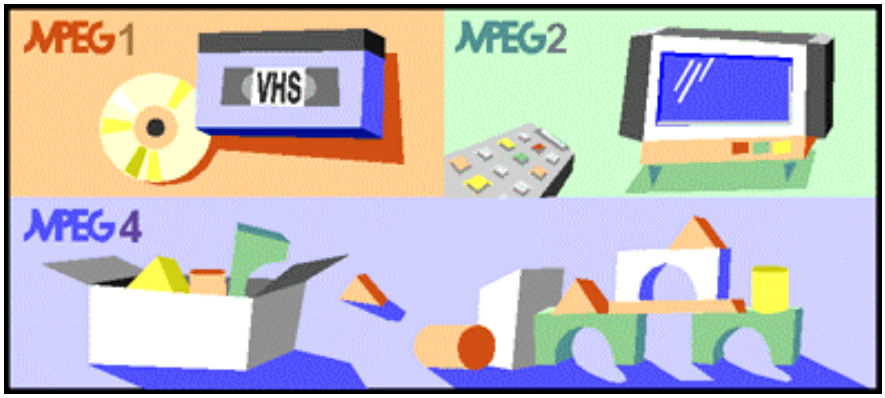
FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

DEPARTAMENTO DE ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

ANÁLISE MPEG



Relatório Final de Projecto

Inês Teixeira de Melo Almeida Santos
Vasco Bernardo Figueiredo Cabral Teles

Coordenação
Engª. Maria Teresa Andrade
Prof. Dr. Eng. Artur Pimenta Alves

Julho 2000

Índice

1. Introdução ao MPEG

MPEG significa Moving Pictures Experts Group, e foi constituído pela ISO, a International Standard Organisation, para trabalhar na área da compressão digital.

A norma MPEG conseguiu uma nova abordagem à standardização da codificação audiovisual, e fornece uma solução de carácter único que vai de encontro às necessidades actuais neste campo, tendo sido o seu primeiro objectivo a definição de um algoritmo de codificação de video para aplicação em armazenamento digital, em particular CD-ROM. Contudo, rapidamente se verificou que existia a necessidade de codificação audio e o campo de acção foi ampliado para tentar definir um algoritmo que pudesse ser utilizado praticamente por todas as aplicações, de sistemas multimédia, a transmissão de televisão, ou aplicações de comunicações como *video-on-demmand*.

Contudo, a ligação a outras organizações de standards não foi esquecida, e em cada reunião MPEG são estudados documentos dessas outras organizações, e elaboradas declarações de cooperação.

1.1 Um pouco de História

O primeiro projecto do Grupo, o MPEG-1, foi publicado em 1993 e consiste num standard de compressão de audio e video e num sistema de multiplexagem para audio e video interlaçado, para que possam ser apresentados em sincronização. Foi aplicado no CD-i e no Video-CD (para video em CD-ROM). Esta norma suporta codificação video com débitos até 1.5Mbit/s, conseguindo uma qualidade semelhante a VHS, e uma qualidade audio de stereo virtual a 192 kbit/s/canal, sendo optimizado para sinais de video não interlaçados.

Ainda em 1990 surgiu a necessidade de um segundo standard de codificação de video a um débito superior num formato interlaçado; foi criado o MPEG-2. É capaz de codificar televisão de qualidade standard com bit-rates de 4 a 9 Mbit/s e televisão de alta definição de 15 a 25 Mbit/s. O MPEG-2 foi publicado em 1995 e desde aí aconteceu um enorme aumento de aplicações e produtos que utilizam este formato digital.

Podemos agora observar a cronologia de algumas das datas mais importantes na história da compressão, à Televisão Digital, passando pelo MPEG, ou pela Televisão de Alta Definição:

- 1970's – Investigação na área da compressão digital conduziu aos algoritmos DCT
- 1988 – Constituído o Motion Pictures Expert Group, MPEG
- 1992 – Combinação entre os formatos MPEG-2 (TV) e MPEG-3 (HDTV)
- 1993 – Definição do Main Profile do MPEG-2, compatível com o MPEG-1
- 1994 – Definição da camada Sistemas do MPEG-2 (ISO 13818-1)
- 1996 – Definição do formato de video 4:2:2
- 1996 – Publicado Conjunto de standards Digital Video Broadcast (DVB)
- 1996 – Demonstração do formato 16:9 em HDTV (1250/50)
- 1996 – EUA adoptam standard de Televisão Digital (DTV) baseado no MPEG-2
- 1997 – Primeiro serviço interactivo do DVB utilizando MPEG-2
- 1998 – Digital Versatile Disk (DVD) utilizando MPEG-2
- 1998 – Active Movie API permitindo que MPEG-2 seja apresentado em computador
- 1998 – Lançamento do serviço DVB-T de televisão terrestre na Grã-Bretanha
- 1998 – Lançamento do serviço DTV nos EUA

1.2 A vantagem do MPEG

A norma MPEG é uma das mais populares técnicas de compressão audio e video pois não é apenas um standard único de compressão, mas sim um leque de ferramentas de codificação standardizadas, que podem ser combinadas flexivelmente para uma utilização adequada a diferentes aplicações, existindo por isso uma descorrelação entre a função de codificação e a aplicação. Uma das razões para este carácter genérico da norma, é a forma em como o MPEG abordou o problema da codificação audiovisual, codificando o video como uma função operando com um vector a três dimensões: duas dimensões da imagem e a terceira relacionada com o tempo, em que apenas são fornecidos parâmetros genéricos associados ao vector. No caso do hardware utilizado,

os standards apenas especificam as funções que o descodificador deve executar com o bitstream recebido.

Com tudo o que foi referido, podemos concluir que ao especificar apenas a sintaxe do bitstream e não a forma como o bitstream é produzido, uma das vantagens do MPEG é a sua abertura a futuros desenvolvimentos e ser susceptível de optimizações sucessivas.

1.3 Compressão

A compressão é a forma de codificar audio e video digital utilizando uma menor quantidade de dados. Surgiu por isso da limitação da capacidade de armazenamento e de largura de banda de transmissão. A compressão tem pois inúmeras vantagens como seja diminuir a largura de banda de transmissão necessária em sistemas que operem em tempo real, ou permitir aplicações de transferência de dados mais velozes que tempo real, ou ainda um formato comprimido poder suportar uma menor densidade de gravação, o que proporciona uma menor sensibilidade a manutenção e a factores ambientais.

No entanto, a compressão não é perfeita, já que tem perdas, no sentido em que o resultado da descodificação não é exactamente igual ao original, havendo uma deterioração da qualidade consoante o formato de compressão utilizado.

1.4 Sistemas de Video

Os sistemas de video composito como o PAL, o NTSC ou o SECAM, são formas de compressão, que utilizam a mesma largura de banda para cor ou para preto-e-branco, no entanto, o MPEG é considerado pelos broadcasters como um formato mais eficiente de substituição destes sistemas.

Para transmissão, podemos escolher entre um débito constante com qualidade variável ou um canal de qualidade constante para um débito variável. Os operadores de redes de telecomunicações preferem um débito constante, como seria de esperar, por razões práticas nos canais. No caso de gravação, como o DVD, é utilizado um débito

variável, que é mais fácil de manipular, por exemplo, acelerando o disco que está a ser lido.

1.5 Codificação

A informação sobre a forma na qual a codificação é efectuada está incluída nos dados comprimidos, para assim ser possível ao descodificador manipular automaticamente todas as "decisões" do codificador.

O MPEG suporta dois tipos de codificação: a *intra-coding*, que é uma técnica que explora a redundância espacial, ou seja, a semelhanças *dentro* da própria imagem, e a *inter-coding*, que explora a redundância temporal, ou seja *entre* imagens consecutivas. A técnica *intra-coding* pode ser utilizada sozinha, como em JPEG, ou é possível ser combinada com a *inter-coding*, como no caso do MPEG.

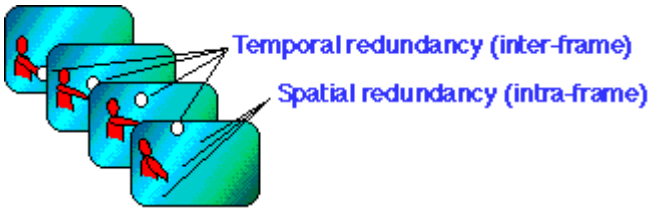


Fig.1.1 – *Inter coding* e *intra coding*

No MPEG são necessários três tipos diferentes de imagens para suportar codificação diferencial e bidireccional e minimização de erros de propagação. São eles: imagens *I*, imagens *P* e imagens *B*. As primeiras são imagens de codificação *intra*, que não necessitam de mais informação do que aquela contida em si própria para serem descodificadas. Como seria de esperar, necessitam de uma maior quantidade de dados que os outros tipos de imagens, e por isso, são apenas transmitidas com frequência baixa: tipicamente a frequência mínima necessária para assegurar os requisitos de acesso aleatório.

Consistem essencialmente em coeficientes de transformada e não contêm vectores de movimento. Este tipo de imagens evita a propagação de erros e permite ao utilizador mudar o canal que está a ser descodificado e visualizado ("channel hopping") com intervalos de tempo relativamente curtos. Em aplicações de difusão de televisão esse intervalo de tempo é usualmente ½ segundo. Depois, temos as imagens P, imagens estimadas de uma imagem anterior, quer seja ela I ou outra imagem P. Os dados de uma

imagem deste tipo consistem de vectores que descrevem a posição da qual, na imagem anterior, cada macrobloco deve ser retirado, e não de coeficientes da transformada que descrevem a correcção que deve ser aplicada a cada macrobloco. Um macrobloco é uma matriz de amostras da imagem original.

As imagens P requerem apenas cerca de metade dos dados necessários a uma imagem I. Finalmente temos as imagens B, que são estimadas a partir de imagens I ou P anteriores ou posteriores à própria imagem. As imagens B são compostas de vectores, tal como as imagens P, mas neste caso referentes a imagens anteriores e posteriores, e contêm também os coeficientes de correcção de movimento, que por serem tão reduzidos, é possível conseguir uma imagem B com cerca de um quarto dos dados de uma imagem I.

Estes tipos de imagens são utilizados para construir o denominado Group of Pictures – GOP – cuja sucessão temporal vai compor a sequência video e o programa. Cada GOP contém normalmente de 12 a 15 imagens e inicia-se com uma imagem I, tendo várias imagens P espaçadas, sendo todas as outras imagens B, com várias combinações. Normalmente cada GOP é caracterizado pelos valores N e M, em que N é o número total de imagens constituintes do GOP e M é o número de imagens P desse GOP. É por exemplo comum em aplicações de difusão de televisão no sistema europeu, utilizar GOPs com N=12 e M=3, isto é GOPs com a combinação IBBPBBP..., isto é, duas imagens B entre cada uma das três imagens P de cada GOP. Podemos observar a constituição de um GOP (com comprimento 9 imagens) e respectivas predições de imagens.

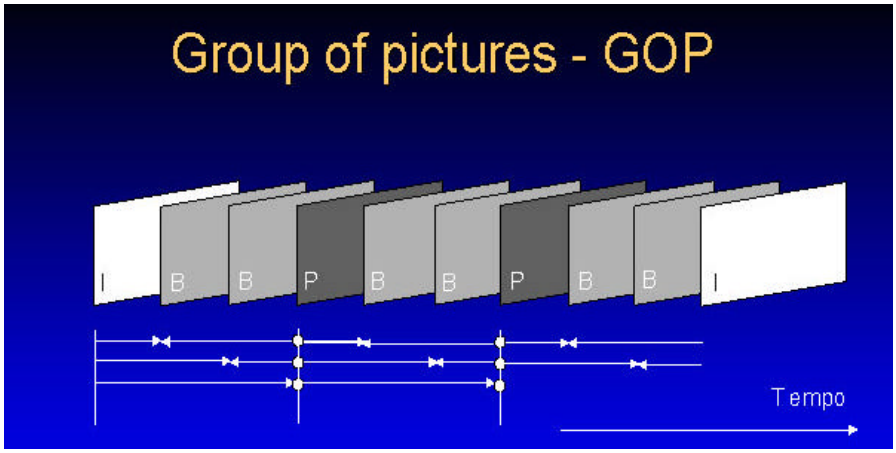


fig.1.2 – *Group of Pictures* de 9 imagens

Como as imagens I têm uma menor qualidade que as outras mas requerem um menor tempo de descodificação e permitem o acesso aleatório (isto é, permitem que o processo de descodificação se possa iniciar no ponto do bitstream onde elas ocorrem), deve ser estabelecido um compromisso entre a qualidade pretendida, o tempo de descodificação e os requisitos de acesso aleatório e re-sincronização.

1.6 Transport Stream

A saída dos codificadores MPEG-audio e MPEG-video é denominada de Elementary Stream. Para uma melhor manipulação pode ser segmentada em blocos de dados, originando o Packetized Elementary Stream – PES.

Estes blocos de dados necessitam de informação suplementar para que seja possível identificar devidamente os dados de video ou audio que transportam, devendo incluir os chamados Time Stamps sem os quais não seria possível obter a sincronização audio/video na apresentação do programa.

Para transmissão e broadcasting, vários programas e respectivos PES podem ser multiplexados numa única stream: a Transport Stream. Esta stream difere da Program Stream (outra forma de multiplexagem que veremos mais adiante) pois na primeira os pacotes PES são subdivididos em pequenos pacotes de comprimento fixo, pacotes TS, nos quais vários programas com diferentes bases de tempo podem ser transportados. Para que isso seja possível, é utilizado um campo no cabeçalho dos pacotes TS, o campo

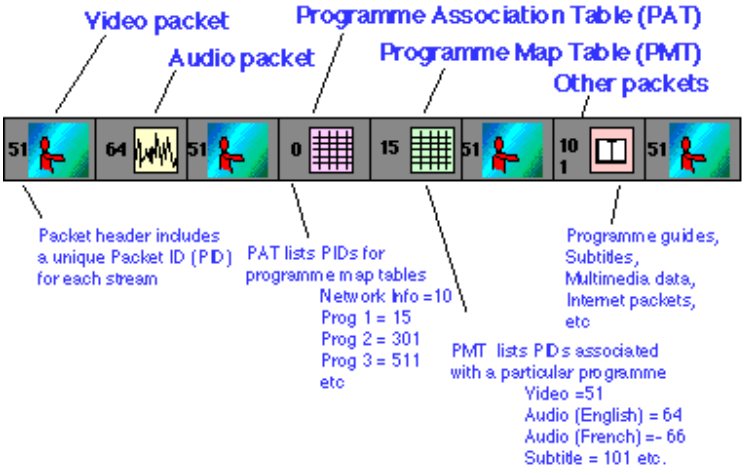


Fig.1.3 – *Transport Stream*

PCR - Program Clock Reference, para transportar amostras da base de tempo de cada programa.

No descodificador, amostras PCR pertencentes a um dado programa, são recolhidas e utilizadas para regenerar uma base de tempo idêntica à base de tempo original utilizada na codificação do programa. É esta base de tempo regenerada (dita "em lock" com a da emissão) que possibilita a apresentação sincronizada de imagens e som, em conjunto com as marcas temporais PTS enviadas nos cabeçalhos dos pacotes PES. Contudo, a Transport Stream é algo mais que apenas uma multiplexagem de audio e video, pois além das pacotes destas streams elementares transporta também metadata. Esta descreve o bitstream incluíndo, entre outra informação, a Program Association Table (PAT) que indica os programas transportados no multiplex, ou a Program Map Table (PMT) para cada um dos programas referidos na PAT. Cada PMT indica a constituição do respectivo programa, fornecendo os identificadores dos pacotes TS (PIDs) que transportam as componentes desse programa (audio, video e dados) e os respectivos tipos das streams (tipo de codificação do audio e video, dados, etc.).

Mais à frente, na explicação do ficheiro de saída do desmultiplexer falaremos mais detalhadamente sobre a Transport Stream e cada elemento relevante para o nosso trabalho.

1.7 Problemas e Falhas

Embora a explicação pareça de alguma forma simples, a Transport Stream no MPEG é uma estrutura bastante complexa, que utiliza tabelas interligadas e identificadores codificados para separar os programas e as streams elementares dentro dos programas em si. Por isso, em tão complexa estrutura é natural que exista a possibilidade de ocorrerem falhas, que podem ser divididas em duas categorias. Na primeira, o sistema de transporte multiplexa e entrega correctamente a informação do codificador ao descodificador sem erros ou *jitter*, mas o codificador ou o descodificador apresenta uma falha. Na segunda categoria, a falha reside no próprio sistema de transporte. É por isso importante o conhecimento da origem da falha e, para essa

origem ser detectada, é necessário verificar se a Transport Stream está de acordo com a sintaxe especificada no MPEG.

Poderão ainda surgir problemas de sincronização, tal como a perda ou erro dos indicadores dessa mesma sincronização originando uma recepção incompleta ou defeituosa da Transport Stream. É possível ocorrerem também erros no protocolo da Transport Stream, o que pode originar problemas na recepção e descodificação dos dados, tal como ser apenas conseguida a imagem mas não o som. Ou ainda, e apesar de uma transmissão correcta dos dados, o *jitter* excessivo pode causar problemas de *timing* (impossibilitar a correcta recuperação da base de tempo). Por tudo isto é necessária e obrigatória uma monitorização e análise constantes, com equipamento adequado.

1.8 O Futuro

Futuramente iremos assistir a uma evolução do MPEG em duas direcções distintas, presenciando uma utilização massiva do formato digital no audiovisual: o desenvolvimento de extensões do MPEG-2 (interfaces em tempo real, entre outras) e o MPEG-4, que é uma norma de codificação por objectos, vocacionada para aplicações a muito baixo débito mas também com especificações para televisão de definição standard e alta definição.

2. Introdução ao MPEG Sistemas

Tal como o MPEG-1, também o MPEG-2 está dividido em várias partes das quais as três principais são: a codificação audio, a codificação video e a gestão de sistema e multiplexagem.

2.1 Definição

A especificação do MPEG Sistemas define como multiplexar várias streams de audio e video numa única stream, como fazer a gestão do buffer no descodificador, como sincronizar as streams no play-back e promover a identificação temporal para cada uma delas.

A definição do MPEG-1 permite que streams elementares que partilhem uma base de tempo comum possam ser multiplexadas num único fluxo utilizando pacotes de tamanho flexível. Esse tamanho é normalmente grande e é escolhido pela própria aplicação. O MPEG-1 é adequado a processamento de software, mas não tem um bom desempenho em ambientes em que os erros sejam comuns. Com o MPEG-2 é atingido um desempenho mais favorável adicionando várias características: conseguindo uma multiplexagem de vários programas com bases de tempo diversas, sendo possível a remultiplexagem, suportando *scrambling*, etc.

2.2 Multiplexagem

A principal função do MPEG Sistemas é fornecer a forma de combinar ou multiplexar, diversos tipos de informação multimédia numa única stream que pode ser transmitida num canal ou armazenada. Existem dois métodos muito utilizados para a multiplexagem de dados de diversas fontes numa única stream. Um é denominado Time Division Multiplexing – TDM – que atribui time slots periódicos a cada uma das streams elementares de audio, video, dados, etc. O outro método também utilizado é

denominado packet multiplexing. Com este método, os pacotes das várias streams elementares são interlaçados uns a seguir aos outros formando uma única stream.

As streams elementares ou as streams MPEG-2 podem ser enviadas a um débito constante –CBR – ou a um débito variável –VBR –, apenas variando o comprimento ou a frequência dos pacotes a enviar. Por exemplo, as elementary streams podem ser enviadas em VBR, mesmo que a stream multiplexada MPEG-2 seja enviada em CBR. Isto possibilita uma outra multiplexagem, a multiplexagem estatística, i.e., quando alguma stream elementar temporariamente não necessita de todos os bits a ela destinados, pode libertar a sua capacidade de canal em proveito de outras streams que temporariamente necessitem mais do que a sua parcela da capacidade de canal.

O MPEG-2 define duas formas de streams de multiplexagem, a Program Stream e a Transport Stream. A primeira é semelhante ao MPEG-1, com algumas características adicionais, como *scrambling, trick modes* (fast-forward, etc.), uma directoria com os conteúdos da multiplexagem e um mapa descrevendo as características das streams que partilham uma base de tempo comum. A Program Stream é utilizada para sistemas de armazenamento e acesso local, onde é importante promover a interactividade, facilitar o processamento de software e onde se admite uma quase total imunidade a erros.

Por sua vez, a Transport Stream é utilizada para sistemas de difusão, nos quais a susceptibilidade ao erro é uma das características mais importantes. Suporta multi-programa com bases de tempo independentes, multiplexados em pacotes de comprimento fixo de 188 bytes. Transporta uma descrição extensa dos conteúdos dos programas no multiplexer e suporta ainda remultiplexagem e *scrambling*. Contudo, não é definido o método de *scrambling*, mas sim o que pode ser *scrambled* e como os dados do controlo de acesso podem ser transmitidos na stream MPEG.

A figura seguinte representa um diagrama de blocos do MPEG Sistemas (a partir da Elementary Stream e até à Program Stream ou Transport Stream) com o processo de codificação, empacotamento e multiplexagem das streams elementares de video e audio até resultar na Program Stream ou na Transport Stream.

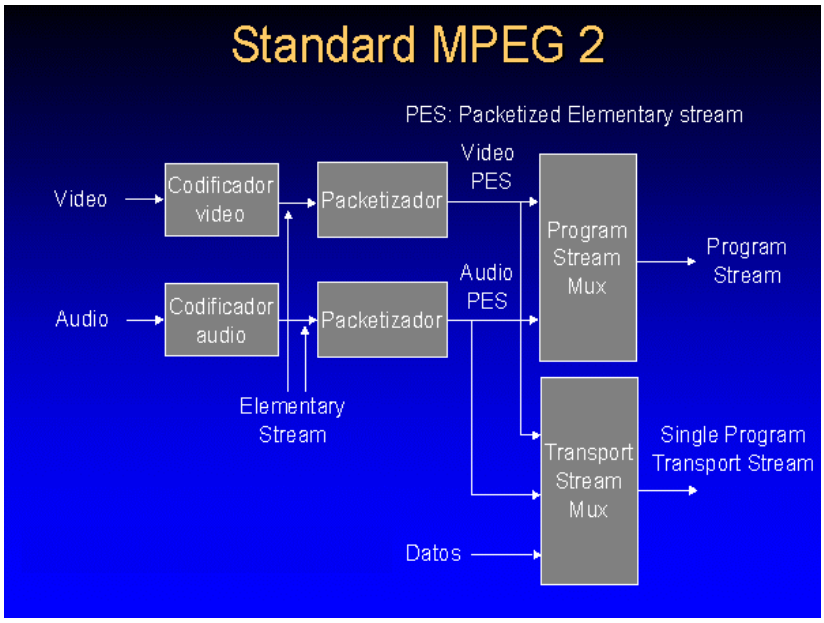


fig.2.1 – Diagrama de blocos do MPEG-2 Sistemas

A transcodificação entre os diferentes formatos MPEG Sistemas é possível e, com as escolhas adequadas de parâmetros, pode ser relativamente simples.

São aqui incluídas definições detalhadas de alguns dos conceitos utilizados, por forma a facilitar a compreensão de alguns aspectos importantes da parte de sistemas da norma MPEG2 cuja abordagem é feita seguidamente.

2.3 Programa

Para tal, começamos pela definição de Programa, que já foi utilizada várias vezes. Quando a norma MPEG refere um programa, significa um único canal ou serviço de broadcast.

2.4 Elementary Stream

Já referimos também a Elementary Stream, que é o nome dado a um único componente codificado digitalmente de um programa, por exemplo video ou audio codificado, podendo um programa conter várias streams elementares, como acontece habitualmente. No caso da transmissão tradicional de televisão, teremos três streams elementares, uma transportando o video codificado, a outra o audio stereo codificado e

a última dados de teletexto. No futuro, as transmissões de televisão irão conter um maior número de streams elementares, o que já começa a acontecer, como seja várias streams de video que transportem diferentes resoluções para a mesma imagem, ou audio e teletexto em diversas línguas, entre outras opções.

2.5 Multiplexador

Interessa também referir o multiplexador. A saída de um multiplexer MPEG-2, é uma stream contínua de dados. Não há restrições quanto ao débito do multiplexer, mas deverá ser pelo menos igual à soma dos débitos das várias streams elementares, e pode ser de débito fixo ou variável, tal como as streams.

Adicionalmente às streams elementares, pode ser incluída no multiplexer uma grande variedade de informação. Por exemplo, um sistema de *time stamps* relacionado com o *clock reference* assegura que streams relacionadas entre si sejam apresentadas em sincronismo no descodificador, como veremos mais adiante. Temos também tabelas de informação de serviço, incluindo parâmetros detalhados da rede, pormenores dos programas dentro do multiplexer e a natureza das várias streams elementares. Podemos também encontrar o suporte para o controlo de *scrambling*, para acesso condicional a uma ou mais streams elementares, apesar de ser de referir que o MPEG não especifica nem o algoritmo de scrambling nem o controlo de acesso.

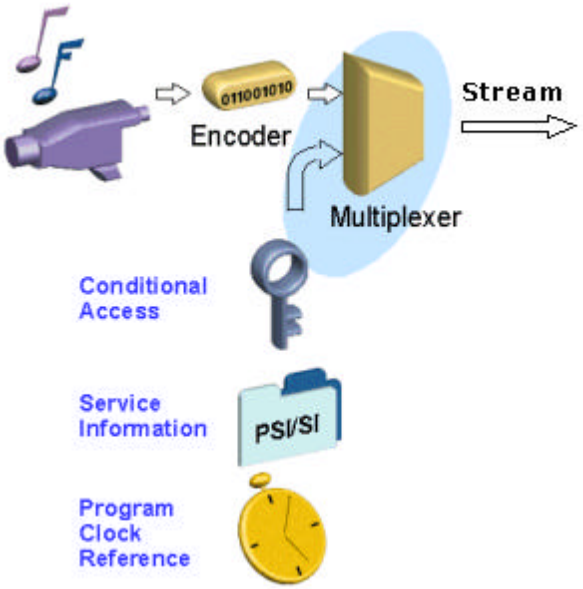


fig.2.2 – Informação de entrada no multiplexer

Poderá ser acomodado ainda um qualquer número de canais com dados privados, que são streams de dados que não são especificados pelo MPEG, e podem ser utilizadas para transportar serviços de dados como teletexto, informação adicional de

serviço específica a uma dada rede, comandos para controlar a modulação e o equipamento de distribuição da rede, ou qualquer outro tipo de dados requeridos a uma aplicação particular.

No entanto, existem algumas características importantes não definidas pelo MPEG, como seja a não existência de qualquer forma de protecção contra erro dentro do multiplexer, pois a protecção contra erro e a modulação da multiplexagem MPEG-2 são escolhidas consoante as características do canal ou do armazenamento, e não são por isso especificadas pela norma. Ou, por exemplo, não existirem definições eléctricas ou físicas para o multiplexer MPEG-2, para que o projectista possa utilizar os níveis de sinal ou os tipos de conectores que melhor se adequem à aplicação desejada.

2.6 Program Stream e Transport Stream

Como referimos anteriormente, o MPEG-2 Sistemas define duas multiplexagens alternativas, a Program Stream e a Transport Stream, cada uma sendo optimizada para um conjunto diferente de aplicações, que iremos seguidamente descrever um pouco mais detalhadamente

A program stream é baseada na multiplexagem estabelecida do MPEG-1, e tal como esta, pode apenas acomodar um único programa, sendo adequada para o armazenamento e recuperação de programas de um DSM (digital storage medium). Uma program stream deverá ser utilizada num meio livre de erros, porque é bastante susceptível a que estes aconteçam. Existem duas razões para este facto: a primeira é devida ao facto da program stream conter uma sucessão de pacotes relativamente longos com comprimentos variáveis em que cada pacote começa com um cabeçalho, e ao ocorrer um erro nesse cabeçalho, o pacote pode ser perdido, podendo causar a perda de uma imagem. Depois, como o comprimento dos pacotes pode variar, o descodificador não pode prever o fim de um pacote e o início de outro, e para isso tem de ler e interpretar o campo que indica o comprimento daquele, no cabeçalho do mesmo. Se acontecer um erro neste campo, o descodificador perderá o sincronismo com a stream, o que poderá resultar na perda de, pelo menos um pacote.

Por sua vez, a Transport Stream foi idealizada para aplicações multi-programa, tal como o broadcasting. Uma única stream pode acomodar vários programas independentes e é constituída por uma sucessão de pacotes de 188 bytes de comprimento denominados *transport packets*. Ao utilizar pacotes pequenos e de comprimento fixo, a transport stream é menos susceptível a erros que a program stream, além de cada pacote poder também ter protecção contra erros simplesmente passando através de um processo de controlo de erro, de que é exemplo a codificação Reed-Solomon. Assim, o melhor desempenho contra erros significa que a transport stream funciona melhor em canais susceptíveis a erros, como o broadcast.

Visto isto, poderá parecer que a transport stream é de longe a melhor das duas alternativas de multiplexagem, com melhor desempenho contra erro e a possibilidade de transportar programas simultâneos. Contudo, é de recordar que a transport stream é uma multiplexagem mais sofisticada que a program stream, e por isso mais difícil de criar e de desmultiplexar. Cada uma das alternativas de multiplexagem deve então ser escolhida consoante a aplicação para a qual irá ser utilizada.

2.7 Packetised Elementary Stream

Para criar uma multiplexagem MPEG-2, é necessário converter cada stream elementar numa *Packetised Elementary Stream* – PES – que consiste em pacotes PES. Estes pacotes são constituídos por um cabeçalho e uma secção de payload, que é composta apenas por bytes de dados retirados sequencialmente da stream elementar original.

Os pacotes PES podem ter um comprimento variável, até um máximo de 64kBytes. O projectista de um multiplexer MPEG-2 pode utilizar esta flexibilidade de várias formas, como por exemplo, utilizar pacotes PES de comprimento fixo ou variar esse mesmo comprimento de forma a alinhar o início das imagens codificadas com o início do payload dos pacotes PES.

2.8 Pacotes de Transporte

Numa transport stream, os pacotes de transporte têm sempre um comprimento de 188 bytes, contendo 4 bytes de cabeçalho, seguidos do Adaptation Field, ou do payload, ou de ambos. Os pacotes PES das várias streams elementares são divididos pelos payloads de um certo número de pacotes de transporte. Contudo, é improvável que um PES-packet preencha exactamente, e por isso o último pacote de transporte poderá ficar incompleto. Então, para que a transport stream seja contínua, é utilizado um Adaptation Field para ocupar esse espaço em excesso. Para minimizar este "gasto" de bytes, o comprimento do pacote PES deve ser cuidadosamente escolhido.

Na figura podemos observar o esquema de um pacote de transporte, e respectivo esquema de cabeçalho.

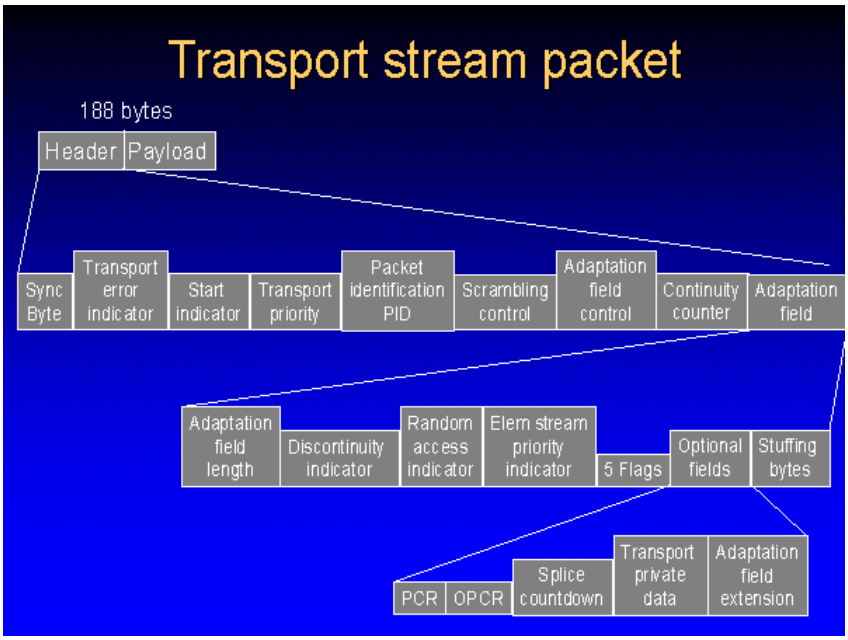


fig.2.3 –Esquema de um pacote de transporte

Como referimos, cada pacote de transporte começa com um cabeçalho de 4 bytes, contendo vários campos, sendo quatro dos quais particularmente importantes. O primeiro byte do cabeçalho é o sync_byte. O valor deste campo é 47h e, por ocorrer cada 188 bytes na transport stream, permite que o descodificador identifique esta repetição e assim reconheça o início de cada novo pacote de transporte. De seguida encontramos o *Packet Identifier* – PID – que serve para distinguir os pacotes de

transporte de uma dada stream elementar daqueles doutra stream, pois uma transport stream pode transportar diversos programas, cada qual contendo várias *packetised elementary streams*. É da responsabilidade do multiplexer assegurar que seja atribuído um valor único de PID a cada stream elementar. Depois podemos observar o campo `payload_unit_start_indicator`, que serve para indicar algo em especial relacionado com o primeiro byte do payload, como por exemplo se esse primeiro byte é também o primeiro byte de um pacote PES. Por último, o campo `continuity_count` é incrementado por cada pacote de transporte sucessivo pertencente à mesma stream elementar. Este facto serve para o descodificador poder detectar a perda ou ganho de um pacote de transporte e tentar corrigir o erro que resultaria de tal facto.

2.9 Presentation Unit e Access Unit

Em MPEG cada imagem não comprimida é denominada Presentation Unit. O codificador comprime cada presentation unit originando uma imagem codificada denominada Access Unit. As access units de video não são todas do mesmo tamanho, dependendo do tipo de codificação ser I, P ou B, e da dificuldade de codificação dessa mesma imagem, como já vimos atrás.

O resultado do MPEG codificar uma sequência video é uma sucessão de video access units, que forma a stream elementar de video, o mesmo acontecendo para o audio. Normalmente, cada access unit de audio contém apenas alguns milisegundos de sinal comprimido.

2.10 Time Stamps

No final, o descodificador sabe quando descodificar cada access unit através dos chamados *Time Stamps*, que indicam explicitamente ao descodificador o momento exacto no qual a access unit deve ser retirada do buffer e descodificada. Se o descodificador utilizar convenientemente os time stamps, então estará a descodificar access units em sincronismo com as outras strings elementares que compõem o programa e assim o buffer nunca sofrerá um overflow ou underflow.

Um time stamp é um valor representando um tempo. Uma função do multiplexer é atribuir time stamps às access units à medida que estas são produzidas pelo codificador. Para o codificador poder atribuir *time stamps*, tem de ter uma referência exacta de relógio, que é fornecida pelo multiplexer. Quando uma nova access unit é gerada pelo codificador, o time stamp é atribuído baseado nessa referência de relógio, indicando o momento em que a access unit será descodificada. Para que isso aconteça com precisão, o descodificador deverá ter um relógio em sincronismo com o do multiplexador. Isto é conseguido incluindo amostras regulares da referência de relógio na multiplexagem efectuada.

Existem dois tipos de time stamps: os Decoding Time Stamps – DTS – que indicam quando a access unit deve ser descodificada, e os Presentation Time Stamps – PTS – que indicam quando deve ser apresentada na saída ou no display.

2.11 Program Specific Information

Numa transport stream, cada pacote de transporte está etiquetado com um PID indicando a que stream elementar pertence o seu payload. Como sabemos, pode haver várias streams elementares de vários programas diversos, por isso a forma como o descodificador consegue determinar qual stream pertence a que programa é incluindo informação adicional na transport stream, que especifique a relação entre os programas disponíveis e os PIDs das suas streams elementares. Esta informação é denominada "Program Specific Information" – PSI – e tem de estar presente em todas as streams de transporte.

Este tipo de informação especificada para a camada Sistemas do MPEG-2 contém quatro tipos de tabelas: a Program Association Table – PAT –, a Program Map Table – PMT –, a Network Information Table – NIT – e a Conditional Access Table – CAT. Cada uma destas tabelas pode ser transportada como payload de um ou mais pacotes de transporte e por isso acomodada numa stream de transporte. Veremos mais adiante em mais detalhe numa situação prática como funcionam a PAT e a PMT, aquando da explicação do ficheiro do desmultiplexer.

2.11.1 PROGRAM ASSOCIATION TABLE

A Program Association Table contém uma lista de todos os programas disponíveis numa stream de transporte, assim como o valor do PID dos pacotes de transporte que contêm a Program Map Table de cada um desses mesmos programas, além de outras informações. Esta tabela apresenta sempre um PID de valor 0x 0000 e é por isso facilmente identificável.

2.11.2 PROGRAM MAP TABLE

Cada programa transportado na transport stream tem uma Program Map Table a si associada. Esta tabela apresenta os detalhes relativos ao programa em si e à stream elementar que o compõe. À PMT mais simples pode ser acrescentada informação sobre descritores especificados no MPEG-2 Sistemas. Eles transportam informação acerca de um programa ou das suas streams elementares. Podem ainda incluir parâmetros de codificação video e audio, identificação da língua, informação "pan-and-scan", detalhes do acesso condicional, informação de direitos de autor, entre outros, como descritores privados para um determinado broadcaster.

2.11.3 NETWORK INFORMATION TABLE

Na PAT, o programa número zero é utilizado para indicar a Network Information Table. Esta tabela é opcional e os seus conteúdos são privados, i.e., são definidos pelo broadcaster ou pelo utilizador e não pelo MPEG. Quando está presente, esta tabela fornece informação acerca da rede que transporta a Transport stream, tal como as frequências dos canais, detalhes de satélite, características de modulação, origem do serviço, nome do serviço e detalhes de redes alternativas que estejam disponíveis.

2.11.4 CONDITIONAL ACCESS TABLE

Finalmente temos a Conditional Access Table. Se qualquer stream elementar numa transport stream está *scrambled*, então a CAT deve estar presente para fornecer detalhes do sistema de *scrambling* em uso e os PIDs dos pacotes de transporte que contenham a informação da gestão e a autorização ao acesso condicional. O formato

desta informação não é especificado pelo MPEG-2 Sistemas por depender do sistema de *scrambling* utilizado.

Voltaremos a falar do MPEG-2 Sistemas um pouco mais à frente, numa situação mais prática, quando explicarmos o ficheiro de saída do desmultiplexer e respectiva sintaxe.

3. Em que consiste o trabalho

A ideia que nos foi apresentada pela Engª. Maria Teresa Andrade e pelo Prof. Artur Pimenta Alves era interessante e consistia em idealizar e concretizar um programa que conseguisse ler o ficheiro de texto de saída de um desmultiplexador, para de seguida separar a informação de diversos tipos contida nesse mesmo ficheiro, como sejam os dados contidos nos pacotes de audio e de video, de forma a ser possível compor os programas transportados e as tabelas PSI, e que contêm não só o número dos programas, mas também outros dados como a codificação a que estão sujeitos, a prioridade ou o identificador dos pacotes que transportam o audio e o video.

Concluída esta primeira fase de execução do programa, seria depois possível analisar toda a informação retirada, com a ajuda de uma interface gráfica para uma utilização simples. No caso da Program Specific Information, essa informação ficaria contida num ficheiro de texto para o qual tinha sido escrita na análise prévia e colocada numa janela aberta na interface gráfica.

Depois seria possível ouvir o audio e visualizar o video, em conjunto ou em separado, cabendo ao programa correr uma aplicação já existente (por exemplo Quick Time ou Media Player), depois de o programa ter seleccionado o ficheiro desejado através de um conjunto de regras definidas por um programa contido no desmultiplexer.

Para ser possível ao utilizador efectuar todas estas operações e executar as funções suportadas pelo programa, seria disponibilizada uma interface gráfica, como já foi referido, que permitisse uma utilização simples e eficiente (*user-friendly*) das capacidades do programa, como seja abrir o ficheiro desejado, analisá-lo, imprimi-lo, ou ainda visualizar e imprimir as tabelas PSI e correr os ficheiros de audio e video.

4. Preliminares

Depois de nos ter sido proposto o Projecto, havia ainda um longo trabalho pela frente antes de realmente começarmos a implementar o programa. Tal como para qualquer trabalho, havia antes que fazer uma fase de investigação e estudo, não só para compreender o ficheiro gerado pelo desmultiplexer, mas também estudar a linguagem na qual iríamos programar, ou escolher o programa compilador.

Ficou decidido com a Engª. Maria Teresa Andrade que a melhor linguagem de programação para a aplicação seria o Java, por ser orientada a objectos, e por isso mais facilmente utilizável, e ainda por ser uma linguagem eficiente para aplicações na Internet, algo que seria para ser executado numa fase posterior ao próprio Projecto.

4.1 Java

Começámos então por estudar Java durante o mês de Abril. Para isso utilizámos várias livros e documentos de sites na Internet. No entanto, deparámo-nos com dificuldades várias devido a nunca termos tido uma boa disciplina de programação durante o Curso, por isso era quase começar do "zero" até adquirirmos conhecimentos de Java tão profundos que nos permitissem conhecer em pormenor cada classe e respectivos métodos, e ainda funcionalidades de programação gráfica.

4.2 MPEG

De seguida, fomos avançando no estudo da norma MPEG, mais precisamente da Camada Sistemas, para obtermos conhecimentos necessários à concretização do trabalho, conhecimentos que foram também sendo adquiridos na disciplina de Televisão Digital

Seguidamente começámos a estudar o ficheiro gerado pelo desmultiplexador. Algo que parecia um pouco confuso, com tantos campos, valores e definições, mas que

aos poucos começou a ficar mais claro à medida que íamos avançando na investigação nos livros e com os esclarecimentos da nossa orientadora.

Depois de compreendermos toda a estrutura e respectivos pormenores do referido ficheiro, bem como os campos mais importantes para o objectivo do nosso trabalho, havia que começar a esboçar o algoritmo do programa que iríamos implementar. Um pouco mais à frente apresentamos a explicação do ficheiro gerado pelo desmultiplexador, de forma a ser mais acessível a compreensão do algoritmo idealizado, que também explicamos.

4.3 A Programação

Para a programação em Java era possível a utilização de diferentes programas compiladores. Optou-se por utilizar o "Forte for Java" da Sun microsystems, pois era aquele mais robusto e além de ser compilador, suporta ainda funcionalidades gráficas bastante úteis. Para isso fizemos o download do programa a partir do site da Sun microsystems.

No entanto, à medida que fomos avançando na implementação do nosso programa, deparámo-nos com algumas falhas no Forte, o que nos levou a utilizar em simultâneo o Ultra Edit, compilador mais simples e com funcionalidades de ajuda bastante úteis. Começámos por aprender a utilizar o Forte, que é um programa não muito complexo do ponto de vista do utilizador, mas extremamente completo, o que o pode tornar um pouco confuso.

4.4 Condições de Trabalho

Para ser possível uma utilização eficiente do Forte, tínhamos necessidade de trabalhar num computador com uma capacidade apreciável, visto aquele ser um programa bastante pesado. Era totalmente impossível trabalhar na Faculdade, antes de mais por lá não ser possível ter um computador "fixo" e depois por não ser possível correr o programa em rede. Para solucionar este novo obstáculo, a nossa orientadora disponibilizou-nos um computador no INESC, algo que foi decisivo para o desenvolvimento e concretização do Projecto.

5. Explicação do desmultiplexador

Como vimos anteriormente, o multiplexador desempenha um papel chave em todo o sistema MPEG. Pudemos observar o seu papel na multiplexagem das várias streams, pois não se transmite apenas um único programa, mas sim vários que são multiplexados; na definição do débito da stream multiplexada e do número de canais de transmissão, factores que condicionam o desempenho de todo o sistema; ao introduzir os *time stamps*, que irão sincronizar a apresentação das várias streams video e audio; ao introduzir informação sobre parâmetros da rede ou sobre os vários programas a serem transportados; ou ainda, ao fornecer o suporte para *scrambling* de acesso condicionado, entre muitas outras funções desempenhadas pelo multiplexer.

Com tudo o que foi referido, podemos ver também a enorme importância do desmultiplexador no sistema. Sem ele, toda a informação construída pelo multiplexador seria em vão, tal como não seria possível a apresentação dos programas transmitidos.

O desmultiplexador tem várias saídas - audio, video, dados - e é nesta última que obtemos o ficheiro de informação que está na base do nosso trabalho. O desmultiplexer contém um programa que recebe um ou mais ficheiros binários que transportam o bitstream multiplexado de acordo com a sintaxe da transport stream, já antes referida. Gera então um ficheiro de texto, com a informação que retira dos campos dos cabeçalhos dos pacotes da transport stream, tal como do payload dos pacotes que transportam as tabelas PSI. Quantos às streams elementares – audio e video – são recuperadas e colocadas em ficheiros separados, para depois seguirem para os respectivos descodificadores.

Entrando agora em maior detalhe, as Transport Streams são descodificadas por um Transport Demultiplexer, que inclui um mecanismo de extracção de relógio, desempacotadas por um Depacketizer, e finalmente, enviadas para descodificação audio e video nos respectivos descodificadores. Os sinais descodificados são enviados para as

respectivas Buffer e Presentation Units que, por sua vez, os enviam para as saídas do display e do altifalante, no tempo correcto de sincronização. De forma idêntica, se forem utilizadas Program Streams, estas serão descodificadas por um Program Stream Demultiplexer e Depacketizer e enviadas para descodificação nos descodificadores audio e video. Tal como em MPEG-1 Sistemas, a informação relativa ao timing está contida nos campos do Clock Reference, que são utilizados para sincronizar o descodificador Systems Time Clock – STC. A apresentação da saída descodificada é controlada pelos Presentation Time Stamps, que são também transportados pelo bitstream.

5.1 Descodificador

Os descodificadores de Sistemas têm que ter em atenção vários tipos de fraquezas da transmissão e multiplexagem que são desconhecidos na altura da codificação das streams elementares. Exceptuando erros de transmissão e perda de pacotes, com os quais os descodificadores PES têm que lidar, o pior efeito da transmissão para os descodificadores de Sistemas é aquele devido à variação do atraso na transmissão, denominado *jitter*, e que origina erros nos SCRs e PCRs recebidos, em proporção à variação daquele mesmo atraso. Para suavizar os efeitos dos erros nos PCRs, os descodificadores utilizam Phase Locked Loops, para recuperar STC estáveis.

Na figura seguinte, podemos observar o diagrama de blocos do descodificador, incluíndo e desmultiplexador, o buffer de entrada e a DCT Inversa.

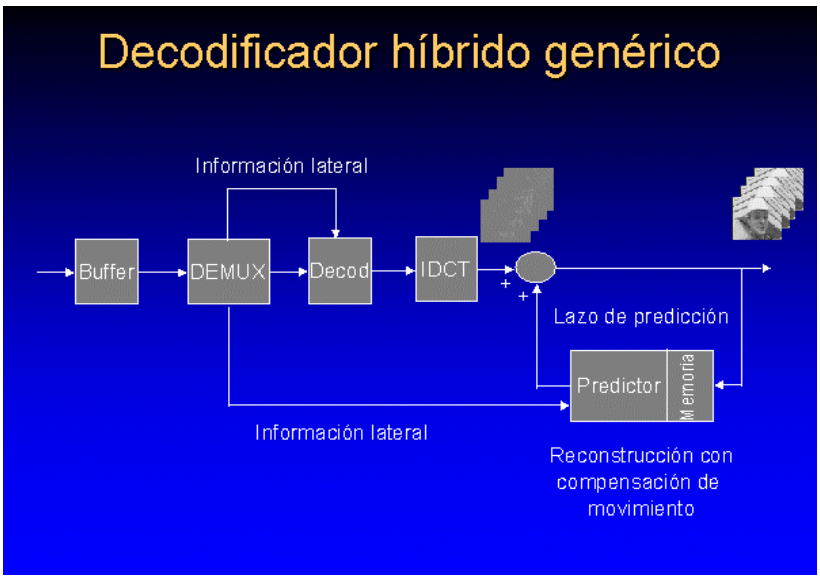


fig.5.1 – Diagrama de blocos do descodificador

5.2 Ficheiro de saída do desmultiplexer

Como foi referido, uma das saídas de dados do desmultiplexador fornece um ficheiro de texto que possui informação variada, e que está na base do programa do nosso Projecto.

Como foi explicado, o ficheiro contém Program Association Tables - PAT's - que fornecem, entre outra, informação sobre o número de programas transportados na stream, a indicação de programas retirados ou inseridos e quando estes estão activos, o identificador dos pacotes - PID's - das várias Program Map Tables - PMT's - dos diferentes programas. Das PMT's, obtemos, entre outra informação, o tipo de streams de video e de audio e respectivos identificadores dos pacotes onde estão contidos os dados dos programas. Além desses dados, alguns pacotes de audio e de video contêm também informação detalhada sobre a Packetized Elementary Stream - PES - e sobre os time stamps, de que falaremos mais à frente. Todos os pacotes possuem um cabeçalho no qual está contida informação sobre erro, a prioridade do programa, eventual *scrambling*, entre outra.

5.2.1 FICHEIRO

Analisando com maior profundidade o ficheiro de saída do desmultiplexer, podemos observar que no início encontramos a Program Association Table, cujo identificador - PID - é sempre 0x 0000.

5.2.2 CABEÇALHO DOS PACOTES

Seguidamente encontramos o cabeçalho que tem a mesma estrutura para todos os pacotes, mas no qual os valores dos campos podem ser alterados, consoante o estado do programa seja ser alterado de pacote para pacote. Os campos mais importantes para o nosso trabalho são o **`transport_error_indicator`**, que indica se o pacote tem algum erro, alterando o valor de 0 para 1 quando o erro é encontrado, o **`transport_priority`**, que indica a prioridade do programa, e que o valor 1 no bit indica uma prioridade maior que os outros pacotes, e o **`transport_scrambling_control`**, que indica se o programa é codificado e o modo de

codificação. Podemos observar um exemplo de um cabeçalho dos pacotes de uma transport stream na figura seguinte.

```
| transport_error_indicator = 0
| payload_unit_start_indicator  = 1
| transport_priority            = 0
| transport_scrambling_control  = 0
| adaptation_field_control      = 1 (payload only)
| continuity_counter = 0 (counter of payloads with the same PID)
```

fig.5.2 - cabeçalho de pacote de uma transport stream

5.2.3 PROGRAM ASSOCIATION TABLE

A seguir ao cabeçalho encontramos a Program Association Section, que contém a informação da PAT, da qual podemos observar um exemplo na figura a seguir. Para o nosso trabalho, a informação mais relevante diz respeito à **version_number** e ao **current_next_indicator**.

```
|PROGRAM ASSOCIATION SECTION**********
| pointer_field = 0 bytes
| table_id = 0
| section_syntax_indicator = 1
| section_length = 25
| transport_stream_id = 2048
| version_number = 0
| current_next_indicator = 1
| section_number = 0
| last_section_number = 0
|
| program_number = 1
| program_map_PID = 0x1010
|
| program_number = 2
| program_map_PID = 0x1011
|
| program_number = 3
| program_map_PID = 0x1012
|
| CRC_32 = 0
```

fig.5.3 – informação da PAT

Podemos ainda observar que é apresentada a informação acerca dos programas, i.e., o seu número e o PID da respectiva Program Map Table.

No exemplo podemos observar que o **current_next_indicator** tem valor 1. Isto significa que os programas referidos pela PAT estão actualizados e prontos para serem transmitidos. Se houver alguma alteração nos programas, o campo **version_number** será incrementado, e o **current_next_indicator** terá valor 0. Apenas quando esta flag voltar a ter valor 1 é que a actualização dos programas estará activa.

O valor do campo **program_map_PID**, apresentado em hexadecimal, indica o identificador dos pacotes que contêm a PMT de cada um dos programas. A lista dos programas e respectivos endereços dos pacotes de PMT termina com o campo CRC_32 =

0. No final do pacote da PAT, tal como nos outros pacotes, é possível ainda encontrar bytes de stuffing. Como o nome indica, estes bytes servem para totalizar os 188 bytes, que é o tamanho de cada pacote, quer seja de PAT, de PMT ou mesmo de dados de video ou de audio.

A seguir ao primeiro pacote da Program Association Table poderá vir outra PAT, se tiver surgido uma eventual alteração (como a adição de um novo programa) caso em que o **version_number** e o **current_next_indicator** têm o comportamento atrás referido. É possível a seguir à PAT aparecerem pacotes nulos, que serão explicados mais adiante, ou então, o pacote da Program Map Table, cujo cabeçalho tem uma estrutura idêntica a todos os outros pacotes, já como referido.

5.2.4 PROGRAM MAP TABLE

Na figura seguinte apresentamos a Program Map Section de uma PMT de um programa.

```
PROGRAM MAP SECTION*************************
| pointer_field = 0 bytes
| table_id                 = 2
| section_syntax_indicator = 1
| section_length = 23
| program_number = 1
| version_number          = 0
| current_next_indicator = 1
| section_number          = 0
| last_section_number     = 0
| PCR_PID = 0x0100
| program_info_length = 0
|
| stream_type = 2   -> ITU-T Rec.H262 | ISO/IEC 13818-2 Video
| elementary_PID = 0x0100
| ES_info_length = 0
|
| stream_type = 3   -> ISO/IEC 11172 Audio
| elementary_PID = 0x0101
| ES_info_length = 0
|
| CRC_32 = 0
```

fig.5.4 - informação da PMT

Podemos observar que são também apresentados os campos **version_number** e o **current_next_indicator** e que apresentam o comportamento descrito anteriormente. De seguida temos as definições do video e do audio e respectivos identificadores dos pacotes que os transportam. O campo **stream_type = 2** indica que a stream é de video com codificação MPEG-2, e o campo **stream_type = 3** representa

audio com codificação MPEG-1. Estes campos indicam o tipo da stream elementar e podem ter variados valores, como podemos observar na tabela seguinte.

Stream_type	Descrição
0×00	Reservado
0×01	MPEG-1 Video
0×02	MPEG-2 Video
0×03	MPEG-1 Audio
0×04	MPEG-2 Audio
0×05	Private_sections
0×06	Pacotes PES contendo streams privadas
0×07	MHEG
0×08	DSM CC
0×09	ITU-T Rec. H.222.1
0×0A	ISO/IEC 13818-6 type A
0×0B	ISO/IEC 13818-6 type B
0×0C	ISO/IEC 13818-6 type C
0×0D	ISO/IEC 13818-6 type D
0×0E	Auxiliar
0×0F - 0×7F	Reservado
0×80 - 0×FF	Utilizador Privado

Fig.5.5 – tipos de stream elementar

A seguir ao **stream_type** temos o **elementary_PID**, que nos fornece o identificador dos pacotes de video e de audio.

5.2.5 PACOTE DE VIDEO

Continuando a seguir o ficheiro, encontramos o primeiro pacote de video, com o PID indicado na PMT, como referido, e com o cabeçalho como explicado anteriormente. Seguidamente é possível encontrar o Adaptation Field, que contém informação sobre o Program Clock Reference, entre outra. Para que o desmultiplexer o reconheça, o campo **adaptation_field_control** no cabeçalho do pacote deve ter valor igual a 3 para indicar um adaptation field seguido de payload. Dentro do Adaptation Field temos uma flag com bastante interesse, a **PCR_flag,** que indica a presença do Program Clock Reference (PCR) se tiver o valor 1. O Adaptation Field é utilizado também para "gastar" espaço em excesso no pacote.

5.2.5.1 PES Packet

Dentro do pacote de video temos o Program Elementary Stream - PES - Packet, que contém os dados de video do programa, isto é, o payload. O primeiro PES Packet a seguir à PMT tem o cabeçalho de PES que contém informação, entre outra, sobre os

Presentation Time Stamps - PTS - e os Decoding Time Stamps - DTS. A flag **PTS_DTS_flags** pode apresentar três valores diferentes: 3, 2 ou 0, correspondendo a estar presente no cabeçalho o DTS e o PTS, somente o PTS ou nenhum dos dois.

Podemos observar na figuras seguinte o primeiro pacote de video a ser encontrado a seguir à PMT, com o Adaptation Field e o PES Packet.

```
-----------------------------------------------
| PID = 0x0100
| transport_error_indicator = 0
| payload_unit_start_indicator  = 1
| transport_priority            = 0
| transport_scrambling_control  = 0
| adaptation_field_control      = 3 (adaptation_field followed by
payload)
| continuity_counter = 0 (counter of payloads with the same PID)
|ADAPTATION FIELD*****************************
| adaptation_field_length = 7
| descontinuity_indicator = 0
| random_access_indicator = 1
| ES_priority_indicator   = 0
| PCR_flag                        = 1
| OPCR_flag                       = 0
| splicing_point_flag             = 0
| transport_private_data_flag     = 0
| adaptation_field_extension_flag = 0
|
| PCR_base = 2
| PCR_extension = 120     (PCR = 2.66667e-05 s)
|
| 0 bytes left - 0 stuffing_bytes (0xFF) detected
|PES PACKET***********************************
| stream_id = 0xE0  ->  Video stream - number 0
| PES_packet_length = 5527 bytes
| PES_scrambling_control    = 0
| PES_priority              = 0
| data_alignment_indicator  = 0
| copyright                 = 0
| original_or_copy          = 0 (copy)
| PTS_DTS_flags             = 3
| ESCR_flag                 = 0
| ES_rate_flag              = 0
| DSM_trick_mode_flag       = 0
| additional_copy_info_flag = 0
| PES_CRC_flag              = 0
| PES_extension_flag        = 0
| PES_header_data_length = 10
|
| PTS = 8100 (0.09 s)
| DTS = 4500 (0.05 s)
| (157 data_bytes)
-----------------------------------------------
```

fig.5.6 – primeiro pacote de video do programa

Na figura seguinte observamos um pacote de video genérico.

```
-----------------------------------------------
| PID = 0x0100
| transport_error_indicator = 0
| payload_unit_start_indicator  = 0
| transport_priority            = 0
| transport_scrambling_control  = 0
| adaptation_field_control      = 1 (payload only)
| continuity_counter = 1 (counter of payloads with the same PID)
|PES PACKET**********************************
| (184 data_bytes)
 -----------------------------------------------
```

fig.5.7 – pacote de video

5.2.6 PACOTE DE AUDIO

Por cada N pacotes de video, é transmitido um pacote de audio. Estes têm o PID indicado na PMT, tal como acontece com o video. A seguir ao cabeçalho encontramos o PES Packet, que contém também os dados de audio do programa. Se for o primeiro pacote de audio a ser encontrado a seguir à PMT, o PES Packet tem cabeçalho onde podemos encontrar informação sobre o PTS e o DTS, verificando a flag **PTS_DTS_flags** como foi explicado atrás . Na figura seguinte apresentamos o primeiro pacote de audio do programa, e a seguir um pacote de audio genérico.

```
-----------------------------------------------
| PID = 0x0101
| transport_error_indicator = 0
| payload_unit_start_indicator  = 1
| transport_priority            = 0
| transport_scrambling_control  = 0
| adaptation_field_control      = 1 (payload only)
| continuity_counter = 0 (counter of payloads with the same PID)
|PES PACKET**********************************
| stream_id = 0xC0  ->  Audio stream - number 0
| PES_packet_length = 968 bytes
| PES_scrambling_control    = 0
| PES_priority              = 0
| data_alignment_indicator  = 0
| copyright                 = 0
| original_or_copy          = 0 (copy)
| PTS_DTS_flags             = 2
| ESCR_flag                 = 0
| ES_rate_flag              = 0
| DSM_trick_mode_flag       = 0
| additional_copy_info_flag = 0
| PES_CRC_flag              = 0
| PES_extension_flag        = 0
| PES_header_data_length = 5
|
| PTS = 6660 (0.074 s)
| (170 data_bytes)
 -----------------------------------------------
```

fig.5.8 – primeiro pacote de audio

```
-----------------------------------------------
| PID = 0x0101
| transport_error_indicator = 0
| payload_unit_start_indicator  = 0
| transport_priority            = 0
| transport_scrambling_control  = 0
| adaptation_field_control      = 1 (payload only)
| continuity_counter = 1 (counter of payloads with the same PID)
|PES PACKET*********************************
| (184 data_bytes)
 -----------------------------------------------
```

fig.5.9 – pacote de audio

5.2.7 PACOTE NULO

No ficheiro do desmultiplexer é possível ainda encontrar pacotes nulos. Este tipo de pacotes tem a função de completar uma eventual capacidade não utilizada da transport stream. Na figura podemos observar a estrutura de um pacote nulo, o qual só contém o cabeçalho e o PID.

```
-----------------------------------------------
| PID = 0x1FFF (Null Packet)
| transport_error_indicator = 0
| payload_unit_start_indicator  = 0
| transport_priority            = 0
| transport_scrambling_control  = 0
| adaptation_field_control      = 1 (payload only)
| continuity_counter = 0 (counter of payloads with the same PID)
 -----------------------------------------------
```

fig.5.10 – pacote nulo

6. Programa

6.1. Breve explicação das classes de Java utilizadas

Para uma melhor compreensão do programa que foi escrito é necessário, relembrar ou conhecer alguns conceitos base da programação em Java, que são apresentados de seguida, sendo de referir que só são apresentados os conceitos utilizados no programa.

6.1.1 LER E ESCREVER

Frequentemente os programas têm que retirar informação de uma fonte externa ou enviar informação para uma fonte externa. A informação pode estar em qualquer lado: no disco, algures na rede, e como no nosso caso, num ficheiro; e pode ser de qualquer tipo: objectos, caracteres, imagens ou sons.

Para receber a informação, o programa abre uma stream numa fonte de informação, neste caso o ficheiro, e lê a informação sequencialmente, como vemos a seguir:

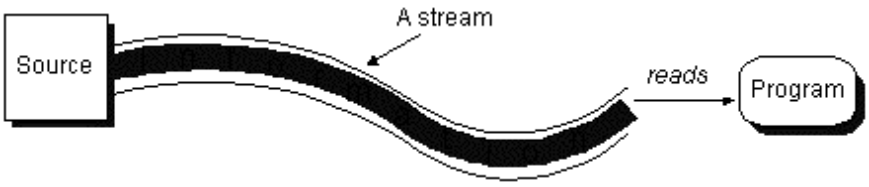


fig.6.1 –Stream da fonte para o programa

Da mesma forma, o programa pode enviar informações para um destino externo abrindo uma stream até ao destino e escrevendo a informação sequencialmente, como podemos ver na figura que se segue:

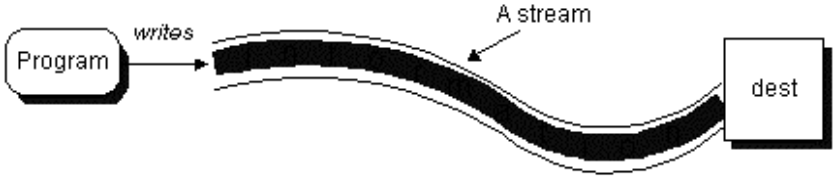


fig.6.2 – Stream do programa para o destino

O algoritmo de leitura e escrita é muito semelhante ao que se apresenta a seguir, não dependendo de onde a informação é lida nem de para onde a informação é escrita:

Reading	**Writing**
open a stream	open a stream
while more information	while more information
read information	write information
close the stream	close the stream

O *package* **java.io** contém uma colecção de classes de streams que suportam estes algoritmos de leitura e escrita. Estas classes estão divididas em duas classes hierárquicas baseadas no tipo de dados (caracteres ou bytes) em que operam.

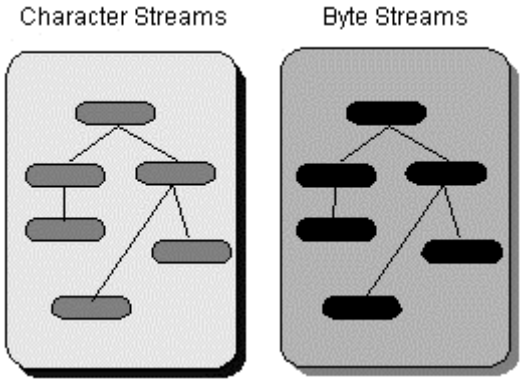


fig.6.3 – classes de streams

Mas, é mais conveniente agrupar as classes baseadas no seu objectivo do que no tipo de dados que elas lêem ou escrevem. Deste modo, podemos agrupar as streams de uma forma "cruzada", dependendo se elas lêem ou escrevem para "sinks" de dados ou se processam a informação à medida que ela é lida ou escrita.

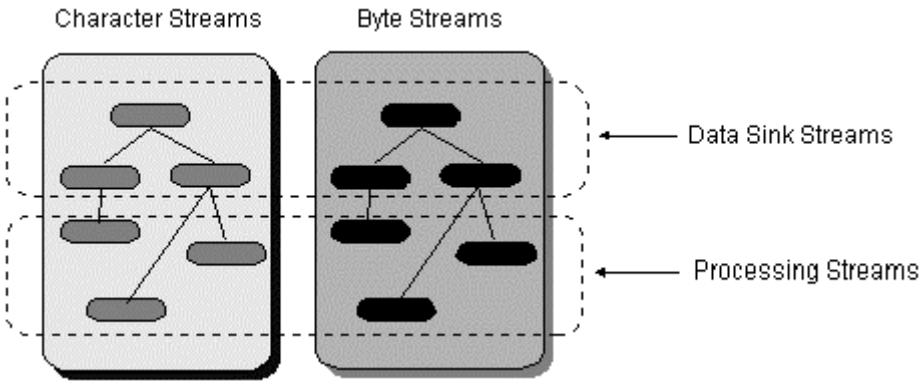


fig.6.4 – classes agrupadas

6.1.2 UMA VISÃO GERAL SOBRE AS I/O STREAMS

Neste ponto vamos descrever os diferentes tipos de streams e apresentar as classes do java.io que as implementam, incidindo mais sobre aquelas que utilizamos na realização do nosso programa.

AS I/O streams dividem-se em Character Streams e Byte Streams, visto serem as últimas as que utilizamos, não vamos descrever as Character Streams.

6.1.2.1 Byte Streams

Os programas devem usar byte streams, derivados das classes `InputStream` e `OutputStream`, para ler e escrever bytes. Estas classes fornecem as API's e implementação para as streams de entrada (streams que lêem bytes) e de saída (streams que escrevem bytes). Estas streams são utilizadas normalmente para ler e escrever dados binários, como imagens e sons.

As subclasses de InputStream e OutputStream fornecem I/O especializada que está dividida em duas categorias: *data sink streams e processing streams.* As figuras seguintes mostram a hierarquia das classes para as byte streams.

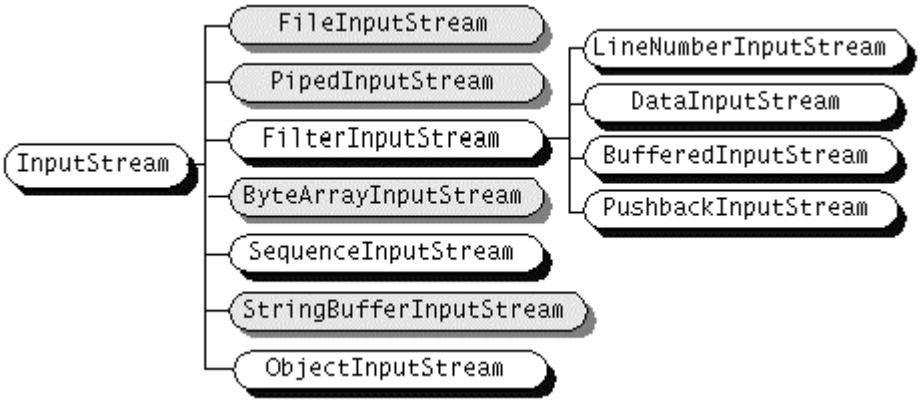


fig.6.5 – hierarquia de byte streams (I)

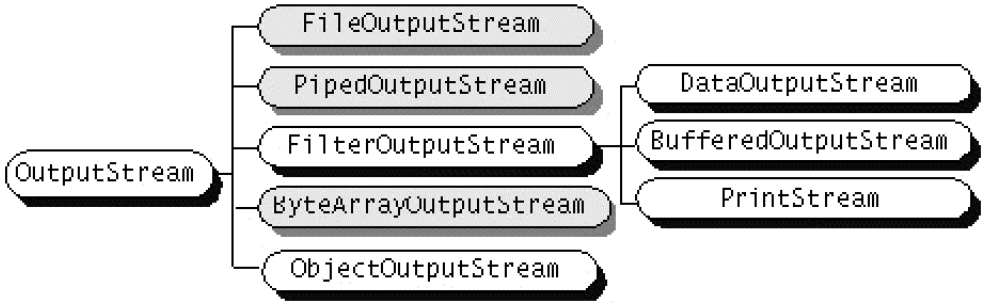


fig.6.6 – hierarquia de byte streams (II)

6.1.3 UTILIZANDO *DATA SINK STREAMS*

As data sink streams lêem de, ou escrevem para data sinks especializadas, tal como string, ficheiros ou pipes. Tipicamente, para cada reader ou input stream utilizado para ler de um tipo fonte de entrada, o **java.io** tem um writer ou output stream paralelo. A tabela seguinte apresenta as data sink streams do **java.io**:

Sink Type	**Character Streams**	**Byte Streams**
Memory	CharArrayReader, CharArrayWriter	ByteArrayInputStream, ByteArrayOutputStream
	StringReader, StringWriter	StringBufferInputStream
Pipe	PipedReader, PipedWriter	PipedInputStream, PipedOutputStream
File	FileReader, FileWriter	**FileInputStream, FileOutputStream**

Tanto o grupo de streams de caracteres como o grupo de streams de bytes contêm pares de classes paralelas que operam nos mesmos data sinks. Descrevemos a seguir aquele que foi utilizado no nosso programa:

`FileInputStream e FileOutputStream` – são chamadas streams files, estas streams são usadas para ler de, ou escrever em ficheiros no sistema nativo de ficheiros.

6.1.4 UTILIZANDO *PROCESSING STREAMS*

As processing streams realizam certo tipo de operações, como o *buffering* ou a codificação de caracteres, à medida que lêem e escrevem. Assim como as data stream sinks, o **java.io** contém pares de streams: uma que realiza uma determina operação durante a leitura e outra que realiza a mesma operação (ou o inverso) durante a escrita. A tabela seguinte mostra as processing streams do java.io:

Process	**CharacterStreams**	**Byte Streams**
Buffering	BufferedReader, BufferedWriter	**BufferedInputStream, BufferedOutputStream**
Filtering	FilterReader, FilterWriter	FilterInputStream, FilterOutputStream
Converting between	InputStreamReader,	

Bytes and Characters	OutputStreamWriter	
Concatenation		SequenceInputStream
Object Serialization		ObjectInputStream, ObjectOutputStream
Data Conversion		**DataInputStream, DataOutputStream**
Counting	LineNumberReader	LineNumberInputStream
Peeking Ahead	PushbackReader	PushbackInputStream
Printing	PrintWriter	PrintStream

Em certos casos, o **java.io** contém character streams e byte streams que realizam o mesmo processamento mas para diferentes tipos de dados. As processing streams que utilizamos são descritas brevemente a seguir:

`BufferedInputStream e BufferedOutputStream` – utiliza o buffer de dados à medida que lê ou escreve, deste modo reduz o número de acessos necessários à fonte de dados original. As buffered streams são tipicamente mais eficientes que as *non-buffered* streams similares.

`DataInputStream e DataOutputStream` – lê e escreve tipos de dados de Java primitivo num formato independente da máquina.

6.1.5 COMO USAR *FILE STREAMS*

As file streams são certamente as streams mais fáceis de compreender. De uma forma simples, as file streams FileInputStream e FileOutputStream, respectivamente lêem de e escrevem para um ficheiro no sistema nativo de ficheiros. Podemos criar uma file stream utilizando um filename (nome do ficheiro) sob a forma de string, de objecto File ou de objecto FileDescriptor.

Para criar um DataInputStream utilizamos:

```
File inputFile = new File("psi.txt");
```

Este código cria um objecto ficheiro que representa o ficheiro nomeado no sistema nativo de ficheiros. File é uma classe muito útil fornecida pelo java.io. O programa utiliza este objecto somente para construir um DataInputStream, no ficheiro psi.txt

6.1.6 COMO USAR *DATAINPUTSTREAM E DATAOUTPUTSTREAM*

O nosso programa abre uma DataInputStream no ficheiro que queremos ler, o ficheiro de saída do desmultiplexer:

```
DataInputStream in = new DataInputStream(new BufferedInputS tream(new
FileInputStream(inputFile)));
```

O DataInputStream, como outras stream de saída filtradas, tem que estar "ligada" a um outro InputStream. No nosso caso, está ligado a um BufferedInputStream e este por sua vez está ligado a um FileInputStream que permite ler de um ficheiro de nome `inputfile`.

O programa lê os dados usando os métodos específicos do DataInputStream `readXXX`.

```
while((s=in.readLine()) != null) {
........
}
```

O método readLine retorna a linha lida, ou null se chegou ao fim do ficheiro. A maior parte dos métodos `readXXX` do DataInputStream não têm esta funcionalidade, isto porque o valor que é retornado para indicar fim de ficheiro pode também ser um valor válido lido da stream. Por exemplo, supondo que usávamos –1 para indicar o fim do ficheiro. Não era possível porque –1 é um valor válido que pode ser lido do inputstream usando readDouble, readInt, ou um outro método que lê números.

Deste modo, os métodos `readXXX` do DataInputStream, "lança" uma EOFException, excepção de fim de ficheiro, quando esta ocorre o ciclo while termina.

O DataOutputStream, como outras stream de saída filtradas, tem que estar "ligada" a um outro OutputStream. No nosso caso, está ligado a um BufferedOutputStream e este por sua vez está ligado a um FileOutputStream que permite escrever para um ficheiro de nome `psi.txt`.

```
File outputFile = new File("psi.txt");
```

```
DataOutputStream psi = new DataOutputStream(new BufferedOutputStream(new
FileOutputStream(outputFile)));
```

De seguida o nosso programa usa os métodos writeXXX do DataOutpuStream para escrever os dados, dependendo do tipos dos mesmos:

```
psi.writeChar('\n');
psi.writeBytes(s);
```

Pode-se verificar que se fecha o stream de saída quando terminamos de o utilizar, assim com o stream de entrada, através da instrução:

```
in.close();
psi.close();
```

6.1.7 E AINDA...

Para além das classes e interfaces apresentadas anteriormente, o java.io contém as seguintes classes também utilizadas no nosso programa:

File – representa um ficheiro do sistema de ficheiros nativo. Podemos criar um objecto ficheiro no sistema de ficheiros nativo e depois pedir informações sobre ele ao objecto (como o pathname completo)

StringTokenizer - divide uma string em tokens, os tokens são as mais pequenas unidades reconhecidas, ainda por um algoritmo de analise de texto.” text-parsing algorithm”, (assim como palavras e símbolos).

6.1.8 COMO COMPARAR STRINGS

Para instânciar a class Collator invocamos o método `getInstance`, para criar uma Collator com um Locale por defeito fazemos:

```
Collator myCollator = Collator.getInstance( );
```

O método `getInstance` retorna um `RuleBasedCollator`, que é uma subclasse do `Collator`. O `RuleBasedCollator` contém um conjunto de regras que determinam, a forma de “sort” das strings para o Locale que especificamos. Estas regras são predefinidas para cada Locale, com estão encapsuladas dentro do `RuleBasedCollator`, o nosso programa não precisa de nenhuma rotina especial para lidar com a forma como as regras variam com a linguagem.

Utilizamos o método Collator.compare para fazermos comparações entre strings independentes. O método compare retorna um inteiro, menor, igual, ou maior que zero quando a primeira string argumento é maior, igual ou menor que uma segunda string argumento. A tabela seguinte mostra alguns exemplos do método Collator.compare:

Collator.compare exemplos

Example	Return Value	Explanation
myCollator.compare("abc", "def")	-1	"abc" is less than "def"
myCollator.compare("rtf", "rtf")	0	the two strings are equal
myCollator.compare("xyz", "abc")	1	"xyz" is greater than "abc"

No nosso programa utilisamos este método da seguinte forma:

```
myCollator.compare(s,s1)
```

este retorna 0 (zero) no caso das strings serem iguais, como foi referido anteriormente.

6.2. Explicação do programa

O objectivo do programa é ler um ficheiro de saída do desmultiplexer cuja estrutura é bastante rígida, como foi explicado anteriormente.

Utilizamos o facto de o Java ser orientado para objectos e dividimos o programa em vários métodos, cada um dos quais realiza uma busca diferente (procura da PAT, de um PID, da PMT).

Depois de uma análise mais cuidada do ficheiro, e também de um estudo mais aprofundado da linguagem de programação em Java, verificámos que devíamos ler o ficheiro de uma forma sequencial de modo a não ser necessário lê-lo desde o inicio cada vez que necessitávamos de encontrar um determinado "componente".

Implementamos então um método principal (procura_PAT), que lê sequencialmente o ficheiro e onde são "chamados" os vários métodos que fazem a busca dos diferentes parâmetros.

Deste modo, numa primeira abordagem pensamos em ler o ficheiro caracter a caracter, procurar palavras chave como PID e ler o valor que o seguia. Concluímos que

este procedimento tornava o programa muito pesado porque se o ficheiro de teste que obtivemos inicialmente tinha 150 páginas, cerca de 300.000 caracteres, visto ter só um programa, a maior parte do ficheiros de saída do desmultiplexer contêm vários programas diferentes e têm mais de 1000 páginas.

Tentámos então encontrar uma forma mais eficiente de ler o ficheiro. Utilizamos as funcionalidades oferecidas pelas streams já descritas para ler de ou escrever para o ficheiro, respectivamente `readLine()` e `writeBytes()`.

Desta forma, a estrutura base do nosso programa é ler o ficheiro linha a linha e comparar cada linha com aquela que queremos encontrar. O início da busca é sempre uma PAT cujo PID tem o valor 0x0000, comparamos então a linha lida com a seguinte:

```
| PID = 0x0000 (Program Association Table)
```

Uma vez encontrada continuamos a ler o ficheiro e à medida que encontramos a flags de interesse escrevemo-las para o ficheiro de saída. Temos que analisar o valor de flags como o `version_number` e o `current_next_indicator` para sabermos se se trata de uma PAT repetida, nova mas ainda não activa ou nova e activa, e, só neste último caso é que a informação nela contida é escrita para o ficheiro de saída.

Segue-se a descrição dos programas contidos na Transport Stream, no mínimo temos um programa que é lido para um vector de strings, isto é, o PID da PMT onde está descrito o programa 1 está na posição 1 do vector, se existirem mais programas realizamos os mesmos procedimentos para cada um deles. Temos então um vector com o PID da PMT de cada programa, e é este vector que vai ser analisado pelo método `procura_PMT`.

O método `procura_PMT` tem como parâmetros o PID de uma PMT e o nome do ficheiro de entrada, seguindo a linha estabelecida pelo método `procura_PAT`, vamos agora procurar uma linha como a apresentada a seguir:

```
| PID = 0xXXXX
```

Este procedimento vai repetir-se desta forma no nosso programa pois é essencial para encontrarmos o pacote desejado.

Depois de encontrado o pacote com o PID desejado, lemos e escrevemos para o ficheiro de saída as flags de interesse, tendo o cuidado de analisar mais uma vez as flags version_number e current_next_indicator.

Seguem-se as definições do vídeo e do áudio e respectivos identificadores dos pacotes que os transportam.

```
| stream_type = 2    -> ITU-T Rec.H262 | ISO/IEC 13818-2 Video
| elementary_PID = 0x0100

| stream_type = 3    -> ISO/IEC 11172 Audio
| elementary_PID = 0x0101
```

Mais uma vez vamos escrever para um vector os PID's das streams, como a primeira stream apresentada é sempre a de vídeo, o PID dos pacotes de vídeo é "guardado" na posição 0 do vector, enquanto o PID dos pacotes de áudio é armazenado na posição 1.

O valor da primeira posição deste vector vai ser o ponto de partida do método procura_video que, como o próprio nome indica tem como função procurar os pacotes com o PID das streams elementares de vídeo.

Como os métodos descritos anteriormente, procura a linha que corresponde ao PID pretendido, depois verifica se o pacote tem Adaptation Field, se existir lê a PCR_flag e verifica o seu valor, se estiver a 1 significa que está presente o Program Clock Reference, logo lemo-lo e escrevemo-lo para o ficheiro de saída.

Dentro do pacote de vídeo temos o Program Elementary Stream - PES - Packet, o cabeçalho do PES, se existir, contém informação, entre outra, sobre os Presentation Time Stamps - PTS - e os Decoding Time Stamps - DTS. O valor da flag PTS_DTS_flags pode ser um de três possíveis, 3, 2 ou 0, dependendo deste valor lemos o DTS e o PTS, só o PTS ou nenhum dos dois.

A busca continua para todos os pacotes de vídeo com o PID referido, escrevendo sempre para o ficheiro as flags de interesse, o PCR, o PTS e DTS.

O método procura_audio tem como parâmetro de entrada o valor da segunda posição do vector retornado pelo método procura_PMT, que é o PID dos pacotes que contém a stream de áudio.

Mais uma vez é realizada a busca entre os pacotes de modo a encontrarmos os pacotes com o PID desejado, quando isso acontece verificamos se o pacote contém o cabeçalho de PES, se sim mais uma vez lemos o DTS e o PTS, só o PTS ou nenhum dos dois, conforme o valor da flag PTS_DTS_flags.

6.3. Fluxogramas

Para ilustrar a explicação do programa feita anteriormente apresentamos nas páginas seguintes um fluxograma para cada método implementado.

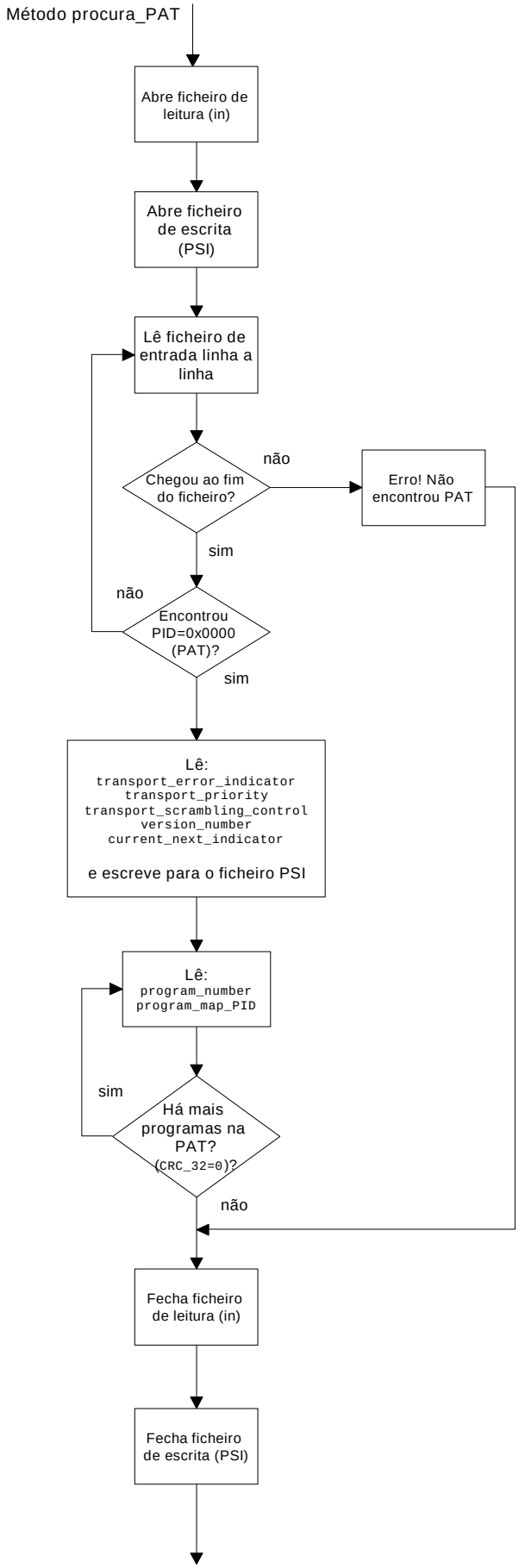


fig.6.7 – método procura_PAT

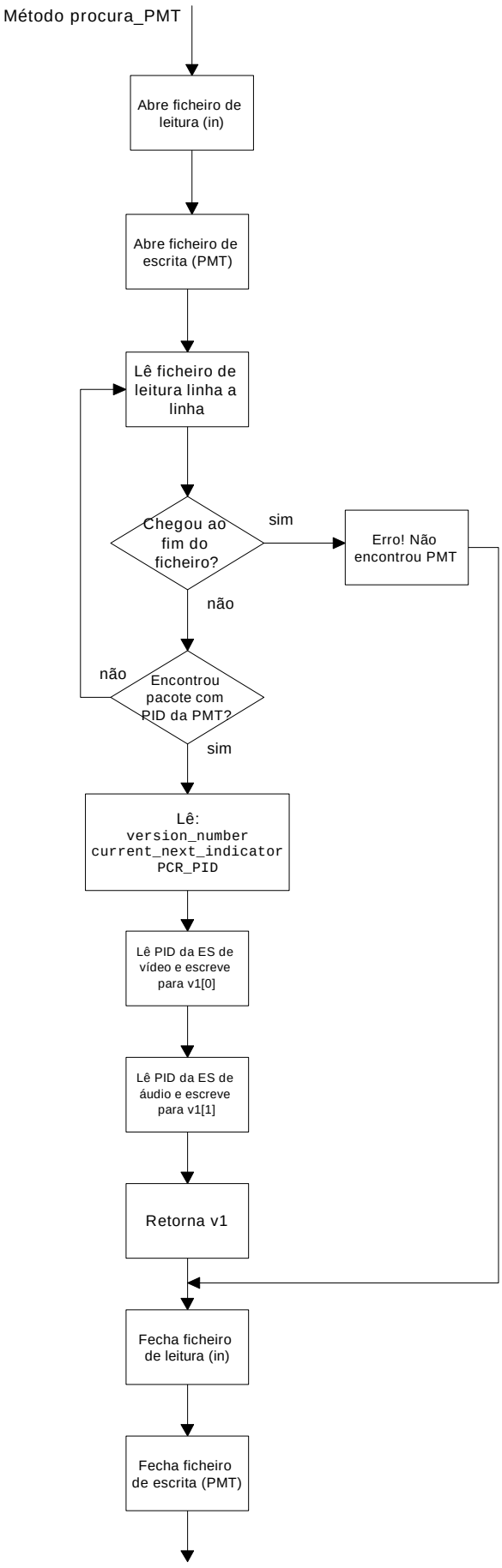


fig.6.8 – método procura_PMT

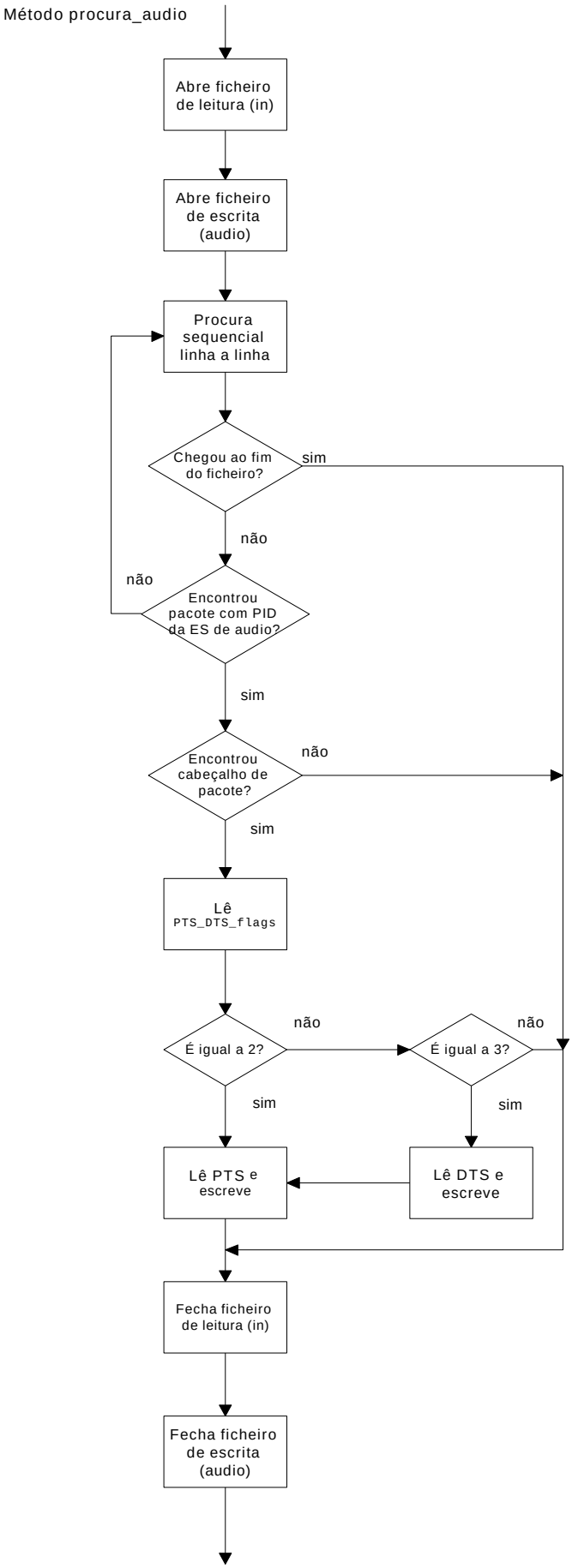


fig.6.9 – método `procura_audio`

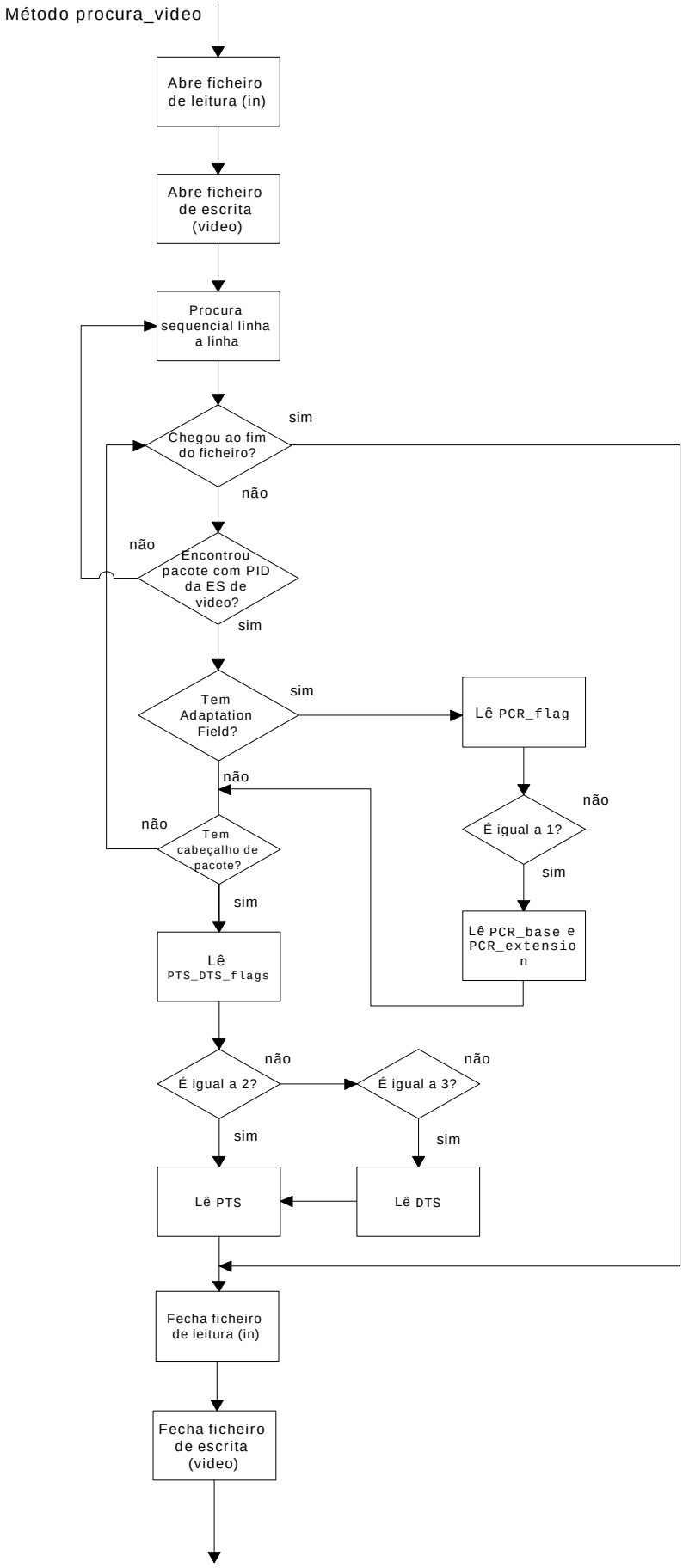


fig.6.9 – método procura_video

7. Interface Gráfica

7.1 A importância das interfaces

Ficou decidido logo desde a fase de idealização do trabalho a execução de uma interface gráfica para o utilizador, pois assim de uma forma simples seria possível aceder às funcionalidades do programa implementado sem ser necessário um estudo exaustivo do mesmo, anterior a uma possível utilização.

No entanto, isto não acontece com várias ferramentas de software implementadas, o que origina uma utilização ineficiente ou mesmo impossível por parte dos utilizadores que não adquiram um estudo aprofundado das mesmas. Por isso uma interface é importante para permitir um fácil acesso de um qualquer utilizador às ferramentas de software, e mesmo que a interface não suporte algumas das funcionalidades implementadas, é possível aceder a um menu de ajuda que facilite a compreensão do programa e assim ser possível uma utilização completa.

7.2 Breve explicação da programação gráfica

Para escrevermos o programa que realiza a interface (GUI - graphical user interfaces), utilizamos os componentes do Swing, que são parte do Java™ Foundation Classes (JFC) e que podem ser utilizados tanto com a plataforma JDK™ 1.1 ou a plataforma Java™ 2.

7.2.1 O QUE É O JFC E O SWING?

O JFC foi apresentado em 1997 na JavaOne Developer Conference e contém os seguintes componentes:

- **Swing Components** – incluem tudo, desde botões a paineis e tabelas.
- **Pluggable Look and Feel Support –** oferece a qualquer programa que use os componentes do Swing a escolha de looks and feels, um mesmo programa pode ter o aspecto e funcionalidades do Java™ ou as do Windows.

- **Accessibility API**
- **Java 2D™ API (Java 2 Platform only)**
- **Drag and Drop Support (Java 2 Platform only)**

7.2.2 QUAIS OS PACKAGES DO SWING QUE DEVEMOS UTILIZAR?

A API do Swing é muito poderosa, flexível e imensa. Felizmente, a maior parte dos programas usa uma pequena parte da API. O nosso programa utiliza os seguintes packages:

- javax.swing
- javax.swing.event (nem sempre necessário)

7.2.3 IMPORTAR OS PACKAGES DO SWING

A linha seguinte importa o package principal do Swing

```
import javax.swing.*;
```

A maior parte dos programas, e também é o caso do nosso, necessitam também de importar os seguintes packages principais do AWT.

```
import java.awt.*;
import java.awt.event.*;
```

7.2.4 CONTENTORES DE NÍVEL SUPERIOR

Todos os programas que apresentam uma Swing GUI, contêm pelo menos um contentor de nível superior do Swing. Para a maior parte dos programas, os contentores de nível superior são instâncias de *JFrame*, *JDialog*, ou (para applets) *JApplet*. Cada objecto *JFrame* implementam única janela principal, cada objecto *Jdialog* implementa uma janela secundária. Cada objecto *Japplet* implementa uma applet's display area numa janela browser. Um contentor de nível superior do Swing oferece o suporte que os components do Swing necessitam para realizarem a gestão da representação e de eventos.

Antes de usarmos um contentor de nível de superior, temos que compreender alguns pontos importantes:

- o Swing oferece como já foi referido três classes gerais de contentores de nível superior: *JFrame*, *JDialog* e *JApplet*
- o Swing contém ainda um contentor intermédio, JInternalFrame, que emita um frame, embora internal frames não sejam exactamente contentores de nível superior;
- para aparecer no écran, cada componente GUI tem que pertencer a uma hierarquia. Cada hierarquia de contentores tem um contentor de nível superior como raiz;
- cada contentor de nível superior tem um content pane que, geralmente, contém os componentes visiveis num contentor GUI de nível superior;
- podemos, opcionalmente uma menu bar a um contentor de nível superior, mas fora do content pane.

Podemos ver na imagem que se segue um frame criado por uma aplicação. O frame contém uma menu bar azul-céu vazia e, no content pane do frame, um label amarelo.

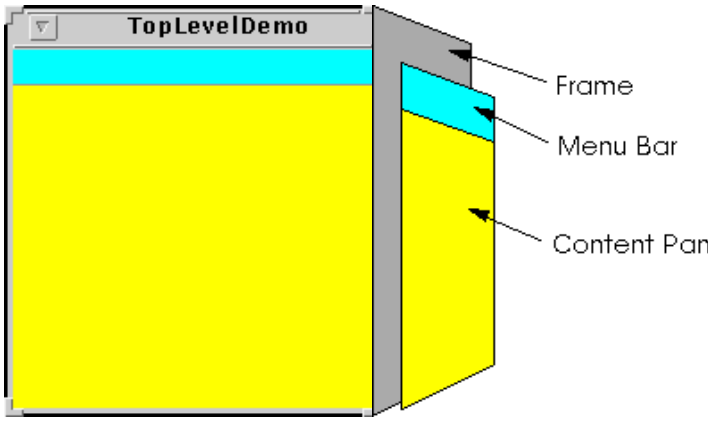


Fig.7.1- Interface TopLevelDemo

E apresentamos a seguir a hierarquia desta simples interface:

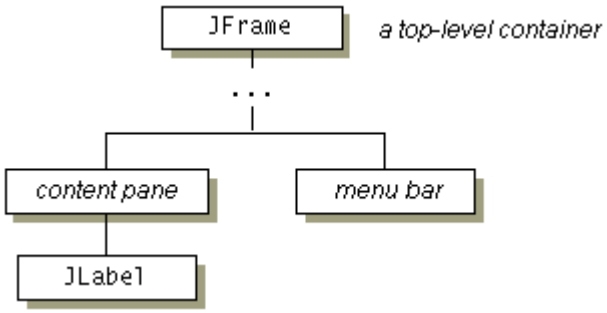


Fig.7.2 –Hierarquia da interface TopLevelDemo

Cada programa que use componentes do Swing tem pelo menos um contentor de nível superior. Este um contentor de nível superior é a raiz da containment hierarquia – a hierarquia que contém todos os componentes do Swing que estão dentro do contentor de nível superior.

7.2.5 ADICIONAR COMPONENTES AO CONTENT PANE

O código que se segue é o código que no exemplo anterior é utilizado para criar o content pane do frame e adicionar-lhe um label amarelo.

```
frame.getContentPane().add(yellowLabel,BorderLayout.CENTER);
```

Como o código mostra, encontramos o content pane de um contentor de alto nível chamando o método `getContentPane`. O content pane por defeito é um simples contentor intermédio inerente à classe *JComponent*, e que utiliza o *BorderLayout* como layout.

7.2.6 COMO USAR INTERNAL FRAMES

Dentro da classe *JInternalFrame* podemos apresentar um *JFrame* como uma janela dentro de outra janela. Usualmente, adicionam-se internal frames a desktop panes, como no nosso caso. A desktop pane, por outro lado, pode ser usada como content pane de um *JFrame*. A desktop pane é uma instância de *JDesktopPane*, que é uma subclasse de *JLayeredPane* que é adicionada à API para fazer a gestão de múltiplas sobreposições de internal frames.

Deve pensar-se com cuidado se se baseia o programa GUI em torno de frames ou internal frames. Comutar entre internal frames e frames ou vice-versa não é

necessariamente uma tarefa simples. No nosso programa optamos por utilizar internal frames.

Podemos ver a imagem em que se mostra uma aplicação que usa dois internal frames (uma delas iconificada) dentro de um frame regular.

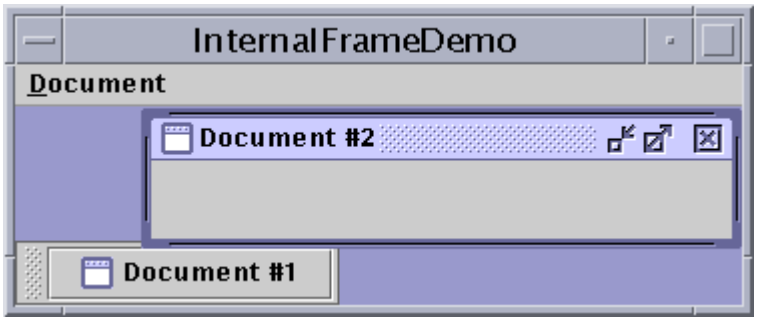


Fig.7.3 – Interface InternalFrameDemo

Podemos também verificar que a decoração de janela de um internal frame reflecte o Java Look & Feel. Embora a janela que contém o internal frame tenha o aspecto do Look & Feel nativo.

7.2.7 ADICIONAR UMA MENU BAR

Todos os contentores de nível superior podem, teoricamente, ter uma menu bar. Na prática, as menu bars aparecem somente em frames, internal frame no nosso caso, ou em applets. Para adicionar uma menu bar a um frame, criamos um objecto *JMenuBar*, inserimos-lhe menus, e depois chamamos o método `setJMenuBar`. A janela TopLevel Demo, apresentada anteriormente adiciona uma menu bar ao frame com a seguinte instrução.

```
frame.setJMenuBar(cyanMenuBar);
```

7.2.8 COMO USAR MENUS

Um menu oferece uma forma mais compacta para permitir ao utilizador escolher uma opção de entre várias.

Os menus são únicos pelo facto de, por convenção, não são colocados com os outros componentes na UI (user interface). Pelo contrário, o menu aparece usualmente em menu bar ou em popup menus.

A figura seguinte mostra a hierarquia inerente às classes relacionadas com o menu:

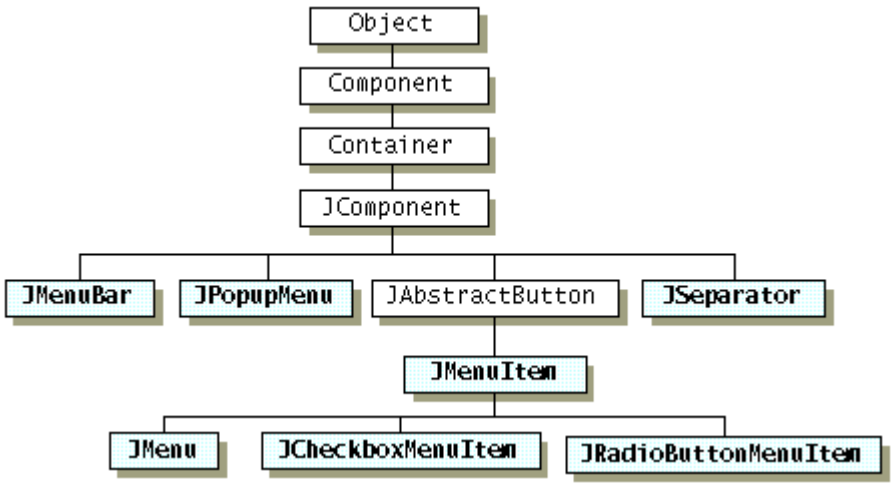


Fig.7.4 – Hierarquia das classe relacionadas com o componente menu

Podemos ver que os menus items (incluindo menus) são simples botões. Podemos perguntar-nos como é que um menu, se é somente um botão, consegue apresentar os seus menus items. A resposta é que quando o menu é activado, ele automáticamente "levanta" um pop menu que mostra os menus items.

O código que se segue é parte da classe *Janela* e cria os menus que aparecem na nossa janela:

```
//no construtor da classe:
JMenuBar jMenuBar1;
JMenu Analisar,Correr,Ajuda;
JMenuItem Abrir,Ajuda;

//cria a menu bar.
JMenuBar1 = new JMenuBar();
setJMenuBar(jMenuBar1);

//constroi o primeiro menu.
Analisar = new JMenu("Analisar");
jMenuBar1.add(Analisar);

//outra forma de criar um menu
Correr = new Jmenu();
Correr.setText("Correr")
jMenuBar1.add(Correr);

//um JMenuItems
Abrir = new JMenuItem("Abrir",KeyEvent.VK_A);
Abrir.setAccelerator(KeyStroke.getKeyStroke(
```

```
        KeyEvent.VK_1, ActionEvent.ALT_MASK));
Analisar.add(Abrir);

//um submenu
Analisar.addSeparator();
InfoPSI = new JMenu("InfoPSI");

PAT = new JMenuItem("PAT");
PAT.setMnemonic(keyEvevt.VK_T);
PAT.setAccelerator(KeyStroke.getKeyStroke(
        KeyEvent.VK_T, ActionEvent.ALT_MASK));
InfoPSI.add(PAT);

video = new JMenuItem("video");
video.setMnemonic(keyEvevt.VK_V);
video.setAccelerator(KeyStroke.getKeyStroke(
        KeyEvent.VK_V, ActionEvent.ALT_MASK));
InfoPSI.add(video);

Analisar.add(InfoPSI);

//constroi outro menu na menu bar.
Ajuda = new JMenu("Ajuda");
JMenuBar1.add(Ajuda);
```

Como o código mostra, para colocarmos uma menu bar numa JInternalFrame, usamos o método setJMenuBar. Para adicionarmos um JMenu a uma JMenuBar, usamos o método add(JMenu). Para adicionarmos menu items e submenus a uma JMenu, usamos o método add(JMenuItem). Outro métodos utilisados como setAccelerator and setMnemonic são discutidos na secção Operações do teclado que se segue.

7.2.9 OPERAÇÕES DO TECLADO

Os menus suportam dois tipos de alternativas do teclado: mnemónica e aceleradores. As *mnemónicas* oferecem a possibilidade de usarmos o teclado para navegar dentro da hierarquia do menu, aumentando a acessibilidade aos programas. Os *aceleradores*, por outro lado, oferecem atalhos de teclado para um acesso rápido aos menus.

A mnemónica é uma tecla que escolhe um menu item já visível. Por exemplo, na janela "MenuLookDemo" ao pressionarmos as teclas Alt e A aparece o primeiro menu. Enquanto este menu está visível se pressionarmos a tecla B (juntamente ou não com o Alt) escolhemos o segundo menu item. Um menu item geralmente mostra a sua mnemónica sublinhando a primeira ocorrência do caracter mnemónica no texto do menu item, como podemos ver no printscrean que se segue:

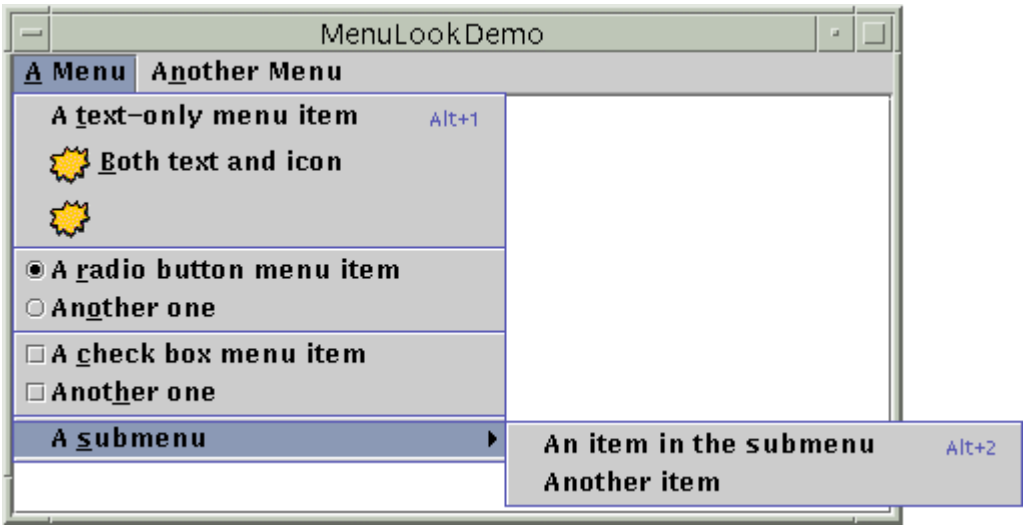


Fig.7.5 – Interface MenuLookDemo

Um acelerador é uma combinação de teclas que escolhe o menu item, esteja ou não visível. Por exemplo, pressionando, simultaneamente as teclas Alt e 2 quando a janela "MenuLookDemo" está activa, escolhemos o primeiro item do primeiro submenu, sem fazer aparecer qualquer menu. Só os menu que não fazem aparecer outros menus podem tem aceleradores.

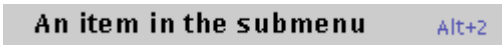


Fig.7.6 – Acelerador de um menu item

Podemos especificar a mnemónica quando estamos a construir o menu ou com o método `setMnemonic`. Para especificar um acelerador, usamos o método `setAccelerator`. Segue-se um exemplo de cada um deles:

```
//especificar a mnemónica quando construímos o menu item:
menuItem = new JMenuItem("A text-only menu item",
                         KeyEvent.VK_T);

//especificar a mnemónica depois de Ter criado o menu item:
menuItem.setMnemonic(KeyEvent.VK_T);

//especificar o acelerador:
menuItem.setAccelerator(KeyStroke.getKeyStroke(
        KeyEvent.VK_T, ActionEvent.ALT_MASK));
```

Como podemos verificar, especificamos a mnemónica utilizando a constante *KeyEvent* que corresponde à tecla que o utilizador pressionou. Para especificar o

acelerador temos que usar o objecto *KeyStroke*, que combina a tecla (especificada pela constante KeyEvent) e uma modifier-key mask (especificada pela constante *ActionEvent*).

7.2.10 GESTÃO DE EVENTOS

Sempre que o utilizador insere um caracter ou pressiona um botão, ocorre um evento. Qualquer objecto pode ser notificado pela ocorrência de um evento. Tudo o que tem que fazer é implementar uma interface apropriada e estar registado como um *event listener* numa *event source* apropriada. Os componentes do Swing podem gerar muitos tipos de eventos, no quadro seguinte apresentam-se alguns exemplos:

Act that results in the event	Listener type
User clicks a button, presses Return while typing in a text field, or chooses a menu item	`ActionListener`
User closes a frame (main window)	`WindowListener`
User presses a mouse button while the cursor is over a component	`MouseListener`
User moves the mouse over a component	`MouseMotionListener`
Component becomes visible	`ComponentListener`
Component gets the keyboard focus	`FocusListener`
Table or list selection changes	`ListSelectionListener`

Cada evento é representado por um objecto que dá a informação sobre o evento e identifica a fonte do evento. As fontes de eventos são tipicamente componentes, mas também podem ser utilizados outros tipos de objectos.

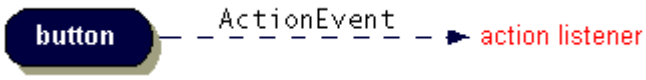


Fig.7.7 –ActionEvent e ActionListener

Todos os event handlers necessitam dos três excertos de código seguintes:

1. Declaração da classe do event handler, o código especifica se a classe implementa uma interface listener ou se estende uma classe que implementa uma interface listener, por exemplo:

2.public class MyClass implements ActionListener {

3.Código que regista a instância da classe event handler tem um listener em um ou mais componentes. Por exemplo:

someComponent.addActionListener(instanceOfMyClass);

4. Código que implementa o método na interface listener. Por exemplo: public void actionPerformed(ActionEvent e) { ...*//code that reacts to the action...*}

O nosso programa tem dois tipos de event handlers. Um tem a gestão dos clics nos botões (action events); o outro tem a gestão sobre o fecho da janela (window events), como podemos verificar na parte de código que se segue:

```
Sair.addActionListener (new java.awt.event.ActionListener(){
   public void actionPerformed(java. awt.event.ActionEvent evt)
   {
        System.exit (0);
   }
}
);
addWindowListener (new java.awt.event.WindowAdapter () {
   public void windowClosing (java.awt.event.WindowEvent evt) {
         exitForm (evt);
   }
}
);
```

7.2.11 COMPONENTES DE TEXTO

Os componentes de texto do Swing apresentam texto e opcionalmente permitem que o utilizador edite texto.

O Swing oferece cinco componentes de texto, juntamente com classes e interfaces, que vão ao encontro mesmo da mais complexa exigência. Embora tenha diferentes utilidades e capacidades, todos os componentes de texto do Swing derivam da mesma superclasse, JtextComponent.

A figura seguinte mostra a hierarquia da classe e organiza as classes de componentes de texto em três grupos:

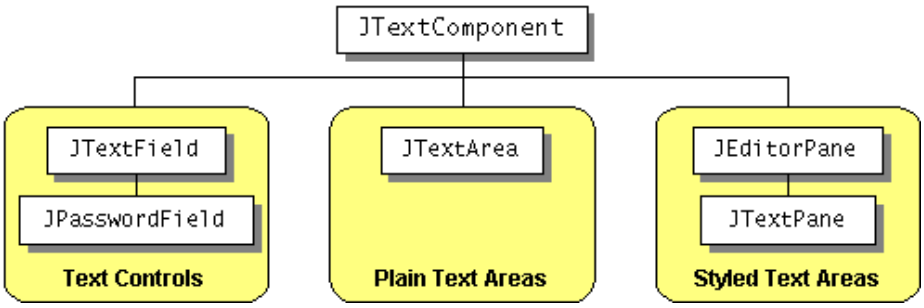


Fig.7.8 – Hierarquia da classe JtextComponent

O quadro seguinte descreve os três grupos de componentes de texto:

Group	Description	Swing Classes
Text Controls	Also known simply as text fields, text controls can display and edit only one line of text and are action-based, like buttons. Use them to get a small amount of textual information from the user and take some action after the text entry is complete.	**JtextField** and its subclass **JpassowrdField**
Plain Text Areas	`JtextArea` can display and edit multiple lines of text. Although a text area can display text in any font, all of the text is in the same font. Use a text area to allow the user to enter unformatted text of any length or to display unformatted help information.	**JtextArea**
Styled Text Areas	A styled text component can display and edit text using more than one font. Some styled text components allow embedded images and even embedded components. Styled text components are powerful and multi-faceted components suitable for high-end needs, and offer more avenues for customization than the other text components. Because they are so powerful and flexible, styled text components typically require more up-front programming to set up and use. The one exception is that editor panes can be easily loaded with formatted text from a URL, which makes them useful for displaying uneditable help information.	**JeditorPane** and its subclass **JtextPane**

7.2.12 TEXT AREA

Uma text area apresenta múltiplas linhas de texto e permite que o utilizador edite texto com o teclado ou rato. Apresentamos a seguir o código que cria uma text area:

```
jTextArea1 = new javax.swing.JtextArea();
jTextArea1.setFont(new Font("Verdana", Font.PLAIN, 14));
jTextArea1.setLineWrap(true);
jTextArea1.setWrapStyleWord(true);
jTextArea1.setEditable(false);
```

O construtor inicializa a text area sem texto. Depois especifica o tipo e tamanho da letra.

Por defeito a text area não "corta" as linhas, mostra o texto numa única linha, e se a text area está definida dentro de um scroll pane, permite ser *scrolled* horizontalmente. O nosso código faz com que as linhas sejam "cortadas" chamando o método `setLineWrap` seguido do método `setWrapStyleWord`, que indica à text area que deve "cortar" as linhas no fim das palavras, e não caracter a caracter.

Para que não seja possivel escrever na text area chamamos o método `setEditable(false.`

Para termos a capacidade de scrolling, colocamos a text area no interior de um scroll pane:

```
areaScrollPane = new JscrollPane(jTextArea);
areaScrollPane.setVerticalScrollBarPolicy(
        JscrollPane.VERTICAL_SCROLLBAR_ALWAYS);
areaScrollPane.setPreferredSize(new Dimension(1000, 1000));
```

7.2.13 COMO USAR SCROLL PANES

Um JscrollPane oferece a possibilidade de termos uma visão *scrollable* de um componente. Quando o écran é limitado, utilizamos um scroll pane para apresentarmos um componente que é demasiado grande ou cujo tamanho varia dinamicamente.

O código para criar um scroll pane pode ser mínimo. Por exemplo, na figura vemos um programa de teste que coloca uma text area num scroll pane porque o tamanho da text area aumenta dinamicamente à medida que o texto é acrescentado:

Fig.7.9 - ScrollPane

O código que segue cria uma text area, torna-a cliente do scroll pane e adiciona o scroll pane a um frame:

```
textArea = new JTextArea(5, 30);
JScrollPane scrollPane = new JScrollPane(textArea);
...
contentPane.setPreferredSize(new Dimension(400, 100));
...
contentPane.add(scrollPane, BorderLayout.CENTER);
```

O programa utiliza a text area como argumento do construtor scroll pane. Isto torna a text area cliente do scroll pane. O scroll pane passa então a gerir todos os acontecimentos: cria as scroll bars, se necessário, redesenha o cliente quando o utilizador move a janela.

Sem a linha de código a bold o scroll pane iria calcular o seu tamanho preferencial de modo que a text area, com o seu tamanho preferencial, caiba completamente dentro do scroll pane. Consequentemente, o scroll pane não teria scroll bars. A linha de código a bold especifica o tamanho preferencial do contentor do scroll pane de modo que este é forçado a ter um tamanho mais reduzido que o preferencial.

Deste modo, quando o programa aparece no écran, o scroll pane tem uma scroll bar vertical.

7.2.14 SCROLL BAR POLICY

No início, a aplicação ScollDemo tem duas scroll bars. Se aumentarmos a janela até ao tamanho do écran, desaparecem as duas scroll bars porque já não são necessárias. Se depois diminuirmos a altura da janela sem alterar a largura, a scroll bar vertical reaparece. Continuando as experiências constatamos que as scroll bars aparecem ou desaparecem conforme são ou não necessárias. Este comportamento é controlado pela *scroll bar policy* do scroll pane. Mais precisamente duas "politicas": uma para cada scroll bar.

A ScrollDemo não explicita as scroll bar policies do scroll pane, utiliza a de defeito, mas podemos especificar as scroll bar policies quando criamos o scroll pane ou altera-las dinamicamente.

Escolhemos utilizar a última e para isso utilizamos o método `setVerticalScrollBarPolicy`. Se quiséssemos também uma scroll bar horizontal podiamos utilizar o método `setHorizontalSctollBarPolicy`.

Os dois métodos utilizam uma das seguintes constantes definidas na interface ScrollPaneConstants (implementada pelo JScollPane).

Policy	Description
`VERTICAL_SCROLLBAR_AS_NEEDED` `HORIZONTAL_SCROLLBAR_AS_NEEDED`	The default. The scroll bar appears when the viewport is smaller than the client and disappears when the viewport is larger than the client.
`VERTICAL_SCROLLBAR_ALWAYS` `HORIZONTAL_SCROLLBAR_ALWAYS`	Always display the scroll bar. The knob disappears if the viewport is large enough to show the whole client.
`VERTICAL_SCROLLBAR_NEVER` `HORIZONTAL_SCROLLBAR_NEVER`	Never display the scroll bar. Use this option if you don't want the user to directly control what part of the client is shown. Perhaps you have an application that requires all scrolling to occur programmatically.

7.2.15 Explicação do programa da interface

O programa que implementa a interface gráfica é constituido por uma classe principal *Janela* e por uma classe auxiliar *ImageFrame*.

A classe auxiliar *ImageFrame* extende o contentor de nível superior *JInternalFrame* e implementa um frame onde se apresenta um text area no interior de um scrool pane.

O construtor da classe tem como argumentos o nome do ficheiro que queremos abrir e o seu path, e tem dois métodos: `initComponents(filename)` que cria e inicializa a text area e o scroll pane, abre o ficheiro de nome `filename` e afixa-o na text area, e o método `setTitle(imagename)` que serve unicamente para especificar o título da janela, neste caso o path do ficheiro que está aberto.

A text area é criada sem qualquer tipo de texto com a instrução:

```
JTextArea1 = new javax.swing.JtextArea() ;
```

e o scroll pane que vai conter a text area, com a seguinte linha de código:

```
areaScrollPane = new JScrollPane(jTextArea1) ;
```

Ao scroll pane é adicionada uma scroll bar vertical e é estabelecido o seu tamanho preferencial.

O método getContentPane retorna o content pane do contentor de alto nível, neste caso do internal frame, e adicionamos-lhe um BoxLayout (a classe BoxLayout coloca todos os componentes numa única coluna ou linha, respeita o tamanho máximo do componente e permite que alinhamos os componentes), assim como os controlos das capacidades da janela (Closable – possibilidade do utilizador fechar o internal frame, Resizable - possibilidade de variar o tamanho do internal frame, Iconifiable - possibilidade de iconificar o internal frame e Maximizable- possibilidade do utilizador maximizar o internal frame).

De seguida abrimos o ficheiro de leitura e utilizando o método read(), para escrevemos o ficheiro na text area.

Para finalizar uma instrução importante:

```
getContenPane().add(areaScrollPane)
```

que adiciona o scroll pane ao contente pane do contentor de alto nível.

A classe principal *Janela* extende o contentor de nível superior *Jframe.*

O construtor da classe inicializa uma desktop e redesenha-a de cada vez que o construtor é chamado, contem ainda um método: `initComponents()` que cria e inicializa a uma menu bar, assim como os vários menus e submenus cada um com as suas especificações.

Os eventos despoletados pela escolha dos diferentes menus items são práticamente idênticos com a excepção da `AbrirActionPerformed()`, `FecharActionPerformed()` e da `exitForm()`.

A acção `AbrirActionPerformed()` abre uma *FileDialog* que permite escolher o ficheiro que queremos analisar de entre os ficheiros, directorias e drives disponiveis. É então chamada a classe *ImageFrame* com o nome do ficheiro e o seu path como argumentos, e adicionado à desktop o internal frame com o método `desktop.add(ifr,javax.swinh.JlayeredPane.DEFAULT_LAYER)`.

O método:

```
desktop.putClienteProperty("JDesktop.dragMode","outline")
```

destina-se tornar o arrastamento da janela mais rápido, isto é, ao arrastarmos a janela só é visivel a orla da mesma.

Ao chamarmos o método `setLocation(x*count, y*count)` fazemos com que os vários frames que são apresentados na janela não se sobreponham e apreçam como cascata.

Por fim é chamado o programa que analisa o ficheiro de texto e cria quatro diferentes ficheiros uma para cada tipo de informação: PAT, PMT, vídeo ou áudio.

A acção `FecharActionPerformed()` limita-se a fechar os frames que estão na janela.

A acção `exitForm()` assim como `SairActionPerformed` limita-se a chamar o método `System.exit(o)` que fecha a janela.

A acção `PATActionPerformed()`, que ocorre quando pressionamos o menu item PAT, é semelhante as acções `PMTActionPerformed()`, `videoActionPerformed()`e `audioActionPerformed()`. Cria duas strings, uma com o path do ficheiro (`pathPAT`) e outra com o nome do ficheiro (`fixPAT`), que obedecem à regra do nome dos ficheiros do programa `Analise` e chama a classe *ImageFrame* com es+

tas duas strings como argumento.

As acções `Ajuda1ActionPerformed()` e `AcercaActionPerformed()` limitam-se a apresentar um frame com os respectivos ficheiros de texto.

7.3 Utilização da interface

Como havíamos explicado, ficou decidido criar uma interface gráfica para que seja simples aceder às ferramentas por nós implementadas. Para isso criámos uma janela para funcionar em ambiente Windows, como aquelas a que estamos acostumados, em que, com um simples clique do rato se pode aceder facilmente aos menus e funções disponíveis.

Na figura podemos observar a janela principal, com uma barra superior que contém o título **Análise MPEG** e que consiste em três menus principais: um **Analisar**, outro **Correr** e um outro **Ajuda**, e ainda dos três botões no canto superior direito, como habitual, com as funções de minimizar, restaurar e fechar a janela, sendo ainda possível alterar o seu tamanho com o movimento e clique do rato ao passá-lo pelas bordas da janela. No canto superior esquerdo encontramos o símbolo da Sun microsystems, o qual abre um menu com as mesmas funcionalidades dos três botões referidos anteriormente.

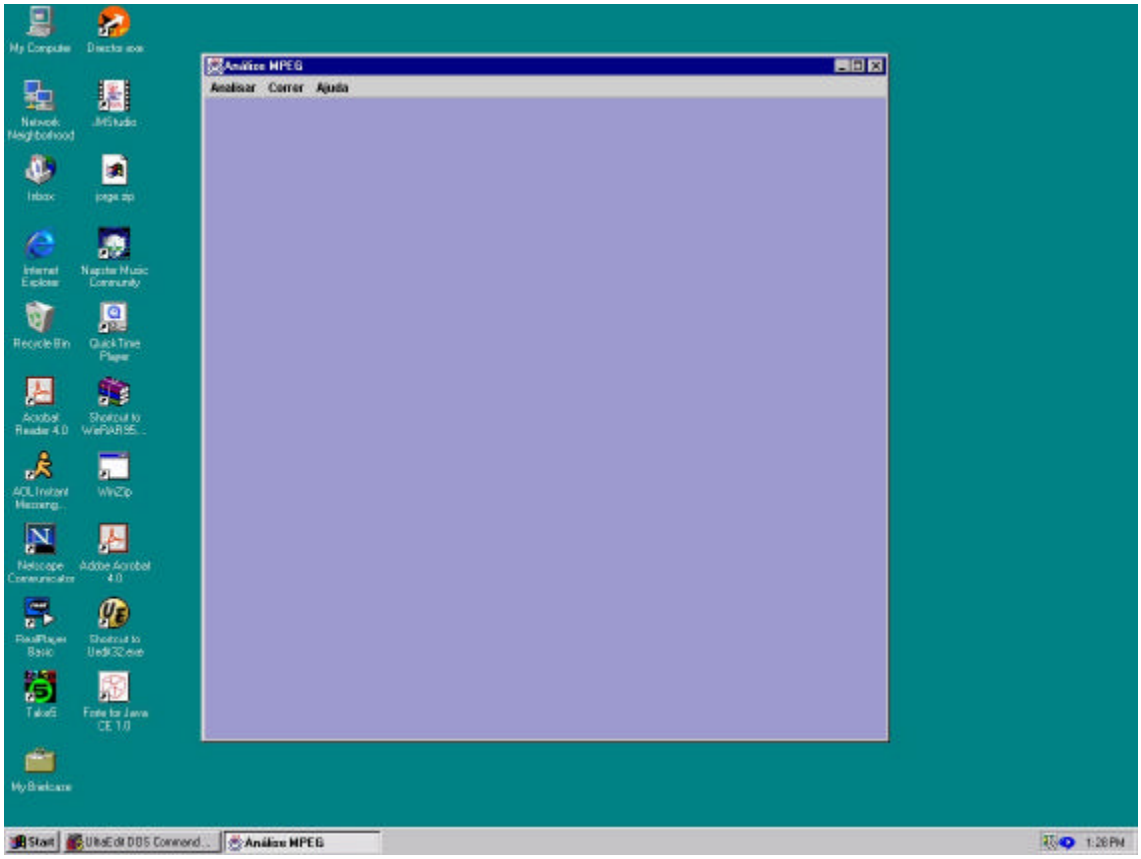


Fig. 7.10 – Janela da Interface de Análise MPEG em ambiente Windows

Começando pelo menu Analisar, podemos abri-lo com um clique do rato, com o qual acedemos a três itens e um submenu: **Abrir**, **Fechar** e **Sair** e **Info PSI**, respectivamente.

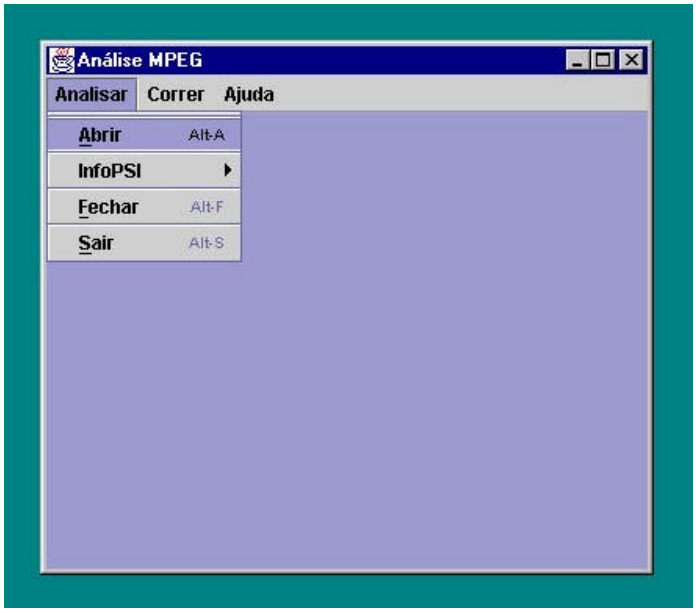


Fig. 7.11 – Menu Analisar

Ao clicar em **Abrir**, temos acesso a uma *file dialog* em que podemos observar os ficheiros, directorias e drives disponíveis, e na qual podemos escolher o ficheiro de texto que queremos analisar. Para aceder a Abrir podemos também utilizar um comando directo pressionando as teclas ALT+A, ou ainda, com o menu Analisar aberto pressionar somente a tecla A, tal como acontece para todos os outros comandos directos que referirmos.

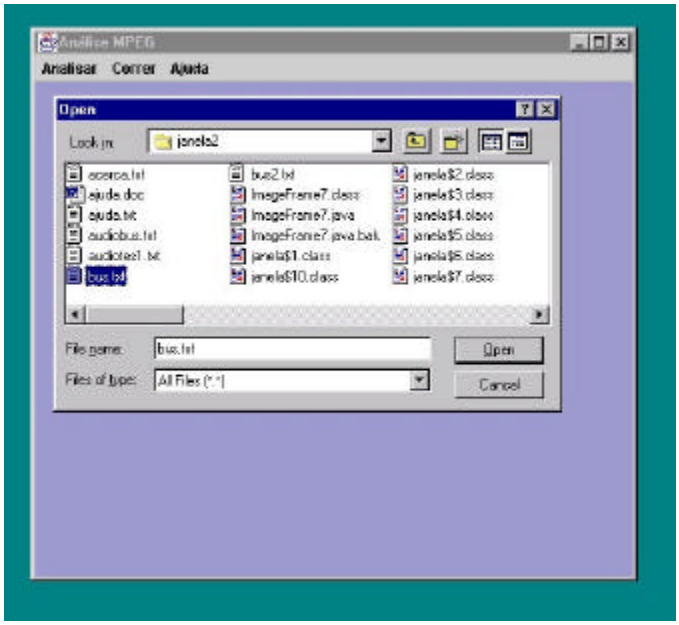


Fig.7.12 –*File Dialog* de abertura do ficheiro do desmultiplexer

Ao abrirmos o ficheiro desejado, uma nova janela irá aparecer dentro da nossa janela principal contendo o ficheiro escolhido de saída do desmultiplexer, sendo possível então observar todo o seu conteúdo.

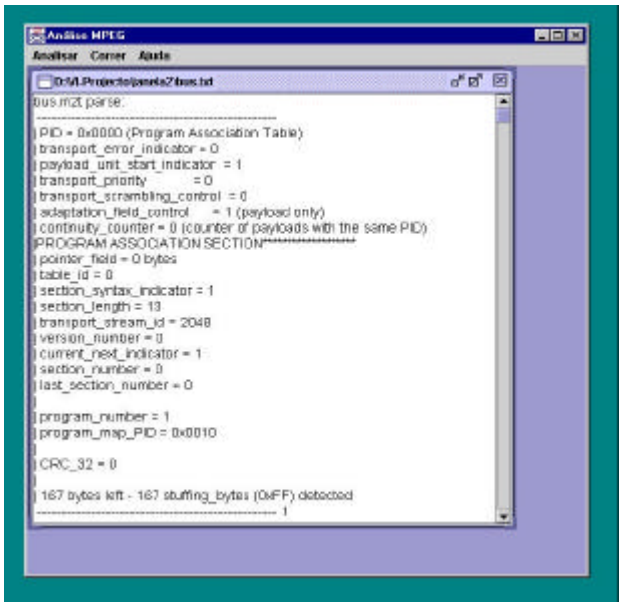


Fig. 7.13 –visualização do ficheiro do desmultiplexer

Porém, quando executamos a função Abrir, não só podemos a visualizar o ficheiro de texto como também estamos a fazer correr todo o nosso programa. Ou seja, estamos a executar toda a análise desejada ao ficheiro, e que estará depois disponível quando acedermos ao submenu **Info PSI**.

Este submenu contém os itens **PAT**, **PMT**, **Video** e **Audio** e com os quais vamos poder visualizar toda a informação retirada da análise do ficheiro de saída de dados do desmultiplexer. Estes dados são escritos em ficheiros de texto separados para cada tipo de informação pretendido, sendo o nome destes ficheiros construído com o tipo de informação escolhida seguido do nome do ficheiro que estamos a analisar, por exemplo PATbus.txt, para um ficheiro de informação sobre a PAT do ficheiro bus.txt.

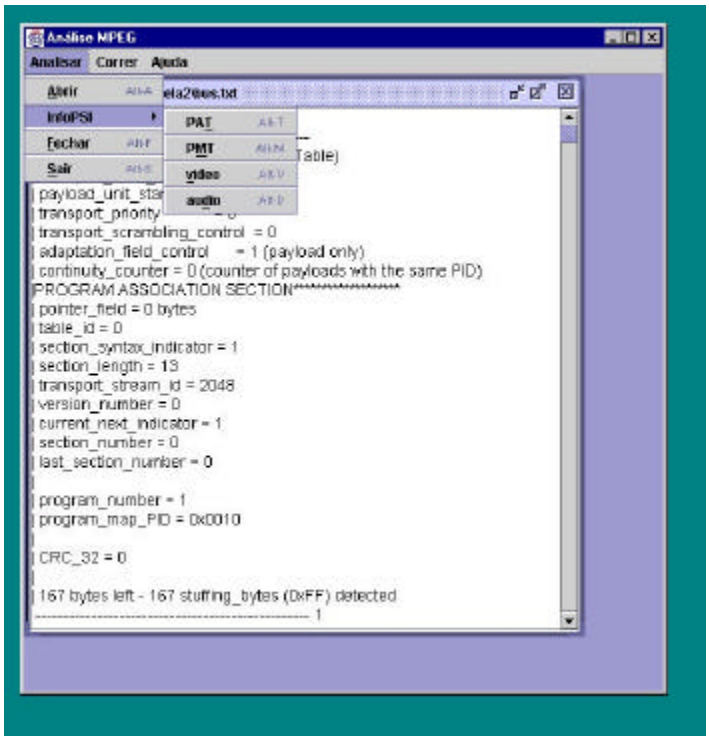


Fig.7.14 - Submenu Info PSI

Para aceder à informação sobre a PAT, basta clicar no item respectivo do submenu ou utilizar as teclas ALT+T e de imediato aparece uma nova janela (em cascata com aquelas já abertas) com toda a informação desejada, e da qual já falámos, sobre a Program Map Table, o mesmo acontecendo para qualquer dos outros três campos: PMT (ALT+M),

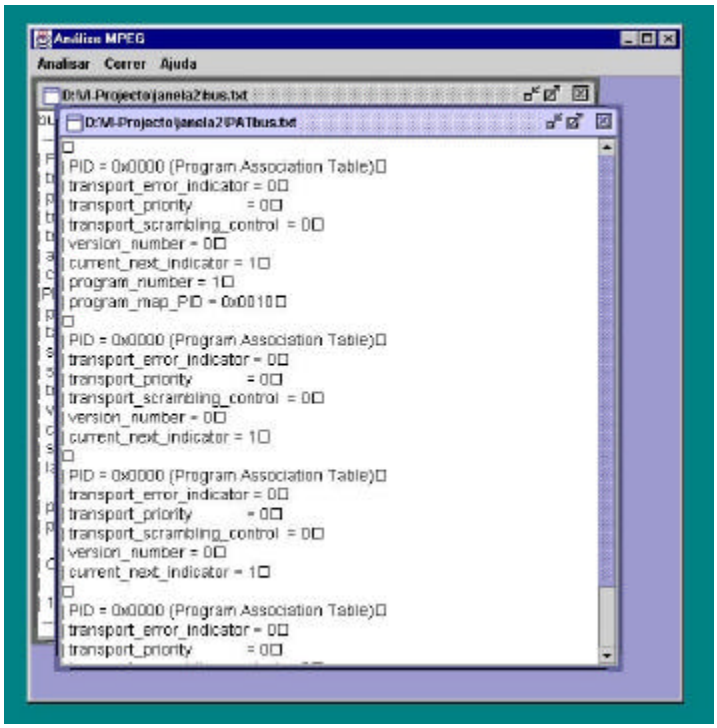


Fig.7.15 - visualização da PAT após a análise, e subjanelas em cascata

Video (ALT+E) e Audio (ALT+I), visto que a análise já havia sido efectuada aquando da abertura do ficheiro desejado. Tal como acontece com a janela principal da interface, também em cada subjanela aberta podemos observar a barra superior com o título da função respectiva, além dos três botões que já referimos para minimizar, restaurar/maximizar e fechar a janela que se encontram no canto superior direito, e ainda com um duplo clique no canto superior esquerdo podemos aumentar automaticamente o tamanho da subjanela até ao tamanho da janela principal. Podemos ainda referir que qualquer destas janelas apenas apresentam a informação, não sendo propositadamente possível escrever nelas, o que iria alterar os ficheiros criados.

De seguida, no menu Analisar, temos a função **Fechar** (ALT+F), que, tal como o próprio nome indica, fecha a janela que estiver activa (tal como no Windows, quando uma janela está activa, a barra superior do título tem a cor azul, enquanto que quando inactiva, a barra apresenta a cor cinzenta).

Finalmente, temos a função **Sair**, que fecha todas as subjanelas ainda abertas assim como a janela principal da interface. Podemos executar esta função com as teclas ALT+S.

Seguidamente temos o menu **Correr**, que tem como função poder fazer a apresentação do audio e do video. Para isso contém três campos: o **Video**, o **Audio**, e o **Video e Audio**, que como é facilmente compreensível, faz a apresentação apenas do video, apenas do audio, ou do audio e video em conjunto, respectivamente. Para chamar cada um destes itens, podemos pressionar as teclas (ALT+E), (ALT+I), (ALT+R), respectivamente.

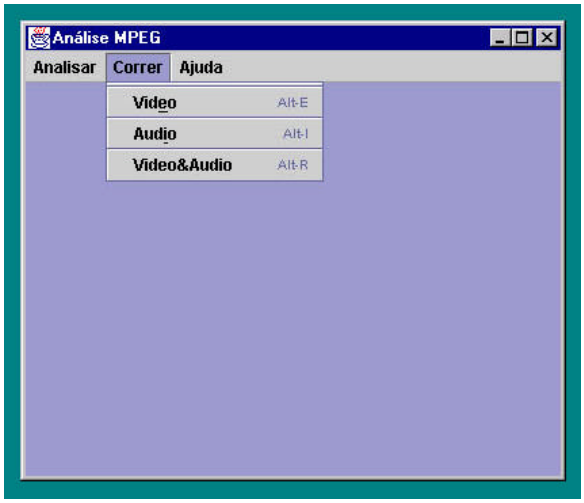


Fig.7.16 – Menu Correr

Para efectuar essa apresentação, o nosso programa chama uma aplicação já existente, como seja o QuickTime ou o Media Player, colocando o ficheiro de dados audio ou video, ou ambos, a correr nessa aplicação.

Para que estas funções de apresentação sejam feitas de forma automática com um simples clique do rato, o nosso programa tem que saber que ficheiros de dados audio e video chamar. Para isso, os nomes desses ficheiros obedecem a uma regra de sintaxe, apresentando um "comportamento" fixo para os diversos programas transportados pela stream. Essa regra é

`inputfilename+PID.streamid` (em hexadecimal)

isto é, mais concretamente, para o nosso ficheiro bus.mt2 teremos um ficheiro de video bus0100.E0, sendo 100 o PID dos pacotes de video e E0 a identificação da stream elementar. Para o ficheiro de dados de audio teremos bus1000.C0, por exemplo.

Temos ainda um menu **Ajuda**, cujo item **Ajuda** (ALT+J) permite ao utilizador encontrar uma explicação sucinta sobre o funcionamento da interface por nós criada, e que é aberta numa janela contendo o texto dessa mesma explicação, tal como acontece para todo o outro tipo de informação apresentada, como explicámos anteriormente. O

utilizador pode ainda observar o item **Acerca** (ALT+L), que contém informação sobre os autores, a versão e a data de execução deste trabalho.

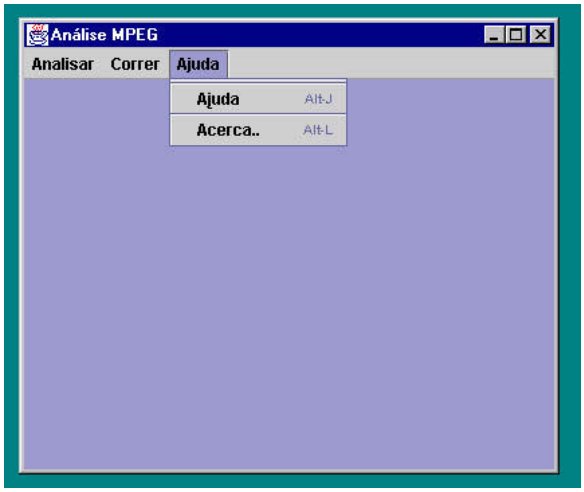


Fig.7.17 – Menu Ajuda

8. Conclusões

Chegámos então ao fim do Relatório do Projecto de final de Curso. Foram quase quatro meses de variados tipos de trabalho: desde procurar e estudar livros sobre Java, até estudar o MPEG com uma certa profundidade em vários livros e documentos, passando por várias reuniões e conversas com a nossa orientadora Engª. Maria Teresa Andrade, ou por dezenas de horas ao computador quer a construir o programa, a efectuar testes, ou a escrever este relatório. Mas agora podemos dizer: conseguimos!

Para trás ficou um trabalho intenso na tentativa de criar algo que nos tinha sido pedido, e que serviria de base a um desenvolvimento futuro para uma compilação de ferramentas de análise MPEG, mas também um conhecimento aprofundado da linguagem Java a um ponto que talvez não tivéssemos imaginado quando começámos, além de nos termos embrenhado no próprio MPEG e sua estrutura, que pensamos ser um dos aspectos mais aliciantes de todo este projecto, como também o é termos executado um trabalho útil numa área em tão grande desenvolvimento.

Como seria de esperar, tivemos de fazer várias adaptações e alterações ao que tínhamos inicialmente previsto. A começar pela a estrutura do programa principal, porque depois de conhecermos cada vez melhor o ficheiro de saída de dados do desmultiplexador, havia que alterar o algoritmo que havíamos idealizado com vista a torná-lo mais eficiente e assim conseguir uma optimização de tempo e capacidade de computação, além de o tornar de fácil utilização e manutenção.

Também tivemos que nos desdobrar para um trabalho de equipa mais eficiente, pois por vezes não era útil estarmos os dois a tentar resolver o mesmo assunto, exceptuando, claro está, quando o ditado "duas cabeças pensam melhor que uma" se podia aplicar.

O projecto deve ser algo de inovador, por isso esperamos que todo o nosso trabalho tenha contribuído de alguma forma para trazer algo de novo e poder servir de base a futuros desenvolvimentos nesta área com vista a uma utilização mais eficiente de todas as capacidades do MPEG.

Faculdade de Engenharia da Universidade do Porto, Julho de 2000

9. Anexos

9.1 Código da classe Analise

```
import java.io.*;
import java.lang.*;
import java.util.*;
import java.text.*;


public class Analise extends Object
{
	public static DataInputStream in;
	public static DataOutputStream pat;
	public static DataOutputStream pmt;
	public static DataOutputStream video;
	public static DataOutputStream audio;
	static String[] v2;

	//-----------------------------------------------//

	public Analise(String filename)
	{
		procura_PAT(filename);

	}


	//-----------------------------------------------//
	//procura o valor da flag "| PTS_DTS_flags = 3

	private String procura_flag(String s)
	{
		String s1="";
		    StringTokenizer st = new StringTokenizer(s);
		while(st.hasMoreTokens())
		{
			String tk1=st.nextToken();
			String tk2=st.nextToken();
			String tk3=st.nextToken();
			String tk4=st.nextToken();
			s1=tk4;
		}
		return s1;
	}


	//-----------------------------------------------//
	//procura o PID na string "program_map_PID=0xXXXX"

	private String procura_PMT_PID(String s)
	{
		String s3="";
		    StringTokenizer st = new StringTokenizer(s);
		while(st.hasMoreTokens())
		{
			String tk1=st.nextToken();
```

```
                if(tk1.equals("program_map_PID"))
                {
                      String tk2=st.nextToken();
                      if(tk2.equals("="))
                      {
                            String tk3=st.nextToken();
                                  s3=tk3;
                      }
                }
          }
          return s3;
    }


    //-----------------------------------------------//
    //procura o PID  na string "elementary_PID = 0xXXXX"

    private String procura_elementary_PID(String s)
    {
          String s3="";
              StringTokenizer st = new StringTokenizer(s);
          while(st.hasMoreTokens())
          {
                String tk1=st.nextToken();
                if(tk1.equals("elementary_PID"))
                {
                      String tk2=st.nextToken();
                      if(tk2.equals("="))
                      {
                            String tk3=st.nextToken();
                                  s3=tk3;
                      }
                }
          }
          return s3;
    }


    //-------------------------------------------------------//
    //procura o pacote com "PID=0x0000", escreve para ficheiro PSI
    //a informação de interesse contida na PAT

      private void procura_PAT(String filename )
    {
        try
        {
                File inputFile = new File(filename);
          DataInputStream in= new DataInputStream(new
BufferedInputStream(new FileInputStream(inputFile)));

          String nome="PAT"+filename;
          File outputFile = new File(nome);
          DataOutputStream pat=new DataOutputStream(new
BufferedOutputStream(new FileOutputStream(outputFile)));

          int i=0,j=1,k=0,l=1,m=1;
          int versao,indicador;
          int versao1=0;
          int indicador1=1;
          String s1="| PID = 0x0000 (Program Association Table)";
          String s2="| CRC_32 = 0";
```

```
            String s,s3;
            String[] v1=new String[500];//vector q contem os PID's dos
programas contidos na PAT
            String[] v3=new String[15];//vector q contem os PID's dos
elementary streams contidos na PMT
            Collator myCollator = Collator.getInstance();

            while((s=in.readLine()) != null)
            {
                  if((myCollator.compare(s, s1))==0)
                  {
                        pat.writeChar('\n');
                        pat.writeBytes(s); //copia p/ fich. pat
"PID=0x0000" (Program Association Table)
                        pat.writeChar('\n');
                        s=in.readLine();
                        pat.writeBytes(s); //copia p/ fich. pat
"transport_error_indicator"
                        pat.writeChar('\n');
                        s=in.readLine();
                        s=in.readLine();
                        pat.writeBytes(s); //copia p/ fich. pat
"transport_priority"
                        pat.writeChar('\n');
                        s=in.readLine();
                        pat.writeBytes(s); //copia p/ fich. pat
"transport_scrambling_control"
                        pat.writeChar('\n');
                        while(k<9)
                        {
                              s=in.readLine();
                              k++;
                        }
                        k=0;

                        pat.writeBytes(s); //copia p/ fich. pat
"version_number"
                        pat.writeChar('\n');
                        s=procura_flag(s);
                        Integer db1 =new Integer(s);//cast de string
para integer
                        versao=db1.intValue();//retorna o valor do
integer como int
                              s=in.readLine();
                        pat.writeBytes(s); //copia p/ fich. pat
"current_next_indicator"
                        pat.writeChar('\n');
                        s=procura_flag(s);
                        Integer db2 =new Integer(s);//cast de string
para integer
                        indicador=db2.intValue();//retorna o valor do
integer como int
                        if(versao==versao1 && indicador==indicador1)
                        {
                              while(i<4)
                              {
                                    s=in.readLine();
                                    i++;
                              }
                              i=0;
```

```
                              pat.writeBytes(s); //copia p/ fich. pat
"program_number"
                              pat.writeChar('\n');
                              s=in.readLine();
                              pat.writeBytes(s); //copia p/ fich. pat
"program_map_PID"
                              pat.writeChar('\n');
                              v1[j]=procura_PMT_PID(s);
                              s3=v1[j];
                              s3=s3+" ";
                              v1[j]=s3;
                              s=in.readLine();
                              s=in.readLine();
                              while((myCollator.compare(s, s2))==-1
||(myCollator.compare(s, s2))==1)
                              {
                                    pat.writeBytes(s);
                                    pat.writeChar('\n');
                                    s=in.readLine();
                                    pat.writeBytes(s);
                                    pat.writeChar('\n');
                                    j++;
                                    v1[j]=procura_PMT_PID(s);
                                    s3=v1[j];
                                    s3=s3+" ";
                                    v1[j]=s3;
                                    s=in.readLine();

                                    s=in.readLine();
                              }
                              while((v1[l])!=null)
                              {
                                    v3=procura_PMT(v1[l],filename);
                                    procura_video(v3[1],filename);
                                    procura_audio(v3[2],filename);
                                    l++;
                              }
                              l=1;j=1;
                              versao1++;
                        }
                  }
            }
            in.close();
                  pat.close();
      }
        catch(Exception e)
      {
        System.out.println("Problema no método procura_PAT");
      }
      }


     //-------------------------------------------------------//
     //procura o pacote com "PID=0xXXXX", escreve para ficheiro PSI
     //a informação de interesse contida na PMT com este PID retorna
     //um vector contendo o PID das elementary streams audio e video

       private String[] procura_PMT(String str,String filename)
     {
         try
         {
```

```
                File inputFile = new File(filename);
            DataInputStream in= new DataInputStream(new
BufferedInputStream(new FileInputStream(inputFile)));
            String nome="PMT"+filename;
            DataOutputStream pmt=new DataOutputStream(new
BufferedOutputStream(new FileOutputStream(nome,true)));

            int i=0,j=1,k=0,l=0,m=0,n=0,p=0;
            int versao,indicador;
            int versao1=0;
            int indicador1=1;
            String s1="| PID = "+ str;
            String s,tipo1,tipo2;
            String[] v1=new String[15];
            Collator myCollator = Collator.getInstance();

            while((s=in.readLine()) != null)
            {
                  if((myCollator.compare(s,s1))==0)
                  {
                        pmt.writeChar('\n');
                        pmt.writeBytes(s); //copia p/ fich. PSI
"PID=0xXXXX" (Program Map Table)
                        pmt.writeChar('\n');
                        while(i<7)
                        {
                              s=in.readLine();
                              i++;
                        }
                        i=0;
                        pmt.writeBytes(s); //copia p/ fich. PSI Program
Map Section
                        pmt.writeChar('\n');
                        while(n<6)
                        {
                              s=in.readLine();
                              n++;
                        }
                        n=0;

                        pmt.writeBytes(s); //copia p/ fich.
version_number
                        pmt.writeChar('\n');
                        s=procura_flag(s);
                        Integer db1 =new Integer(s);//cast de string
para integer
                        versao=db1.intValue();//retorna o valor do
integer como int
                        s=in.readLine();
                        pmt.writeBytes(s); //copia p/ fich.
current_next_indicator
                        pmt.writeChar('\n');
                        s=procura_flag(s);
                        Integer db2 =new Integer(s);//cast de string
para integer
                        indicador=db2.intValue();//retorna o valor do
integer como int
                        if(versao==versao1 && indicador==indicador1)
                        {
                              while(k<3)
                              {
```

```
                                    s=in.readLine();
                                    k++;
                              }
                              k=0;
                              pmt.writeBytes(s); //copia p/ fich.
PCR_PID
                              pmt.writeChar('\n');
                              while(l<3)
                              {
                                    s=in.readLine();
                                    l++;
                              }
                              l=0;
                              tipo1=s;
                              pmt.writeBytes(tipo1); //copia p/ fich.
stream type
                              pmt.writeChar('\n');
                              s=in.readLine();
                              pmt.writeBytes(s); //copia p/ fich.
elementary_PID
                              pmt.writeChar('\n');
                              v1[j]=procura_elementary_PID(s); //vector
com o PID do stream video ou audio
                              while(m<3)
                              {
                                    s=in.readLine();
                                    m++;
                              }
                              m=0;
                              tipo2=s;
                              pmt.writeBytes(tipo2); //copia p/ fich.
stream type
                              pmt.writeChar('\n');
                              s=in.readLine();
                              pmt.writeBytes(s); //copia p/ fich.
elementary_PID
                              pmt.writeChar('\n');
                              j++;
                              v1[j]=procura_elementary_PID(s); //vector
com o PID do stream video ou audio
                              v2=v1;
                              versao1++;
                        }
                  }
            }
            in.close();
                  pmt.close();
        }
          catch(Exception e)
        {
          System.out.println("Problema no método procura_PMT");
        }
        return v2;
        }


      //-------------------------------------------------------- //
      //procura o pacote com "PID=0xXXXX"=str, escreve para ficheiro
      //video a informação de video com este PID

        private void procura_video(String str,String filename)
```

```
      {
            try
            {
                        File inputFile = new File(filename);
                  DataInputStream in = new DataInputStream(new
BufferedInputStream(new FileInputStream(inputFile)));
                  String nome="video"+filename;
                  DataOutputStream video =new DataOutputStream(new
BufferedOutputStream(new FileOutputStream(nome,true)));

                  int i=0,j=0,k=0,l=0,n=0,p=0,q=0,r=0,t=0,v=0;
                  String s;
                  String s1="| PID = "+ str+" ";
                  String s2="|ADAPTATION
FIELD*****************************";
                  String s3="";
                  String s4="";
                  String s5="| stream_id = 0xE0   ->  Video stream -
number 0";
                  Collator myCollator = Collator.getInstance();

                  while((s=in.readLine()) != null)
                  {
                        if((myCollator.compare(s,s1))==0)//se o PID do
pacote for o q procuramos
                        {
                              while(i<7)
                              {
                                    s=in.readLine();
                                    i++;
                              }
                              i=0;
                              if((myCollator.compare(s,s2))==0)//se
tiver Adaptation Field..
                              {
                                    while(k<5)
                                    {
                                          s=in.readLine();
                                          k++;
                                    }
                                    k=0;
                                    video.writeChar('\n');
                                    video.writeBytes(s1);//copia p/
fich. o PID do pacote
                                    video.writeChar('\n');
                                    s4=procura_flag(s);//procuramos o
valor da flag "| PCR_flag = 0"
                                    Integer db3 =new Integer(s4);//cast
de string para integer
                                    v=db3.intValue();//retorna o valor
do integer como int
                                    switch(v)
                                    {
                                          case 0:     while(p<7) //nao
tem PCR
                                                {
                                                      s=in.readLine();
                                                      p++;
                                                }
                                                p=0;
```

```
                                                      break;//última linha
lida é "|PES PACKET*******"
                                                case 1:     while(n<6) //tem
PCR
                                                      {
                                                            s=in.readLine();
                                                            n++;
                                                      }
                                                      n=0;

      video.writeBytes(s);//copia p/ fich. "PCR_base"
                                                      video.writeChar('\n');
                                                      s=in.readLine();

      video.writeBytes(s);//copia p/ fich. "PCR_extension"
                                                      video.writeChar('\n');
                                                      while(j<3)
                                                      {
                                                            s=in.readLine();
                                                            j++;
                                                      }
                                                      j=0;
                                                      break;//última linha
lida é "|PES PACKET********"
                                          }
                                    }
                                    s=in.readLine();
                                    if((myCollator.compare(s,s5))==0)//se
tiver cabeçalho de pacote
                                    {
                                          while(l<7)
                                          {
                                                s=in.readLine();
                                                l++;
                                          }
                                          l=0;
                                          s3=procura_flag(s);//procuramos o
valor da flag "| PTS_DTS_flags = 3"
                                          Integer db2 =new Integer(s3);//cast
de string para integer
                                          t=db2.intValue();//retorna o valor
do integer como int
                                          switch(t)
                                          {
                                                case 0:     break;
                                                case 2:     while(q<9)
                                                      {
                                                            s=in.readLine();
                                                            q++;
                                                      }
                                                      q=0;

      video.writeBytes(s);//copia p/ fich. "PTS"
                                                      video.writeChar('\n');
                                                      break;
                                                case 3:     while(r<9)
                                                      {
                                                            s=in.readLine();
                                                            r++;
                                                      }
                                                      r=0;
```

```
      video.writeBytes(s);//copia p/ fich. "PTS"
                                                video.writeChar('\n');
                                                s=in.readLine();

      video.writeBytes(s);//copia p/ fich. "DTS"
                                                video.writeChar('\n');
                                                break;
                                          }
                                    }
                              }
                        }
                        in.close();
                              video.close();
                  }
                  catch(Exception e)
                  {
                         System.out.println("Problema no método
procura_video");
                  }
      }


      //------------------------------------------------------------//
      //procura o pacote com "PID=0xXXXX"=str, escreve para ficheiro
      //audio a informação de audio com este PID

        private void procura_audio(String str,String filename)
      {
            try
            {
                        File inputFile = new File(filename);
                  DataInputStream in = new DataInputStream(new
BufferedInputStream(new FileInputStream(inputFile)));
                  String nome="audio"+filename;
                  DataOutputStream audio =new DataOutputStream(new
BufferedOutputStream(new FileOutputStream(nome,true)));

                  int i=0,l=0,q=0,r=0,t=0;
                  String s;
                  String s1="| PID = "+ str+" ";
                  String s2="| stream_id = 0xC0  ->  Audio stream -
number 0";
                  String s3="";
                  Collator myCollator = Collator.getInstance();

                  while((s=in.readLine()) != null)
                  {
                        if((myCollator.compare(s,s1))==0)//se o PID do
pacote for o q procuramos
                        {
                              while(i<8)
                              {
                                    s=in.readLine();
                                    i++;
                              }
                              i=0;
                              //última linha lida é "| stream_id = 0xC0
->  Audio stream - number 0"
                              if((myCollator.compare(s,s2))==0)//se
tiver cabeçalho de pacote
```

```
                                {
                                      while(l<7)
                                      {
                                            s=in.readLine();
                                            l++;
                                      }
                                      l=0;
                                      audio.writeChar('\n');
                                      audio.writeBytes(s1);//copia p/
fich. o PID do pacote
                                      audio.writeChar('\n');
                                      s3=procura_flag(s);//procuramos o
valor da flag "| PTS_DTS_flags = 3"
                                      Integer db2 =new Integer(s3);//cast
de string para integer
                                      t=db2.intValue();//retorna o valor
do integer como int
                                      switch(t)
                                      {
                                            case 0:     break;
                                            case 2:     while(q<9)
                                                  {
                                                        s=in.readLine();
                                                        q++;
                                                  }
                                                  q=0;

      audio.writeBytes(s);//copia p/ fich. "PTS"
                                                  audio.writeChar('\n');
                                                  break;
                                            case 3:     while(r<9)
                                                  {
                                                        s=in.readLine();
                                                        r++;
                                                  }
                                                  r=0;

      audio.writeBytes(s);//copia p/ fich. "PTS"
                                                  audio.writeChar('\n');
                                                  s=in.readLine();

      audio.writeBytes(s);//copia p/ fich. "DTS"
                                                  audio.writeChar('\n');
                                                  break;
                                      }
                                      t=0;
                                }
                          }
                    }
                    in.close();
                          audio.close();
              }
              catch(Exception e)
              {
                     System.out.println("Problema no método
procura_audio");
              }
      }

}
```

9.2 Código da classe ImageFrame

```
import java.awt.*;
import javax.swing.*;
import javax.swing.text.*;
import java.io.*;
import java.lang.Object;


public class ImageFrame extends javax.swing.JInternalFrame
{

  private javax.swing.JTextArea jTextArea1;
  private javax.swing.JScrollPane areaScrollPane;
  public static InputStreamReader in;

  static int openFrameCount = 0;
  static final int xOffset = 10, yOffset = 10;


  //-------------------------------------------------------
  public ImageFrame(String imageName,String filename)
  {
    initComponents(filename);
    setTitle (imageName);
  }


  //-------------------------------------------------------
  private void initComponents (String filename)
  {
    jTextArea1 = new javax.swing.JTextArea ();
    areaScrollPane = new JScrollPane(jTextArea1);


areaScrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_S CROLLBAR
_ALWAYS);
    areaScrollPane.setPreferredSize(new Dimension(1000, 1000));

    getContentPane ().setLayout (new javax.swing.BoxLayout
(getContentPane (), 0));
    setClosable (true);
    setResizable (true);
    setIconifiable (true);
    setMaximizable (true);

    jTextArea1.setFont(new Font("Verdana", Font.PLAIN, 14));
    jTextArea1.setLineWrap(true);
    jTextArea1.setWrapStyleWord(true);
    jTextArea1.setEditable (false);

    try
    {
            File inputfile = new File(filename);
        InputStreamReader in= new InputStreamReader(new
DataInputStream(new BufferedInputStream(new
FileInputStream(filename))));
      String desc="";
```

```
        jTextArea1.read(in,desc);
            in.close();
    }
    catch(Exception e)
    {
            System.out.println("Erro na abertura do ficheiro");
    }

    getContentPane ().add (areaScrollPane);
  }
}
```

9.3 Código da classe Janela

```
import javax.swing.*;
import java.lang.*;
import java.awt.*;
import java.awt.event.*;

public class janela extends javax.swing.JFrame {


//--------------------------------------
//|       Declaração de Variaveis    |
//--------------------------------------
  private javax.swing.JMenuBar jMenuBar1;
  private javax.swing.JMenu Analisar;
  private javax.swing.JSeparator jSeparator5;
  private javax.swing.JMenuItem Abrir;
  private javax.swing.JSeparator jSeparator7;
  private javax.swing.JMenu InfoPSI;
  private javax.swing.JSeparator jSeparator1;
  private javax.swing.JMenuItem PAT;
  private javax.swing.JSeparator jSeparator2;
  private javax.swing.JMenuItem PMT;
  private javax.swing.JSeparator jSeparator3;
  private javax.swing.JMenuItem video;
  private javax.swing.JSeparator jSeparator4;
  private javax.swing.JMenuItem audio;
  private javax.swing.JSeparator jSeparator6;
  private javax.swing.JMenuItem Fechar;
  private javax.swing.JSeparator jSeparator11;
  private javax.swing.JMenuItem Sair;
  private javax.swing.JMenu Correr;
  private javax.swing.JSeparator jSeparator8;
  private javax.swing.JMenuItem Video;
  private javax.swing.JSeparator jSeparator9;
  private javax.swing.JMenuItem Audio;
  private javax.swing.JSeparator jSeparator10;
  private javax.swing.JMenuItem VideoEAudio;
  private javax.swing.JMenu Ajuda;
  private javax.swing.JSeparator jSeparator12;
  private javax.swing.JMenuItem Ajuda1;
  private javax.swing.JSeparator jSeparator13;
  private javax.swing.JMenuItem Acerca;


  private JLayeredPane desktop;
  private java.awt.FileDialog fd;
  private Analise ler;
  private ImageFrame ifr=null;
```

```
  private ImageFrame ifr1=null;
  private ImageFrame ifr2=null;
  private ImageFrame ifr3=null;
  private ImageFrame ifr4=null;
  private ImageFrame ifr5=null;
  private ImageFrame ifr6=null;

  private int count=0;
  private final int x=20,y=24;


//--------------------------------------
//|          Cria nova janela       |
//--------------------------------------
  public janela() {

    desktop = new JDesktopPane();
    desktop.repaint();
    initComponents ();
  }


//--------------------------
//|    initComponents       |
//--------------------------
  private void initComponents ()
  {
    jMenuBar1 = new javax.swing.JMenuBar ();
    Analisar = new javax.swing.JMenu ();
    jSeparator5 = new javax.swing.JSeparator ();
    Abrir = new javax.swing.JMenuItem ("Abrir",KeyEvent.VK_A);
    jSeparator7 = new javax.swing.JSeparator ();
    InfoPSI = new javax.swing.JMenu ();
    jSeparator1 = new javax.swing.JSeparator ();
    PAT = new javax.swing.JMenuItem ("PAT",KeyEvent.VK_T);
    jSeparator2 = new javax.swing. JSeparator ();
    PMT = new javax.swing.JMenuItem ("PMT",KeyEvent.VK_M);
    jSeparator3 = new javax.swing.JSeparator ();
    video = new javax.swing.JMenuItem ("video",KeyEvent.VK_V);
    jSeparator4 = new javax.swing.JSeparator ();
    audio = new javax.swing.JMenuItem ("audio",KeyEvent.VK_D);
    jSeparator6 = new javax.swing.JSeparator ();
    Fechar = new javax.swing.JMenuItem ("Fechar",KeyEvent.VK_F);
    jSeparator11 = new javax.swing.JSeparator ();
    Sair = new javax.swing.JMenuItem ("Sair",KeyEv ent.VK_S);
    Correr = new javax.swing.JMenu ();
    jSeparator8 = new javax.swing.JSeparator ();
    Video = new javax.swing.JMenuItem ("Video",KeyEvent.VK_E);
    jSeparator9 = new javax.swing.JSeparator ();
    Audio = new javax.swing.JMenuItem ("Audio ",KeyEvent.VK_I);
    jSeparator10 = new javax.swing.JSeparator ();
    VideoEAudio = new javax.swing.JMenuItem
("VideoEAudio",KeyEvent.VK_R);

    Ajuda = new javax.swing.JMenu ();
    jSeparator12 = new javax.swing.JSeparator ();
    Ajuda1 = new javax.swing.JMenuItem ("Ajuda1",KeyEvent.VK_J);
    jSeparator13 = new javax.swing.JSeparator ();
    Acerca = new javax.swing.JMenuItem ("Acerca",KeyEvent.VK_L);
    jMenuBar1.setPreferredSize (new java.awt.Dimension(125, 23));
```

```
      Analisar.setText ("Analisar");
      Analisar.add (jSeparator5);

      Abrir.setText ("Abrir");
        Abrir.addActionListener (new java.awt.event.ActionListener () {
            public void actionPerformed (java.awt.event.ActionEvent evt)
{
                  AbrirActionPerformed (evt);
                  }
        }
        );
        Abrir.setMnemonic(KeyEvent.VK_A);
        Abrir.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_A,
ActionEvent.ALT_MASK));
        Abrir.setLabel ("Abrir");
        Analisar.add (Abrir);
        Analisar.add (jSeparator7);

        InfoPSI.setText ("InfoPSI");
        InfoPSI.setLabel ("InfoPSI");
      InfoPSI.add (jSeparator1);

        PAT.setText ("PAT");
        PAT.setMnemonic(KeyEvent.VK_T);
        PAT.setAccelerator(KeyStroke.getKeyStroke(KeyEv ent.VK_T,
ActionEvent.ALT_MASK));
        PAT.setLabel ("PAT");
        PAT.addActionListener (new java.awt.event.ActionListener () {
            public void actionPerformed (java.awt.event.ActionEvent evt)
{
                        PATActionPerformed (evt);
                  }
            }
        );
        InfoPSI.add (PAT);
        InfoPSI.add (jSeparator2);

        PMT.setText ("PMT");
        PMT.setMnemonic(KeyEvent.VK_M);
        PMT.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_M,
ActionEvent.ALT_MASK));
        PMT.setLabel ("PMT");
        PMT.addActionListener (new java.awt.event.ActionListener () {
            public void actionPerformed (java.awt.event.ActionEvent evt)
{
                        PMTActionPerformed (evt);
                  }
        }
        );
        InfoPSI.add (PMT);
        InfoPSI.add (jSeparator3);

        video.setText ("video");
        video.setMnemonic(KeyEvent.VK_V);
        video.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_V,
ActionEvent.ALT_MASK));
        video.setLabel ("video");
        video.addActionListener (new java.awt.event.ActionListener () {
            public void actionPerformed (java.awt.event.ActionEvent evt)
{
```

```
                        videoActionPerformed (evt);
                    }
        }
        );
        InfoPSI.add (video);
        InfoPSI.add (jSeparator4);

        audio.setText ("audio");
        audio.setMnemonic(KeyEvent.VK_D);
        audio.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_D,
ActionEvent.ALT_MASK));
        audio.setLabel ("audio");
        audio.addActionListener (new java.awt.event.ActionListener () {
            public void actionPerformed (java.awt.event.ActionEvent evt)
{
                        audioActionPerformed (evt);
                    }
        }
        );
        InfoPSI.add (audio);
        Analisar.add (InfoPSI);
        Analisar.add (jSeparator6);

        Fechar.setText ("Fechar");
        Fechar.setMnemonic(KeyEvent.VK_F);
        Fechar.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_F,
ActionEvent.ALT_MASK));
        Fechar.setLabel ("Fechar");
        Analisar.add (Fechar);
        Analisar.add (jSeparator11);

        Sair.setText ("Sair");
        Sair.setMnemonic(KeyEvent.VK_S);
        Sair.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S,
ActionEvent.ALT_MASK));
        Sair.setLabel ("Sair");
        Sair.addActionListener (new java.awt.event.ActionListener () {
            public void actionPerformed (java.awt.event.ActionEvent evt)
{
                        SairActionPerformed (evt);
            }
        }
        );
        Analisar.add (Sair);

      jMenuBar1.add (Analisar);

      Correr.setText ("Correr");
      Correr.setLabel ("Correr");
        Correr.add (jSeparator8);

        Video.setText ("Video");
        Video.setMnemonic(KeyEvent.VK_E);
        Video.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_E,
ActionEvent.ALT_MASK));
        Video.addActionListener (new java.awt.event.ActionListener () {
          public void actionPerformed (java.awt.event.ActionEvent evt) {
            VideoActionPerformed (evt);
          }
        }
        );
```

```
        Correr.add (Video);
        Correr.add (jSeparator10);

        Audio.setText ("Audio");
        Audio.setMnemonic(KeyEvent.VK_I);
        Audio.setAccelerator(KeyStroke.getKeyStroke(KeyEvent. VK_I,
ActionEvent.ALT_MASK));
        Audio.addActionListener (new java.awt.event.ActionListener () {
          public void actionPerformed (java.awt.event.ActionEvent evt) {
            AudioActionPerformed (evt);
          }
        }
        );
        Correr.add (Audio);
        Correr.add (jSeparator9);

        VideoEAudio.setText ("Video&Audio");
        VideoEAudio.setMnemonic(KeyEvent.VK_R);
        VideoEAudio.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_R,
ActionEvent.ALT_MASK));
        VideoEAudio.addActionListener (new java.awt.event.ActionListener
() {
          public void actionPerformed (java.awt.event.ActionEvent evt) {
            VideoEAudioActionPerformed (evt);
          }
        }
        );
        Correr.add (VideoEAudio);
            jMenuBar1.add (Correr);


            Ajuda.setText ("Ajuda");
            Ajuda.setLabel ("Ajuda");


        Ajuda.add (jSeparator12);
        Ajuda1.setText ("Ajuda");
        Ajuda1.setMnemonic(KeyEvent.VK_J);
        Ajuda1.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_J,
ActionEvent.ALT_MASK));
        Ajuda1.setLabel ("Ajuda");
        Ajuda1.addActionListener (new java.awt.event.ActionListener () {
          public void actionPerformed (java.awt.event.ActionEven t evt) {
            Ajuda1ActionPerformed (evt);
          }
        }
        );

        Ajuda.add (Ajuda1);
      Ajuda.add (jSeparator13);

        Acerca.setText ("Acerca..");
        Acerca.setMnemonic(KeyEvent.VK_L);
        Acerca.setAccelerator(K eyStroke.getKeyStroke(KeyEvent.VK_L,
ActionEvent.ALT_MASK));
        Acerca.setLabel ("Acerca..");
        Acerca.addActionListener (new java.awt.event.ActionListener () {
          public void actionPerformed (java.awt.event.ActionEvent evt) {
            AcercaActionPerformed (evt);
          }
        }
```

```
        );

        Ajuda.add (Acerca);
      jMenuBar1.add (Ajuda);

      getContentPane ().setLayout (new javax.swing.BoxLayout
(getContentPane (), 0));
      setTitle ("Análise MPEG");
      addWindowListener (new java.awt.event.WindowAdapter () {
                  public void windowClosing (java.awt.event.WindowEvent
evt) {
                  exitForm (evt);
            }
      }
      );

      setJMenuBar (jMenuBar1);
      desktop.setOpaque(true);
      getContentPane().add(desktop);

      setSize (700, 700);
      setLocation (200, 50);
      setVisible(true);

  }



//--------------------
//|   Eventos         |
//--------------------

      private void AbrirActionPerformed (java.awt.event.ActionEvent evt)
//event_AbrirActionPerformed
      {
            fd = new java.awt.FileDialog (this);
            fd.show ();

            if (fd.getFile () == null) return;
            ifr = new ImageFrame7 (fd.getDirectory () + fd.getFile
(),fd.getFile ());
            ifr.setVisible(true);
            desktop.add (ifr, javax.swing.JLayeredPa ne.DEFAULT_LAYER);
            try
            {
                        ifr.setSelected(true);
            }
            catch (java.beans.PropertyVetoException e){}

      desktop.putClientProperty("JDesktopPane.dragMode","outline");//ao
movermos a janela so e desenhado o border
            ifr.setSize (500, 500);
            ifr.setLocation(x*count, y*count);
            count++;

            ler = new Analise(fd.getFile ());

      }

      //---------------------------------------------------------
      //---------------------------------------------------------
```

```
       private void PATActionPerformed (java.awt.event.ActionEvent evt)
       {
            String pathPAT=fd.getDirectory () + "PAT" + fd.getFile ();
            String fixPAT="PAT" + fd.getFile ();
            ifr1 = new ImageFrame7 (pathPAT,fixPAT);
            ifr1.setVisible(true);
            desktop.add (ifr1, javax.swing.JLayeredPane.DEFAULT_LAYER);
            try {
                        ifr1.setSelected(true);
            }
            catch (java.beans.PropertyVetoException e){}

      desktop.putClientProperty("JDesktopPane.dragMode","outline");//ao
movermos a janela so e desenhado o border
            ifr1.setSize (500, 500);
            ifr1.setLocation (x*count, y*count);
            count++;
       }

      //--------------------------------------------------------
      private void PMTActionPerformed (java.awt.event.ActionEvent evt)
      {
            String pathPMT=fd.getDirectory () + "PMT" + fd.getFile ();
            String fixPMT="PMT" + fd.getFile ();
            ifr2 = new ImageFrame7 (pathPMT,fixPMT);
            ifr2.setVisible(true);
            desktop.add (ifr2, javax.swing.JLayeredPane.DEFAULT_LAYER);
            try {
                        ifr2.setSelected(true);
            }
            catch (java.beans.PropertyVetoException e){}

      desktop.putClientProperty("JDesktopPane.dragMode","outline");//ao
movermos a janela so e desenhado o border
            ifr2.setSize (500, 500);
            ifr2.setLocation (x*count, y*count);
            count++;
      }

      //--------------------------------------------------------
      private void videoActionPerformed (java.awt.event.ActionEvent evt)
      {
            String pathvideo=fd.getDirectory () + "video" + fd.getFile
();
            String fixvideo="video" + fd.getFile ();
            ifr3 = new ImageFrame7 (pathvideo,fixvideo);
            ifr3.setVisible(true);
            desktop.add (ifr3, javax.swing.JLayeredPane.DEFAULT_LAYER);
            try {
                        ifr3.setSelected(true);
            }
            catch (java.beans.PropertyVetoException e){}

      desktop.putClientProperty("JDesktopPane.dragMode","outline");//ao
movermos a janela so e desenhado o border
            ifr3.setSize (500, 500);
            ifr3.setLocation (x*count, y*count);
            count++;
      }

      //--------------------------------------------------------
```

```
      private void audioActionPerformed (java.awt.event.ActionEvent evt)
      {
            String pathaudio=fd.getDirectory () + "audio" + fd.getFile
();
            String fixaudio="audio" + fd.getFile ();
            ifr4 = new ImageFrame7 (pathaudio,fixaudio);
            ifr4.setVisible(true);
            desktop.add (ifr4, javax.swing.JLayeredPane.DEFAULT_LAYER);
            try {
                        ifr4.setSelected(true);
            }
            catch (java.beans.PropertyVetoException e){}

      desktop.putClientProperty("JDesktopPane.dragMode","outline");//ao
movermos a janela so e desenhado o border
            ifr4.setSize (500, 500);
            ifr4.setLocation (x*count, y*count);
            count++;
      }

      //---------------------------------------------------------
      //---------------------------------------------------------
      private void FecharActionPerformed (java.awt.event.ActionEvent
evt)
      {
            ifr.setVisible(false);
            ifr.dispose();
            ifr1.setVisible(false);
            ifr1.dispose();
            ifr2.setVisible(false);
            ifr2.dispose();
            ifr3.setVisible(false);
            ifr3.dispose();
            ifr4.setVisible(false);
            ifr4.dispose();
            //desktop.repaint();
      }

      //---------------------------------------------------------
      private void SairActionPerformed (java.awt.event.ActionEvent  evt)
//event_SairActionPerformed
      {
            System.exit (0);
      }

      //---------------------------------------------------------
      //---------------------------------------------------------
      private void VideoActionPerformed (java.awt.event.ActionEvent evt)
      {

      }

      private void AudioActionPerformed (java.awt.event.ActionEvent evt)
      {

      }

      private void VideoEAudioActionPerformed
(java.awt.event.ActionEvent evt)
      {
```

```
	}

	//---------------------------------------------------
	//---------------------------------------------------
	private void Ajuda1ActionPerformed (java.awt.event.ActionEvent
evt)
	{
		String pathPAT= "ajuda.txt";
		String fixPAT="ajuda.txt";
		ifr5 = new ImageFrame7 (pathPAT,fixPAT);
		ifr5.setVisible(true);
		desktop.add (ifr5, javax.swing.JLayeredPane.DEFAULT_LAYER);
		try {
				ifr5.setSelected(true);
		}
		catch (java.beans.PropertyVetoException e){}

	desktop.putClientProperty("JDesktopPane.dragMode","outline");//ao
movermos a janela so e desenhado o border
		ifr5.setSize (500, 500);
		ifr5.setLocation (x*count, y*count);
		count++;
	}

	//---------------------------------------------------
	private void AcercaActionPerformed (java.awt.event.ActionEvent
evt)
	{
		String pathPAT="acerca.txt";
		String fixPAT="acerca.txt";
		ifr6 = new ImageFrame7 (pathPAT,fixPAT);
		ifr6.setVisible(true);
		desktop.add (ifr6, javax.swing.JLayeredPane.DEFAULT_LAYER);
		try {
				ifr6.setSelected(true);
		}
		catch (java.beans.PropertyVetoException e){}

	desktop.putClientProperty("JDesktopPane.dragMode","outline");//ao
movermos a janela so e desenhado o border
		ifr6.setSize (500, 500);
		ifr6.setLocation (x*count, y*count);
		count++;
	}

	//---------------------------------------------------
	//---------------------------------------------------
	private void exitForm(java.awt.event.WindowEvent evt)  /** Exit
the Application */
	{
		System.exit (0);
	}

	//---------------------------------------------------
	//---------------------------------------------------
	public static void main (String args[])
	{
		new janela ().show ();
	}

}
```

10. Bibliografia

[1] Digital Video: an introduction to MPEG-2 –B.Haskell, A.Puri, A.Netravali - Chapman & Hall, 1997

[2] Digital Television, MPEG-1, MPEG-2 and principles of the DVB system –H.Benoit –Arnold, 1997

[3] A Guide to MPEG Fundamentals and Protocol Analysis (Including DVB and ATSC) –Tektronix, 1998

[4] MPEG-2, What it is and what it isn t, Colloquium organised by Professional Group E14 –IEE, 1995

[5] The Development of na Integrated Audiovisual Coding Standard: MPEG – L.Chiariglione – Proceedings of the IEEE, Vol.83, No.2, 1995

[6] MPEG2 system bitstreams parser: ISO/IEC 13818-1 (NO721 - June 94), version 18 –A.Bragança – INESC/Programa Ciencia, 8 de Novembro de 1994

[7] Aprenda Java Já –S.Davis –McGraw-Hill de Portugal, 1998

[8] The Java Language Environment, a white paper – J.Gosling, H.McGilton – Sun microsystems, JavaSoft, 1996

[9] Thinking in Java –B.Eckel –Prentice Hall PTR, 1998

[10] Java Sourcebook –E.Anuff –John Wiley & Sons, 1996

[11] The Java Programming Language –K.Arnold, J.Gosling –Sun microsystems, 1996

[12] Java Examples in a Nutshell –D.Flanagan –O Riley & Associates Inc., 1997

[13] Java Language Reference –M.Grand - O Riley & Associates Inc., 1997

11. Sites de consulta

www.am.hhi.de/mpeg-video/papers/sikora/mpeg1_2.htm

www.cs.sfu.ca/undergrad/CourseMaterials/CMPT479/material/notes/Chap4/Chap4.2/Chap4.2.html

www.mpeg.org/MPEG/video.html

www.wwgsolutions.com/appnotes/mpeg/mpeg_dolby_elementary_stream_analysis.html

www.wwgsolutions.com/appnotes/mpeg/mpegapp.html

www.zenith.com/mpeg_tutorial

gps-tsc.upc.es/imatge/MAIN/TEI

java.sun.com/docs/books/tutorial

java.sun.com/products/jdk/1.2/docs/api/