição num ambiente escolar, preservando o valor intrínseco dos conteúdos cinematográficos enquanto obra de arte.

O protótipo desenvolvido cumpriu na totalidade os objetivos definidos, interpretando corretamente os ficheiros auxiliares de *metadata* que compõe um Digital Cinema Package recebidos na sua entrada e transformando devidamente os ficheiros de vídeo, áudio e legendas num ficheiro MP4 com vídeo codificado no formato H.264/AVC e áudio codificado no formato AAC, de acordo com os parâmetros definidos pelo utilizador na interface gráfica.

Face ao resultados obtidos com o protótipo, a empresa decidiu seguir para a implementação no sistema de *ingest* da empresa do processo de transcodificação definido para os vários ficheiros audiovisuais presentes num Digital Cinema Package.

O elevado tempo despendido a reajustar determinados módulos que não estavam completamente funcionais e a implementar outros módulos que ainda não existiam no motor de transcodificação do sistema de *ingest* da empresa fez com que a verificação dos requisitos definidos para a plataforma não fosse cumprida na totalidade (para os vídeos de saída do sistema de *ingest* da empresa) durante o período que circunscreve esta dissertação. Os alicerces estão no entando bem lançados para a curto prazo essa implementação ser realizada integralmente.

9.2 Trabalho Futuro

A análise e transformação dos Digital Cinema Packages realizada teve como referência as características dos DCP's enviados pela Cinemateca Portuguesa, que continham vídeo num único *reel*, conteúdo desencriptado e ficheiros únicos de vídeo e áudio. O standard da DCI acerca dos DCP's prevê que a sua estrutura possa ser mais complexa, pelo que no futuro poderá justificar-se uma melhoria do código desenvolvido nesta dissertação, abarcando funcionalidades como a desencriptação e verificação da integridade dos ficheiros recebidos num DCP, e a leitura de múltiplos *reels* de uma Composition Playlist.

O projeto em que a presente dissertação esteve inserida terá um tempo de implementação de três anos, pelo que mesmo quando todas as funcionalidades estiverem completamente implementadas, apenas quando as várias instituições visadas pela plataforma — escolas, órgãos de tutela do sistema educativo, organismos detentores dos direitos de exploração dos conteúdos audiovisuais e arquivos de cinema — fizerem uma utilização contínua e proveitosa do sistema para a educação e formação do público em ambiente escolar é que o sucesso do projeto poderá ser celebrado.

Anexo A

Script de análise de Digital Cinema Packages

```
1  import sys
2  import os
3  import subprocess
4  import xml.etree.ElementTree as ET
5
6
7  #-------------------------VARRE XMLs------------------------------
8
9
10 # Dada uma pasta, identifica filename da PKL
11 def getPklFullFilename(inFolder, outFolder):
12
13   global pklFullFilename
14   global inputFolder
15   global outputFolder
16   inputFolder = inFolder
17   outputFolder = outFolder
18
19   for filename in os.listdir(inputFolder):
20     if filename.endswith('.xml'):
21       fullFilename = inputFolder+'\\'+filename # Calcula o path completo, pasta +
             ficheiro
22       tree = ET.parse(fullFilename)
23       root = tree.getroot()
24       for elements in root.iter():
25         elements.tag = elements.tag.split("}")[1][0:]
26       if root.tag == 'PackingList':
27         pklFullFilename = fullFilename
28         print pklFullFilename
29
30   return pklFullFilename
```

```
31
32
33 # Dada uma pasta, identifica filenames das CPLs e guarda em listaDeCompositions
34 def getListaDeCompositions(folder):
35
36   global listaDeCompositions
37   listaDeCompositions = []
38
39   for filename in os.listdir(folder):
40     if filename.endswith('.xml'):
41       fullFilename = folder+'\\'+filename # Calcula o path completo, pasta +
             ficheiro
42       tree = ET.parse(fullFilename)
43       root = tree.getroot()
44       for elements in root.iter():
45         elements.tag = elements.tag.split("}")[1][0:]
46       if root.tag == 'CompositionPlaylist':
47         listaDeCompositions.append(fullFilename)
48
49   # Aborta o programa se nao encontrou nenhuma CPL na pasta de input
50   if len(listaDeCompositions) == 0:
51     print "\nNao ha nenhuma CPL nesta pasta. Por favor indique uma pasta correcta."
52     sys.exit()
53
54   return listaDeCompositions
55
56
57 # Ve se ha legendas, em que idiomas
58 def getSubsIdiomas(listaDeCompositions):
59
60   global subsIdiomas
61   subsIdiomas = []
62
63
64   for i in range(len(listaDeCompositions)):
65     tree = ET.parse(listaDeCompositions[i])
66     root = tree.getroot()
67     # Remove o {namespace} de todos os elementos
68     for elements in root.iter():
69       elements.tag = elements.tag.split("}")[1][0:]
70     # Ve se ha legendas e adiciona aos idiomas
71     for elements in root.iter('AssetList'):
72       if elements.find('MainSubtitle') == None:
73         subsIdiomas.append('sem legendas')
74       else:
75         for elements in root.iter('MainSubtitle'):
76           idioma = elements.find('Language').text
77           subsIdiomas.append(idioma)
78
```

```
79    return subsIdiomas
80
81  def iniciarEncoding(selectedPNC, selectedEXT, selectedSubPNC, selectedSubEXT):
82
83    # Gera versao PNC apenas
84
85    if (selectedPNC and not selectedEXT):
86      res = 'a'
87
88      # Ve qual e' o index da sub escolhida na lista
89      pos = subsIdiomas.index(selectedSubPNC)
90
91      cplEscolhida = listaDeCompositions[pos]
92
93      print cplEscolhida
94
95      # #-------------------------LEGENDAS------------------------------
96
97      # Importa a cpl.xml
98
99      tree = ET.parse(cplEscolhida)
100     root = tree.getroot()
101
102     # Remove o {namespace} de todos os elementos
103
104     for elements in root.iter():
105       elements.tag = elements.tag.split("}")[1][0:]
106
107     # Ve se ha legendas na CPL, e obtem o ID
108
109     subtitlesExist = 0
110
111     for element in root.iter('MainSubtitle'):
112       subtitlesExist = 1
113       # Id
114       subsLanguage = element.find('Language').text
115       subsId = element.find('Id').text
116       print '\nId das legendas: ' + subsId
117       # EditRate (frames)
118       subsEditRate = element.find('EditRate').text # Valor tipo '25 1'
119       subsEditRate = subsEditRate.split(" ") # Lista [25 1]
120       subsEditRate = subsEditRate[0] # Fica com o primeiro valor, o desejado
121       subsEditRate = int(subsEditRate)
122       # EntryPoint
123       subsEntryPointFrames = int(element.find('EntryPoint').text)
124       subsEntryPointSeconds = float(subsEntryPointFrames)/subsEditRate
125
126     # Importa a pkl.xml
127
```

```
128     tree = ET.parse(pklFullFilename)
129     root = tree.getroot()
130
131     # Remove o {namespace} de todos os elementos
132
133     for elements in root.iter():
134       elements.tag = elements.tag.split("}")[1][0:]
135
136     # Esta parte apenas se houve legendas
137
138     if subtitlesExist == 1:
139
140       # Descobre o path/filename das legendas
141       for element in root.iter('Asset'):
142         if element.find('Id').text == subsId:
143           # Identifica o Original Filename
144           subsOriginalFilename = element.find('OriginalFileName').text
145           # Calcula o Filename
146           aux = subsOriginalFilename.split('/')
147           subsFilename = aux[len(aux)-1]
148           # Corrige o Original Filename. Muda os / para \
149           subsOriginalFilename = subsOriginalFilename.replace('/','\\')
150           # Path
151           subsPath = inputFolder + '\\' + subsOriginalFilename
152           print '\nPath da legenda identificado: ' + subsPath
153
154
155       # Calcula o filename agnostico (sem .formato)
156       aux = subsFilename.split('.') # Separa o path por '.'
157       subsFilenameAgno = '.'.join(aux[0:(len(aux)-1)])
158       # Identifica o formato das legendas
159       subsFormat = aux[len(aux) - 1] # O formato e o ultimo elemento da lista
160       print '\nFormato das legendas: ' + subsFormat
161
162       # Calcula o path agnostico (sem .formato)
163       aux = subsPath.split('.') # Separa o path por '.'
164       subsPathAgno = '.'.join(aux[0:(len(aux)-1)])
165
166       # Se o formato for MXF, converte primeiro para XML, e depois para SRT
167
168       if subsFormat == 'mxf':
169
170         # Chama o cmd, converte as legendas de MXF para XML
171
172         comando = 'asdcp-unwrap "{}" "{}"'
173
174         comando = comando.format(subsPath, outputFolder + '\\' + subsFilenameAgno +
              '.xml')
175
```

```
176         subprocess.call(comando)
177
178
179         # Chama o cmd, converte as legendas de XML para SRT
180
181         comando = 'SubtitleEdit /convert "{}" srt /overwrite /offset:00:00:00:{}'
182
183         comando = comando.format(outputFolder + '\\' + subsFilenameAgno + '.xml',
                subsEntryPointSeconds)
184
185         subprocess.call(comando)
186
187       # Se o formato for XML, converte logo para SRT
188
189       if subsFormat == 'xml':
190
191         comando = 'SubtitleEdit /convert {} srt /overwrite /outputfolder:{} /offset
              :00:00:00:{}'
192
193         comando = comando.format(subsPath, outputFolder, subsEntryPointSeconds)
194
195         subprocess.call(comando)
196
197       # Adapta o path das legendas SRT para por no ffmpeg
198       # Os \ passam a /, e o : tem de ser escapado
199
200       subsPathOutSrt = outputFolder + '\\' + subsFilenameAgno + '.srt'
201       subsPathOutSrtFfmpeg = subsPathOutSrt.replace('\\','/').replace(':','\\:')
202
203
204     # #--------------------------VIDEO---------------------------------------
205
206
207     # Pega outra vez na CPL
208
209     tree = ET.parse(cplEscolhida)
210     root = tree.getroot()
211
212     # Remove o {namespace} de todos os elementos
213
214     for elements in root.iter():
215       elements.tag = elements.tag.split("}")[1][0:]
216
217     # Info do video na CPL
218
219     for element in root.iter('MainPicture'):
220       # ID
221       videoId = element.find('Id').text
222       print '\n\nId do video: ' + videoId
```

```
223       # Frame Rate
224       FrameRate = element.find('FrameRate').text # Valor tipo '25 1'
225       FrameRate = FrameRate.split(" ") # Lista [25 1]
226       FrameRate = FrameRate[0] # Fica com o primeiro valor, o desejado
227       # EditRate (frames)
228       videoEditRate = element.find('EditRate').text # Valor tipo '25 1'
229       videoEditRate = videoEditRate.split(" ") # Lista [25 1]
230       videoEditRate = videoEditRate[0] # Fica com o primeiro valor, o desejado
231       videoEditRate = int(videoEditRate)
232       # EntryPoint
233       videoEntryPointFrames = int(element.find('EntryPoint').text)
234       videoEntryPointSeconds = float(videoEntryPointFrames)/videoEditRate
235
236     # Pega outra vez na PKL
237
238     tree = ET.parse(pklFullFilename)
239     root = tree.getroot()
240
241     # Remove o {namespace} de todos os elementos
242
243     for elements in root.iter():
244       elements.tag = elements.tag.split("}")[1][0:]
245
246     # Descobre o original filename do video
247
248     for element in root.iter('Asset'):
249       if element.find('Id').text == videoId:
250         # No DCP do filme Fuligem nao existe o campo OriginalFileName, por isso
              tive de adicionar este if
251         # Neste DCP o nome do ficheiro estava no campo AnnotationText, mas nao sei
              se vai ser sempre assim
252         # Este DCP esta encriptado de certa forma
253         if element.find('OriginalFileName') == None:
254           videoOriginalFilename = element.find('AnnotationText').text
255           print '\nvideoOriginalFilename: ' + videoOriginalFilename
256         else:
257           videoOriginalFilename = element.find('OriginalFileName').text
258           print '\nvideoOriginalFilename: ' + videoOriginalFilename
259
260
261
262     # Calcula o path do video
263     videoPath = inputFolder + '\\' + videoOriginalFilename
264     print '\nPath do video identificado: ' + videoPath
265
266
267     # #----------------------AUDIO-------------------------------------------
268
269
```

```
270     # Pega outra vez na CPL
271
272     tree = ET.parse(cplEscolhida)
273     root = tree.getroot()
274
275     # Remove o {namespace} de todos os elementos
276
277     for elements in root.iter():
278       elements.tag = elements.tag.split("}")[1][0:]
279
280     # Procura ID do Audio (MainSound) na CPL
281
282     for element in root.iter('MainSound'):
283       # Id
284       audioId = element.find('Id').text
285       print '\nId do audio: ' + audioId
286       # EditRate (frames)
287       audioEditRate = element.find('EditRate').text # Valor tipo '25 1'
288       audioEditRate = audioEditRate.split(" ") # Lista [25 1]
289       audioEditRate = audioEditRate[0] # Fica com o primeiro valor, o desejado
290       audioEditRate = int(audioEditRate)
291       # EntryPoint
292       audioEntryPointFrames = int(element.find('EntryPoint').text)
293       audioEntryPointSeconds = float(audioEntryPointFrames)/audioEditRate
294
295     # Pega outra vez na PKL
296
297     tree = ET.parse(pklFullFilename)
298     root = tree.getroot()
299
300     # Remove o {namespace} de todos os elementos
301
302     for elements in root.iter():
303       elements.tag = elements.tag.split("}")[1][0:]
304
305     # Descobre o original filename do audio
306
307     for element in root.iter('Asset'):
308       if element.find('Id').text == audioId:
309         # No DCP do filme Fuligem nao existe o campo OriginalFileName, por isso
              tive de adicionar este if
310         # Neste DCP o nome do ficheiro estava no campo AnnotationText, mas nao sei
              se vai ser sempre assim
311         # Este DCP esta encriptado de certa forma
312         if element.find('OriginalFileName') == None:
313           audioOriginalFilename = element.find('AnnotationText').text
314           print '\nAudioOriginalFilename: ' + audioOriginalFilename
315         else:
316           audioOriginalFilename = element.find('OriginalFileName').text
```

```
317         print '\nOriginalFileName do audio identificado: ' +
                audioOriginalFilename
318
319     # Calcula o path do audio
320     audioPath = inputFolder + '\\' + audioOriginalFilename
321     print '\nPath do audio identificado: ' + audioPath
322
323
324
325     # #-------------------------FFmpeg-----------------------------
326
327     if subtitlesExist == 1:
328
329       comando = 'ffmpeg -ss {} -i "{}" -ss {} -i "{}" -i moglogo100.png -
              filter_complex "[0:v] scale=1280:-2 [sc2]; [sc2] subtitles=\'{}\' [sub2]"
               -map "[sub2]" -map 1 -pix_fmt yuv420p -c:v libx264 -preset ultrafast -
              crf 21 -y "{}"'
330
331       comando = comando.format(videoEntryPointSeconds, videoPath,
              audioEntryPointSeconds, audioPath, subsPathOutSrtFfmpeg, outputFolder + '
              \\' + 'output720.mp4')
332
333     if subtitlesExist == 0:
334
335
336       comando = 'ffmpeg -ss {} -i "{}" -ss {} -i "{}" -i moglogo100.png -
              filter_complex "[0:v] scale=1280:-2 [sc2]" -map "[sc2]" -map 1 -pix_fmt
              yuv420p -c:v libx264 -preset ultrafast -crf 21 -y "{}"'
337
338       comando = comando.format(videoEntryPointSeconds, videoPath,
              audioEntryPointSeconds, audioPath, outputFolder + '\\' + 'output720.mp4')
339
340     subprocess.call(comando)
341
342
343   # GERA VERSAO EXT APENAS
344
345   elif (not selectedPNC and selectedEXT):
346     res = 'b'
347
348     # Ve qual e' o index da sub escolhida na lista
349     pos = subsIdiomas.index(selectedSubEXT)
350
351     cplEscolhida = listaDeCompositions[pos]
352
353     print cplEscolhida
354
355     # #-------------------------LEGENDAS-----------------------------
356
```

```
357     # Importa a cpl.xml
358
359     tree = ET.parse(cplEscolhida)
360     root = tree.getroot()
361
362     # Remove o {namespace} de todos os elementos
363
364     for elements in root.iter():
365       elements.tag = elements.tag.split("}")[1][0:]
366
367     # Ve se ha legendas na CPL, e obtem o ID
368
369     subtitlesExist = 0
370
371     for element in root.iter('MainSubtitle'):
372       subtitlesExist = 1
373       # Id
374       subsLanguage = element.find('Language').text
375       subsId = element.find('Id').text
376       print '\nId das legendas: ' + subsId
377       # EditRate (frames)
378       subsEditRate = element.find('EditRate').text # Valor tipo '25 1'
379       subsEditRate = subsEditRate.split(" ") # Lista [25 1]
380       subsEditRate = subsEditRate[0] # Fica com o primeiro valor, o desejado
381       subsEditRate = int(subsEditRate)
382       # EntryPoint
383       subsEntryPointFrames = int(element.find('EntryPoint').text)
384       subsEntryPointSeconds = float(subsEntryPointFrames)/subsEditRate
385
386     # Importa a pkl.xml
387
388     tree = ET.parse(pklFullFilename)
389     root = tree.getroot()
390
391     # Remove o {namespace} de todos os elementos
392
393     for elements in root.iter():
394       elements.tag = elements.tag.split("}")[1][0:]
395
396     # Esta parte apenas se houve legendas
397
398     if subtitlesExist == 1:
399
400       # Descobre o path/filename das legendas
401       for element in root.iter('Asset'):
402         if element.find('Id').text == subsId:
403           # Identifica o Original Filename
404           subsOriginalFilename = element.find('OriginalFileName').text
405           # Calcula o Filename
```

```
406           aux = subsOriginalFilename.split('/')
407           subsFilename = aux[len(aux)-1]
408           # Corrige o Original Filename. Muda os / para \
409           subsOriginalFilename = subsOriginalFilename.replace('/','\\')
410           # Path
411           subsPath = inputFolder + '\\' + subsOriginalFilename
412           print '\nPath da legenda identificado: ' + subsPath
413
414
415       # Calcula o filename agnostico (sem .formato)
416       aux = subsFilename.split('.') # Separa o path por '.'
417       subsFilenameAgno = '.'.join(aux[0:(len(aux)-1)])
418       # Identifica o formato das legendas
419       subsFormat = aux[len(aux) - 1] # O formato e o ultimo elemento da lista
420       print '\nFormato das legendas: ' + subsFormat
421
422       # Calcula o path agnostico (sem .formato)
423       aux = subsPath.split('.') # Separa o path por '.'
424       subsPathAgno = '.'.join(aux[0:(len(aux)-1)])
425
426       # Se o formato for MXF, converte primeiro para XML, e depois para SRT
427
428       if subsFormat == 'mxf':
429
430         # Chama o cmd, converte as legendas de MXF para XML
431
432         comando = 'asdcp-unwrap "{}" "{}"'
433
434         comando = comando.format(subsPath, outputFolder + '\\' + subsFilenameAgno +
                '.xml')
435
436         subprocess.call(comando)
437
438
439         # Chama o cmd, converte as legendas de XML para SRT
440
441         comando = 'SubtitleEdit /convert "{}" srt /overwrite /offset:00:00:00:{}'
442
443         comando = comando.format(outputFolder + '\\' + subsFilenameAgno + '.xml',
              subsEntryPointSeconds)
444
445         subprocess.call(comando)
446
447       # Se o formato for XML, converte logo para SRT
448
449       if subsFormat == 'xml':
450
451         comando = 'SubtitleEdit /convert {} srt /overwrite /outputfolder:{} /offset
              :00:00:00:{}'
```

```
452
453         comando = comando.format(subsPath, outputFolder, subsEntryPointSeconds)
454
455         subprocess.call(comando)
456
457       # Adapta o path das legendas SRT para por no ffmpeg
458       # Os \ passam a /, e o : tem de ser escapado
459
460       subsPathOutSrt = outputFolder + '\\' + subsFilenameAgno + '.srt'
461       subsPathOutSrtFfmpeg = subsPathOutSrt.replace('\\','/').replace(':','\\:')
462
463
464     # #---------------------------VIDEO----------------------------------------
465
466
467     # Pega outra vez na CPL
468
469     tree = ET.parse(cplEscolhida)
470     root = tree.getroot()
471
472     # Remove o {namespace} de todos os elementos
473
474     for elements in root.iter():
475       elements.tag = elements.tag.split("}")[1][0:]
476
477     # Info do video na CPL
478
479     for element in root.iter('MainPicture'):
480       # ID
481       videoId = element.find('Id').text
482       print '\n\nId do video: ' + videoId
483       # Frame Rate
484       FrameRate = element.find('FrameRate').text # Valor tipo '25 1'
485       FrameRate = FrameRate.split(" ") # Lista [25 1]
486       FrameRate = FrameRate[0] # Fica com o primeiro valor, o desejado
487       # EditRate (frames)
488       videoEditRate = element.find('EditRate').text # Valor tipo '25 1'
489       videoEditRate = videoEditRate.split(" ") # Lista [25 1]
490       videoEditRate = videoEditRate[0] # Fica com o primeiro valor, o desejado
491       videoEditRate = int(videoEditRate)
492       # EntryPoint
493       videoEntryPointFrames = int(element.find('EntryPoint').text)
494       videoEntryPointSeconds = float(videoEntryPointFrames)/videoEditRate
495
496     # Pega outra vez na PKL
497
498     tree = ET.parse(pklFullFilename)
499     root = tree.getroot()
500
```

```
501     # Remove o {namespace} de todos os elementos
502
503     for elements in root.iter():
504       elements.tag = elements.tag.split("}")[1][0:]
505
506     # Descobre o original filename do video
507
508     for element in root.iter('Asset'):
509       if element.find('Id').text == videoId:
510         # No DCP do filme Fuligem nao existe o campo OriginalFileName, por isso
              tive de adicionar este if
511         # Neste DCP o nome do ficheiro estava no campo AnnotationText, mas nao sei
              se vai ser sempre assim
512         # Este DCP esta encriptado de certa forma
513         if element.find('OriginalFileName') == None:
514           videoOriginalFilename = element.find('AnnotationText').text
515           print '\nvideoOriginalFilename: ' + videoOriginalFilename
516         else:
517           videoOriginalFilename = element.find('OriginalFileName').text
518           print '\nvideoOriginalFilename: ' + videoOriginalFilename
519
520
521
522     # Calcula o path do video
523     videoPath = inputFolder + '\\' + videoOriginalFilename
524     print '\nPath do video identificado: ' + videoPath
525
526
527     # #----------------------AUDIO-----------------------------------------
528
529
530     # Pega outra vez na CPL
531
532     tree = ET.parse(cplEscolhida)
533     root = tree.getroot()
534
535     # Remove o {namespace} de todos os elementos
536
537     for elements in root.iter():
538       elements.tag = elements.tag.split("}")[1][0:]
539
540     # Procura ID do Audio (MainSound) na CPL
541
542     for element in root.iter('MainSound'):
543       # Id
544       audioId = element.find('Id').text
545       print '\nId do audio: ' + audioId
546       # EditRate (frames)
547       audioEditRate = element.find('EditRate').text # Valor tipo '25 1'
```

```
548       audioEditRate = audioEditRate.split(" ") # Lista [25 1]
549       audioEditRate = audioEditRate[0] # Fica com o primeiro valor, o desejado
550       audioEditRate = int(audioEditRate)
551       # EntryPoint
552       audioEntryPointFrames = int(element.find('EntryPoint').text)
553       audioEntryPointSeconds = float(audioEntryPointFrames)/audioEditRate
554
555     # Pega outra vez na PKL
556
557     tree = ET.parse(pklFullFilename)
558     root = tree.getroot()
559
560     # Remove o {namespace} de todos os elementos
561
562     for elements in root.iter():
563       elements.tag = elements.tag.split("}")[1][0:]
564
565     # Descobre o original filename do audio
566
567     for element in root.iter('Asset'):
568       if element.find('Id').text == audioId:
569         # No DCP do filme Fuligem nao existe o campo OriginalFileName, por isso
              tive de adicionar este if
570         # Neste DCP o nome do ficheiro estava no campo AnnotationText, mas nao sei
              se vai ser sempre assim
571         # Este DCP esta encriptado de certa forma
572         if element.find('OriginalFileName') == None:
573           audioOriginalFilename = element.find('AnnotationText').text
574           print '\nAudioOriginalFilename: ' + audioOriginalFilename
575         else:
576           audioOriginalFilename = element.find('OriginalFileName').text
577           print '\nOriginalFileName do audio identificado: ' +
                audioOriginalFilename
578
579     # Calcula o path do audio
580     audioPath = inputFolder + '\\' + audioOriginalFilename
581     print '\nPath do audio identificado: ' + audioPath
582
583
584
585     # #-------------------------FFmpeg-----------------------------
586
587     if subtitlesExist == 1:
588
589       comando = 'ffmpeg -ss {} -i "{}" -ss {} -i "{}" -i moglogo100.png -
            filter_complex "[0:v] scale=-2:576 [sc1]; [sc1] subtitles=\'{}\' [sub1];
            [sub1][2] overlay=10:10 [ovr1]; [ovr1] drawtext=fontfile=/Windows/Fonts/
            arial.ttf:timecode=\'00\:00\:00\:00\':r={}:x=(w-tw)/2:y=(0.7*h):fontsize
```

```
        =60:fontcolor=white:box=1:boxcolor=0x00000099 [tc1]" -map "[tc1]" -map 1
        -pix_fmt yuv420p -c:v libx264 -preset ultrafast -crf 21 -y "{}"'
590
591     # Solucao do FJSevilla a funcionar, finalmente, apos dia todo a ver formas de
            lidar com espacos, aspas, parametros, etc
592     # Os {} no comando sao substituidos
593
594     comando = comando.format(videoEntryPointSeconds, videoPath,
        audioEntryPointSeconds, audioPath, subsPathOutSrtFfmpeg, FrameRate,
        outputFolder + '\\' + 'output576.mp4')
595
596   if subtitlesExist == 0:
597
598     comando = 'ffmpeg -ss {} -i "{}" -ss {} -i "{}" -i moglogo100.png -
        filter_complex "[0:v] scale=-2:576 [sc1]; [sc1][2] overlay=10:10 [ovr1];
        [ovr1] drawtext=fontfile=/Windows/Fonts/arial.ttf:timecode
        =\'00\:00\:00\:00\':r={}:x=(w-tw)/2:y=(0.7*h):fontsize=60:fontcolor=white
        :box=1:boxcolor=0x00000099 [tc1]" -map "[tc1]" -map 1 -pix_fmt yuv420p -c
        :v libx264 -preset ultrafast -crf 21 -y "{}"'
599
600     comando = comando.format(videoEntryPointSeconds, videoPath,
        audioEntryPointSeconds, audioPath, FrameRate, outputFolder + '\\' + '
        output576.mp4')
601
602   subprocess.call(comando)
603
604   # GERA AS 2 VERSOES
605   # Faz 2 vezes o processo habitual de abrir os ficheiros xml, etc
606
607
608 else:
609   res = 'c'
610
611   # ---PARTE DO PNC
612
613   # Ve qual e' o index da sub escolhida na lista
614   posPNC = subsIdiomas.index(selectedSubPNC)
615
616   cplEscolhidaPNC = listaDeCompositions[posPNC]
617
618   print cplEscolhidaPNC
619
620   # #------------------------LEGENDAS-----------------------------
621
622   # Importa a cpl.xml
623
624   tree = ET.parse(cplEscolhidaPNC)
625   root = tree.getroot()
626
```

```
627     # Remove o {namespace} de todos os elementos
628
629     for elements in root.iter():
630       elements.tag = elements.tag.split("}")[1][0:]
631
632     # Ve se ha legendas na CPL, e obtem o ID
633
634     subtitlesExistPNC = 0
635
636     for element in root.iter('MainSubtitle'):
637       subtitlesExistPNC = 1
638       # Id
639       subsLanguagePNC = element.find('Language').text
640       subsIdPNC = element.find('Id').text
641       print '\nId das legendas: ' + subsIdPNC
642       # EditRate (frames)
643       subsEditRatePNC = element.find('EditRate').text # Valor tipo '25 1'
644       subsEditRatePNC = subsEditRatePNC.split(" ") # Lista [25 1]
645       subsEditRatePNC = subsEditRatePNC[0] # Fica com o primeiro valor, o desejado
646       subsEditRatePNC = int(subsEditRatePNC)
647       # EntryPoint
648       subsEntryPointFramesPNC = int(element.find('EntryPoint').text)
649       subsEntryPointSecondsPNC = float(subsEntryPointFramesPNC)/subsEditRatePNC
650
651     # Importa a pkl.xml
652
653     tree = ET.parse(pklFullFilename)
654     root = tree.getroot()
655
656     # Remove o {namespace} de todos os elementos
657
658     for elements in root.iter():
659       elements.tag = elements.tag.split("}")[1][0:]
660
661     # Esta parte apenas se houve legendas
662
663     if subtitlesExistPNC == 1:
664
665       # Descobre o path/filename das legendas
666       for element in root.iter('Asset'):
667         if element.find('Id').text == subsIdPNC:
668           # Identifica o Original Filename
669           subsOriginalFilenamePNC = element.find('OriginalFileName').text
670           # Calcula o Filename
671           auxPNC = subsOriginalFilenamePNC.split('/')
672           subsFilenamePNC = auxPNC[len(auxPNC)-1]
673           # Corrige o Original Filename. Muda os / para \
674           subsOriginalFilenamePNC = subsOriginalFilenamePNC.replace('/','\\')
675           # Path
```

```
676             subsPathPNC = inputFolder + '\\' + subsOriginalFilenamePNC
677             print '\nPath da legenda identificado: ' + subsPathPNC
678
679
680         # Calcula o filename agnostico (sem .formato)
681         auxPNC = subsFilenamePNC.split('.') # Separa o path por '.'
682         subsFilenameAgnoPNC = '.'.join(auxPNC[0:(len(auxPNC)-1)])
683         # Identifica o formato das legendas
684         subsFormatPNC = auxPNC[len(auxPNC) - 1] # O formato e o ultimo elemento da
                lista
685         print '\nFormato das legendas: ' + subsFormatPNC
686
687         # Calcula o path agnostico (sem .formato)
688         auxPNC = subsPathPNC.split('.') # Separa o path por '.'
689         subsPathAgnoPNC = '.'.join(auxPNC[0:(len(auxPNC)-1)])
690
691         # Se o formato for MXF, converte primeiro para XML, e depois para SRT
692
693         if subsFormatPNC == 'mxf':
694
695           # Chama o cmd, converte as legendas de MXF para XML
696
697           comando = 'asdcp-unwrap "{}" "{}"'
698
699           comando = comando.format(subsPathPNC, outputFolder + '\\' +
                  subsFilenameAgnoPNC + '.xml')
700
701           subprocess.call(comando)
702
703
704           # Chama o cmd, converte as legendas de XML para SRT
705
706           comando = 'SubtitleEdit /convert "{}" srt /overwrite /offset:00:00:00:{}'
707
708           comando = comando.format(outputFolder + '\\' + subsFilenameAgnoPNC + '.xml'
                  , subsEntryPointSecondsPNC)
709
710           subprocess.call(comando)
711
712         # Se o formato for XML, converte logo para SRT
713
714         if subsFormatPNC == 'xml':
715
716           comando = 'SubtitleEdit /convert {} srt /overwrite /outputfolder:{} /offset
                  :00:00:00:{}'
717
718           comando = comando.format(subsPathPNC, outputFolder,
                  subsEntryPointSecondsPNC)
719
```

```
720         subprocess.call(comando)
721
722       # Adapta o path das legendas SRT para por no ffmpeg
723       # Os \ passam a /, e o : tem de ser escapado
724
725       subsPathOutSrtPNC = outputFolder + '\\' + subsFilenameAgnoPNC + '.srt'
726       subsPathOutSrtFfmpegPNC = subsPathOutSrtPNC.replace('\\','/').replace(':','
              \\:')
727
728
729     # #---------------------------VIDEO----------------------------------------
730     # Pega outra vez na CPL
731
732     tree = ET.parse(cplEscolhidaPNC)
733     root = tree.getroot()
734
735     # Remove o {namespace} de todos os elementos
736
737     for elements in root.iter():
738       elements.tag = elements.tag.split("}")[1][0:]
739
740     # Info do video na CPL
741
742     for element in root.iter('MainPicture'):
743       # ID
744       videoIdPNC = element.find('Id').text
745       print '\n\nId do video: ' + videoIdPNC
746       # Frame Rate
747       FrameRatePNC = element.find('FrameRate').text # Valor tipo '25 1'
748       FrameRatePNC = FrameRatePNC.split(" ") # Lista [25 1]
749       FrameRatePNC = FrameRatePNC[0] # Fica com o primeiro valor, o desejado
750       # EditRate (frames)
751       videoEditRatePNC = element.find('EditRate').text # Valor tipo '25 1'
752       videoEditRatePNC = videoEditRatePNC.split(" ") # Lista [25 1]
753       videoEditRatePNC = videoEditRatePNC[0] # Fica com o primeiro valor, o
              desejado
754       videoEditRatePNC = int(videoEditRatePNC)
755       # EntryPoint
756       videoEntryPointFramesPNC = int(element.find('EntryPoint').text)
757       videoEntryPointSecondsPNC = float(videoEntryPointFramesPNC)/videoEditRatePNC
758
759     # Pega outra vez na PKL
760
761     tree = ET.parse(pklFullFilename)
762     root = tree.getroot()
763
764     # Remove o {namespace} de todos os elementos
765
766     for elements in root.iter():
```

```
767       elements.tag = elements.tag.split("}")[1][0:]
768
769     # Descobre o original filename do video
770
771     for element in root.iter('Asset'):
772       if element.find('Id').text == videoIdPNC:
773         # No DCP do filme Fuligem nao existe o campo OriginalFileName, por isso
              tive de adicionar este if
774         # Neste DCP o nome do ficheiro estava no campo AnnotationText, mas nao sei
              se vai ser sempre assim
775         # Este DCP esta encriptado de certa forma
776         if element.find('OriginalFileName') == None:
777           videoOriginalFilenamePNC = element.find('AnnotationText').text
778           print '\nvideoOriginalFilename: ' + videoOriginalFilenamePNC
779         else:
780           videoOriginalFilenamePNC = element.find('OriginalFileName').text
781           print '\nvideoOriginalFilename: ' + videoOriginalFilenamePNC
782
783
784
785     # Calcula o path do video
786     videoPathPNC = inputFolder + '\\' + videoOriginalFilenamePNC
787     print '\nPath do video identificado: ' + videoPathPNC
788
789
790     # #----------------------AUDIO-------------------------------------------
791
792
793     # Pega outra vez na CPL
794
795     tree = ET.parse(cplEscolhidaPNC)
796     root = tree.getroot()
797
798     # Remove o {namespace} de todos os elementos
799
800     for elements in root.iter():
801       elements.tag = elements.tag.split("}")[1][0:]
802
803     # Procura ID do Audio (MainSound) na CPL
804
805     for element in root.iter('MainSound'):
806       # Id
807       audioIdPNC = element.find('Id').text
808       print '\nId do audio: ' + audioIdPNC
809       # EditRate (frames)
810       audioEditRatePNC = element.find('EditRate').text # Valor tipo '25 1'
811       audioEditRatePNC = audioEditRatePNC.split(" ") # Lista [25 1]
812       audioEditRatePNC = audioEditRatePNC[0] # Fica com o primeiro valor, o
              desejado
```

```
813       audioEditRatePNC = int(audioEditRatePNC)
814       # EntryPoint
815       audioEntryPointFramesPNC = int(element.find('EntryPoint').text)
816       audioEntryPointSecondsPNC = float(audioEntryPointFramesPNC)/audioEditRatePNC
817
818     # Pega outra vez na PKL
819
820     tree = ET.parse(pklFullFilename)
821     root = tree.getroot()
822
823     # Remove o {namespace} de todos os elementos
824
825     for elements in root.iter():
826       elements.tag = elements.tag.split("}")[1][0:]
827
828     # Descobre o original filename do audio
829
830     for element in root.iter('Asset'):
831       if element.find('Id').text == audioIdPNC:
832         # No DCP do filme Fuligem nao existe o campo OriginalFileName, por isso
               tive de adicionar este if
833         # Neste DCP o nome do ficheiro estava no campo AnnotationText, mas nao sei
               se vai ser sempre assim
834         # Este DCP esta encriptado de certa forma
835         if element.find('OriginalFileName') == None:
836           audioOriginalFilenamePNC = element.find('AnnotationText').text
837           print '\nAudioOriginalFilename: ' + audioOriginalFilenamePNC
838         else:
839           audioOriginalFilenamePNC = element.find('OriginalFileName').text
840           print '\nOriginalFileName do audio identificado: ' +
                  audioOriginalFilenamePNC
841
842     # Calcula o path do audio
843     audioPathPNC = inputFolder + '\\' + audioOriginalFilenamePNC
844     print '\nPath do audio identificado: ' + audioPathPNC
845
846   # ---PARTE DO EXT
847
848     # Ve qual e' o index da sub escolhida na lista
849     posEXT = subsIdiomas.index(selectedSubEXT)
850
851     cplEscolhidaEXT = listaDeCompositions[posEXT]
852
853     print cplEscolhidaEXT
854
855     # #------------------------LEGENDAS-----------------------------
856
857     # Importa a cpl.xml
858
```

```
859     tree = ET.parse(cplEscolhidaEXT)
860     root = tree.getroot()
861
862     # Remove o {namespace} de todos os elementos
863
864     for elements in root.iter():
865       elements.tag = elements.tag.split("}")[1][0:]
866
867     # Ve se ha legendas na CPL, e obtem o ID
868
869     subtitlesExistEXT = 0
870
871     for element in root.iter('MainSubtitle'):
872       subtitlesExistEXT = 1
873       # Id
874       subsLanguageEXT = element.find('Language').text
875       subsIdEXT = element.find('Id').text
876       print '\nId das legendas: ' + subsIdEXT
877       # EditRate (frames)
878       subsEditRateEXT = element.find('EditRate').text # Valor tipo '25 1'
879       subsEditRateEXT = subsEditRateEXT.split(" ") # Lista [25 1]
880       subsEditRateEXT = subsEditRateEXT[0] # Fica com o primeiro valor, o desejado
881       subsEditRateEXT = int(subsEditRateEXT)
882       # EntryPoint
883       subsEntryPointFramesEXT = int(element.find('EntryPoint').text)
884       subsEntryPointSecondsEXT = float(subsEntryPointFramesEXT)/subsEditRateEXT
885
886     # Importa a pkl.xml
887
888     tree = ET.parse(pklFullFilename)
889     root = tree.getroot()
890
891     # Remove o {namespace} de todos os elementos
892
893     for elements in root.iter():
894       elements.tag = elements.tag.split("}")[1][0:]
895
896     # Esta parte apenas se houve legendas
897
898     if subtitlesExistEXT == 1:
899
900       # Descobre o path/filename das legendas
901       for element in root.iter('Asset'):
902         if element.find('Id').text == subsIdEXT:
903           # Identifica o Original Filename
904           subsOriginalFilenameEXT = element.find('OriginalFileName').text
905           # Calcula o Filename
906           auxEXT = subsOriginalFilenameEXT.split('/')
907           subsFilenameEXT = auxEXT[len(auxEXT)-1]
```

```
908             # Corrige o Original Filename. Muda os / para \
909             subsOriginalFilenameEXT = subsOriginalFilenameEXT.replace('/','\\')
910             # Path
911             subsPathEXT = inputFolder + '\\' + subsOriginalFilenameEXT
912             print '\nPath da legenda identificado: ' + subsPathEXT
913
914
915         # Calcula o filename agnostico (sem .formato)
916         auxEXT = subsFilenameEXT.split('.') # Separa o path por '.'
917         subsFilenameAgnoEXT = '.'.join(auxEXT[0:(len(auxEXT)-1)])
918         # Identifica o formato das legendas
919         subsFormatEXT = auxEXT[len(auxEXT) - 1] # O formato e o ultimo elemento da
                lista
920         print '\nFormato das legendas: ' + subsFormatEXT
921
922         # Calcula o path agnostico (sem .formato)
923         auxEXT = subsPathEXT.split('.') # Separa o path por '.'
924         subsPathAgnoEXT = '.'.join(auxEXT[0:(len(auxEXT)-1)])
925
926         # Se o formato for MXF, converte primeiro para XML, e depois para SRT
927
928         if subsFormatEXT == 'mxf':
929
930           # Chama o cmd, converte as legendas de MXF para XML
931
932           comando = 'asdcp-unwrap "{}" "{}"'
933
934           comando = comando.format(subsPathEXT, outputFolder + '\\' +
                  subsFilenameAgnoEXT + '.xml')
935
936           subprocess.call(comando)
937
938
939           # Chama o cmd, converte as legendas de XML para SRT
940
941           comando = 'SubtitleEdit /convert "{}" srt /overwrite /offset:00:00:00:{}'
942
943           comando = comando.format(outputFolder + '\\' + subsFilenameAgnoEXT + '.xml'
                  , subsEntryPointSecondsEXT)
944
945           subprocess.call(comando)
946
947         # Se o formato for XML, converte logo para SRT
948
949         if subsFormatEXT == 'xml':
950
951           comando = 'SubtitleEdit /convert {} srt /overwrite /outputfolder:{} /offset
                  :00:00:00:{}'
952
```

```
953         comando = comando.format(subsPathEXT, outputFolder,
                subsEntryPointSecondsEXT)
954
955         subprocess.call(comando)
956
957       # Adapta o path das legendas SRT para por no ffmpeg
958       # Os \ passam a /, e o : tem de ser escapado
959
960       subsPathOutSrtEXT = outputFolder + '\\' + subsFilenameAgnoEXT + '.srt'
961       subsPathOutSrtFfmpegEXT = subsPathOutSrtEXT.replace('\\','/').replace(':','
              \\:')
962
963     # #--------------------------VIDEO---------------------------------------
964
965     # Pega outra vez na CPL
966
967     tree = ET.parse(cplEscolhidaEXT)
968     root = tree.getroot()
969
970     # Remove o {namespace} de todos os elementos
971
972     for elements in root.iter():
973       elements.tag = elements.tag.split("}")[1][0:]
974
975     # Info do video na CPL
976
977     for element in root.iter('MainPicture'):
978       # ID
979       videoIdEXT = element.find('Id').text
980       print '\n\nId do video: ' + videoIdEXT
981       # Frame Rate
982       FrameRateEXT = element.find('FrameRate').text # Valor tipo '25 1'
983       FrameRateEXT = FrameRateEXT.split(" ") # Lista [25 1]
984       FrameRateEXT = FrameRateEXT[0] # Fica com o primeiro valor, o desejado
985       # EditRate (frames)
986       videoEditRateEXT = element.find('EditRate').text # Valor tipo '25 1'
987       videoEditRateEXT = videoEditRateEXT.split(" ") # Lista [25 1]
988       videoEditRateEXT = videoEditRateEXT[0] # Fica com o primeiro valor, o
              desejado
989       videoEditRateEXT = int(videoEditRateEXT)
990       # EntryPoint
991       videoEntryPointFramesEXT = int(element.find('EntryPoint').text)
992       videoEntryPointSecondsEXT = float(videoEntryPointFramesEXT)/videoEditRateEXT
993
994     # Pega outra vez na PKL
995
996     tree = ET.parse(pklFullFilename)
997     root = tree.getroot()
998
```

```
999     # Remove o {namespace} de todos os elementos
1000
1001    for elements in root.iter():
1002      elements.tag = elements.tag.split("}")[1][0:]
1003
1004    # Descobre o original filename do video
1005
1006    for element in root.iter('Asset'):
1007      if element.find('Id').text == videoIdEXT:
1008        # No DCP do filme Fuligem nao existe o campo OriginalFileName, por isso
              tive de adicionar este if
1009        # Neste DCP o nome do ficheiro estava no campo AnnotationText, mas nao sei
              se vai ser sempre assim
1010        # Este DCP esta encriptado de certa forma
1011        if element.find('OriginalFileName') == None:
1012          videoOriginalFilenameEXT = element.find('AnnotationText').text
1013          print '\nvideoOriginalFilename: ' + videoOriginalFilenameEXT
1014        else:
1015          videoOriginalFilenameEXT = element.find('OriginalFileName').text
1016          print '\nvideoOriginalFilename: ' + videoOriginalFilenameEXT
1017
1018    # Calcula o path do video
1019    videoPathEXT = inputFolder + '\\' + videoOriginalFilenameEXT
1020    print '\nPath do video identificado: ' + videoPathEXT
1021
1022    # #----------------------AUDIO-------------------------------------------
1023
1024    # Pega outra vez na CPL
1025
1026    tree = ET.parse(cplEscolhidaEXT)
1027    root = tree.getroot()
1028
1029    # Remove o {namespace} de todos os elementos
1030
1031    for elements in root.iter():
1032      elements.tag = elements.tag.split("}")[1][0:]
1033
1034    # Procura ID do Audio (MainSound) na CPL
1035
1036    for element in root.iter('MainSound'):
1037      # Id
1038      audioIdEXT = element.find('Id').text
1039      print '\nId do audio: ' + audioIdEXT
1040      # EditRate (frames)
1041      audioEditRateEXT = element.find('EditRate').text # Valor tipo '25 1'
1042      audioEditRateEXT = audioEditRateEXT.split(" ") # Lista [25 1]
1043      audioEditRateEXT = audioEditRateEXT[0] # Fica com o primeiro valor, o
            desejado
1044      audioEditRateEXT = int(audioEditRateEXT)
```

```
1045        # EntryPoint
1046        audioEntryPointFramesEXT = int(element.find('EntryPoint').text)
1047        audioEntryPointSecondsEXT = float(audioEntryPointFramesEXT)/audioEditRateEXT
1048
1049      # Pega outra vez na PKL
1050
1051      tree = ET.parse(pklFullFilename)
1052      root = tree.getroot()
1053
1054      # Remove o {namespace} de todos os elementos
1055
1056      for elements in root.iter():
1057        elements.tag = elements.tag.split("}")[1][0:]
1058
1059      # Descobre o original filename do audio
1060
1061      for element in root.iter('Asset'):
1062        if element.find('Id').text == audioIdEXT:
1063          # No DCP do filme Fuligem nao existe o campo OriginalFileName, por isso
                 tive de adicionar este if
1064          # Neste DCP o nome do ficheiro estava no campo AnnotationText, mas nao sei
                 se vai ser sempre assim
1065          # Este DCP esta encriptado de certa forma
1066          if element.find('OriginalFileName') == None:
1067            audioOriginalFilenameEXT = element.find('AnnotationText').text
1068            print '\nAudioOriginalFilename: ' + audioOriginalFilenameEXT
1069          else:
1070            audioOriginalFilenameEXT = element.find('OriginalFileName').text
1071            print '\nOriginalFileName do audio identificado: ' +
                   audioOriginalFilenameEXT
1072
1073      # Calcula o path do audio
1074      audioPathEXT = inputFolder + '\\' + audioOriginalFilenameEXT
1075      print '\nPath do audio identificado: ' + audioPathEXT
1076
1077      # #-------------------------FFmpeg-----------------------------
1078
1079      if subtitlesExistPNC == 0 and subtitlesExistEXT == 0:
1080
1081        comando = 'ffmpeg -ss {} -i "{}" -ss {} -i "{}" -i moglogo100.png -
             filter_complex "[0:v] scale=-2:576 [sc1]; [sc1][2] overlay=10:10 [ovr1];
             [ovr1] drawtext=fontfile=/Windows/Fonts/arial.ttf:timecode
             =\'00\:00\:00\:00\':r={}:x=(w-tw)/2:y=(0.7*h):fontsize=60:fontcolor=white
             :box=1:boxcolor=0x00000099 [tc1]" -map "[tc1]" -map 1 -pix_fmt yuv420p -c
             :v libx264 -preset ultrafast -crf 21 -y "{}" -filter_complex "[0:v] scale
             =1280:-2 [sc2]" -map "[sc2]" -map 1 -pix_fmt yuv420p -c:v libx264 -preset
              ultrafast -crf 21 -y "{}"'
1082
```

```
1083        comando = comando.format(videoEntryPointSecondsEXT, videoPathEXT,
                audioEntryPointSecondsEXT, audioPathEXT, FrameRateEXT, outputFolder + '\\
                ' + 'output576.mp4', outputFolder + '\\' + 'output720.mp4')
1084
1085      if subtitlesExistPNC == 0 and subtitlesExistEXT == 1:
1086
1087        comando = 'ffmpeg -ss {} -i "{}" -ss {} -i "{}" -i moglogo100.png -
                filter_complex "[0:v] scale=-2:576 [sc1]; [sc1] subtitles=\'{}\' [sub1];
                [sub1][2] overlay=10:10 [ovr1]; [ovr1] drawtext=fontfile=/Windows/Fonts/
                arial.ttf:timecode=\'00\:00\:00\:00\':r={}:x=(w-tw)/2:y=(0.7*h):fontsize
                =60:fontcolor=white:box=1:boxcolor=0x00000099 [tc1]" -map "[tc1]" -map 1
                -pix_fmt yuv420p -c:v libx264 -preset ultrafast -crf 21 -y "{}" -
                filter_complex "[0:v] scale=1280:-2 [sc2] -map "[sc2]" -map 1 -pix_fmt
                yuv420p -c:v libx264 -preset ultrafast -crf 21 -y "{}"'
1088
1089        comando = comando.format(videoEntryPointSecondsEXT, videoPathEXT,
                audioEntryPointSecondsEXT, audioPathEXT, subsPathOutSrtFfmpegEXT,
                FrameRateEXT, outputFolder + '\\' + 'output576.mp4', outputFolder + '\\'
                + 'output720.mp4')
1090
1091      if subtitlesExistPNC == 1 and subtitlesExistEXT == 0:
1092
1093        comando = 'ffmpeg -ss {} -i "{}" -ss {} -i "{}" -i moglogo100.png -
                filter_complex "[0:v] scale=-2:576 [sc1]; [sc1][2] overlay=10:10 [ovr1];
                [ovr1] drawtext=fontfile=/Windows/Fonts/arial.ttf:timecode
                =\'00\:00\:00\:00\':r={}:x=(w-tw)/2:y=(0.7*h):fontsize=60:fontcolor=white
                :box=1:boxcolor=0x00000099 [tc1]" -map "[tc1]" -map 1 -pix_fmt yuv420p -c
                :v libx264 -preset ultrafast -crf 21 -y "{}" -filter_complex "[0:v] scale
                =1280:-2 [sc2]; [sc2] subtitles=\'{}\' [sub2]" -map "[sub2]" -map 1 -
                pix_fmt yuv420p -c:v libx264 -preset ultrafast -crf 21 -y "{}"'
1094
1095        comando = comando.format(videoEntryPointSecondsEXT, videoPathEXT,
                audioEntryPointSecondsEXT, audioPathEXT, FrameRateEXT, outputFolder + '\\
                ' + 'output576.mp4', subsPathOutSrtFfmpegPNC, outputFolder + '\\' + '
                output720.mp4')
1096
1097      if subtitlesExistPNC == 1 and subtitlesExistEXT == 1:
1098
1099        comando = 'ffmpeg -ss {} -i "{}" -ss {} -i "{}" -i moglogo100.png -
                filter_complex "[0:v] scale=-2:576 [sc1]; [sc1] subtitles=\'{}\' [sub1];
                [sub1][2] overlay=10:10 [ovr1]; [ovr1] drawtext=fontfile=/Windows/Fonts/
                arial.ttf:timecode=\'00\:00\:00\:00\':r={}:x=(w-tw)/2:y=(0.7*h):fontsize
                =60:fontcolor=white:box=1:boxcolor=0x00000099 [tc1]" -map "[tc1]" -map 1
                -pix_fmt yuv420p -c:v libx264 -preset ultrafast -crf 21 -y "{}" -
                filter_complex "[0:v] scale=1280:-2 [sc2]; [sc2] subtitles=\'{}\' [sub2]"
                 -map "[sub2]" -map 1 -pix_fmt yuv420p -c:v libx264 -preset ultrafast -
                crf 21 -y "{}"'
1100
```

```
1101        comando = comando.format(videoEntryPointSecondsEXT, videoPathEXT,
                audioEntryPointSecondsEXT, audioPathEXT, subsPathOutSrtFfmpegEXT,
                FrameRateEXT, outputFolder + '\\' + 'output576.mp4',
                subsPathOutSrtFfmpegPNC, outputFolder + '\\' + 'output720.mp4')
1102
1103
1104      subprocess.call(comando)
1105
1106    return res
```

Anexo B

Página web de interface com o utilizador

B.1 Código front-end

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">

    <title>Cinemateca</title>

    <style>
      html {
        font-family: sans-serif;
      }

      body {
        background-color: #FFFFFF;
        width: 50%;
        max-width: 800px;
        min-width: 480px;
        margin: 0 auto;
      }

      h1 {
        text-align: center;
      }

      h1 a {
        text-decoration: none;
        color: #000000;
      }
```

```
29
30      p {
31        text-decoration: none;
32        color: #000000;
33      }
34
35      h3 {
36        text-align: center;
37        color: #000000;
38      }
39
40      fieldset {
41        border: 0;
42      }
43
44    </style>
45
46    <script src="https://code.jquery.com/jquery-3.3.1.min.js"
47    integrity="sha256-FgpCb/KJQlLNfOu91ta32o/NMZxltwRo8QtmkMRdAu8="
48    crossorigin="anonymous">
49    </script>
50
51  </head>
52
53  <body>
54    <h1><a href="http://172.25.120.193:8000/cinemateca">Cinemateca Portuguesa</a></
        h1>
55    <h3>Conversor DCP</h3>
56
57    <div id="upload">
58      <p>Seleccione a pasta de um DCP:</p>
59      <input id="folderIn" type="file" webkitdirectory>
60      <br><br>
61      <button id="buttonUpload" type="button">Upload</button>
62    </div>
63
64    <div id="selectInfo">
65    </div>
66
67    <div id="output">
68    </div>
69
70    <script>
71
72      function createFormFromJson(data){
73
74        $("#upload").html('');
75
76        // Proxy PNC
```

```
77          $("#selectInfo").append('<form id="formPNC"><label>Gerar vers o 720p para
                o Plano Nacional de Cinema <input type="checkbox" id="proxyPNC" name="
                proxyPNC"></label></form>');
78
79          $("#selectInfo").append('<fieldset id="fieldsetPNCsubs" disabled><form id="
                formPNCsubs"></form></fieldset>');
80
81          for (var i=0; i< data.length; i++) {
82            $("#formPNCsubs").append('<label><input type="radio" class="subsPNC" name
                  ="subsPNC" value="' + data[i] + '" />' + data[i] + '</label><br>')
83          };
84
85          // Proxy EXT
86          $("#selectInfo").append('<form id="formEXT"><label>Gerar vers o 576p para
                consulta externa <input type="checkbox" id="proxyEXT" name="proxyEXT"><
                /label></form>');
87
88          $("#selectInfo").append('<fieldset id="fieldsetEXTsubs" disabled><form id="
                formEXTsubs"></form></fieldset>');
89
90          for (var i=0; i< data.length; i++) {
91            $("#formEXTsubs").append('<label><input type="radio" class="subsEXT" name
                  ="subsEXT" value="' + data[i] + '" />' + data[i] + '</label><br>')
92          }
93
94          $("#selectInfo").append('<br><button id="buttonEncoding" type="submit">
                Iniciar encoding</button>');
95        }
96
97
98        $("#buttonUpload").click(function() {
99
100         $.ajax({
101           url: '/cinemateca/upload',
102           type: 'GET',
103           success: function(response) {
104             console.log(response);
105             var subsIdiomas = response;
106             // Cria form com as legendas
107             createFormFromJson(subsIdiomas);
108
109
110           },
111           error: function(error) {
112               console.log(error);
113           }
114         })
115       });
116
```

```
117     // Deixa escolher as legendas do PNC só se quiser gerar esse proxy
118     $("#selectInfo").on('change', "#proxyPNC", function() {
119
120       if (this.checked){
121         $("#fieldsetPNCsubs").prop('disabled', false);
122       }
123       else {
124         $("#fieldsetPNCsubs").prop('disabled', true);
125       }
126
127     });
128
129     // Deixa escolher as legendas do EXT só se quiser gerar esse proxy
130     $("#selectInfo").on('change', "#proxyEXT", function() {
131
132       if (this.checked){
133         $("#fieldsetEXTsubs").prop('disabled', false);
134       }
135       else {
136         $("#fieldsetEXTsubs").prop('disabled', true);
137       }
138
139     });
140
141     // Lê o click no bot o Iniciar Encoding
142
143     // Este .on é para seleccionar elementos que foram criados dinamicamente
144     // $(staticAncestors).on(eventName, dynamicChild, function() {});
145     $("#selectInfo").on('click', "#buttonEncoding", function() {
146
147       var A = $('#proxyPNC').is(':checked');
148       var B = $('#formPNCsubs input:checked').length;
149       var C = $('#proxyEXT').is(':checked');
150       var D = $('#formEXTsubs input:checked').length;
151
152
153       if ( ( A && (B>0) & !C ) || ( C && (D>0) && !A ) || ( A && (B>0) && C && (D
              >0) ) ) {
154
155         // tive de pôr estes parametros num objecto, para depois passar ao flask
                como json
156         // se fosse um array ele nao reconhecia bem
157         var parameters = {
158           selectedPNC: $('#proxyPNC').is(':checked'),
159           selectedEXT: $('#proxyEXT').is(':checked'),
160           selectedSubPNC: $('input[name=subsPNC]:checked').val(),
161           selectedSubEXT: $('input[name=subsEXT]:checked').val()
162         };
163
```

```
164           if (parameters['selectedSubPNC'] == null) {
165             parameters['selectedSubPNC'] = 0;
166           }
167
168           if (parameters['selectedSubEXT'] == null) {
169             parameters['selectedSubEXT'] = 0;
170           }
171
172           console.log(parameters.selectedSubPNC);
173
174
175           $.ajax({
176             url: '/cinemateca/encoding',
177             type: 'POST',
178             contentType: "application/json; charset=utf-8",
179             data: JSON.stringify(parameters),
180
181             success: function(response) {
182                 console.log(response);
183             },
184             error: function(error) {
185                 console.log(error);
186             }
187           });
188
189         }
190
191         else {
192           alert("N o seleccionou nenhuma opç o ");
193         }
194
195       });
196
197     </script>
198   </body>
199 </html>
```

B.2 Código back-end

```
from flask import Flask, jsonify, render_template, Response, request, redirect,
    url_for
import DCPv2web, teste
import settings
import sys

# Argumentos de entrada no cmd
inFolder = sys.argv[1]
outFolder = sys.argv[2]

# rotas Flask
app = Flask(__name__)

@app.route('/cinemateca')
def cinemateca():

    return render_template('cinemateca.html')


@app.route("/cinemateca/upload", methods = ['GET'])
def upload():
    pklFullFilename = DCPv2web.getPklFullFilename(inFolder, outFolder)
    listaDeCompositions = DCPv2web.getListaDeCompositions(inFolder)
    subsIdiomas = DCPv2web.getSubsIdiomas(listaDeCompositions)

    return jsonify(subsIdiomas)

@app.route("/cinemateca/encoding", methods = ['POST'])
def encoding():

    data = request.json

    # vai buscar os parametros de proxies e subs
    ret = DCPv2web.iniciarEncoding(data['selectedPNC'], data['selectedEXT'], data['
        selectedSubPNC'], data['selectedSubEXT'])

    print ret

    return jsonify("x")
```

Referências

[1] W3c video on the web - report, Abril 2008. URL: https://www.w3.org/2007/08/video/report.

[2] Dmitriy Vatolin, Dmitriy Kulikov, Dr. Mikhail Erofeev, Stanislav Dolganov, e Sergey Zvezdakov. Msu codec comparison 2017 - part v: High quality encoders. Relatório técnico, CS MSU Graphics Media Lab, Video Group, Janeiro 2018.

[3] YouTube Engineering e Developers Blog. Vp9: Faster, better, buffer-free youtube videos, Abril 2015. URL: https://youtube-eng.googleblog.com/2015/04/vp9-faster-better-buffer-free-youtube.html.

[4] Digital cinema system specification version 1.2. Relatório técnico, Digital Cinema Initiatives, LLC, Outubro 2012.

[5] Cyberhomes. Home cinema surround sound: Breathtaking audio, Acedido a 20 de Junho de 2018. URL: https://www.cyberhomes.co.uk/what-we-do/home-cinema-installations/home-cinema-audio-surround-sound/.

[6] Brad Allen. Resolution/aspect ratio cheat sheet v1.6.

[7] Chameleon DG. Resolution / aspect ratio cheat sheet, Fevereiro 2015. URL: http://blog.chameleondg.com/post/111891072017/resolution-aspect-ratio-cheat-sheet.

[8] SMPTE. Smpte metadata registers - view of labels, Acedido a 10 de Junho de 2018. URL: https://registry.smpte-ra.org/view/published/labels_view.html.

[9] Reto Kromer. Matroska and ffv1: One file format for film and video archiving? Relatório técnico, Journal of Film Preservation,o, Abril 2017.

[10] Nicola Mazzanti. New challenges for moving image archives: digital delivery, digital preservation. the edcine project's approach. Relatório técnico, Cinema Expert Group, Junho 2009.

[11] Emanuel Lorrain. A short guide to choosing a digital format for video archiving masters, Março 2014. URL: https://www.scart.be/?q=en/content/short-guide-choosing-digital-format-video-archiving-masters.

[12] Sustainability of digital formats: Planning for library of congress collections, Acedido a 5 de Março de 2018. URL: https://www.loc.gov/preservation/digital/formats/index.html.

[13] Recommendations on preservation files for use in the digitization of analog audio and video recordings. Relatório técnico, National, Provincial and Territorial Archives Conference Audiovisual Working Group, Novembro 2015.

[14] Apple prores white paper. Relatório técnico, Apple Inc., Abril 2018.

[15] Bruce Devlin, Matt Beard, Phil Tudor, e Jim Wilkinson. *The MXF Book: an Introduction to the Material eXchange Format*. Focal Press, 2006.

[16] Anush Moorthy Jan De Cock, Aditya Mavlankar e Anne Aaron. A large-scale video codec comparison of x264, x265 and libvpx for practical vod applications. *Applications of Digital Image Processing XXXIX,*, 2016.

[17] ITU-T e ISO/IEC JTC 1. Advanced video coding for generic audiovisual services. Relatório técnico, ITU-T Rec. H.264 and ISO/IEC 14496-10, 2003.

[18] ITU-T e ISO/IEC JTC 1. High efficiency video coding. Relatório técnico, ITU-T Rec. H.265 and ISO/IEC 23008-2, 2013.

[19] Adrian Grange, Peter de Rivaz, e Jonathan Hunt. Vp9 bitstream & decoding process specification. Relatório técnico, Google Inc., Março 2016.

[20] Phil Layton. World cup 2018 in uhd hdr on bbc iplayer, Maio 2018. URL: https://www.bbc.co.uk/rd/blog/2018-05-uhd_hdr_world_cup_2018.

[21] James Bankoski, Matthew Frost, e Adrian Grange. The internet needs a competitive, royalty-free video codec. *APSIPA Transactions on Signal and Information Processing*, 6:e13, 2017.

[22] Peter de Rivaz e Jack Haughton. Av1 bitstream & decoding process specification. Relatório técnico, The Alliance for Open Media, Junho 2018.

[23] Zhou Wang, A. C. Bovik, H. R. Sheikh, e E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, Abril 2004.

[24] Ben Juurlink, Mauricio Alvarez-Mesa, Chi Ching Chi, Arnaldo Azevedo, Cor Meenderinck, e Alex Ramirez. *Scalable Parallel Programming Applied to H.264/AVC Decoding*. Springer-Verlag New York, 2012.

[25] Harishankar Murugan, Kamisetty Rao, e Nyeon Kim. Multiplexing h.264 video with aac audio bit streams. 22:371–383, Janeiro 2009.

[26] SMPTE 428-2-2006. Matroska and ffv1: One file format for film and video archiving? Relatório técnico, Journal of Film Preservation,o, Abril 2017.

[27] Tim Ryan. Subtitle specification (xml file format) for dlp cinema projection technology. Relatório técnico, Texas Instruments, 2005.

[28] St 428-7:2010 - smpte standard - digital cinema distribution master - subtitle. *SMPTE ST 428-7:2010*, Dezembro 2010.

[29] Rfc 4122 - a universally unique identifier (uuid) urn namespaces, Julho 2005. URL: https://tools.ietf.org/html/rfc4122l.

[30] T. Wiegand, G. J. Sullivan, G. Bjontegaard, e A. Luthra. Overview of the h.264/avc video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, Julho 2003.

[31] St 298:2009 - smpte standard - universal labels for unique identification of digital data. *SMPTE ST 298:2009*, Junho 2009.

[32] Atsc standard: Digital audio compression (ac-3, e-ac-3). Relatório técnico, Advanced Television Systems Committee, Dezembro 2012.