Universidade de Aveiro
2009
Departamento de Electrónica, Telecomunicações e Informática

José Maurício Faria Andrade

Sistema de difusão de televisão experimental da Universidade de Aveiro

Universidade de Aveiro Departamento de Electrónica,
2009 Telecomunicações e Informática

José Maurício Faria Andrade

Sistema de difusão de televisão experimental da Universidade de Aveiro

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestrado Integrado em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica do Dr. António José Nunes Navarro Rodrigues, Professor auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

O júri

Prof. Dr. Paulo Miguel Nepomuceno Pereira Monteiro
Professor Associado da Universidade de Aveiro.

Prof. Dr. Artur Pimenta Alves
Professor Catedrático da Faculdade de Engenharia da Universidade do Porto.

Prof. Dr. António José Nunes Navarro Rodrigues
Professor Auxiliar da Universidade de Aveiro.

Agradecimentos

Agradeço de modo sincero e profundo a todos que, de algum modo, colaboraram comigo na realização deste trabalho.

Em destaque devo agradecer ao Professor Doutor António José Nunes Navarro Rodrigues, meu orientador científico, por todo o apoio, orientação, disponibilidade fornecida e condições proporcionadas, além da oportunidade que me deu na realização deste trabalho muito aliciante. Agradeço também, pelo acompanhamento e conhecimentos que me foram transmitidos ao longo da realização do trabalho.

Também cabe-me agradecer ao Engenheiro Nuno Coelho, meu colega no laboratório de Vídeo e Televisão Digital, por também me ter acompanhado e transmitido alguns conhecimentos, além do fornecido pelo meu orientador científico.

Ao Instituto de Telecomunicações de Aveiro, por me ter dado todas as condições de trabalho em termos de recursos humanos, instalações e equipamentos necessários para o desenvolvimento deste trabalho.

Índice

Índice.....1

Índice de figuras.....5

Índice de tabelas.....8

Capítulo 1 – Introdução

1.1 Resumo.....9

1.2 Introdução.....11

1.3 Motivação digital vs analógico.....13

Capítulo 2 – Digital Vídeo Broadcasting

2.1 DVB.....15

2.2 DVB-T.....16

2.2.1 Modulação.....17

2.3 MPEG-2.....20

2.4 Conceitos de MPEG-2 Sistemas.....21

2.4.1 Pacotes Nulos.....21

2.4.2 PES – *Packetized Elementary Stream*.....21

2.4.3 Sinalização de serviços.....21

Capítulo 3 – Transporte em DVB-T

3.1 Introdução.....25

3.2 *Real-Time Transport Protocol* (RTP).....27

3.3 *User Datagram Protocol* (UDP).....32

3.4 *Internet Protocol* (IP).....35

3.5 *Multiprotocol Encapsulation* (MPE).....42

3.6 MPEG-2 *Transport Stream* (MPEG-2 TS).....48

Capítulo 4 – Receptores DVB-T

4.1 Introdução.....54

4.2 Tecnologias mais comuns.....55

4.2.1 Modo de Diversidade.....55

4.2.2 Modo de Duplo Sintonizador.....56

4.3 Dispositivos em mercado para computadores pessoais.....57

Capítulo 5 – DirectShow

5.1 Introdução.....61

5.2 Filtros.....63

5.3 Grafo de filtros.....66

5.4 Componentes de construção de um grafo de filtros.....67

5.5 *Filter Graph Manager*.....67

5.5.1 Transições de estado nos filtros.....68

5.5.2 Interacção entre aplicação e grafo de filtros.....69

5.5.3 Interacção do *hardware* e grafo de filtros.....70

5.6 Ligações entre filtros.....71

5.6.1 *Media types*.....71

5.6.2 *Transports*.....72

5.6.3 *Allocators* e *media samples*.....73

5.6.4 *Intelligent connect*.....76

5.7 Implementação de filtros.....76

5.7.1 *Classes*.....77

5.7.2 Criação de pinos.....77

Capítulo 6 – Desenvolvimento da aplicação

6.1 Introdução.....78

6.2 Problemas do insersor de IP da Rohde&Schwarz.....78

6.3 Implementação do encapsulador.....80

6.4 Implementação do filtro.....82

6.5 Desenvolvimento da aplicação.....84

6.5.1 Ferramentas auxiliares à construção do grafo.....85

6.5.2 Desenvolvimento do grafo de filtros.....85

6.4.3 Interface da aplicação.....88

Capítulo 7 – Resultados.....90

Capítulo 8 – Conclusão e trabalho futuro.....94

Referências.....96

Anexos

Anexo 1 – encapsulador.cpp.....101

Anexo 2 – Dump_m4u.cpp.....107

Anexo 3 **-** FilterGraphTools.cs.....127

Anexo 4 – BDAGraphBuilder.cs.....130

Índice de figuras

Figura 1. Ilustração do sistema DVB. 10

Figura 2. Logo Oficial DVB. 15

Figura 3. Logo Oficial DVB-T. 16

Figura 4. Distribuição global dos sistemas de difusão de televisão digital terrestre. 17

Figura 5. Diagrama de constelação 16-QAM e alocação de bits..... 18

Figura 6. Estrutura das portadoras OFDM…..... 19

Figura 7. Relação entre as tabelas PSI. 23

Figura 8. Codificação na fonte e Camada de sistema..... 26

Figura 9. Relação entre os vários protocolos que compõem o MPEG-2 TS. 27

Figura 10. Estrutura do protocolo RTP. 28

Figura 11. Estrutura do cabeçalho RTP. 29

Figura 12. Estrutura do protocolo UDP. 33

Figura 13. Estrutura do protocolo IP. 36

Figura 14. Estrutura do cabeçalho IP. 36

Figura 15. Estrutura do protocolo MPE. 43

Figura 16. Estrutura do cabeçalho MPE. 43

Figura 17. Mapeamento do endereço MAC. 44

Figura 18. Estrutura dos pacotes TS. 49

Figura 19. Estrutura do *Adaptation Field.* 51

Figura 20. Receptor *Diversity* com n antenas. 55

Figura 21. Cenário 1. 56

Figura 22. Cenário 2. 56

Figura 23. Cenário 3. 56

Figura 24. Terratec – Cinergy DT USB XS Diversity. 58

Figura 25. Terratec - Cinergy HTC USB XS HD. 58

Figura 26. Terratec - Cinergy T USB XE. 59

Figura 27. Terratec - Cinergy 2400i DT. 59

Figura 28. Terratec - Cinergy T Express. 60

Figura 29. Logo de DirectX Media SDK. 61

Figura 30. Diagrama genérico de uma aplicação em DirectShow. 63

Figura 31. Exemplos de *Source filters*. 64

Figura 32. Exemplos de *Transform filters*. 65

Figura 33. Exemplos de *Renderer filters*. 65

Figura 34. Grafo de filtros em GraphEdit. 66

Figura 35. Interacção entre aplicação e grafo de filtros. 69

Figura 36. Exemplo de um filtro *wrapper*. 70

Figura 37. Ligações entre dois filtros. 74

Figura 38. Insersor R&S – Problema 1…..79

Figura 39. Insersor R&S – Problema 2.....80

Figura 40. Geração do fluxo de dados MPEG-2 TS e emissão.....81

Figura 41. Diagrama de blocos - Encapsulador.....81

Figura 42. Recepção do fluxo de dados MPEG-2 TS e descodificação.....83

Figura 43. Diagrama de blocos - Desencapsulador.....83

Figura 44. Logo de Visual Studio 2005. 84

Figura 45. Grafo de filtros gerado pela aplicação desenvolvida. 87

Figura 46. Interface de definição dos parâmetros de sintonia. 88

Figura 47. Interface "mãe" da aplicação desenvolvida. 88

Figura 48. Interface de informação acerca da aplicação. 89

Figura 49. Análise ao ficheiro de teste. 91

Figura 50. Ficheiro resultante do algoritmo de encapsulamento. 91

Figura 51. Fluxo de dados emitidos no sistema DVB-T. 92

Figura 52. Resultado da aplicação do algoritmo de desencapsulamento.....93

Índice de tabelas

Tabela 1. Significados de *type of service*. 38

Tabela 2. Significados de *flags*. 39

Tabela 3. Exemplos de campos de *option*. 40

Tabela 4. Descrição das possibilidades em *payload scrambling control*. 45

Tabela 5. Descrição das possibilidades em *address scrambling control*. 46

Tabela 6. Significados possíveis de *LLC SNAP flag*. 46

Tabela 7. Valores para *Packet IDentifier*. 50

Tabela 8. Valores para *transport scrambling control*. 50

Tabela 9. Valores para *adaptation field control*. 50

Capítulo 1

Introdução

1.1 Resumo

O presente projecto descreve uma arquitectura para aprendizagem electrónica (e-learning) através da recepção de um sinal de televisão digital terrestre (DVB-T – *Digital Vídeo Broadcasting - Terrestrial*). De acordo com este conceito, nesta dissertação foi desenvolvida uma aplicação usando DirectShow, responsável pela recepção do sinal IP (Internet Protocol) sobre DVB-T, sinal este que transporta toda a informação e é característico do nosso sistema DVB. A aplicação é um componente fundamental no sistema de difusão de televisão digital (DVB), com respectiva emissão, numa primeira fase por todo o campus e proximidades da Universidade de Aveiro, de modo a permitir a recepção de aulas interactivas através de um sinal de televisão digital. Os alunos devem possuir sistemas de recepção compatíveis com o sistema DVB. Nomeadamente um dispositivo USB ou PCMCIA a ligar ao computador pessoal. Para além disso, devem ainda instalar a aplicação desenvolvida neste projecto. É pressuposto ainda a utilização de outras ferramentas de comunicação, como por exemplo uma aplicação de *chat*, de modo a permitir interactividade em tempo real entre alunos e professor. Por fim, o sistema pode ainda ser completado com a inclusão de uma câmara (webcam) e um microfone em cada computador pessoal, permitindo a existência de sinal de vídeo e áudio no canal de retorno.

Em seguida é apresentada uma ilustração do possível sistema de difusão de televisão digital (DVB), desenvolvido para a transmissão das aulas interactivas.

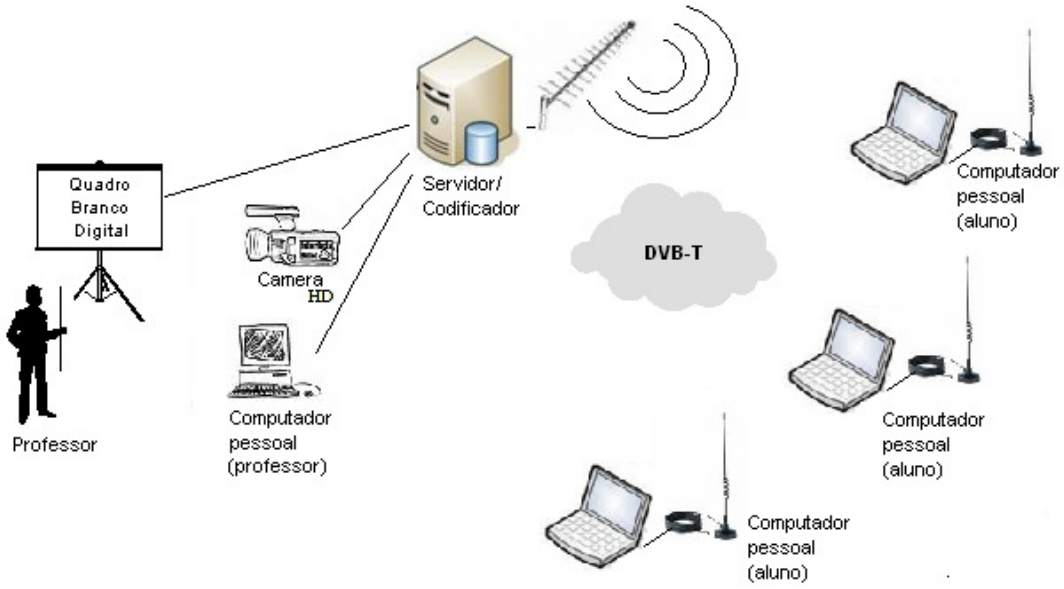


Figura 1. Ilustração do sistema DVB.

1.2 Introdução

Actualmente é constatado que cada vez mais o cenário de ensino acompanha o avanço tecnológico, pelo que o ensino à distância, como por exemplo através de uma transmissão de sinal de televisão digital, é uma realidade interessante que ainda revela determinadas vantagens em relação ao ensino presencial praticado na nossa universidade, e é um conceito que não requer um grande investimento monetário por todos os que tencionem aderir. As aulas interactivas transmitidas por um sinal de televisão digital, possibilitam aos alunos assistirem e interagirem com a aula em qualquer local que possua o mínimo de qualidade para a recepção do sinal emitido, além de que o pressuposto da criação dessas mesmas aulas possibilitam a gravação e disponibilização em servidores próprios para o efeito, permitindo o acesso posterior aos conteúdos, para por exemplo a alunos que não lhes foi possível visualizar no horário corrente da aula ou até mesmo apenas para rever conteúdos passados. Uma outra vantagem é que, com base em estudos realizados por grupos de pesquisa, está demonstrado que o compartilhar interactivo de documentos, e o uso de áudio e vídeo em tempo real, faz com que assuntos antes cansativos nas salas de aula tradicionais, se tornem em assuntos muito mais atraentes aos alunos e por consequente um aumento de motivação no processo de aprendizagem.

Como referido anteriormente, numa primeira instância este projecto destina-se a uma experiência de difusão, que apenas circunscreve o campus e proximidades da Universidade de Aveiro, contudo deixa em aberto a possibilidade futura de um sistema de difusão com suporte a um maior alcance, permitindo que os alunos possam assistir às respectivas aulas interactivas a partir de suas residências, evitando assim possíveis transtornos de deslocações até a universidade.

A estrutura seguida neste documento relativo ao trabalho, inicia-se com a apresentação de alguns conceitos e noções teóricos acerca dos sistemas DVB-T, e em seguida temos uma exposição detalhada sobre os protocolos de comunicação utilizados para o transporte dos dados. Logo depois são apresentados alguns dispositivos no mercado actual, que podem ser utilizados para a recepção do sinal no sistema DVB-T. Seguem-se algumas noções básicas acerca de *Microsoft DirectShow* e por fim é apresentado todo trabalho desenvolvido.

Relativamente à aplicação desenvolvida neste projecto, a sua implementação foi realizada usando a ferramenta de desenvolvimento de *software Microsoft Visual Studio* e elaborada segundo programação em *Microsoft Directshow*. Além disso foi tido em conta a possível utilização por utilizadores menos experientes em níveis tecnológicos, pelo que a sua utilização é simples e facilmente perceptível.

1.3 Motivação Digital vs Analógico

A grande vantagem dos sistemas digitais de difusão é a eficiente utilização do espectro de frequências e a redução de radiações de potência quando comparados com os sistemas analógicos de difusão, mantendo as mesmas áreas de cobertura. Outra grande vantagem deste tipo de sistemas é a possibilidade de implementação de uma rede denominada SFN (Single Frequency Network), o que indica que as estações vizinhas umas das outras, usam a mesma frequência mantendo o sinal adjacente sem interferências. O sistema digital transmite fluxos de dados, o que significa que podem ser utilizados para comunicações de sinais de dados (ex: Serviços de Internet), além dos sinais de televisão. Os dados consistem em fluxos de bit em MPEG-2, o que implica a utilização de compressão, permitindo a difusão de 4 ou 5 programas num só canal de televisão com 8MHz de largura de banda.

A principal diferença entre os dois formatos de sinal de televisão, assenta na forma como a respectiva informação é transportada, desde a fonte até ao receptor. O sinal analógico é transmitido em ondas contínuas, enquanto o sinal digital implica uma transmissão discreta de informação. Este simples facto permite que o sistema digital seja mais imune a erros de transmissão, o que para os telespectadores se traduz numa significante melhoria na qualidade de sinal recebido, nomeadamente a nível de nitidez de imagem e possíveis interferências. Para além disso, permite também a inclusão de novas funcionalidades, tais como o formato de imagem "*wide screen*" (16:9), o guia de programa interactivo (IPG – *Interactive Program Guide*) ou até mesmo um sistema de som 5.1.

Do ponto de vista das empresas de transmissão, também existem expressivas vantagens com a utilização do formato digital, visto que é

possível, como já referido, incluir mais canais de televisão num dado espectro de radiofrequências, possibilitando assim uma redução de custos e um aumento da oferta sem comprometer a qualidade. Por exemplo em Portugal, nomeadamente no que se refere aos operadores RTP, SIC e TVI, considera-se que bastaria em princípio a ocupação de um único canal de radiofrequência (correspondente a 8MHz na faixa UHF) para a difusão do sinal digital.

Deve-se ainda salientar que a transmissão digital permite o desenvolvimento de aplicações e serviços extremamente atraentes.

Capítulo 2
Digital Video Broadcasting (DVB)

2.1 DVB

Digital Video Broadcasting (DVB) corresponde a um conjunto de normas internacionalmente aceites para a transmissão de televisão digital, mantido pelo *DVB Project*, que consiste actualmente num consórcio com mais de 280 membros, entre eles difusores, fabricantes, operadores de redes, programadores e entidades reguladoras, que definiram e mantêm actualizados um conjunto de normas técnicas para a transmissão global de televisão digital e serviços de dados.

Figura 2. Logo Oficial DVB.

As normas técnicas são publicadas pelo Comité do Conjunto Técnico (JTC - *Joint Technical Committee*) do Instituto Europeu de Normas para Telecomunicações (ETSI - *European Telecommunications Standards Institute*), Comité Europeu para Normas de Electrotécnica (CENELEC - *European Committee for Electrotechnical Standardization*) e União Europeia de Emissão (EBU - *European Broadcasting Union*).

Um sistema DVB utiliza um protocolo de comunicação denominado por MPEG *Transport Stream* para a transmissão dos seus dados e é normalizado segundo um de vários subsistemas de difusão DVB, cabo (DVB-C), satélite (DVB-S, DVB-S2 e DVB-SH) e terrestre (DVB-T e DVB-T2), definindo assim a camada física e a camada de ligação de dados de um sistema de difusão, os tipos de pacotes a serem transmitidos, a codificação de canal e a modulação para cada um dos subsistemas DVB,

especificando o conjunto de normas para a televisão digital, que incluem a transmissão do sinal e o serviços de dados associado.

Estes subsistemas de distribuição diferem essencialmente no tipo de modulação utilizado e nos respectivos códigos de correcção de erro associados.

O subsistema DVB utilizado para a transmissão do nosso sinal de televisão digital, corresponde ao de televisão digital terrestre (DVB-T), embora o propósito deste projecto possa ser implementado sob outra qualquer tecnologia de rede de banda larga.

2.2 DVB-T

Figura 3. Logo Oficial DVB-T.

Digital Video Broadcasting – Terrestrial (DVB-T) é uma norma do consórcio Europeu para a televisão digital terrestre, publicada em ETSI EN 300 744[1], *Digital Video Broadcasting(DVB); Framing structure, channel coding and modulation for digital terrestrial television, June 2004*. Além disso temos a norma ETSI TS 101 154[2], *Digital Video Broadcasting(DVB); Implementation guidelines for the use of Video and Audio Coding in Broadcasting Applications based on the MPEG-2 Transport Stream, May 2004* que define detalhes sobre os aperfeiçoados métodos de codificação utilizados em DVB-T, como o MPEG-2 e mais recentemente o H.264/MPEG-4 AVC e ainda sistemas de codificação de áudio.

À semelhança do que aconteceu com a televisão analógica em que existiam vários formatos de distribuição (NTSC, PAL e SECAM), a televisão digital terrestre depara-se com o mesmo "problema". Existem actualmente 4 sistemas de difusão de sinal de televisão digital terrestre distintos, o ATSC de origem nos Estados Unidos da América, o ISDB-T de origem Japonesa, o DMB-T de origem Chinesa, e por fim temos o DVB-T Europeu que se encontra numa boa posição para se transformar no sistema global mais utilizado, como se pode verificar na imagem seguinte.

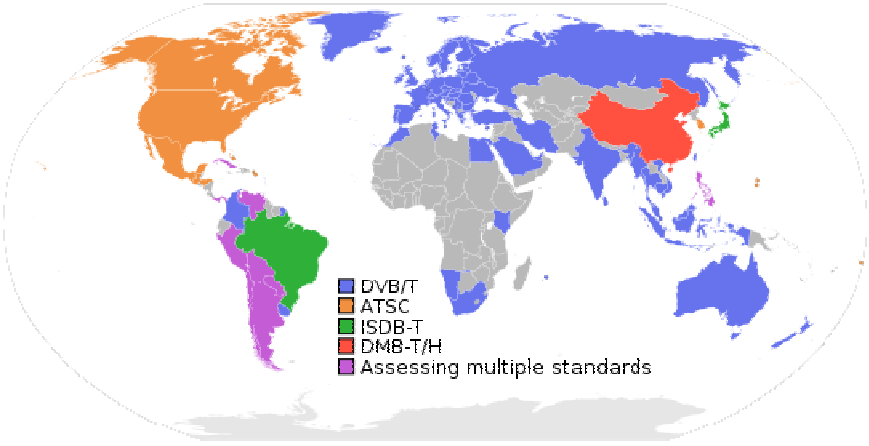


Figura 4. Distribuição global dos sistemas de difusão de televisão digital terrestre.

2.2.1 Modulação

O sistema de difusão DVB-T utiliza modulação OFDM (Orthogonal Frequency Division Multiplexing) com transmissão por múltiplas portadoras. Existem dois modos designados por 2K e 8K, que utilizam respectivamente 1705 e 6817 portadoras, em que cada uma dessas portadoras é modulada separadamente e transmitida no canal de televisão de 8MHz. A modulação utilizada para estas portadoras é tipicamente QPSK (Quadrature Phase-Shift Keying), 16-QAM (Quadrature amplitude modulation) ou 64-QAM. Cada sinal pode ser dividido em duas componentes designadas por "In Phase" (I) e "Quadrature Phase" (Q), desfasadas em 90˚ entre elas. Este tipo de modulação pode ser observado

na figura seguinte que apresenta um diagrama de constelação, onde os dois eixos representam as duas componentes (I e Q). Este caso corresponde a uma modulação 16-QAM, com 16 estados possíveis de alocação, e em que cada símbolo representa 4 bits.

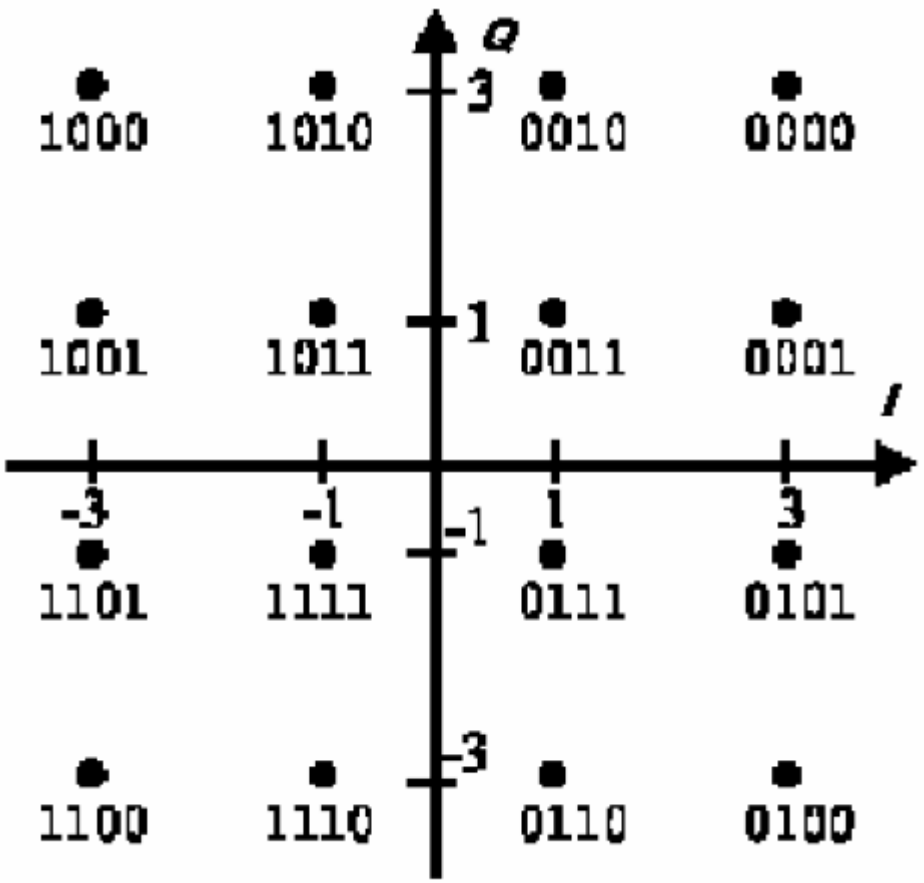


Figura 5. Diagrama de constelação 16-QAM e alocação de bits.

A estrutura das portadoras é ilustrada na figura seguinte. O eixo horizontal representa a frequência e o eixo vertical representa o tempo. O canal de 8MHz consiste no conjunto de portadoras, espaçadas respectivamente de 4462Hz ou 1116Hz de acordo com o modo 2K ou 8K referidos anteriormente. Existe um conjunto de portadoras reservadas, denominadas por portadoras TPS (Transmission Parameter Signalling) que não contêm dados, e que apenas transportam informação do modo de transmissão para o receptor. Assim sendo, o número total de portadoras que

podem ser utilizadas para o transporte de dados são 1512 e 6048 respectivamente para os dois modo de transmissão (2K e 8K), e o débito resultante corresponde a um valor entre 4,97 e 31,66 Mbit/s, dependendo da modulação utilizada (QPSK, 16-QAM ou 64-QAM), do modo de transmissão (2K ou 8K), da taxa de código (CR – Code rate) utilizado para a correcção de erros e o intervalo de guarda (GI – Guard Interval) seleccionado. O intervalo de guarda indica o pequeno espaço de tempo existente entre dois símbolos consecutivos, pelo que a transmissão não é contínua. Este tempo de guarda permite uma recepção adequada, eliminando os erros causados pela propagação multi-caminho.

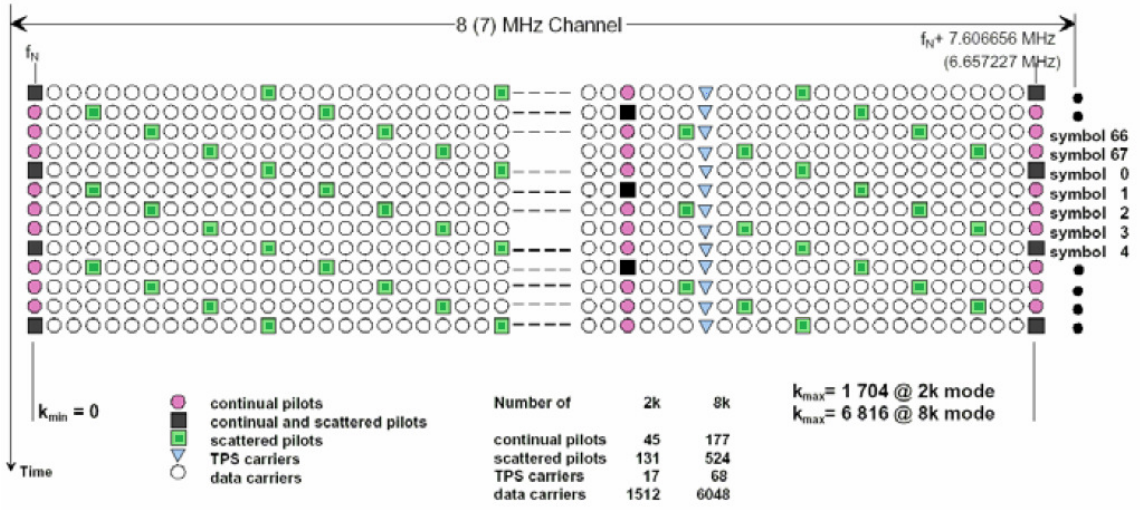


Figura 6. Estrutura das portadoras OFDM.

É ainda de salientar que recentemente, foi normalizada a norma DVB-T2 que permite um ganho de débito na ordem dos 40%, relativamente ao DVB-T.

2.3 MPEG-2

O MPEG-2 é a norma de codificação de vídeo digital (MPEG-2 Vídeo) e áudio associado (MPEG-2 Áudio), utilizada para o sinal de televisão digital difundido no sistema DVB-T. Descreve uma combinação de compressão de vídeo e áudio que permite a transmissão destes dados utilizando uma menor largura de banda, aumentado o desempenho de aplicações em tempo real como uma aplicação para recepção de televisão digital terrestre. Esta norma foi especificada em 1993 pelo MPEG – *Moving Picture Expert Group* e é internacionalmente aceite e publicada, em ISO/IEC 13818-1[3], *Information technology – Generic coding of moving pictures and associated áudio information: Systems,* pela Organização Internacional para Normas (ISO – *International Organization for Standardization*), Comissão Internacional de Electrotécnica (IEC - *International Electrotechnical Commission*). Algumas partes da norma foram ainda desenvolvidas em colaboração com o grupo denominado por União Internacional de Telecomunicações (ITU – *International Telecommunication Union*).

Esta norma descreve o núcleo do formato seguido para os dados difundidos em sistemas de televisão digital terrestre (DVB-T).

Estão definidos dois métodos para o transporte dos dados. Um dos métodos chama-se *Program Stream* e é aplicado em situações em que temos de transportar dados que necessitam de uma fiabilidade minimamente razoável, como por exemplo DVDs. O outro método é o utilizado na transmissão dos dados num sistema DVB-T e é designado por *Transport Stream*, pois permite o transporte dos dados de vídeo digital e áudio, onde é de maior importância o desempenho a nível temporal que a nível de fiabilidade em termos de perda de dados.

2.4 Conceitos de MPEG-2 Sistemas

2.4.1 Pacotes nulos

Em transmissões DVB é necessário manter constante um determinado débito de transmissão (*bitrate*), pelo que de modo a satisfazer essa imposição, é inserindo no fluxo de dados pacotes nulos adicionais. Para tal foi reservado o PID com o valor 0x1FFF que indica ao receptor que deve ignorar o conteúdo desse pacote. O PID é, como detalhado no próximo capítulo, o elemento identificador de cada entidade (fluxo elementar) no fluxo de dados multiplexados.

2.4.2 PES – Packetized Elementary Stream

Packetized Elementary Stream resulta de um processo de empacotamento dos dados pertencentes a um fluxo elementar (ES – *Elementary Stream*). Os pacotes PES resultam simplesmente de o acrescentar ao conteúdo dos pacotes elementares, um cabeçalho que possui informação directamente relacionada ao conteúdo do fluxo elementar, como por exemplo, identifica os diferentes fluxos.

2.4.3 Sinalização de serviços

A sinalização de serviços difundidos segue uma estrutura de sinalização muito particular. Um programa ou serviço, como por exemplo o sinal de televisão digital terrestre emitido, é sinalizado num fluxo de transporte por uma tabela denominada por *Program Map Table* (PMT) que possui um PID específico e que lista todos os PIDs correspondentes aos

fluxos elementares pertencentes a esse programa. Descreve-se em seguida com mais detalhe as tabelas de sinalização.

Program Specific Information **(PSI)** – corresponde ao conjunto de tabelas que especificam ao descodificador toda a informação relevante ao processamento de um determinado programa. Essas são a tabela de associação ao programa (PAT – *Program Association Table*), tabela de mapa do programa (PMT – *Program Map Table*), tabela de acesso condicional (CAT – *Conditional Access Table*) e tabela de informação de rede (NIT – *Network Information Table*). Em MPEG-2 não existe um formato especificado para as duas últimas tabelas indicadas.

Program Association Table **(PAT)** – tabela que lista todos os programas presentes no fluxo de transporte. Todos esses programas listados são identificados por um valor de 16 bits designado por *program_number* com o seu correspondente PID para a PMT.
O valor 0x0000 para *program_number* é reservado para especificar o PID da tabela NIT.
Os pacotes TS que contêm a informação da PAT têm sempre o valor de PID igual a 0x0000.

Program Map Table **(PMT)** – tabela que contém informações acerca de um determinado programa. Existe uma PMT por cada programa e é identificada por um PID específico. Uma PMT indica todos os PIDs dos fluxos elementares que contêm informação relevante a um determinado programa.

Na figura seguinte é ilustrado a relação entre as tabelas constituintes da PSI.

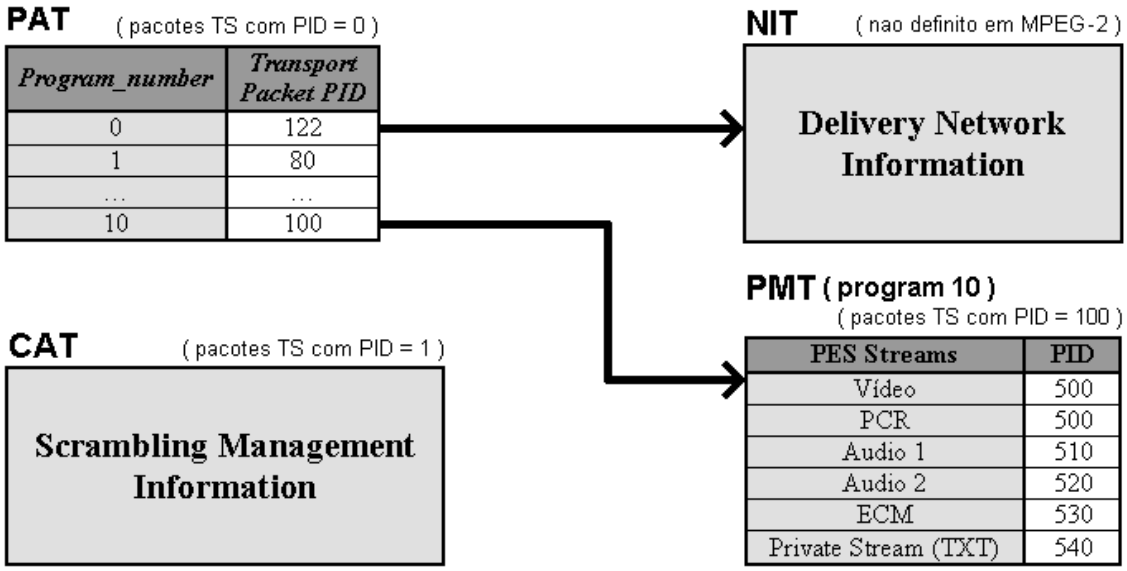


Figura 7. Relação entre as tabelas PSI.

O DVB ainda acrescentou outras tabelas, em seguida apresentadas:

Network Information Table **(NIT)** – providencia informação acerca da rede física.

Bouquet Association Table **(BAT)** – apresenta uma lista de serviços e outras informações a cada conjunto de serviços identificados como apenas uma entidade.

Service Description Table **(SDT)** – descreve os serviços no sistema, como por exemplo o nome do serviço ou até mesmo o fornecedor do serviço.

***Event Information Table* (EIT)** – contém informação relativa aos eventos, como por exemplo o nome do evento, o tempo de início e a duração.

***Running Status Table* (RST)** – indica o estado de um evento, ou seja, indica o estado *running* ou *not running*.

***Stuffing Table* (ST)** – indica a existência de secções que devem ser invalidadas.

***Time and Date Tables* (TDT)** – providencia informação acerca do tempo e data actual.

***Time Offset Table* (TOT)** – providencia informação acerca do deslocamento de tempo local.

Capítulo 3

Transmissão dos dados DVB-T

3.1 Introdução

Os sistemas DVB-T providenciam um meio de entregar *MPEG-2 Transport Stream* (MPEG-2 TS) através de uma variedade de modos de transmissão. Alguns exemplos de transmissão de dados são o *download* de software através de satélite, cabo ou ligações terrestres, entrega de serviços Internet através de canais de transmissão e televisão interactiva.

Existem várias áreas de aplicações com diferentes requisitos para o transporte dos dados, como por exemplo *Data Piping*, *Data Streaming, Multiprotocol Encapsulation* (MPE) que é a utilizada na transmissão dos dados no sistema DVB-T, entre outras.

A transmissão dos dados é o ponto-chave da tecnologia de televisão digital, principalmente se for utilizada a tecnologia IP. Foi essa a motivação do princípio implementado neste trabalho.

Na figura seguinte pode-se observar um diagrama de blocos onde é apresentado o processamento dos dados que compõem o nosso sinal, desde a codificação na fonte, a passagem à camada de sistema e por fim os métodos de encapsulamento que originam o fluxo MPEG-2 TS.

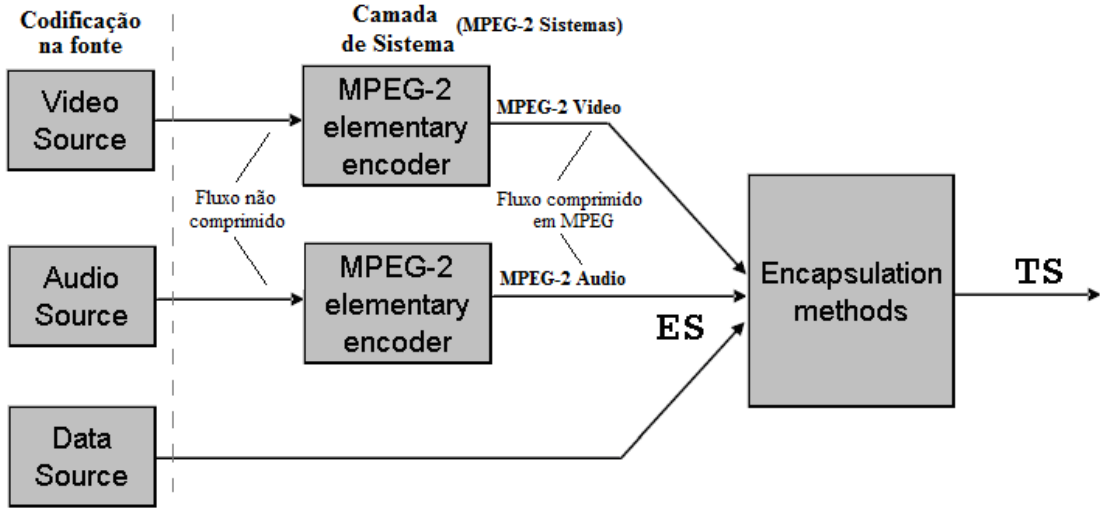


Figura 8. Codificação na fonte e Camada de sistema.

Os fluxos elementares (ES – *Elementary Stream*) de vídeo MPEG-2, áudio MPEG-2 e dados de informação, sincronismo e compressão que compõem a informação de um determinado programa, são encapsulados num protocolo de transporte em tempo real (RTP – *Real-Time Transport Protocol*), tendo em conta que se pretende uma transmissão de dados em televisão digital terrestre, resultando num conjunto de fluxos elementares empacotados (PES – *Packetized Elementary Stream*) correspondentes ao vídeo, áudio e dados de informação. Contudo um pacote RTP não pode só por si ser transmitido numa rede, pelo que este protocolo é por sua vez encapsulado num outro de transferência denominado por *User Datagram Protocol* (UDP). Desta feita, de modo a permitir a transferência destes pacotes em redes IP, o pacote UDP é encapsulado no protocolo de Internet (IP – *Internet Protocol*). Neste ponto, é então utilizado o protocolo MPE e o MPEG-2 TS como já referido, permitindo assim a transmissão dos dados no sistema DVB-T.

Na figura 7 é ilustrado a relação entre os vários tipos de pacotes correspondentes aos protocolos de comunicação utilizados, e que serão apresentados com algum detalhe posteriormente. Estes compõem assim o fluxo de transporte (MPEG-2 TS) transmitido no sistema DVB-T.

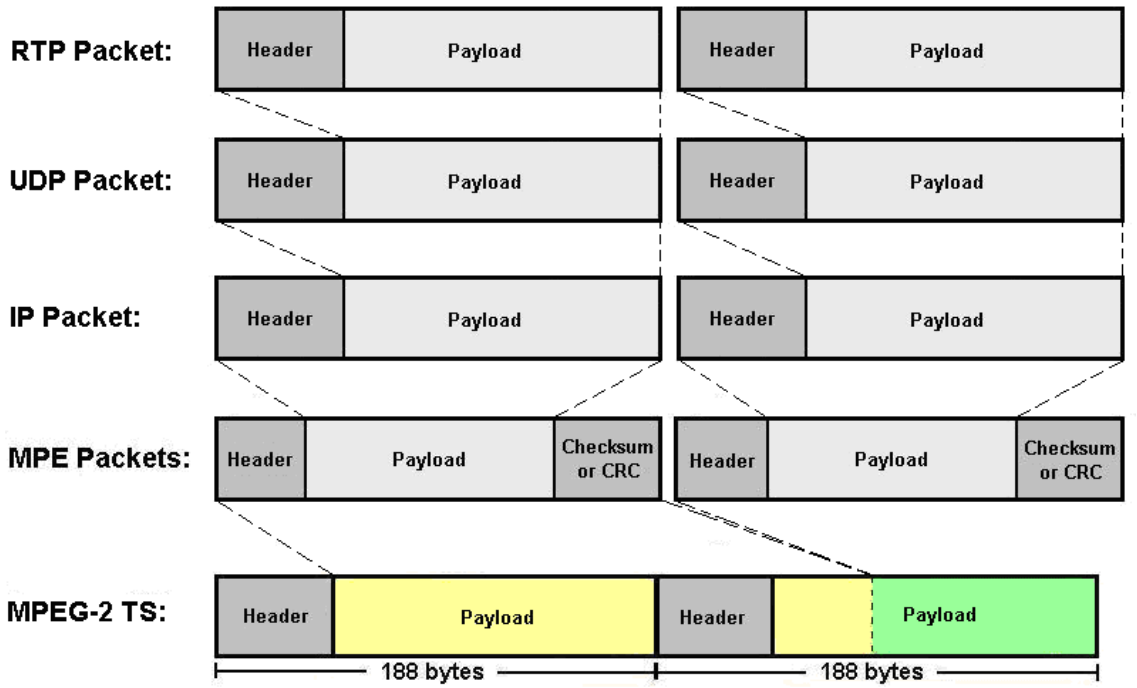


Figura 9. Relação entre os vários protocolos que compõem o MPEG-2 TS.

3.2 *Real-Time Transport Protocol* (RTP)

O RTP é um protocolo de comunicação desenvolvido pelo *Audio-Video Transport Working Group* da organização *Internet Engineering Task Force* (IETF) e foi publicado pela primeira vez em 1996 como RFC 1889[7], *RTP: A Transport Protocol for Real-Time Applications*. Desde então foram feitas actualizações e foi novamente publicado em 2003 como RFC 3550[8], *RTP: A Transport Protocol for Real-Time Applications*.

Este protocolo destina-se a comunicações onde a transferência de dados surge em tempo real, permite a definição de tempos de amostragem, ou seja, facilita a sincronização de, por exemplo fluxos elementares, como é o caso do áudio e vídeo associados para televisão digital. Define como deve ser feita a fragmentação dos fluxos de dados, adicionando a cada fragmento informação de sequência e de tempo de entrega.

Este protocolo, só por si, não garante o envio de dados em tempo real, mas fornece mecanismos para recepção e envio em aplicações que suportem a transferência de dados. Normalmente este protocolo é, como já referido anteriormente, encapsulado no protocolo UDP que será apresentado em seguida. As aplicações que utilizem o protocolo RTP, são pouco sensíveis as perdas de pacotes, contudo dão muita importância ao atraso na entrega de pacotes, pelo que a utilização do protocolo UDP torna-se numa escolha mais adequada quando comparada com outra possibilidade, como o protocolo TCP (*Transmission Control Protocol*) que requer mais tempo de processamento sobre os pacotes transmitidos.

O protocolo RTP e respectivo cabeçalho estão organizados da seguinte maneira:

Figura 10. Estrutura do protocolo RTP.

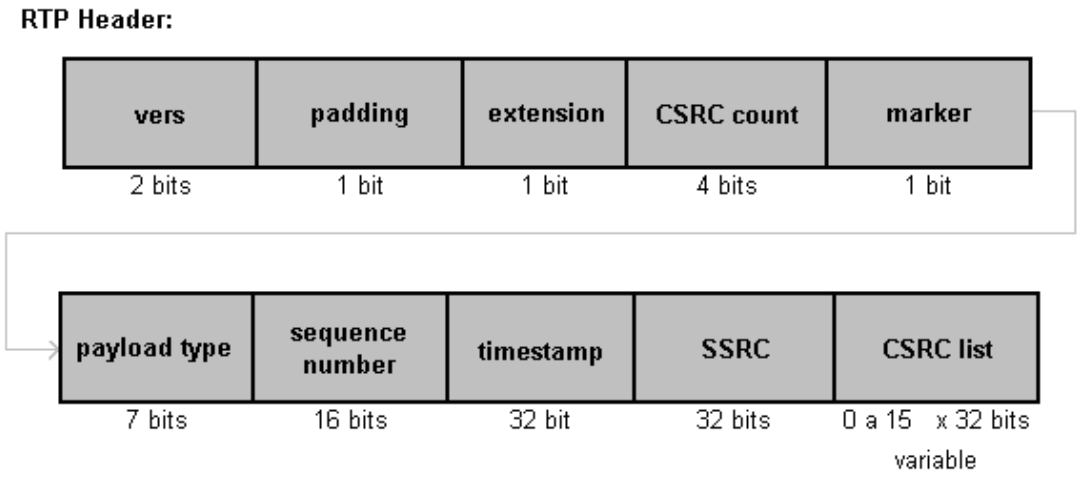


Figura 11. Estrutura do cabeçalho RTP.

- ***Header* (variável):** cabeçalho do pacote RTP.

  - *vers* (2 bits): identifica a versão do protocolo RTP. Actualmente este campo possui o valor 2;

  - *padding* (1 bit): quando activo a "1" indica a utilização de bytes extra de enchimento no final do pacote, que não fazem parte dos dados. O último octeto (8 bits) indica o número de octetos que devem ser ignorados da zona de dados. Esta ferramenta pode ser útil em casos que se pretenda utilizar algum algoritmo de encriptação que necessite de pacotes de tamanho fixo;

  - *extension* (1 bit): quando activo a "1" designa que existe uma extensão ao cabeçalho do pacote e só depois a zona de dados;

  - *CSRC count* (4 bits): indica o número de identificadores de fontes extra, ou seja, fontes adicionais que contribuíram para o conteúdo dos dados no pacote;

- *marker* (1 bit): utilizado a nível da aplicação e definido segundo um determinado perfil. Quando este bit toma o valor "1" significa que os dados possuem especial relevância para a aplicação, e no caso contrário toma então o valor "0";

- *payload type* (7 bits): indica o formato dos dados, determinando assim a interpretação que a aplicação deve seguir sobre esses dados;

- *sequence number* (16 bits): valor de incremento por cada pacote RTP enviado e pode ser utilizado pelo receptor para a detecção de pacotes perdidos e reformar a sequência correcta dos pacotes. O valor inicial deste campo é aleatório;

- *timestamp* (32 bits): indica o instante de tempo da amostragem do primeiro octeto da zona de dados do pacote. O instante de amostragem deve derivar de um relógio de incremento monotónico e linear no tempo, permitindo assim os cálculos de sincronização e *jitter* (variação estatística do retardo na entrega de dados em uma rede). A resolução do relógio deve ser pelo menos suficiente para a precisão pretendida e para a medição *jitter* na entrega dos pacotes. Temos ainda que a frequência de relógio depende do formato dos dados a serem transportados, sendo especificada através de um perfil ou na especificação pelo campo que define o formato dos dados, como já mencionado.

- *SSRC* (32 bits): identifica a fonte de sincronização do fluxo de dados. Este valor é escolhido ao acaso com a intenção de não haver fontes com o mesmo valor para este campo;

- *CSRC list* (0 a 15 x 32 bits): lista que identifica as diferentes fontes que contribuíram para o conteúdo dos dados no pacote. O número de identificadores é então indicado em *CSRC count* e apenas podem ser indicados 15, mesmo que existam mais a contribuir para a zona de dados.

- ***Payload* (variável):** zona de dados no pacote RTP.

Após a apresentação do protocolo RTP, passa-se em seguida a apresentar o segundo protocolo (UDP) de encapsulamento dos dados.

3.3 *User Datagram Protocol* (UDP)

O UDP é um protocolo de comunicação projectado por David P. Reed em 1980 e formalmente publicado como norma em RFC 768[9], *User Data Protocol, August 1980*.

Este protocolo apenas fornece um serviço mínimo de transporte de dados, oferecendo um conjunto limitado de serviços para a troca de mensagens entre computadores numa rede que utilize protocolo de internet (IP - *Internet Protocol*). Este protocolo é então em seguida encapsulado no protocolo IP, podendo deste modo ser transmitido em redes IP.

O protocolo UDP não permite identificar a sequência pela qual os pacotes são recebidos, o que significa que uma aplicação que utilize este protocolo deve confirmar que recebeu toda a informação da mensagem e que esta se encontra pela ordem correcta. Além disso não é confiável, do ponto de vista em que não há qualquer tipo de garantia que os pacotes chegam ao receptor, caso sejam requeridas garantias é necessário implementar uma série de estruturas de controlo, tais como *timeouts*, retransmissões, *acknowlegments*, controle de fluxo, etc. Contudo essa garantia já é suportada pelo protocolo do nível superior, já apresentado anteriormente (protocolo RTP).

Além disso o protocolo UDP não suporta a divisão de uma mensagem, no nosso caso um pacote RTP, por vários pacotes UDP.

O UDP é muito utilizado em aplicações em que se pretende obter um fluxo de dados em tempo real, como no nosso caso em que temos uma aplicação para recepção de televisão digital, visto que este protocolo é muito simples não necessitando assim de muito tempo para processamento

dos pacotes que são compostos praticamente só por dados. Também é muito utilizado quando o fluxo admite perda ou o corromper de parte de seu conteúdo, tal como os fluxos de vídeo digital e áudio associados que compõem o sinal.

Este protocolo permite a utilização de dois serviços, importantes para a implementação da aplicação de recepção de televisão digital, não possibilitados no próximo protocolo de encapsulamento (protocolo IP, em seguida apresentado), que são a identificação das portas utilizadas nos dispositivos, permitindo assim a diferenciação dos pedidos de diferentes utilizadores. Além disso, possui um campo de verificação (*checksum*) que possibilita verificar se os dados chegaram intactos. É um protocolo que deverá necessariamente ser utilizado para o suporte de *broadcasting* e *multicasting*.

Na figura 10 pode-se observar o formato do protocolo UDP.

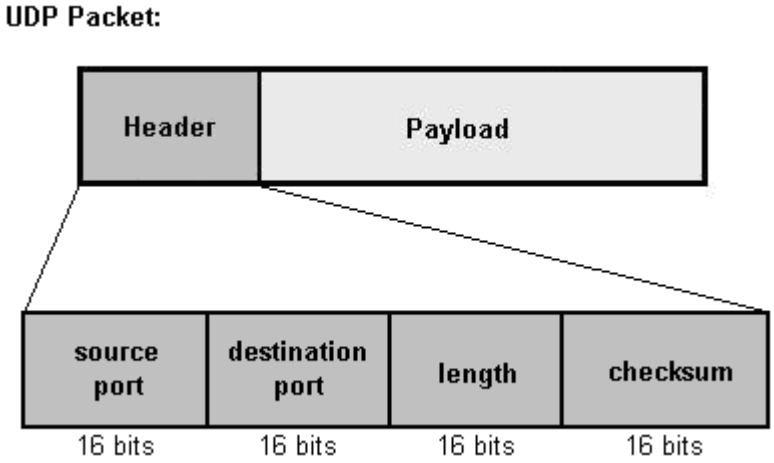


Figura 12. Estrutura do protocolo UDP.

- ***Header* (64 bits):** cabeçalho do pacote UDP.

  - *source port* (16 bits): identifica a porta de comunicação utilizada pelo emissor. A porta de origem específica geralmente a porta desejada de resposta, mas em comunicações *broadcast* este campo pode ser omitido, tomando o valor zero;

  - *destination port* (16 bits): identifica a porta de comunicação utilizada no receptor;

  - *length* (16 bits): indica o comprimento em bytes do cabeçalho UDP mais o dos dados encapsulados, logo o vamos mínimo é 8. Teoricamente o valor máximo seria de 65.535 bytes, contudo na prática estamos a utilizar valores muito mais reduzidos para o comprimento do pacote;

  - *checksum* (16 bits): utilizado para detectar a existência de erros no cabeçalho e nos dados do pacote UDP. Se este valor toma o valor 0xFFFF significa que esta ferramenta não é utilizada neste protocolo.

- ***Payload* (variável):** zona de dados no pacote UDP.

3.4 Internet Protocol (IP)

O protocolo IP foi desenvolvido pelo *Internet Engineering Task Force* (IETF) e a norma correspondente foi publicada pela primeira vez como RFC 760[10], *DoD Standard Internet Protocol, January 1980.* Mais tarde, foi publicada uma nova versão denominada por RFC 791[11], *Internet Protocol, Darpa Internet Program, Sptember 1981.*

O IP é um protocolo de comunicação entre duas ou mais máquinas de rede para encaminhamento dos dados. O pacote resultante deste protocolo é geralmente designado por pacote IP ou IP *Datagram.* No nosso caso, é utilizada a versão 4 do protocolo IP, pelo que os pacotes também podem ainda ser denominados por pacote IPv4 ou IPv4 *Datagram,* e como os outros protocolos de rede possui uma estrutura característica, apresentada mais à frente.

Este protocolo oferece um serviço não confiável (também chamado por melhor esforço), ou seja, não existem quaisquer garantias, do ponto de vista que podem ocorrer pacotes desordenados, duplicados de outros pacotes, e até mesmo podem ser perdidos por completo. Estes aspectos devem ser tomados em conta nos protocolos de nível superior, no nosso caso os protocolos UDP e RTP.

O pacote IPv4 é conceptualmente dividido em duas partes. Temos primeiro um cabeçalho IP (*Header*) onde encontramos especificações de endereço e de controlo. Numa última secção do pacote temos a zona de dados (*Payload*) que transporta a informação a ser transmitida na rede, no nosso caso estes dados correspondem a um pacote de protocolo UDP, como já referido.

Em seguida é então apresentada a estrutura do pacote IPv4 e respectivo cabeçalho:

Figura 13. Estrutura do protocolo IP.

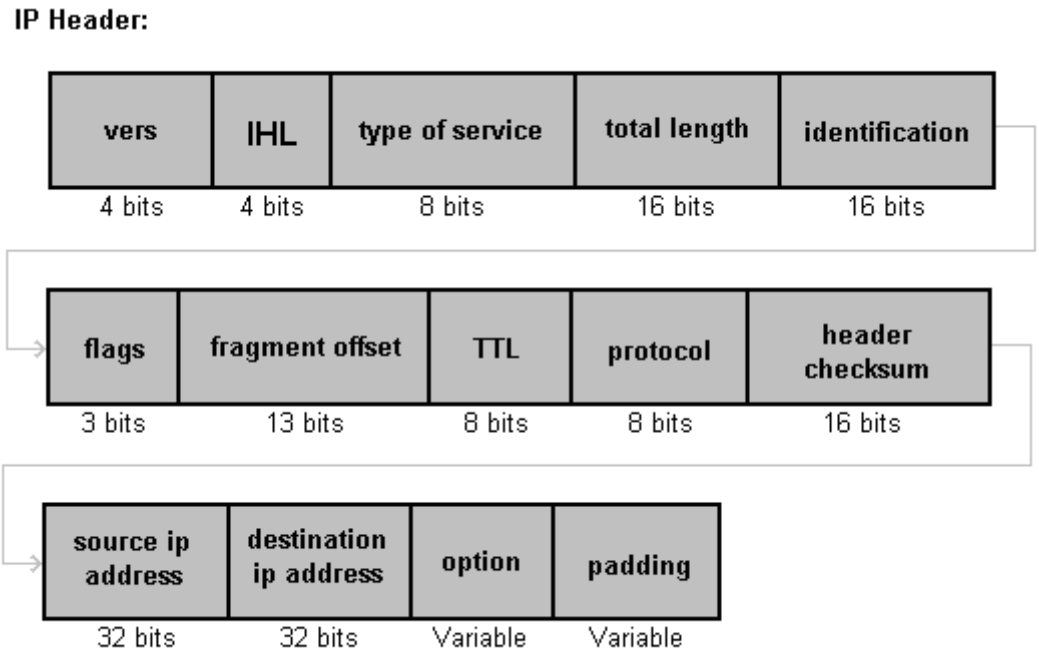


Figura 14. Estrutura do cabeçalho IP.

- ***Header* (variável):** cabeçalho do pacote IPv4.

  - *vers* (4 bits): identifica a versão do protocolo IP utilizada para gerar o pacote. Visto que na nossa situação, como já referido, é

utilizado o pacote IPv4, este campo tomará o valor 4. O propósito desta informação é garantir compatibilidade entre as maquinas na rede que podem eventualmente estar a correr versões diferentes de protocolo IP. Em geral, maquinas que correm uma versão mais antiga de protocolo IP rejeita os pacotes IP gerados por outras versões mais recentes, assumindo que a versão mais antiga pode não ser capaz de interpretar correctamente o pacote de versão mais nova;

- *IHL* (4 bits): *Internet Header Length* especifica o tamanho do cabeçalho IPv4, incluindo os campos de *option* e *padding*, em palavras de 32 bits, ou seja, em bytes é necessário multiplicar o valor contido neste campo por 4;

- *type of service* (8 bits): contém informação que permite uma gestão de qualidade de recursos, como por exemplo definir prioridades na entrega de pacotes IPv4. Contudo, esta ferramenta nunca foi amplamente utilizada como definida de origem, pelo que o seu significado tem sido sequencialmente redefinido para a utilização de uma técnica denominada por *Differentiated Services* (DS). Este campo é utilizado com alguma importância, em aplicações que necessitam do envio de pacotes em tempo real. Este campo é subdividido pelos significados apresentados na tabela seguinte:

***Precedence* (bits 0-2)**	**Descrição**
111	Network Control
110	Internetwork Control
101	CRITIC/ECP
100	Flash Override
011	Flash
010	Immediate
001	Priority
000	Routine
***Delay* (bit 3)**	**Descrição**
0	Normal Delay
1	Low Delay
***Throughput* (bit 4)**	**Descrição**
0	Normal Throughput
1	Low Throughput
***Reliability* (bit 5)**	**Descrição**
0	Normal Reability
1	Low Reability
***Reserved* (bits 6-7)**	**Descrição**
-	Reservado para utilização futura

Tabela 1. Significados de *type of service*.

- *total length* (16 bits): especifica o tamanho total em bytes do pacote IPv4. O tamanho máximo permitido é então de 65.535 bytes, contudo os pacotes IPv4 utilizados são de tamanho muito menor, como já referido no caso do protocolo de nível superior (protocolo RTP);

- *identification* (16 bits): este campo foi inicialmente utilizado para identificar os respectivos fragmentos de um pacote IPv4. Alguns estudos experimentais sugerem que este campo pode ser utilizado também para outros propósitos, como por exemplo adicionar pacotes com informação de rastreio, de forma a ajudar a pesquisar pacotes anteriores com endereços fonte falsificados;

- *flags* (3 bits): 3 bits utilizados para controlar e identificar fragmentos, sendo estes bits por ordem do de maior grau para o menor, apresentados na tabela seguinte:

Bit	Descrição
Reserved (1 bit)	Deve tomar sempre o valor zero.
DF (1 bit)	*Don't fragment* quando activo a "1", indica que o pacote não deve ser fragmentado. Utilizado por exemplo em situações em que o dispositivo de destino não possui recursos suficientes para suportar a desfragmentação. No caso do nosso sinal DVB este valor é sempre zero.
MF (1 bit)	*More Fragments* quando colocado a "0", significa que este pacote é o ultimo fragmento de uma mensagem, e quando colocado a "1" indica que ainda existem mais fragmentos a receber, ou seja, este valor será sempre "1" e apenas "0" no último pacote de uma mensagem fragmentada. Na situação em que não ocorre fragmentação, análoga à do nosso sinal DVB, este valor é sempre "0".

Tabela 2. Significados de *flags*.

- *fragment offset* (13 bits): quando existe fragmentação de uma mensagem é aqui especificado em unidades de 8 bytes (64 bits) o offset ou posição relativo a toda a mensagem do fragmento que segue neste pacote IPv4. Um primeiro fragmento tem então o valor zero;

- *TTL* (8 bits): *Time To Live* significa tempo de vida dos pacotes IPv4 e ajuda a prevenir que estes persistam na rede. As máquinas de rede vão sequencialmente reduzindo este valor por cada vez que recolocam o pacote na rede, até que este tome o valor zero e seja descartado;

- *protocol* (8 bits): neste campo é especificado o protocolo utilizado na porção de dados do pacote IPv4. Por exemplo, a utilização do protocolo UDP para a porção de dados, é identificado com o valor decimal 17;

- *header checksum* (16 bits): é o campo de verificação para o cabeçalho do pacote IPv4. É uma forma de detectar a consistência do cabeçalho e este valor é ajustado e verificado ao longo do caminho que percorre. Apenas envolve verificação do cabeçalho e não dos dados transportados pelo pacote;

- *source ip address* (32 bits): endereço IP de 32 bits do dispositivo de origem do pacote. Os dispositivos intermédios no caminho do pacote não alteram este valor permitindo assim sempre a identificação do dispositivo gerador do pacote;

- *destination ip address* (32 bits): endereço IP de 32 bits do dispositivo destino do pacote. Novamente, este é o valor do último dispositivo a que se destina o pacote, independentemente se o caminho engloba outros dispositivos intermediários;

- *option* (variável): campos de cabeçalho adicionais, normalmente não utilizados. Utilizados para vários propósitos, como por exemplo:

option	Descrição
Security	Especifica o nível de segurança do pacote, utilizado por exemplo em aplicações militares.
Record route	Indica que cada dispositivo deve anexar o seu endereço.

Tabela 3. Exemplos de campos de *option*.

- *padding* (variável): utilizado para o caso em que sejam inseridos qualquer tipo de campos adicionais de cabeçalho (*option*) e que o numero de bits utilizados não seja múltiplo de 32. Nessa situação são inseridos bits com o valor "0" de modo a colocar o cabeçalho com um tamanho múltiplo de 32 bits (4 bytes).

- ***Payload* (variável):** zona de dados do pacote IPv4 que contém o pacote de protocolo UDP.

3.5 *Multiprotocol Encapsulation* (MPE)

O MPE é um protocolo de transmissão de dados definido para DVB pelo *European Telecommunications Standards Institute* (ETSI) e publicado em ETSI EN 301192[14], *Digital Video Braodcasting(DVB); Specification for Data Broadcasting, April 2008,* em conformidade com ISO/IEC 13818-6[6], *Information technology – Generic coding of moving pictures and associated áudio information: Extensions for DSM-CC, September 1998*.

A transmissão dos dados é feita através de um encapsulando dos pacotes resultantes de outros protocolos de comunicação de nível superior e o método foi optimizado para o transporte do protocolo de Internet (IP - *Internet Protocol*), e corre sobre o MPEG-2 *Transport Stream*, em redes de televisão digital. Além disso permite também o transporte de qualquer tipo de outro protocolo de comunicação desde que siga um encapsulamento LLC/SNAP.

Este protocolo também suporta *unicast* (datagramas enviados a um único receptor), *multicast* (datagramas enviados a um grupo de receptores) e *broadcast* (datagramas enviados a todos os receptores).

No nosso caso de transmissão o protocolo MPE é responsável por encapsular cada pacote IP, adicionando um cabeçalho MPE e um checksum de verificação de erro de bit, resultando num novo pacote denominado por PDU (*Protocol Data Unit*) ou simplesmente por pacote MPE. Depois disso, o pacote resultante é então fragmentado resultando numa na série final de pacotes MPEG2 TS.

Visto a necessidade natural das transmissões em redes DVB, o nível de segurança e privacidade sobre os dados ser um factor muito importante, o processo de encapsulamento é fundamental visto permitir encriptação dos dados e alteração dinâmica dos endereços MAC.

Nas figuras seguintes são apresentados a estrutura do protocolo MPE e em detalhe o respectivo cabeçalho.

Figura 15. Estrutura do protocolo MPE.

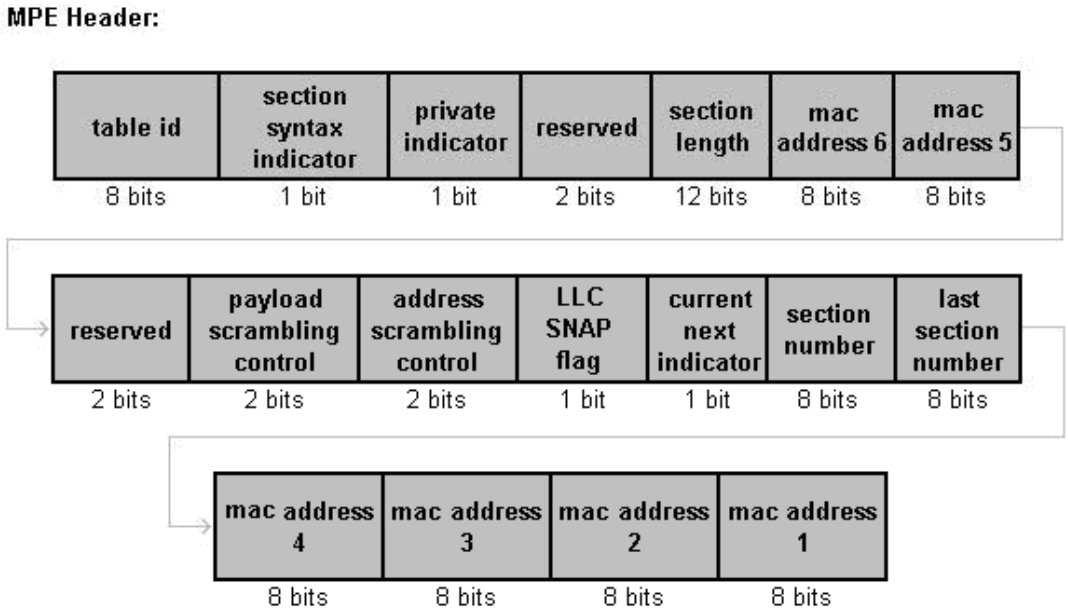


Figura 16. Estrutura do cabeçalho MPE.

- ***Header* (96 bits):** cabeçalho do pacote MPE.

  - *table id* (8 bits): campo que identifica de inicio de pacote MPE de valor fixo 0x3E;

- *section syntax indicator* (1 bit): quando activo a "1", indica a presença do campo CRC. Quando este campo toma o valor "0", indica a presença do campo *checksum*.

- *private indicator* (1 bit): indica o transporte de dados privativos;

- *reserved* (2 bits): reservado para utilização futura. De momento estes 2 bits são colocados a "1";

- *section length* (12 bits): indica o número de bytes logo em seguida a este campo, que ainda pertencem a este mesmo pacote MPE. Assim sendo o tamanho total do pacote pode ser encontrado, somando o valor 3 (bytes de cabeçalho anteriores a este campo) ao valor contido em *section length*;

- *mac address* [1..6] (48 bits): este é um campo que contém o endereço MAC do destinatário. O endereço MAC é fragmentado em 6 campos denominados por *mac address* 1 até *mac address* 6, em que o primeiro contém o byte mais significativo do endereço MAC e o ultimo o byte menos significativo. Na figura seguinte é ilustrado o mapeamento do endereço MAC na secção de cabeçalho do pacote MPE:

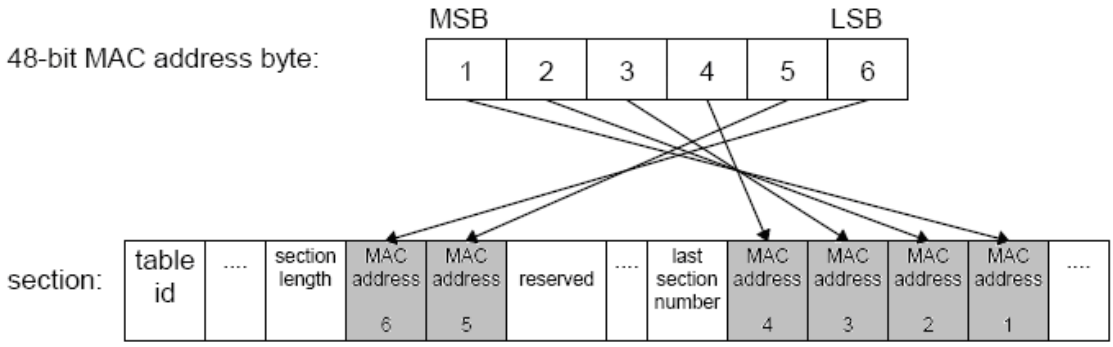


Figura 17. Mapeamento do endereço MAC.

A ordem dos bits nos respectivos bytes não é invertida, pelo que o bit mais significativo no byte é transmitido em primeiro lugar. Os campos que compõem o endereço MAC podem simplesmente transportar a informação ordenada do endereço, ou podem conter informação desordenada de acordo com o controlo definido para o desordenamento destes campos;

- *reserved* (2 bits): reservado para utilização futura. De momento estes 2 bits são colocados a "1";

- *payload scrambling control* (2 bits): define o modo de desordenamento da zona de dados do pacote MPE, ou seja entre o último campo do cabeçalho (*mac address* 1) e o checksum do pacote. O método de desordenamento é privado do utilizador. Na tabela seguinte são apresentados os respectivos significados para os casos possíveis neste campo.

Valor	Descrição
00	Zona de dados não desordenada.
01	Definido pelo serviço.
10	Definido pelo serviço.
11	Definido pelo serviço.

Tabela 4. Descrição das possibilidades em *payload scrambling control*.

- *address scrambling control* (2 bits): define o modo de desordenamento do endereço MAC em causa. Este campo permite uma alteração dinâmica do endereço MAC e o método de desordenamento é privado do utilizador. Na tabela seguinte são apresentados os respectivos significados para os casos possíveis neste campo;

Valor	Descrição
00	Endereço MAC não desordenado.
01	Definido pelo serviço.
10	Definido pelo serviço.
11	Definido pelo serviço.

Tabela 5. Descrição das possibilidades em *address scrambling control*.

- *LLC SNAP flag* (1 bit): indica como deve ser interpretada a zona de dados do pacote;

Valor	Descrição
0	A zona de dados corresponde a um pacote IP sem encapsulamento LLC/SNAP.
1	A zona de dados corresponde a um pacote encapsulado segundo LLC/SNAP, e o tipo de dados encaminhados é definido pela estrutura desse encapsulamento.

Tabela 6. Significados possíveis de *LLC SNAP flag*.

- *current next indicator* (1 bit): indica se em seguida seguem mais pacotes MPE, logo este bit deve tomar sempre o valor "1";

- *section number* (8 bits): utilizado em situações de por exemplo fragmentação de um determinado segmento de dados por vários pacotes, este campo deve indicar a posição deste fragmento em relação a todo o conjunto de fragmentos que compõem a segmento de dados. Em situações que não existe fragmentação, este valor deve ser zero;

- *last section number* (8 bits): utilizado em situações de por exemplo fragmentação de um determinado segmento de dados por vários pacotes, este campo deve indicar o número do último pacote MPE com informação fragmentada pertencente a esse segmento de dados. Em situações que não existe fragmentação, este valor deve ser zero;

- ***Payload* (variável):** zona de dados do pacote MPE que contém o pacote de protocolo IP. Estes dados podem estar desordenados segundo o definido no controlo de desordenamento deste campo.

- ***Checksum* (32 bits):** campo de verificação que indica se o pacote MPE está intacto e este valor é calculado sobre todo o pacote MPE.

Os pacotes MPE são em seguida encapsulados em pacote TS, protocolo explicado em detalhe na secção seguinte.

3.6 MPEG-2 *Transport Stream* (TS)

O MPEG-2 TS é um protocolo de comunicação desenvolvido pelo *Moving Picture Expert Group* (MPEG) e publicado pelo *International Organization for Standardization* (ISO) e *International Electrotechnical Commission* (IEC) na primeira parte da norma ISO/IEC 13818-1[3], *Information technology — Generic coding of moving pictures and associated áudio information: Systems, December 2000.* Algumas partes desta norma ainda foram desenvolvidas em colaboração com o sector de normas do grupo *International Telecommunication Union* (ITU).

MPEG-2 TS permite a multiplexagem e compressão de fluxos elementares (ES – *Elementary Streams*) de áudio, vídeo e dados com informação para a sincronização entre esses mesmos fluxos, produzindo assim um programa ou serviço, tal como o de sinal de televisão digital terrestre. No caso da transmissão de MPEG-2 TS em DVB os fluxos elementares foram devidamente encapsulados (PES - *Packetized Elementary Stream*) até ao protocolo MPE, como já referido anteriormente.

Toda a informação de um determinado programa é transmitida numa única frequência, e os pacotes de transmissão são de tamanho fixo e reduzido.

Cada pacote pertencente ao fluxo de transporte (TS – *Transport Stream*) tem um tamanho fixo de 188 bytes, e a sua estrutura inicia-se sempre com 4 bytes que compõem o cabeçalho (TS *header*) seguidos por 184 bytes para um campo de adaptação com bytes de enchimento (*adaptation field*) como extensão do cabeçalho e/ou zona de dados (*payload*).

Na figura seguinte apresenta-se a estrutura dos pacotes TS e do seu cabeçalho, com uma identificação um pouco detalhada acerca dos vários campos que os compõem.

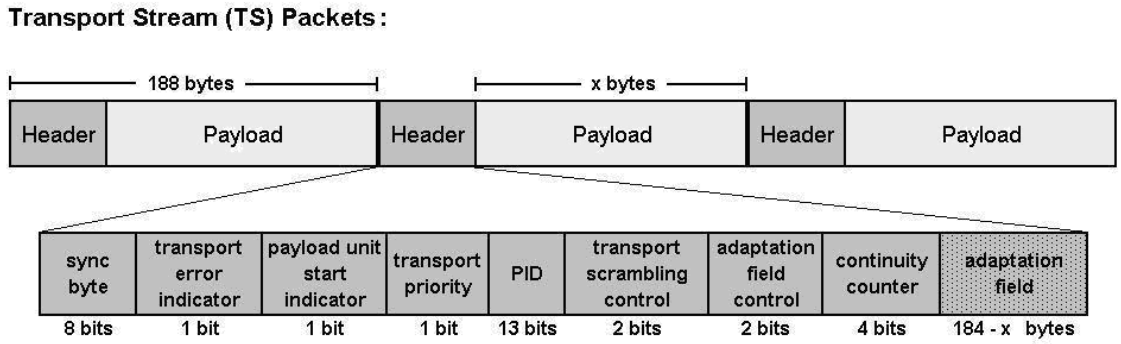


Figura 18. Estrutura dos pacotes TS.

- ***Header* (variável):** cabeçalho do pacote TS.

  - *sync* byte (8 bits): byte inicial no cabeçalho com o valor fixo 0x47 e que identifica o início de um pacote TS;

  - *transport error indicator* (1 bit): quando activo a "1", identifica um erro de bit no pacote;

  - *payload unit start indicator* (1 bit): quando activo indica o início de uma nova unidade neste pacote TS, como por exemplo um novo pacote MPE ou uma secção PSI (*Program Specific Information*);

  - *transport priority* (1 bit): este valor activo declara que este pacote TS tem maior prioridade que os outros que têm o mesmo PID (*Paket IDentifier*) mas que possuem este valor a "0";

- *PID* (13 bits): é um dos campos do pacote TS com maior importância, visto indicar que todos os pacotes que possuem o mesmo valor em PID, pertencem ao mesmo fluxo elementar ou a uma tabela. Existem alguns valores de PID que têm um determinado significado, pelo que em seguida se apresenta uma tabela com o significado na gama de PIDs permitida;

Valor	Descrição
0x0000	Tabela de associação a um programa. (PAT – *Program Association Table*)
0x0001	Tabela de acesso condicionado. (CAT – *Condicional Acess Table*)
0x0002 … 0x000F	Reservado
0x0010 … 0x1FFE	Disponível para PES, tabelas de programas, tabelas de redes, etc.
0x1FFF	Pacote nulo.

Tabela 7. Valores para *Packet IDentifier*.

- *transport scrambling control* (2 bits): este campo indica o modo de desordenamento no pacote TS.

Valor	Descrição
00	Não existe desordenamento no pacote TS.
01	Reservado para futuro, usado em DVB
10	Pacote TS desordenado por uma chave par.
11	Pacote TS desordenado por uma chave ímpar.

Tabela 8. Valores para *transport scrambling control*.

- *adaptation field control* (2 bits): indica se o cabeçalho do pacote TS é seguido por um campo de adaptação e/ou por uma zona de dados. Em seguida é apresentado os valores e correspondente descrição.

Valor	Descrição
00	Reservado para futuro, usado por ISO/IEC
01	Apenas zona de dados
10	Apenas campo de adaptação
11	Campo de adaptação seguido por zona de dados

Tabela 9. Valores para *adaptation field control*.

- *continuity counter* (4 bits): valor de incremento em cada pacote TS com o mesmo PID. Este valor é retornado a zero após chegar ao valor máximo e no caso em que temos adaptation field control com o valor 00 ou 10, o continuity counter não deve ser alterado. Este campo é uma ferramenta importante para a determinação de pacotes perdidos.

- *adaptation field* (variável): esta é a zona de bytes de enchimento seguida por bytes de dados (payload), permitindo assim uma adaptação ao tamanho do pacote de 188 bytes.

- ***Payload* (variável):** secção de dados com 184 bytes menos o tamanho do *adaptation field*. Estes dados correspondem a secções ou ate mesmo pacotes MPE completos.

Em seguida é apresentada a estrutura relativa ao campo de adaptação utilizado neste protocolo.

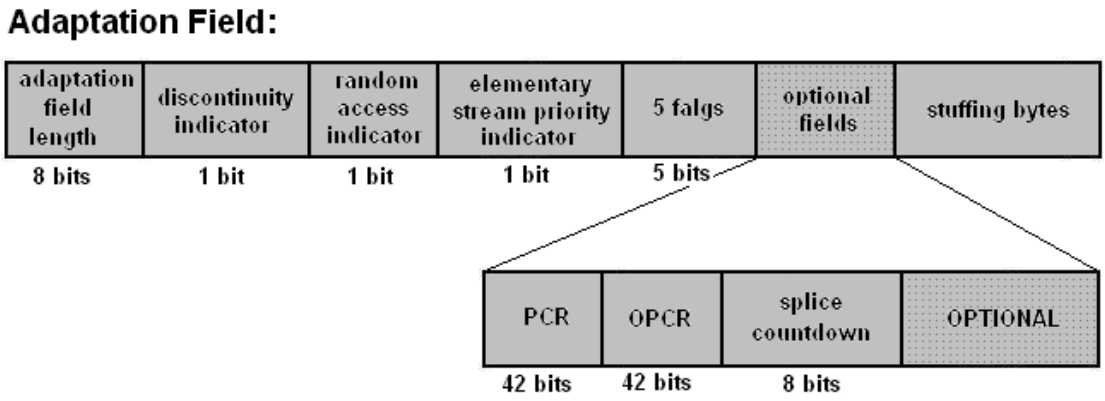


Figura 19. Estrutura do *Adaptation Field*.

- *adaptation field length* (8 bits): especifica o número de bytes que se seguem, logo após este campo, que completam o *adaptation field*;

- *discontinuity indicator* (1 bit): indica uma descontinuidade no *continuity counter* apresentado no cabeçalho do pacote TS, quando activo a "1";

- *random access indicator* (1 bit): toma o valor "1", de modo a especificar o início de uma sequência de vídeo ou de áudio na zona de dados do pacote TS;

- *elementary stream priority indicator* (1 bit): utilizado para especificar maior prioridade deste pacote, quando activo a "1";

- *5 flags* (5 bits): indicam a existência ou não dos campos adicionais que se podem encontrar no campo de adaptação, sendo estas indicações:

- *PCR flag* (1 bit): "1" indica que o campo de adaptação contém *PCR*;

- *OPCR flag* (1 bit): "1" indica a existência de *OPCR*;

- *Splicing point flag* (1 bit): indica que existe *Splice Countdown* no campo de adaptação, quando activo a "1";

- *Transport private data flag* (1 bit): "1" indica a existência de bytes com informação privada neste campo de adaptação;

- *Adaptation field extension flag* (1 bit): indica a existência ou não de uma extensão no campo de adaptação de modo a obtermos um valor de bits múltiplo de 8;

- *optional fields* (variável): como indicado anteriormente, estes campos que se seguem são apenas opcionais e dependem das indicações nas 5 *flags* anteriores. No âmbito do trabalho a realizar, apenas optou-se por utilizar o campo PCR, por isso, não serão desenvolvidos detalhes acerca dos outros campos que se seguem.

- ***PCR* (42 bits):** *Program Clock Reference*, como o próprio nome indica, este campo especifica um relógio de referência, útil por exemplo para determinar o tempo e velocidade de apresentação em fluxos de dados MPEG2. Os primeiros 33 bits são baseados num relógio a 90KHz, incrementado por cada Hertz ou ciclo, e os restantes 9 bits de extensão são baseados num relógio de 27MHz;

- *stuffing bytes* (variável): bytes de adaptação, não representam qualquer tipo de informação. Utilizados apenas para adaptar toda a informação útil no pacote TS a um tamanho fixo de 188 bytes necessários para obter o comprimento fixo de um pacote deste tipo.

O protocolo MPEG2 – TS transporta finalmente todos os dados a serem transmitidos no sinal do nosso sistema DVB-T. No próximo passa-se a apresentar alguns conceitos e exemplares de dispositivos receptores do sinal referido.

Capítulo 4

Receptores DVB-T

4.1 Introdução

A especificação da norma DVB-T em 1994, com base nos requisitos comerciais e técnicos existentes na respectiva data, definiam como principal objectivo a criação de um sistema que permitisse uma recepção estacionária de sinal de televisão digital terrestre, recorrendo a apenas antenas fixas. Esses requisitos pretendiam também que o novo sistema de transmissão permitisse:

- Similaridade aos sistemas DVB-S e DVB-C;
- Uso de redes facilmente configuráveis utilizando uma única frequência, que coexistisse com os sistemas analógicos em vigor sem interferir com estes;
- Receptores de baixo custo;
- Estudo da possibilidade móvel.

O projecto DVB-T apresenta, actualmente, um conjunto de técnicas que permitem realizar além da recepção estacionária através de um canal gausseano e uma antena fixa, a recepção em dispositivos portáteis e também em dispositivos móveis com antenas de dimensões reduzidas, como receptores em automóveis, ou seja, a transmissão móvel tornou-se uma realidade, mantendo o baixo custo dos receptores necessários.

Os dispositivos actuais que se podem encontram no mercado, suportam um conjunto variado de tecnologias que implementam determinados conceitos e vantagens na recepção num sistema DVB-T.

4.2 Tecnologias mais comuns

4.2.1 Modo de Diversidade

O modo de diversidade é uma tecnologia baseada no conceito de recepção de sinal por mais do que uma antena, e tem sido largamente estudada e desenvolvida, tendo em conta que tem apresentado um conjunto significante de vantagens na recepção de sinal DVB-T. Está já actualmente presente em alguns dispositivos de recepção, com a utilização em simultâneo de, por exemplo duas antenas, e permite que os sinais recebidos por essas duas antenas se combinem formando um novo sinal. Deste modo temos uma expressiva melhoria na recepção de sinal, o que representa uma vantagem muito importante, permitindo até que, por vezes não seja necessário a instalação de antenas exteriores ou recorrer a acessórios adicionais, como antenas activas. A melhoria na recepção de sinal é ainda importante em situações de recepção em dispositivos portáteis, tendo em conta a possibilidade de utilização em locais onde existem constantemente objectos em movimento, que podem gerar diferentes tipos de reflexão e por consequente degradar o sinal, tais como pessoas ou veículos em deslocação. Além das vantagens apresentadas anteriormente, a melhor qualidade de sinal recebido através da utilização desta tecnologia, permite ainda uma recepção móvel muito mais viável, principal conceito para o constante estudo e desenvolvimento do modo de diversidade. Por exemplo, temos no mercado alguns dispositivos com duas antenas que permitem a recepção móvel até uma velocidade de 160 km/h.

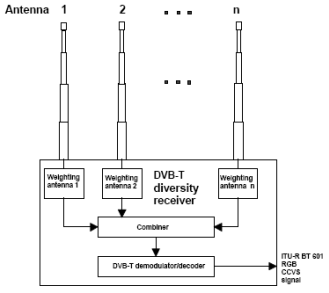
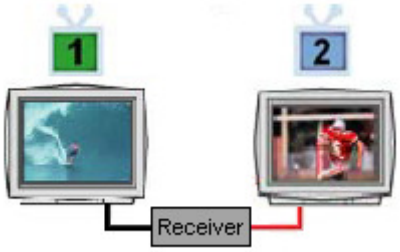


Figura 20. Receptor *Diversity* com n antenas.

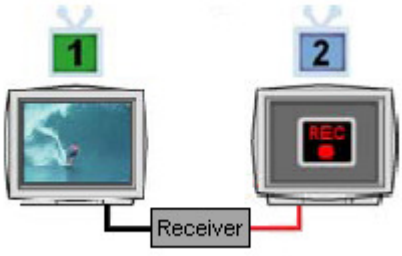
4.2.2 Modo de Duplo Sintonizador

O modo de duplo sintonizador é uma tecnologia relacionada com o modo de diversidade, apresentado anteriormente, visto que também tem por conceito a utilização de mais do que um sintonizador, neste caso, temos a utilização de dois. Esta tecnologia é praticável em situações em que o sinal recebido é suficiente para a utilização de apenas uma antena, pelo que o dispositivo entra em funcionamento com dois sintonizadores individuais, e que permitem a recepção de dois programas diferentes ao mesmo tempo. Ideal para a visualização de um programa enquanto se grava outro, ou outras situações, que em seguida são ilustradas:



Cenário 1: Temos o receptor a apresentar os diferentes programas recebidos em dois ecrãs distintos.

Figura 21. Cenário 1.



Cenário 2: Neste caso temos a ilustração de uma situação em estamos em simultâneo a visualizar um programa e gravar outro.

Figura 22. Cenário 2.

Cenário 3: Outra situação possível é a visualização de dois programas distintos num só ecrã.

Figura 23. Cenário 3.

4.3 Dispositivos em mercado para computadores pessoais

No mercado actual, existem imensos produtos de variadas marcas, que podem ser utilizados como dispositivo de recepção do sinal de televisão digital emitido, em qualquer tipo de computador pessoal. Numa simples pesquisa pela Internet, podemos encontrar muitos modelos de receptores desse tipo. Em seguida serão apresentados alguns exemplares, e respectivas características principais, a sua interface de comunicação que nos indica a que tipo de computadores se destina a utilização do dispositivo, as tecnologias por este suportadas, e por fim o preço que é um factor muito influente nas decisões de compra de produtos. Uma outra característica que também deve ser tomada em linha de conta é o suporte para, além da recepção do sinal de televisão digital, a recepção de outros tipos de sinal de televisão e até sinal de rádio, o que torna o dispositivo numa ferramenta com mais praticabilidade. Os dispositivos apresentados correspondem apenas a resultados da marca TERRATEC, visto corresponder à utilizada na execução deste projecto, e também porque os dispositivos de outras marcas são em tudo idênticos, e apenas se pretende evidenciar as diferentes possibilidades de equipamentos, tanto a nível de interface como de suporte de tecnologias diversificadas.

Terratec – Cinergy DT USB XS Diversity

Figura 24. Terratec – Cinergy DT USB XS Diversity.

Características:

- Dispositivo de interface USB, o que permite uma fácil utilização nos diversos cenários de recepção possível (estacionária, portátil e móvel);
- Implementa as tecnologias de diversidade e de duplo sintonizador;
- Suporta DVB-T;
- Preço: 99.99€

Terratec - Cinergy HTC USB XS HD

Figura 25. Terratec - Cinergy HTC USB XS HD.

.

Características:

- Dispositivo de interface USB, para fácil utilização em todo o tipo de computadores;
- Suporta DVB-T, DVB-C, Televisão analógica e rádio;
- Suporta HDTV;
- Preço: 99.99€

Terratec - Cinergy T USB XE

Figura 26. Terratec - Cinergy T USB XE.

Características:

- Dispositivo de interface USB, de fácil utilização em todo o tipo de computadores;
- Suporta DVB-T;
- Preço: 29.99€

Terratec - Cinergy 2400i DT

Figura 27. Terratec - Cinergy 2400i DT.

Características:

- Dispositivo de interface PCI Express, para uma utilização em situações de recepção estacionária em computadores fixos.

- Suporta a tecnologia de duplo sintonizador, com apenas a utilização de uma ligação a uma antena, a distribuição do sinal pelos dois sintonizadores é efectuada internamente.
- Suporta DVB-T.
- Preço: 59.99€

Terratec - Cinergy T Express

Figura 28. Terratec - Cinergy T Express.

Características:

- Dispositivo de recepção com interface ExpressCard/34, para uma utilização em computadores portáteis.
- Suporta DVB-T.
- Preço: 59.99€

Pode-se concluir que o preço de mercado destes dispositivos enunciados é relativamente baixo, devido ao facto da existência de uma vasta gama de dispositivos e de variadas marcas, como por exemplo a PINNACLE, a FREECOM, além da TERRATEC já referida. Estes dispositivos suportam a aplicação desenvolvida neste projecto. Assim sendo pode-se afirmar que estes dispositivos são de fácil acessibilidade a todos os alunos e podem ser utilizados para a recepção do sinal do nosso sistema DVB-T.

Capítulo 5

DirectShow

5.1 DirectShow

Neste capítulo serão apenas apresentadas algumas noções e conceitos acerca de DirectShow, deixando de parte detalhes mais aprofundados que não são necessários à implementação da aplicação de recepção de televisão digital. Algumas destas informações podem ser estudadas com mais detalhe no site da plataforma *Microsoft* de suporte a aplicativos http://msdn.microsoft.com[12], ou ainda em diversos livros como por exemplo "*Programming Microsoft Directshow For Digital Video And Television"*[13]*, Mark D. Pesce, by Microsoft 2003*.

Microsoft DirectShow é uma interface de programação para aplicações (API – *Application Programming Interface*) suportadas pela plataforma *Microsoft Windows,* com uma arquitectura direccionada a fluxos multimédia.

Figura 29. Logo de DirectX Media SDK.

Suporta uma vasta variedade de formatos, incluindo *Advanced Systems Format* (ASF), *Motion Picture Experts Group* (MPEG), *Audio-Video Interleaved* (AVI), *MPEG Audio Layer-3* (MP3), e ficheiros de som WAV. Permite a captura a de fluxos multimédia a partir de dispositivos analógicos ou digitais e ainda detecta e utiliza automaticamente aceleração hardware de vídeo e áudio sempre que possível, suportando na mesma sistemas sem aceleração hardware.

A *DirectShow* é baseada em *Component Object Model* (COM) e em muitas aplicações não é necessário a implementação de novos objectos COM, pois a API já disponibiliza uma vasta gama destes. Contudo, para a implementação da aplicação de recepção do sinal de televisão digital terrestre, não existem todos os objectos necessários para o processamento desejado sobre o fluxo de dados, pelo que será desenvolvido pelo menos um novo objecto.

DirectShow é então uma ferramenta essencial para a implementação de todo o tipo de aplicações que envolvam qualquer tipo de processamento em fluxos ou ficheiros multimédia, como por exemplo conversores de formato de ficheiro, aplicações de edição de vídeo, reprodutores de multimédia, aplicações para captura de áudio-vídeo como o caso da pretendida para a recepção do sinal DVB-T.

A principal característica é a facilidade de criação de aplicações multimédia para a plataforma Windows, isolando a aplicação de toda a complexidade dos transportes de fluxos de dados, dos diferentes dispositivos de hardware e sincronização dos dados.

Para a apresentação dos dados de vídeo e áudio, *DirectShow* utiliza sempre que possível *DirectDraw* e *DirectSound* respectivamente. Estas tecnologias permitem o envio eficiente dos fluxos de vídeo e áudio para os respectivos dispositivos de hardware.

No diagrama seguinte podemos observar a relação genérica entre uma aplicação, os componentes *DirectShow* e alguns dos componentes de hardware e software que o *DirectShow* também suporta.

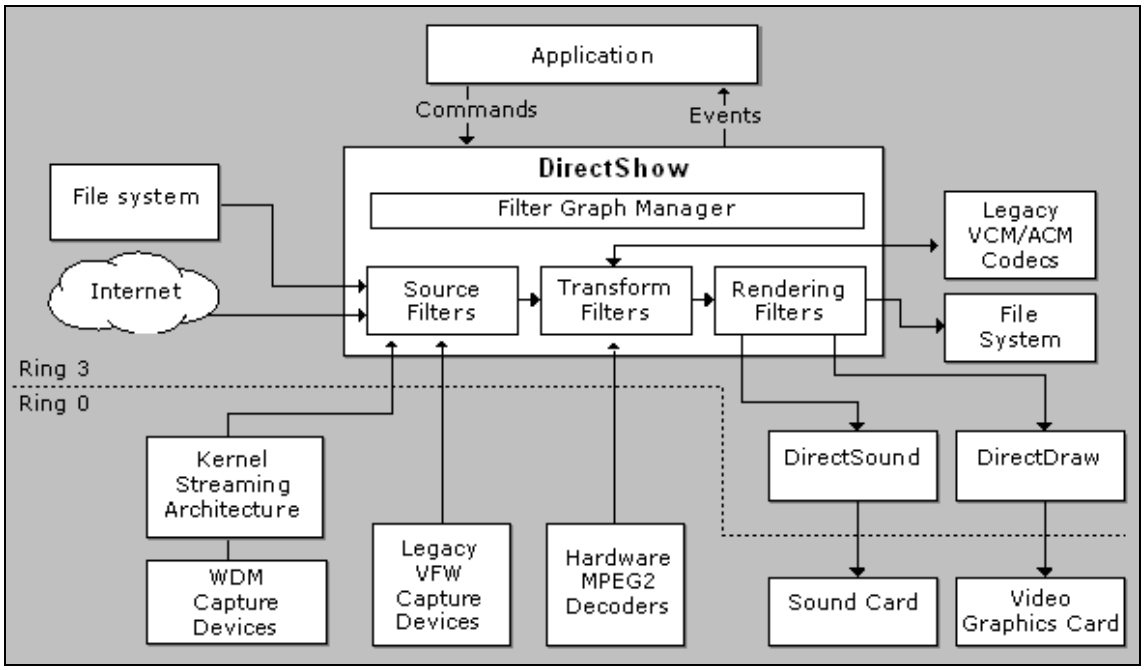


Figura 30. Diagrama genérico de uma aplicação em *DirectShow*.

5.2 Filtros

Um filtro é basicamente um módulo ou um estágio. A aplicação baseada em *DirectShow* subdivide uma tarefa multimédia complexa, como por exemplo a captura e apresentação do sinal de televisão digital em causa, por uma sequência fundamental de fases de processamento. Cada uma destas fases é então colocada num módulo de certo modo independente e denominado por filtro.

Um filtro é um componente em software que permite de algum modo uma operação sobre um fluxo de dados multimédia. Por exemplo um filtro *DirectShow* pode efectuar uma leitura de um ficheiro, obter vídeo através de um dispositivo de captura de vídeo, fornecer os dados a placas gráficas e/ou de som, etc.

A ligação entre os filtros é possível devido à utilização de pinos de entrada e/ou de saída. Os pinos de entrada são utilizados para a recepção de toda a informação a ser processada pelo filtro, e os pinos de saída possuem a informação pretendida após o respectivo processamento pelo filtro. Por exemplo um filtro descodificador de vídeo MPEG-1 recebe à entrada fluxos MPEG codificados, e coloca na saída uma série de frames de vídeo descomprimidas.

Os pinos permitem um mecanismo de ligação de um modo desejado entre filtros, admitindo assim uma implementação de um conjunto variado de funções mais complexas.

Os filtros *DirectShow* são caracterizados segundo três tipos fundamentais:

- *Source filters*: estes filtros providenciam os fluxos de dados fonte, como por exemplo temos na nossa situação a utilização de um filtro deste tipo responsável para a captura do sinal de televisão digital, que em seguida entrega esse fluxo de dados aos seguintes filtros que compõem todo o processamento sobre o sinal recebido.

Figura 31. Exemplos de *Source filters*.

- *Transform filters*: estes, como o próprio nome indica, transformam os dados que recebem segundo um efeito desejado e implementado pelo filtro, temos como exemplo o filtro que utilizamos para seleccionar um

determinado fluxo identificado por um PID específico. A principal característica comum neste tipo de filtros é que apenas possuem um pino de entrada e um de saída.

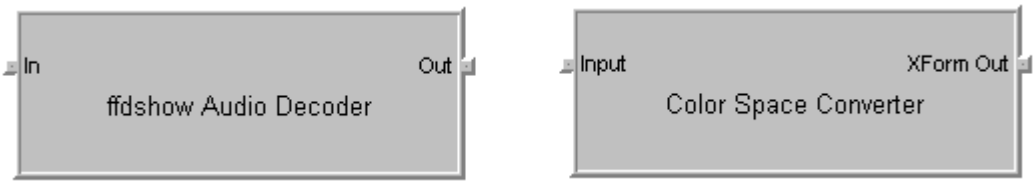


Figura 32. Exemplos de *Transform filters*.

- *Renderer filters*: este tipo de filtros é utilizado para a apresentação final após o processamento sobre o fluxo de dados recebido pelos filtros anteriores, temos então os filtros que são responsáveis pela reprodução do vídeo e áudio que compõem o sinal de televisão digital.

Figura 33. Exemplos de *Renderer filters*.

Os filtros encontram-se sempre num de três estados possíveis de funcionamento:

- *Running*: indica que o filtro se encontra a processar dados.
- *Stopped*: na situação em que o filtro se encontra parado, ou seja, não está a processar dados.
- *Paused*: utilizado para iniciar a recepção dos dados antes de passar ao estado *Running*, deste modo permitindo que a transição responda imediatamente.

5.3 Grafo de filtros

Para a implementação de uma determinada tarefa complexa em DirectShow devemos em primeiro lugar desenvolver um grafo, denominado por *filter graph,* composto por instâncias aos filtros necessários e respectivas conexões, de modo a representar o modo de propagação dos fluxos de dados.

Em seguida é apresentada um exemplo de um grafo de filtros, que proporciona a reprodução de um ficheiro de áudio, e em que se pode observar a utilização dos três tipos de filtros descritos anteriormente. Este grafo foi gerado através da utilização da ferramenta *GraphEdit* disponibilizada junto da plataforma SDK do Windows.

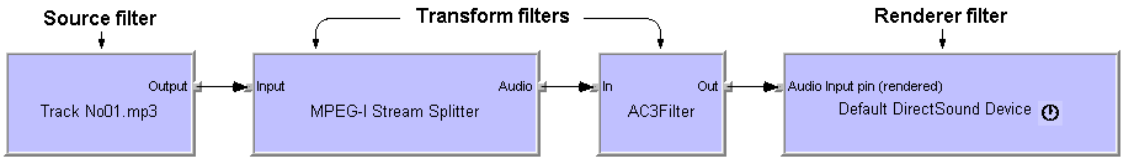


Figura 34. Grafo de filtros em GraphEdit.

GraphEdit é então uma ferramenta que se pode utilizar simplesmente para a implementação e teste de um grafo, ou até mesmo para verificar se a aplicação implementou correctamente o grafo de filtros desejado.

Durante o processo de construção de um grafo através desta ferramenta, o utilizador escolhe os filtros pretendidos que se encontram registados no Windows, e em seguida efectua a ligação desejada entre esses filtros definindo o modo de propagação dos fluxos de dados. Após a implementação do grafo de filtros, é permitido a sua execução de modo a que o utilizador visualize o resultado obtido.

5.4 Componentes de construção de um grafo de filtros

A *Directshow* providencia vários componentes que podem ser utilizados para a construção de um grafo de filtros. Em seguida serão apresentados e descritos os relevantes ao propósito da aplicação:

- *Filter Graph Manager*: Este objecto controla todo o grafo de filtros. Suporta um vasto conjunto de interfaces como por exemplo IGraphBuilder e IMediaControl.

- *Capture Graph Builder*: este objecto dispõe um conjunto de métodos adicionais para a construção de um grafo de filtros. Foi desenvolvido para o propósito de grafos com o objectivo de captura de vídeo, mas é muitas vezes utilizado em outros tipos de grafos. Suporta a interface ICaptureGraphBuilder2.

- *Filter Mapper* e *System Device Enumerator*: Estes objectos são responsáveis por localizar os filtros registados no sistema ou que representam dispositivos hardware (*Wrapper filters*).

5.5 *Filter Graph Manager*

O *Filter Graph Manager* é um objecto COM responsável por gerar o grafo de filtros pretendido para executar uma determinada tarefa complexa. Além disso controla cada filtro constituinte do grafo, coordena as alterações de estados dos filtros, estabelece um relógio de referência para sincronismo, comunica com a aplicação em caso de ocorrência de determinados eventos que ocorrem no grafo de filtros e providencia métodos que permitem à aplicação construir o grafo, ou seja, adicionar filtros e efectuar conexões e desconexões.

Todos os filtros utilizam o mesmo relógio de referência (*reference clock*), escolhido pelo *Filter Graph Manager*, o que permite a sincronização dos fluxos de dados.

5.5.1 Transições de estado nos filtros

Como já referido, o *Filter Graph Manager* é responsável por coordenar todas as alterações de estados nos filtros constituintes do grafo. Este objecto efectua a alteração de estado dos filtros desde o filtro *renderer* até ao filtro fonte de todo o fluxo de dados, deste modo prevenindo que não seja perdida qualquer tipo de informação e que o grafo de filtros não entre em algum tipo de bloqueio.

As transições de estado nos filtros mais críticas são entre os estados de *paused* e *stopped*:

- *Stopped* para *Paused*: Quando um filtro transita do estado *stopped* para o estado *paused*, passa a estar preparado a receber dados para processar. O filtro fonte do fluxo de dados a propagar-se pelo grafo é o último filtro a transitar ao estado *paused*, permitindo assim que quando este iniciar a distribuir dos dados, todos os outros filtros já estão no estado *paused* e preparados a receber os respectivos dados a processar, pelo que não serão perdidos fluxos de informação.

- *Paused* para *Stopped*: Nesta transição de estado, um filtro liberta os dados que possui, o que possibilita o desbloqueio de qualquer filtro seguinte que se encontre em espera desses dados. De acordo com a ordem pela qual o *Filter Graph Manager* efectua a alteração de estados dos filtros no grafo, não serão perdidos quaisquer segmentos de informação, mesmo que um filtro liberte um determinado conjunto de dados antes de receber o comando para alterar de estado, os filtros seguintes já encontram-se no estado *stopped*, pelo que esses dados são rejeitados.

5.5.2 Interacção entre aplicação e grafo de filtros

Através da utilização do *Filter Graph Manager* a aplicação não necessita nem de controlar directamente o processo de encaminhamento dos fluxos de dados nem as transições de estados dos filtros. A aplicação apenas efectua chamadas de nível superior ao *Filter Graph Manager*, como por exemplo "*Run*" que indica que os fluxos de dados devem propagar-se ao longo do grafo. Para um acesso mais directo às operações sobre os fluxos de dados, os filtros devem ser acedidos directamente através das interfaces COM de suporte aos respectivos filtros.

De modo geral, uma aplicação deve então efectuar três tarefas, ilustradas na figura seguinte e posteriormente descritas.

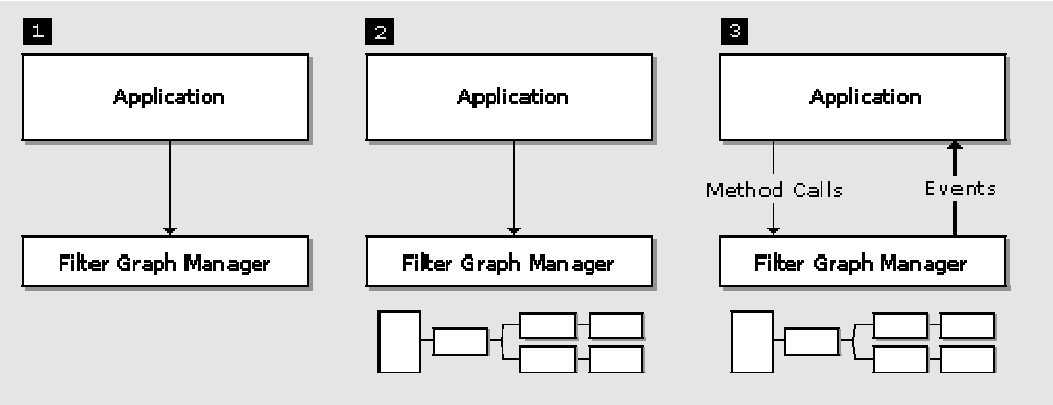


Figura 35. Interacção entre aplicação e grafo de filtros.

Em primeiro lugar a aplicação deve criar uma instância ao componente *Filter Graph Manager* e em seguida é utilizado esse componente para a criação do grafo de filtros (*filter graph*) necessários à execução da tarefa pretendida. Por fim temos, também através do *Filter Graph Manager*, o controlo sobre o grafo de filtros e respectivo fluxo de dados que se propaga pelos filtros, e a resposta da aplicação às notificações dos eventos que recebe.

Após a conclusão da tarefa pretendida, a aplicação liberta o *Filter Graph Manager* e por consequente todos os filtros.

DirectShow é baseado em COM, então o *Filter Graph Manager* e todos os filtros são objectos COM.

5.5.3 Interacção do hardware e grafo de filtros

Todos os filtros *Directshow* são blocos em software que executam uma determinada tarefa. Um dispositivo hardware, como por exemplo uma placa de vídeo, é representada por um filtro correspondente e assim é possível a sua inclusão num grafo de filtros *Directshow*. Todos os filtros que representam um dispositivo hardware são denominados por filtros *wrapper*.

Os filtros *wrapper* expõem interfaces COM utilizadas para fornecer e receber informação do filtro, e representam as capacidades do dispositivo.

Assim sendo os filtros *wrapper* fornecem um método de controlo da aplicação sobre um determinado dispositivo, não exigindo complexidades de programação, pois todos os detalhes de interacção com o dispositivo hardware são deixados ao cargo do respectivo filtro.

Figura 36. Exemplo de um filtro *wrapper*.

5.6 Ligações entre filtros

5.6.1 *Media types*

Media Types é um mecanismo universal utilizado em DirectShow para descrever o formato de dados em cada ponto do *Filter Graph*. Por exemplo no caso de um grafo para a reprodução de um ficheiro do tipo AVI, temos em certo ponto a separação em fluxos de vídeo e de áudio, pelo que os dados devem ser identificado pelo seu tipo.

Uma ligação entre dois filtros só é possível se os respectivos pinos que possibilitam a ligação acordarem em relação ao tipo de dados a serem transmitidos.

Media types são definidos segundo a estrutura AM_MEDIA_TYPE e contem os seguintes campos de informação:

- *Major type*: é uma GUID que define a categoria dos dados.

- *Subtype*: é outra GUID e define um formato mais específico, ou seja, providencia ainda mais informação acerca do tipo de dados além do Major type.

- *Format block*: descreve o formato dos dados em detalhe, como por exemplo indica o tamanho de uma imagem. Esta informação é alocada separadamente, sendo referenciada pelo ponteiro pbFormat contido na estrutura AM_MEDIA_TYPE.

A utilização do *Format block* para definir o tipo de dados torna redundante a informação contida em *Major type* e *Subtype*, contudo estes dois campos de informação são essenciais para situações em que se quer

simplesmente identificar o tipo de dados sem detalhar o formato dos mesmos.

A estrutura de *Media Types* possui ainda alguns campos apenas opcionais que podem ser utilizados para indicar informação adicional:

- *lSampleSize*: indica o tamanho de cada *sample*. Se este campo possui o valor zero, significa que o tamanho da *sample* pode variar.

- *bFixedSizeSamples*: indica, no caso de tomar o valor booleano "TRUE" que deve ser tomada em conta a informação contida em lSampleSize e em caso contrário deve ser ignorada.

- *bTemporalCompression*: no caso de conter o valor booleano "FALSE", indica que todas as frames são key frames.

5.6.2 *Transports*

Directshow suporta um conjunto de possíveis protocolos, chamados de *transports*, que possibilitam a propagação dos dados media ao longo dos grafos de filtros. Quando dois filtros estão conectados, estes têm de suportar o mesmo tipo de transporte sobre os dados entre os pinos que suportam a ligação.

A maior parte dos filtros *Directshow* colocam os dados media na memória principal do sistema, este tipo de transporte é denominado por *local memory transport*. Estão definidos dois mecanismos de *local memory transport*, o *push model* que utiliza a interface *IMemInputPin* e indica que um filtro fonte gera os dados e entrega ao filtro receptor, e o *pull model* que utiliza a interface *IAsyncReader*, indica que o filtro fonte aguarda o pedido de entrega de dados pelo filtro *parser* a este conectado.

O *push model* é o mecanismo mais utilizado.

5.6.3 *Allocators* e *media samples*

Os fluxos de dados propagam-se através de conexões desde um pino de saída até um outro de entrada, em buffers que correspondem simplesmente a *arrays* de bytes.

O modo mais comum de entrega de dados por um pino de saída é efectuando a chamada ao método *IMemInputPin::Receive* no pino de entrada, ou seja, no pino que recebe os dados (*push model*, como já referido), contudo existem outros mecanismos que permitem a transmissão dos dados entre os dois pinos.

O objecto COM responsável pela alocação de memória necessária para colocar o respectivo conjunto de buffers que suporta a ligação, é denominado por *allocator* e possui a interface *IMemAllocator*. Um *allocator* está sempre associado a uma ligação entre dois pinos, e pode ainda ser partilhado por duas ou mais conexões de pinos.

Quando dois pinos efectuam uma ligação, um dos dois pinos deve definir um *allocator* responsável pela ligação. DirectShow define uma sequência de métodos que podem ser utilizados para estabelecer qual dos pinos providencia o *allocator*. Os pinos também acordam em relação às características dos buffers que o *allocator* deve criar, como por exemplo o número de buffers e respectivo tamanho.

Antes do início do fluxo de dados pelo grafo de filtros, o *allocator* cria o respectivo conjunto de buffers, os quais serão preenchidos com dados pelo filtro fonte da ligação, deste modo permitindo que o filtro receptor possa obter os dados pretendidos. Contudo o filtro fonte não indica directamente ao receptor os respectivos ponteiros ao conjunto de buffers,

essa tarefa é deixada ao cargo de objectos COM denominados por *media samples* criados pelo *allocator* para controlar esses buffers.

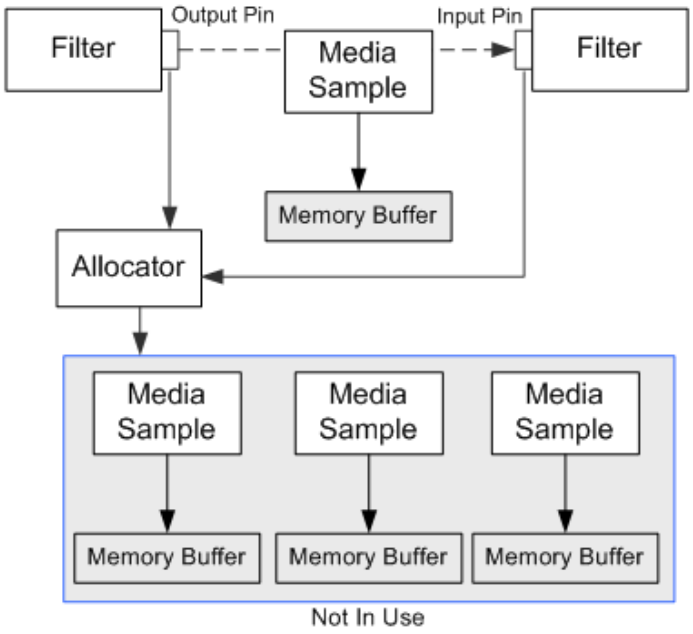


Figura 37. Ligações entre dois filtros.

Os objectos *media samples* possuem uma interface chamada *IMediaSample* que contém um conjunto de informações:

- ponteiro para o buffer subjacente;
- *time stamp*: define o tempo de amostragem, utilizado pelo filtro de apresentação dos dados, de modo a sincronizar os dados;
- algumas *flags*: indicam vários propósitos como por exemplo uma situação de pausa de dados desde a *sample* anterior;
- opcionalmente também possui o *media type*: que providencia um modo aos filtros de alteração do tipo de dados fornecidos. Normalmente este campo não existe pelo que indica que o filtro não alterou o tipo de dados desde a última *sample*.

Sempre que um filtro requer a recepção de um *buffer* de dados para continuar o seu processamento, este efectua um pedido ao *allocator* através da interface com a chamada a *IMemAllocator::GetBuffer*. No caso em que o *allocator* possui alguma *sample*, que no momento não se encontra em utilização por um outro filtro, é de imediato devolvido um ponteiro a indicar a posição da *sample*, e em seguida o filtro efectua o respectivo processamento e entrega de resultado. No caso contrário, o filtro permanece em espera até que surja um *sample* disponível.

Enquanto um filtro está sobre a posse de um buffer, este incrementa o contador de referência da *sample*, assim permitindo ao *allocator* saber se um determinado buffer se encontra disponível ou não para ser reutilizado. Este método previne que um filtro substitua os dados num buffer que ainda estão a ser utilizados por outro buffer.

No caso concreto dos filtros *renderer*, a recepção de uma *sample* implica que o filtro verifique o *time stamp* e espere até que o relógio de referência do grafo de filtros indique que a respectiva *sample* deve ser amostrada. Após a amostragem, o filtro liberta a *sample*, contudo está não retorna à área de alojar *samples* permitindo a sua sobreposição, até que o seu contador de referência indique o valor zero, significando deste modo que todos os filtros libertaram a respectiva *sample*.

5.6.4 *Intelligent Connect*

Intelligent Connect é um mecanismo que o *Filter Graph Manager* utiliza para a construção de grafos de filtros e consiste num conjunto de algoritmos que seleccionam e adicionam determinados filtros ao grafo. Este método de construção é utilizado quando se pretende deixar ao cargo do *Filter Graph Manager* a construção de todo ou apenas parte do grafo de filtros.

Quando o *Filter Graph Manager* necessita de filtros adicionais para completar o grafo, este segue um determinado procedimento:

1- Se já existe um filtro no grafo, e pelo menos um pino não conectado, o *Filter Graph Manager* tenta utilizar esse filtro.

2- No caso contrário, o *Filter Graph Manager* procura no registo por filtros que podem ser utilizados para a ligação de acordo com o tipo de media de dados. Cada filtro possui um valor que indica o quão este é normalmente utilizado em grafos de filtros, pelo que o *Filter Graph Manager* escolhe o de valor mais elevado.

5.7 Implementação de filtros

Como já referido anteriormente, para a realização da nossa aplicação de recepção de sinal de televisão digital, é pretendido um determinado processamento sobre o fluxo de dados que ainda não é disponibilizado pelos filtros existentes e registados no nosso sistema *Windows*, pelo que nesta secção será apresentado o processo de implementação de novos filtros.

5.7.1 *Classes*

A maior parte dos filtros são implementados com base num conjunto de classes C++ disponibilizadas em *DirectShow SDK*, denominadas por *DirectShow Base Classes.*

A biblioteca base de classes disponibilizada define *CBaseFilter* como a classe mãe para todo o tipo de filtros *DirectShow*. Contudo existem já desenvolvidas, e derivadas da classe mãe, outras classes de filtros mais especificadas ao tipo de filtros a que se destinam, como por exemplo *CTransformFilter* destinada aos filtros de transformação.

Para o desenvolvimento de um novo filtro, devemos em primeiro lugar determinar a classe mais adequada ao tipo de filtro pretendido.

5.7.2 Criação de pinos

Um filtro deve criar um ou mais pinos de comunicação, e o número de pinos pode ser fixado no momento de desenvolvimento do filtro ou então podemos deixar ao cargo do filtro a tarefa de criar novos pinos sempre que necessite de uma nova ligação.

Os pinos geralmente derivam de uma classe mãe chamada *CBasePin*, ou então, como no caso das classes destinadas aos filtros, existem outras classes projectadas ao tipo de pino a que se destinam, como *CBaseInputPin*.

Os pinos pertencentes a um filtro devem ser declarados como variáveis na classe de filtros.

Algumas classes de filtros definem automaticamente os pinos a serem criados pelo filtro, contudo se o filtro derivar da classe mãe (*CBaseFilter*), é necessária a declaração de pinos como referido.

Capítulo 6
Desenvolvimento da aplicação

6.1 Introdução

Tendo em conta o estudo a realizado nos capítulos anteriores, e a existência de técnicas já desenvolvidas para o desencapsulamento dos dados inseridos no protocolo RTP, optou-se apenas por efectuar a recepção dos dados e desencapsular até ser obtido um fluxo de dados que contém apenas pacotes de protocolo RTP.

Numa primeira fase do estudo, optou-se pela utilização de um encapsulador proveniente do insersor de IP da *Rohde&Schwarz*, existente no laboratório de estudo, cujo processamento sobre os fluxos de dados seguia um conjunto específico de características. Contudo através deste método foram verificadas algumas dificuldades no desenvolvimento do filtro receptor, capaz de desencapsular os dados. Por este facto determinou-se que seria mais vantajoso desenvolver além do filtro referido e da aplicação para a recepção do sinal, uma aplicação capaz de efectuar o encapsulamento de protocolo RTP até ser obtido um fluxo de dados em *Transport Stream* MPEG-2. Em seguida esses dados foram colocados num insersor de um dispositivo da DEKTEC (DTA-140) instalado num computador do laboratório de vídeo, de modo a poderem ser transmitidos.

6.2 Problemas do insersor de IP da *Rohde&Schwarz*

Como referido o insersor utilizado inicialmente possuía um conjunto de características específicas que originaram diversos problemas na implementação do filtro. Essas características são mencionadas em seguida:

- Os pacotes de protocolo MPEG-2 TS nunca possuíam campo de adaptação, ao contrário do especificado nas normas, essa zona é substituída por dados do(s) próximo(s) pacote(s). No caso de ocorrer esta situação, o primeiro *Byte* na zona de dados do pacote MPEG-2 TS corresponde a uma indicação ao deslocamento (*offset*) em *Bytes* ao início do segundo pacote MPE contido nesta zona de dados. Logo este *Byte* tem de ser descartado, pois não possui conteúdo útil de protocolo MPE.

Esta situação ainda levanta um outro problema que é, na ocorrência de espaço livre na zona de dados, capaz de suportar mais do que um dos próximos pacotes de protocolo MPE, apenas obtém-se a informação da posição de inicio do primeiro pacote, não havendo qualquer tipo de indicação relativa ao segundo pacote MPE.

Na figura 36 é ilustrado um exemplo da situação mencionada:

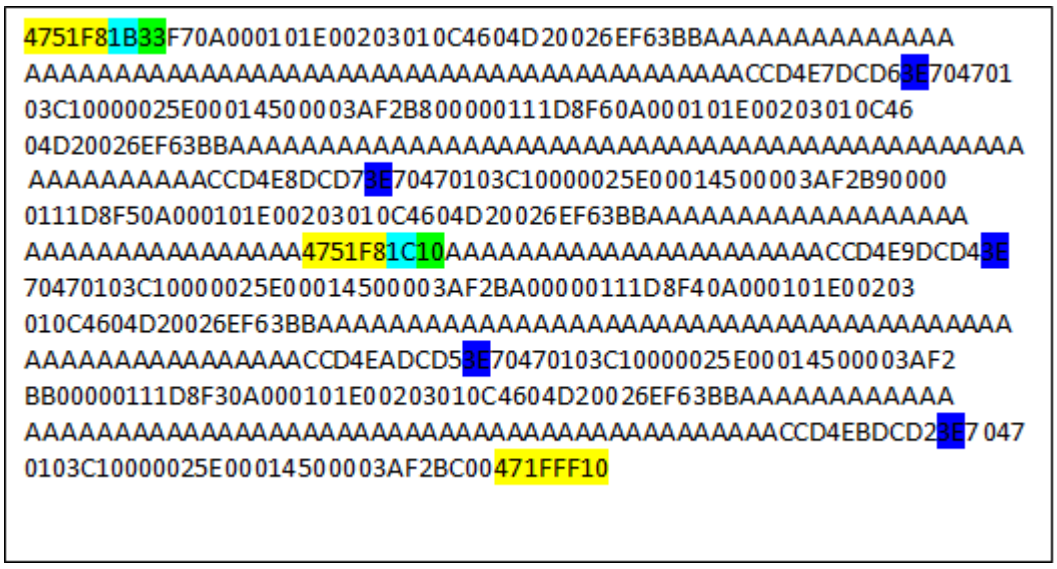


Figura 38. Insersor R&S - Problema 1.

Na figura anterior pode-se observar o início de vários pacotes de protocolo MPE dentro de apenas uma zona de dados de um pacote MPEG-2 TS. O início de um pacote MPE está sinalizado por 0x3E. Verifica-se ainda o *Byte* 0x10 que indica apenas o deslocamento ao início do primeiro pacote de protocolo MPE.

- Um outro problema verifica-se no caso de apenas restar um *Byte* livre na zona de dados do protocolo MPEG-2 TS. Neste caso a situação anterior referida não é verificada. O insersor de IP R&S nesta situação coloca no *Byte* livre o valor 0xFF, e este representa um *Stuffing Byte,* ou seja, este conteúdo deve ser ignorado.

Na figura seguinte pode-se observar a situação referida:

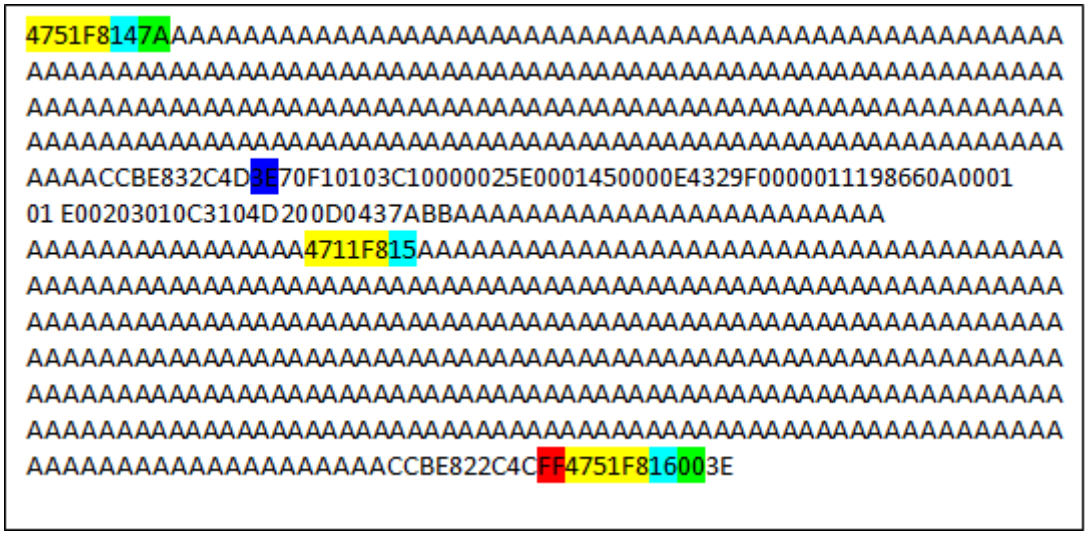


Figura 39. Insersor R&S - Problema 2.

Pode-se identificar na figura a existência de um Byte com o valor 0xFF no final da zona de dados do pacote de protocolo MPEG-2 TS.

6.3 Implementação do encapsulador

No desenvolvimento do encapsulador referido, foram seguidas as especificações apresentadas anteriormente no capítulo 3, acerca dos protocolos de comunicação utilizados para encapsular os dados. Todo o código desenvolvido responsável pela tarefa a efectuar, segue em anexo 1.

Na figura 38 pode-se obter uma visão geral da utilização do software desenvolvido para o processamento de encapsulação desejado e respectiva emissão dos dados (MPEG-2 *transport Stream*) gerados.

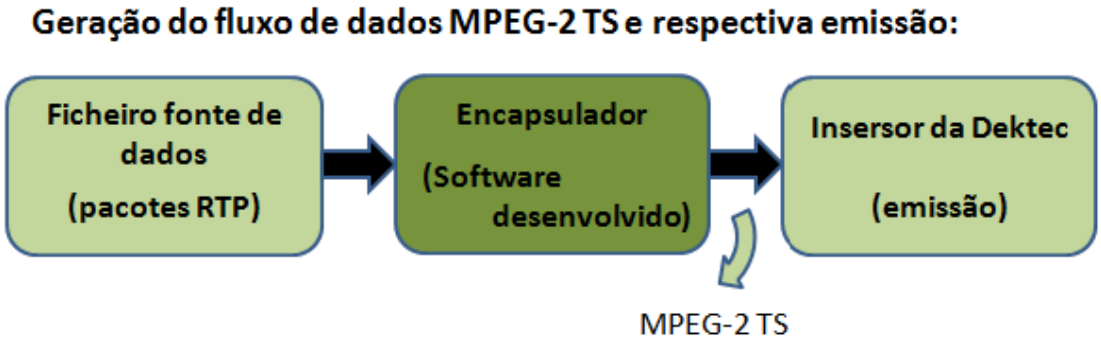


Figura 40. Geração do fluxo de dados MPEG-2 TS e emissão.

Em seguida é apresentado um diagrama de blocos onde é ilustrado toda a base de funcionamento do software de encapsulamento desenvolvido.

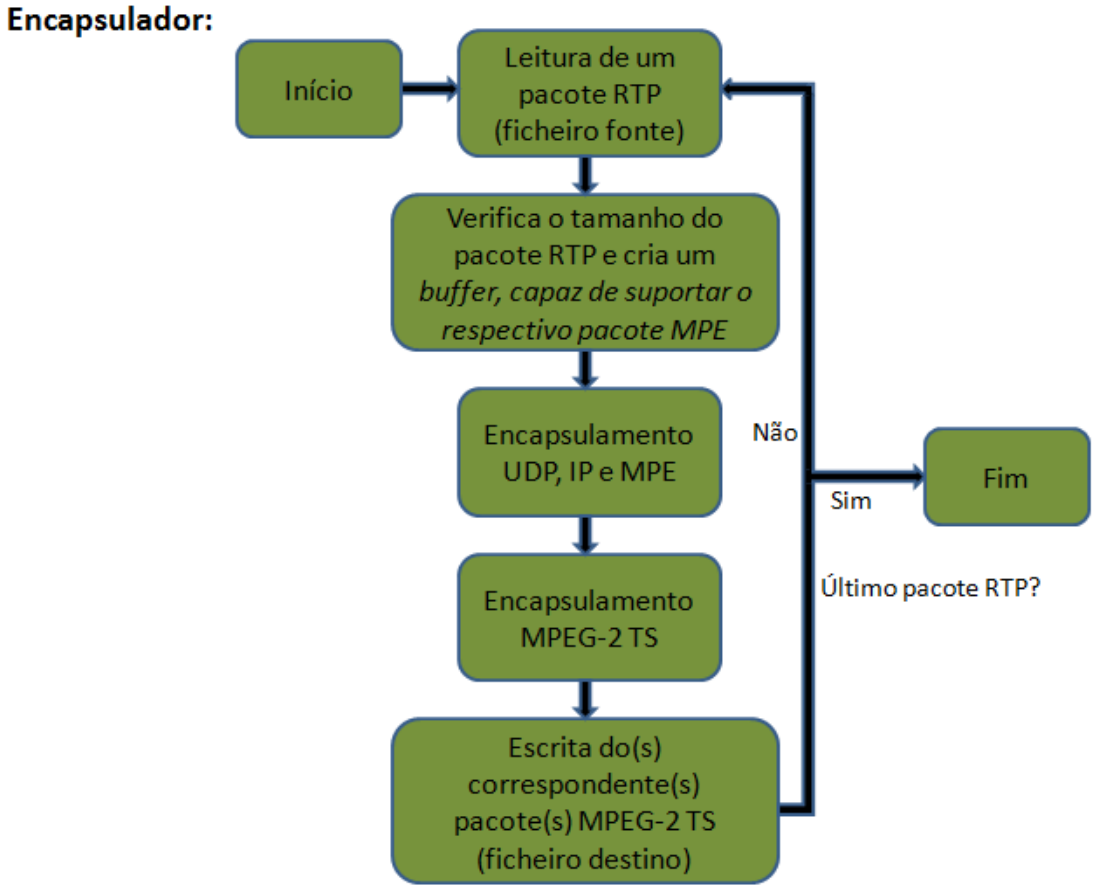


Figura 41. Diagrama de blocos - Encapsulador.

6.4 Implementação do Filtro

Após a implementação do encapsulador referido anteriormente, passou-se ao desenvolvimento do filtro necessário introduzir na aplicação que irá receber os dados DVB-T. Este filtro será responsável por tratar os dados de modo a ser obtido um fluxo de dados apenas com pacotes de protocolo RTP, deste modo sendo possível depois aplicar os métodos de desencapsulamento de protocolo RTP e retirar os dados MPEG-2 que representam o resultado final e esperado no propósito da realização deste estudo.

Visto o objectivo do trabalho tomar por conceito um estudo sobre a recepção dos dados DVB-T, e de modo a facilitar um pouco a tarefa a realizar, optou-se por apenas salvaguardar os dados recebidos e só depois analisar o resultado. Assim sendo, o filtro desenvolvido teve por base uma "*sample*" fornecida junto da plataforma SDK da *Microsoft*, foram efectuadas algumas adaptações e por fim foi inserido um algoritmo de desencapsulamento deste MPEG-2 *Transport Stream* até protocolo RTP. A "*sample*" referida é denominada por filtro DUMP e tem por objectivo o salvaguardar dos dados recebidos no seu pino de entrada, num ficheiro previamente definido.

O filtro desenvolvido é apresentado em anexo 2.

Na figura seguinte pode-se, de forma análoga à apresentada no caso do software do encapsulador, obter a visão geral da utilização do filtro, e neste caso também da aplicação final, de modo a adquirir uma melhor compreensão sobre a aplicabilidade do software desenvolvido.

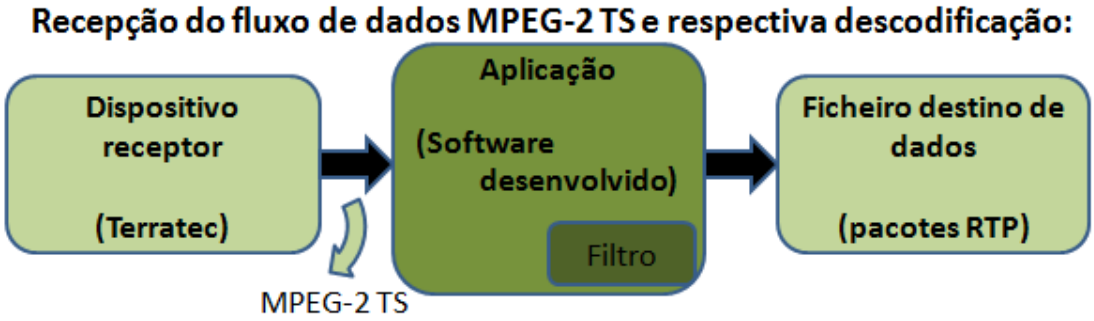


Figura 42. Recepção do fluxo de dados MPEG-2 TS e descodificação.

Na figura 41 apresenta-se um diagrama de blocos para o caso do software desenvolvido, relativo ao filtro de desencapsulamento.

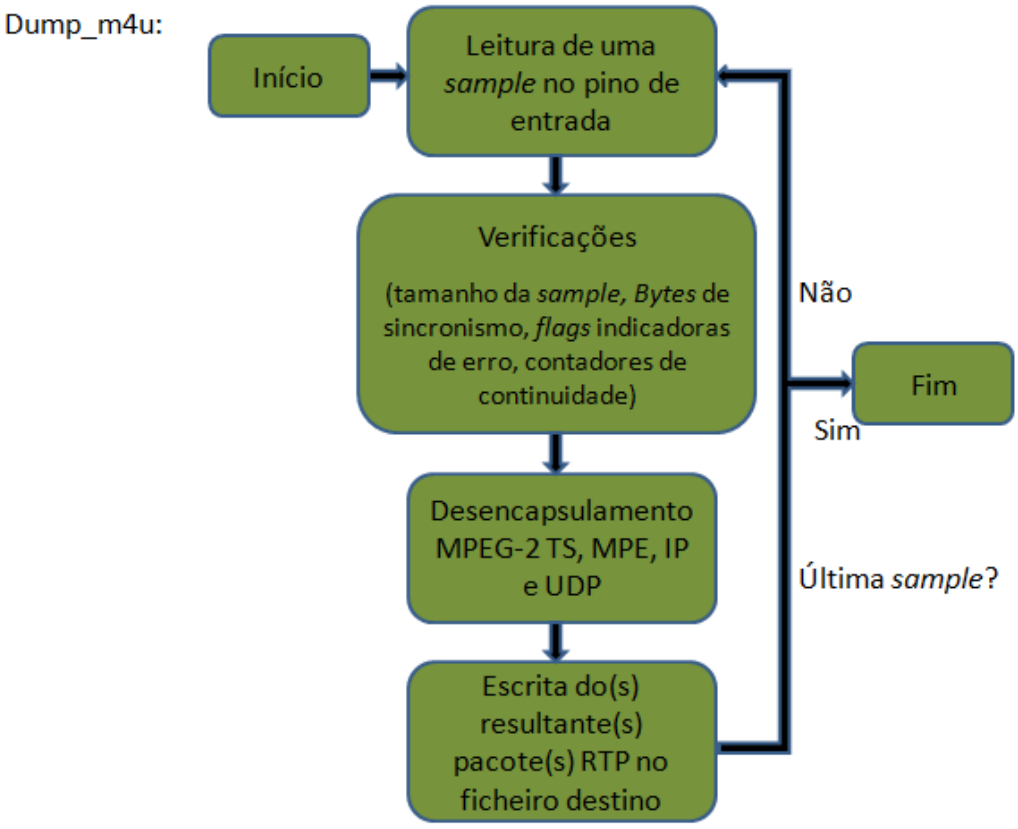


Figura 43. Diagrama de blocos - Desencapsulador.

6.5 Desenvolvimento da aplicação

Figura 44. Logo de Visual Studio 2005.

Um projecto baseado na arquitectura DirectShow pode ser implementado através da linha de comandos básica da plataforma do Windows ou, como utilizado para o desenvolvimento da aplicação de recepção de televisão digital, podemos utilizar a ferramenta de desenvolvimento de software Microsoft Visual Studio, mais precisamente a versão de 2005.

Visual Studio é essencialmente dedicado a linguagens de programação como Visual Basic (VB), C, C++ (*C Plus Plus*), C# (*C Sharp*) e J# (*Jey Sharp*), e permite um ambiente de programação de software tanto para utilizadores mais experientes como para principiantes.

Todas as aplicações baseadas em DirectShow devem utilizar um ficheiro de referência, que contém os componentes necessários e fundamentais da arquitectura para a implementação da aplicação em causa. O ficheiro utilizado foi então o DirectShowLib-2005.dll disponibilizado em DirectShow.Net.

Numa reflexão inicial acerca da aplicação a ser desenvolvida neste projecto, e após alguma pesquisa, identificou-se a existência de uma aplicação denominada por DTVViewer-2005, já desenvolvida e disponibilizada em DirectShow.Net, destinada à recepção de um sinal genérico de televisão digital. Contudo esta aplicação não satisfazia o propósito do projecto a realizar, pelo que apenas foi utilizada para modelo de estudo.

6.5.1 Ferramentas auxiliares à construção do grafo de filtros

Inicialmente desenvolveu-se um conjunto de ferramentas auxiliares, destinadas aos vários propósitos na construção do grafo de filtros pretendido, como por exemplo o identificar e adicionar filtros.

Assim sendo temos o ficheiro "*FilterGraphTools.cs*", que segue como anexo 3 ao relatório do projecto, onde estão apresentadas as ferramentas desenvolvidas e aqui mencionadas:

AddFilterFromClsid( ); - Adiciona um filtro ao grafo através do respectivo identificador exclusivo.

IsThisComObjectInstalled( ); - Verifica se um determinado objecto COM se encontra disponibilizado.

RemoveAllFilters( ); - Remove e liberta todos os filtros utilizados no grafo gerado pela aplicação.

SaveGraphFile( ); - Salva o grafo de filtros gerado pela aplicação num ficheiro especificado.

6.5.2 Desenvolvimento do grafo de filtros

Procedeu-se ao desenvolvimento da estrutura base que implementa a tarefa pretendida, ou seja, o grafo de filtros que define o modo de processamento sobre os fluxos de dados a receber. Toda essa implementação está contida no ficheiro "*BDAGraphBuilder.cs*" apresentado em anexo 4.

Em seguida serão apresentados os detalhes essenciais dessa estrutura:

BuildGraph( ); - A função que cria o grafo de filtros. Esta é composta por um conjunto de outras funções que implementam cada uma das fases de construção do grafo pretendido:

AddNetworkProviderFilter( ); - Adiciona um filtro que identifica os parâmetros de sintonia ao programa pretendido, como por exemplo a frequência de canal.

AddMPEG2DemuxFilter( ); - Adiciona o filtro MPEG2-Demultiplexer de onde poderemos extrair o fluxo de dados que correspondente ao programa pretendido. Assim sendo, além de adicionar o Demultiplexer, nesta função também foram definidos alguns parâmetros, como por exemplo o PID, de modo a filtrar e seleccionar apenas o fluxo de dados desejado.

AddAndConnectBDABoardFilters( ); - Enumera os dispositivos BDA registados no sistema e determina um que possa ser utilizado para a recepção do sinal de televisão digital pretendido, de acordo com os parâmetros de sintonia definidos.

AddAndConnectCustomFilter( ); - Adiciona o filtro elaborado na realização deste projecto e conecta ao *MPEG2-Demultiplexer*, permitindo assim obter o resultado final do processamento sobre o fluxo de dados.

Determinou-se assim que este era o grafo de filtros necessário e suficiente para a execução da tarefa desejada no propósito da aplicação desenvolvida neste projecto:

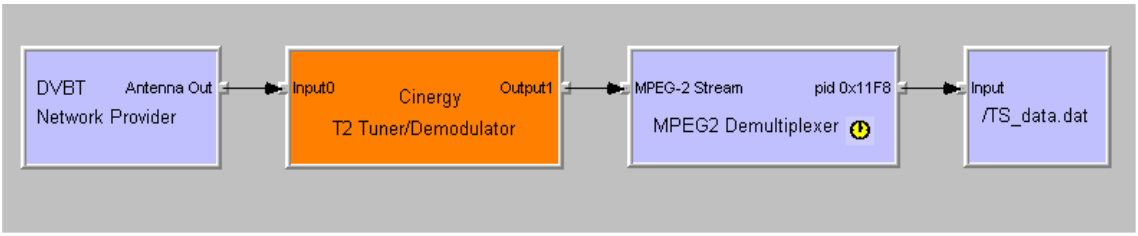


Figura 45. Grafo de filtros gerado pela aplicação desenvolvida.

Além da função principal de suporte à aplicação, ainda podem encontrar neste ficheiro as seguintes funções:

BDAGraphBuilder( ); - Permite o controlo da aplicação sobre o grafo de filtros;

SubmitTuneRequest( ); - Permite a definição dos parâmetros de sintonia do programa pretendido, como por exemplo a frequência de canal;

SaveGraph( ); - Permite salvar o grafo de filtros gerado pela aplicação;

RunGraph( ); - Especifica o início de operação, ou seja, determina o início de propagação e respectivo processamento do fluxo de dados pelo grafo de filtros;

Decompose( ); - Utilizada para parar o processamento de dados e libertar todos os recursos utilizados pelo grafo de filtros.

6.5.3 Interface da aplicação

Após a conclusão do núcleo de processamento sobre os fluxos de dados, elaborou-se a aplicação que interage com o grafo de filtros, produzindo deste modo o efeito desejado sobre o sinal DVB-T a receber.

Desenvolveu-se primeiro uma interface que permite ao utilizador definir os parâmetros de sintonia ao programa desejado. Em seguida é apresentada uma imagem ilustrativa:

Figura 46. Interface de definição dos parâmetros de sintonia.

Como podemos observar na figura, o utilizador apenas necessita de introduzir a frequência de sinal ao programa pretendido.

Por fim foi projectada a interface "mãe" da aplicação, onde é permitido ao utilizador efectuar as tarefas pretendidas, como por exemplo dar início à recepção e processamento dos dados DVB-T, produzindo o produto final do propósito da realização desta aplicação. Em seguida é apresentada uma imagem da interface desenvolvida:

Figura 47. Interface "mãe" da aplicação desenvolvida.

Em seguida serão identificadas as opções disponibilizadas ao utilizador nesta interface "mãe" da aplicação desenvolvida.

Tune Request – Permite dar início à recepção de dados, efectuando o pedido à interface de definição de parâmetros acerca da frequência de sinal a utilizar, e posteriormente iniciando então a pretendida recepção de sinal.

Decompose – Determina o final da ligação efectuada, ou seja, finaliza a recepção de dados.

EXIT – Fecha a aplicação.

Save Graph – Implementado mais intenção do estudo realizado, pois não é necessário a sua utilização pelo utilizador comum. Esta função permite o salvaguardar do grafo de filtros produzido pela aplicação, e assim verificar se este se adequa ao esperado.

About – Indica informação adicional acerca da aplicação, em seguida é apresentada uma figura que ilustra o resultado da utilização desta função.

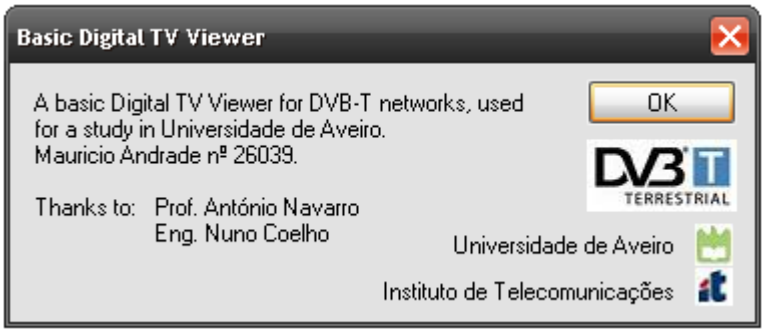


Figura 48. Interface de informação acerca da aplicação.

Capítulo 7

Resultados

Após toda a implementação das ferramentas necessárias à realização do estudo em causa, passou-se à verificação dos resultados obtidos. Inicialmente efectuou-se uma análise aos resultados através da utilização exclusiva dos algoritmos desenvolvidos para o encapsulamento dos dados de protocolo RTP em pacotes *Transport Stream* MPEG-2, e respectivo desencapsulamento de modo a obter novamente o ficheiro com apenas os dados iniciais de protocolo RTP.

Para a realização desta verificação foi utilizado um ficheiro cuja informação consistia apenas em pacotes RTP, e para uma análise mais detalhada acerca dos dados contidos no ficheiro e nos respectivos resultados a analisar foi utilizada uma ferramenta de visualização e analise a ficheiros em hexadecimal, como por exemplo *Hex Workshop* da *BreakPoint Software*.

Em seguida serão apresentadas algumas imagens ilustrativas das análises efectuadas ao ficheiro utilizado para teste e respectivos resultados obtidos.

Na figura seguinte é apresentado o ficheiro utilizado para teste dos algoritmos implementados. É possível identificar no início dos dados o cabeçalho do protocolo RTP que encapsula os dados de vídeo.

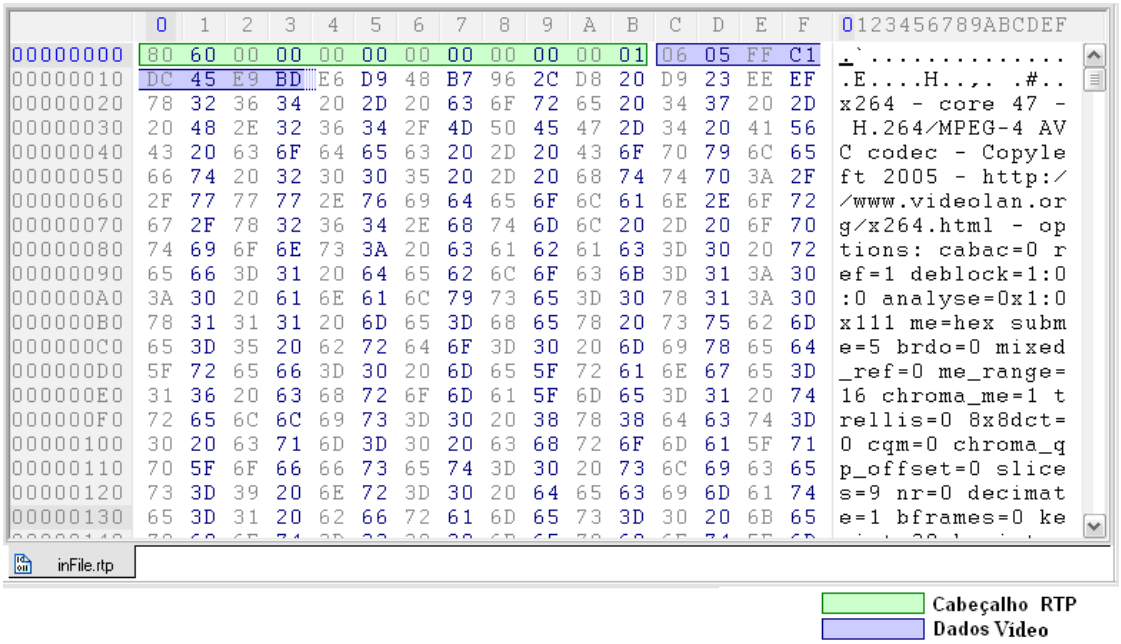


Figura 49. Análise ao ficheiro de teste.

Em seguida é apresentada uma imagem onde podem ser observados os encapsulamentos referidos anteriormente. Assim sendo, podemos verificar os dados de vídeo e áudio encapsulados até serem obtidos os pacotes *Transport Stream* MPEG-2 que irão ser transmitidos no nosso sistema DVB-T. Este ficheiro foi obtido após aplicar ao de teste o algoritmo de encapsulamento implementado.

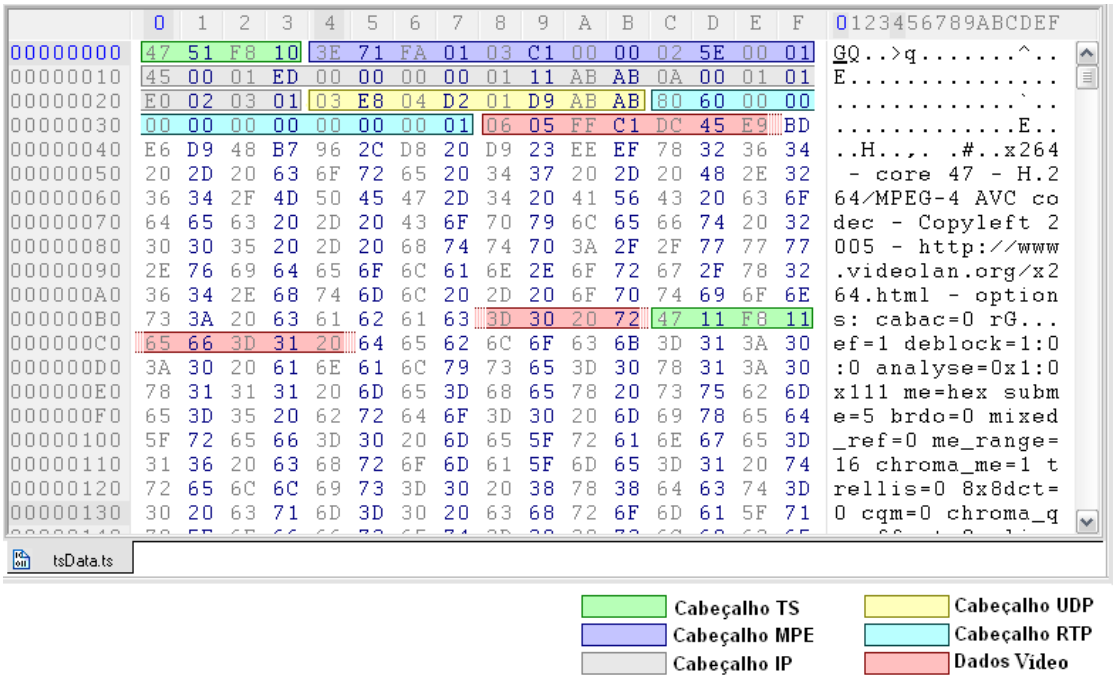


Figura 50. Ficheiro resultante do algoritmo de encapsulamento.

Após ser gerado o ficheiro com exclusivamente pacotes *Transport Stream* MPEG-2, foi inserido num insersor fornecido junto do dispositivo DTA-140 da Dektec instalado num computador no laboratório de vídeo, e efectuou-se uma transmissão via cabo, até uma entrada no mesmo dispositivo, deste modo sendo possível efectuar uma captura dos dados transmitidos. Em seguida é apresentado o resultado dessa experimentação. Podemos verificar que além dos pacotes existentes no ficheiro gerado, o insersor adicionou ao fluxo de dados pacotes nulos, devido à imposição de manter constante uma taxa de transmissão (*bitrate*) numa transmissão DVB-T, como referido anteriormente no capítulo 2.

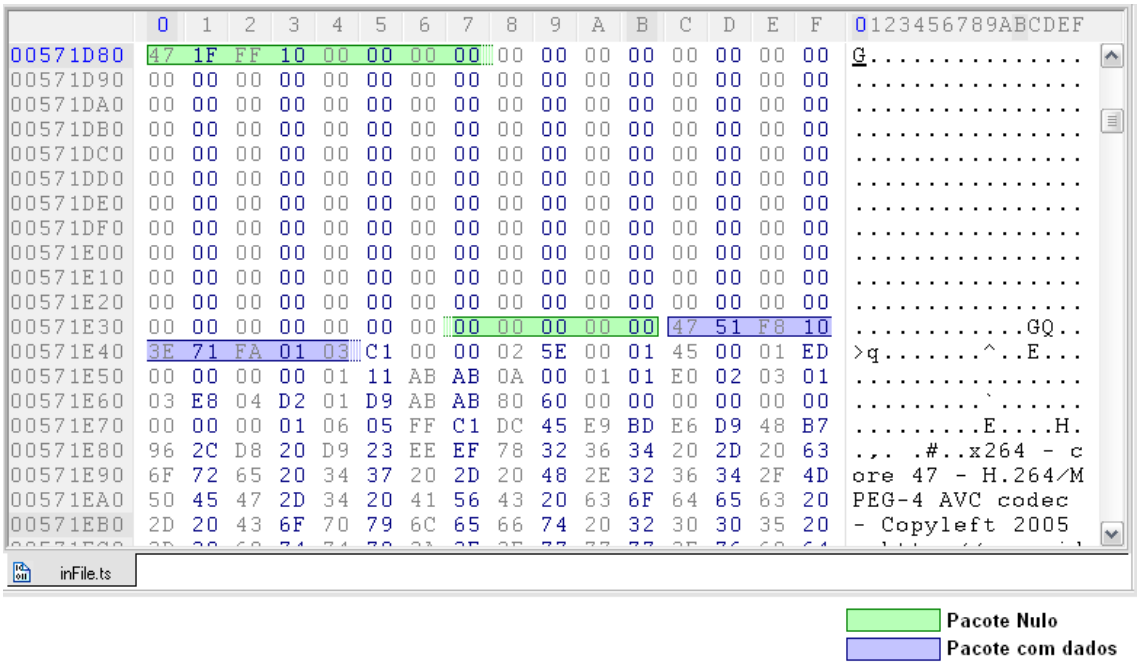


Figura 51. Fluxo de dados emitidos no sistema DVB-T.

Por fim foi aplicado à captura efectuada o algoritmo de desencapsulamento, e analisado o resultado obtido que é apresentado em seguida. Podemos verificar que os dados obtidos correspondem exactamente aos do ficheiro inicial de teste. De modo a facilitar a tarefa de análise ao conteúdo do ficheiro final, foi utilizada a ferramenta de

comparação de ficheiros, disponibilizada na aplicação *Hex Workshop* e verifica-se que de facto os ficheiros são idênticos.

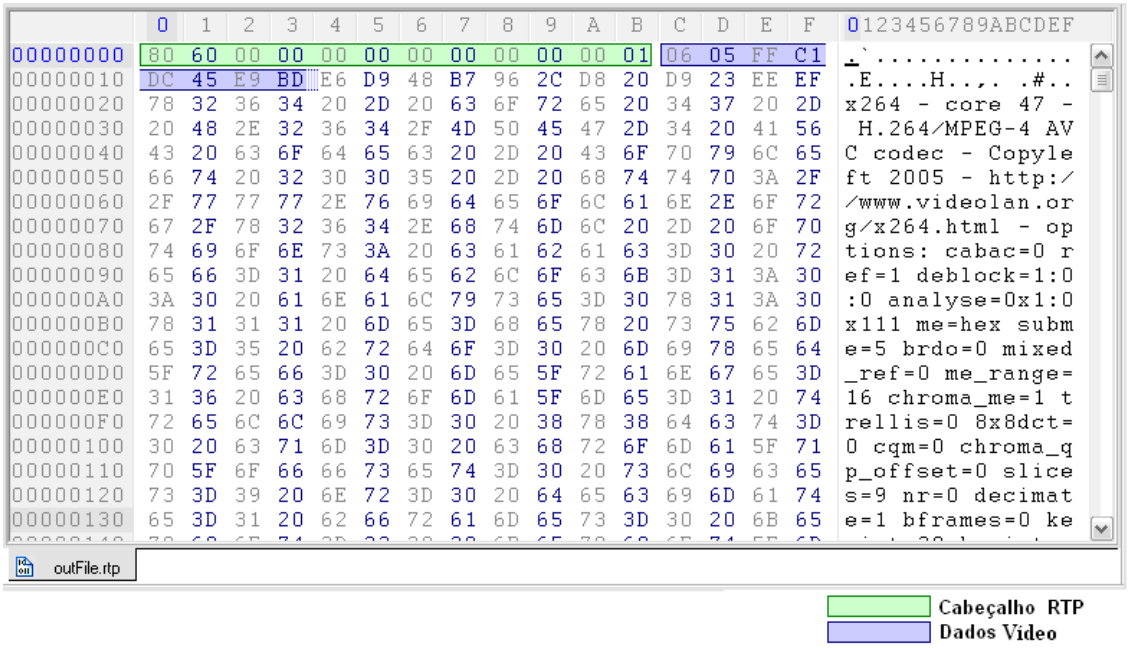


Figura 52. Resultado da aplicação do algoritmo de desencapsulamento.

Após toda esta verificação procedeu-se à experimentação de uma transmissão aérea terrestre, de modo a analisar o sistema DVB-T pretendido. Os resultados obtidos foram análogos aos anteriores, com excepção em alguma perda de dados, de acordo com a qualidade de sinal recebido. Em situações de alta qualidade de sinal recebido, os resultados obtidos são mesmo iguais aos da transmissão via cabo.

Capítulo 8
Conclusão e trabalho futuro

O estudo realizado permite determinar com sucesso a possibilidade de uma transmissão de aulas interactivas através de um sinal de televisão digital terrestre (DVB-T).

Este conceito permite obter alguns benefícios como o acesso remoto dos alunos às referidas aulas, sem se deslocarem ao local onde a aula é assistida presencialmente. Possibilita ainda o acesso aos conteúdos de uma determinada aula num momento posterior, ou seja, uma aula interactiva transmitida em sinal DVB-T, é digitalizada e deste modo pode ser salvaguardada num servidor próprio para o efeito e emitida, satisfazendo um determinado pedido de um utilizador. Além disso este método de ensino pode ser de certo modo motivador, inspirando interesse à aprendizagem pelo facto de se basear num modo tecnológico mais avançado.

Relativamente ao trabalho realizado, deve-se salientar que numa primeira fase, o insersor utilizado para gerar os fluxos de dados a serem emitidos (insersor de IP R&S), possuía um conjunto de características específicas, e surgiram determinados obstáculos de resolução complexa, dificultando a tarefa de implementação do filtro a ser desenvolvido para inclusão na aplicação de recepção ao sinal de televisão digital terrestre. Este facto tornava a realização deste trabalho, além de complexa, muito morosa, pelo que compensou em todos os aspectos alterar de estratégia e optar por desenvolver um algoritmo encapsulador e utilizar o insersor da Dektec, como referido no capítulo 7.

No âmbito do trabalho realizado optou-se apenas por salvaguardar o conteúdo dos dados recebidos e só depois observar o resultado, o que bastaria para verificar a possibilidade de sucesso do estudo em causa, e deste modo simplificar um pouco a tarefa a realizar. Contudo, de modo a obter uma aplicação final mais praticável ao utilizador comum é recomendado a apresentação em tempo real dos dados recebidos, o que possibilita obter o efeito de aula interactiva desejado, utilizando a aplicação desenvolvida neste projecto com núcleo de todo o processamento.

Um outro aspecto que deve ser entendido como uma possibilidade de trabalho futuro é a medição deste sistema DVB-T em situações de recepção diferentes, como por exemplo, medir a qualidade subjacente do vídeo recebido em ambiente móvel, e comparar os resultados com os obtidos e apresentados na publicação "*Mobile HDTV at 140km/h*", *IEEE International Symposium on Consumer Electronics*, Nuno Coelho, Nelson Cabral, David Marques, Antonio Navarro, Vila Moura, Portugal, Abril 2008.

Referências

[1] ETSI EN 300 744, *Digital Video Broadcasting(DVB); Framing structure, channel coding and modulation for digital terrestrial television, June 2004*.

[2] ETSI TS 101 154, *Digital Video Broadcasting(DVB); Implementation guidelines for the use of Video and Audio Coding in Broadcasting Applications based on the MPEG-2 Transport Stream, May 2004.*

[3] ISO/IEC 13818-1, *Information technology – Generic coding of moving pictures and associated áudio information: Systems, December 2000.*

[4] ISO/IEC 13818-2, *Information technology – Generic coding of moving pictures and associated áudio information: Video, September 1995*.

[5] ISO/IEC 13818-3, *Information technology – Generic coding of moving pictures and associated áudio information: Audio, September 1995*.

[6] ISO/IEC 13818-6, *Information technology – Generic coding of moving pictures and associated áudio information: Extensions for DSM-CC, September 1998*.

[7] RFC 1889, *RTP: A Transport Protocol for Real-Time Applications, January 1996*.

[8] RFC 3550, *RTP: A Transport Protocol for Real-Time Applications, July 2003*.

[9] RFC 768, *User Data Protocol, August 1980*.

[10] RFC 760*, DoD Standard Internet Protocol, January 1980.*

[11] RFC 791, *Internet Protocol, Darpa Internet Program, Sptember 1981*.

[12] *http://msdn.microsoft.com*

[13] *Programming Microsoft Directshow For Digital Video And Television, Mark D. Pesce, 2003 by Microsoft Corporation.*

[14] ETSI EN 301192[6], *Digital Video Braodcasting(DVB); Specification for Data Broadcasting, April 2008.*

[15] *http://www.dvb.org/.*

[16] *http://www.geocities.com/intro_to_multimedia/RTP/index.html.*

[17] *http://www.protocolbase.net/protocols/protocol_RTP Video.php.*

[18] *http://www.tcpipguide.com/free/t_IPDatagramGeneralFormat.htm.*

[19] *"Estudo de Caso no Curso de Ciência da Computação/UFPEL: Aulas Remotas Utilizando Streaming de Vídeo e Chat como Ferramenta de Comunicação Interativa", Kelly Hannel, Verônica Burmann da Silva , Raymundo Ferreira Filho, Ricardo Azambuja Silveira , May 2005.*

[20] *http://en.wikipedia.org/wiki/Main_Page.*

[21] *Apontamentos das aulas de Televisão Digital da Universidade de Aveiro.*

[22] *Apontamentos das aulas de Televisão Digital da Universidade do Porto.*

[23] *"Mobile DVB-T using antenna diversity receivers", por Gerard Faria, IBC 2001- September, Amsterdam.*

[24] *"Digital Video Broadcasting – Terrestrial (DVB-T)", Sérgio Mota, Luís Luz e Bruno Henriques, Instituto Superior Técnico – Lisboa.*

[25] *"Simulation and Design of IP over DVB using Multi-Protocol Encapsulation and Ultra Lightweight Encapsulation", Teh Chee Hong, Wan Tat Chee, Rahmat Budiarto - National Computer Science Postgraduate Colloquium 2005 (NaCSPC'05), Penang, Malaysia June 27-28, 2005*

[26] *"Redes de difusão DVB-T", Armindo Machado, Vidal Fernandes – Trabalho de Televisão Digital 2002/2003, Faculdade de Engenharia da Universidade do Porto.*

[27] "*DVB-T OFDM modulation system", József Biró, Endre Borbély*

Anexos

Anexo 1

encapsulador.cpp

```
// encapsulador.cpp : Defines the entry point for the console
application.
//    Mauricio Andrade n 26039

#include "stdafx.h"
#include <stdio.h>
#include <io.h>
#include <fcntl.h>
#include <sys\stat.h>
#include <string.h>


int _tmain(int argc, _TCHAR* argv[])
{
      FILE *inFile, *outFile1, *outFile2, *sizesFile;

      inFile = fopen("inFile.rtp", "rb");
      outFile1 = fopen("mpeData.data", "wb");
      outFile2 = fopen("tsData.ts", "wb");
      sizesFile = fopen("sizes.txt","r");

      if(inFile != NULL)
      {
      if(outFile1 != NULL)
      {
      if(outFile2 != NULL)
      {
      if(sizesFile != NULL)
      {

      //coloca o ponteiro no inicio dos ficheiros
      fseek(inFile, NULL, SEEK_SET);
      fseek(outFile1, NULL, SEEK_SET);
      fseek(sizesFile, NULL, SEEK_SET);

      //MPE
      char mpe1 = 0x3E; //table ID (0x3E)
      char mpe2 = 0x70; //section syntax indicator & private indicator
& reserved (to do OR with lenght)
      char mpe4 = 0x01, mpe5 = 0x03, mpe9 = 0x02, mpe10 = 0x5E, mpe11
= 0x00, mpe12 = 0x01;//mac 01.00.5E.02.03.01 (é invertido)
      char mpe6 = 0xC1; //reserved & payload scrambling control & LLC
SNAP flag & current next indicator
      char mpe7 = 0x00; //section number
      char mpe8 = 0x00; //last section number

      //IP
      char ip1 = 0x45;              //versao do protocolo IP (IPv4)
      char ip2 = 0x00;              //Type of service
      char ip7 = 0x00, ip8 = 0x00;//Flags & Fragment Offset
      char ip9 = 0x01;              //Time To Live
      char ip10 = 0x11;             //Protocol (UDP = 17 = 0x11)
      char ip13 = 0x0A, ip14 = 0x00, ip15 = 0x01, ip16 = 0x01;
      //Source IP Address (10.0.1.1)
```

```
	char ip17 = 0xE0, ip18 = 0x02, ip19 = 0x03, ip20 = 0x01;
	//Destination IP Address (224.2.3.1 - multicast)
	int t = 0;                              //contador para o campo
Identification (0x0000 - 0xFFFF)

	//UDP
	char udp1=0x03, udp2=0xE8;    //source port 1000
	char udp3=0x04, udp4=0xD2;    //dest port 1234

	char ck=0xAB;                           //para os checksum (tudo a
'1')
	int size;         //size do pacote RTP

	int a = 0;        //variavel de controlo para terminar o
encapsulamento
	int count = 0;    //contador para TS

	char buf[2044];
	char *pbuf = buf;

	int valor = 0;

	do{
		//le o valor do size do pacote RTP
		a = fscanf(sizesFile, "%d", &size);
		if(size > 2000)
		{
			FILE *log;
			log = fopen("logFile.txt", "w");
			fprintf(log, "ERROR: \tSize of RTP datagram must be
less than 2000 bytes.\n\tLast packet size is: %d", size);
			fclose(log);
			return 0;
		}

		if(a != EOF)
		{
			//prepara o buffer
			memset(pbuf, 0, sizeof(buf));
			pbuf = buf;

			//preenchimento cabeçalho MPE
			*(pbuf) = mpe1;                 //table id (0x3E)
			*(pbuf + 1) = (((size + 8 + 20 + 13)/256) | mpe2);
	//total lenght + checksum
			*(pbuf + 2) = (size + 8 + 20 + 13);
			*(pbuf + 3) = mpe4;                     //MAC6
			*(pbuf + 4) = mpe5;             //MAC5
			*(pbuf + 5) = mpe6;             //reserved & payload
scrambling control & LLC SNAP flag & current next indicator
			*(pbuf + 6) = mpe7;             //section number
			*(pbuf + 7) = mpe8;             //last section number
			*(pbuf + 8) = mpe9;             //MAC4
			*(pbuf + 9) = mpe10;    //MAC3
			*(pbuf + 10) = mpe11;   //MAC2
			*(pbuf + 11) = mpe12;   //MAC1


			//preenchimento do cabeçalho IP
			*(pbuf + 12) = ip1;                     //versao &
internet header lenght
```

```
                    *(pbuf + 13) = ip2;                 //type of
service
                    *(pbuf + 14) = ((size + 8 + 20)/256);     //total
lenght
                    *(pbuf + 15) = (size + 8 + 20);
                    *(pbuf + 16) = (t/256);       //identification
(neste caso como um contador ate 0xFFFF)
                    *(pbuf + 17) = t;
                    *(pbuf + 18) = ip7;                 //flags &
fragment offset
                    *(pbuf + 19) = ip8;
                    *(pbuf + 20) = ip9;                 //time to live
                    *(pbuf + 21) = ip10;          //Protocol (UDP = 17)
                    *(pbuf + 22) = ck;                  //header
checksum
                    *(pbuf + 23) = ck;
                    *(pbuf + 24) = ip13;          //Source IP Address
                    *(pbuf + 25) = ip14;
                    *(pbuf + 26) = ip15;
                    *(pbuf + 27) = ip16;
                    *(pbuf + 28) = ip17;          //Destination IP
Address
                    *(pbuf + 29) = ip18;
                    *(pbuf + 30) = ip19;
                    *(pbuf + 31) = ip20;


                    //preenchimento do cabeçalho UDP
                    *(pbuf + 32) = udp1;                //source port
                    *(pbuf + 33) = udp2;
                    *(pbuf + 34) = udp3;                //det port
                    *(pbuf + 35) = udp4;
                    *(pbuf + 36) = ((size + 8)/256);//total lenght
                    *(pbuf + 37) = (size + 8);
                    *(pbuf + 38) = ck;                        //checksum
                    *(pbuf + 39) = ck;


                    //preenchimento dos dados RTP
                    for(int i = 0; i < size; i++)
                    {
                          *(pbuf + 40 + i) = getc(inFile);
                    }


                    //preenchimento do Checksum MPE
                    *(pbuf + 40 + size) = ck;
                    *(pbuf + 40 + size + 1) = ck;
                    *(pbuf + 40 + size + 2) = ck;
                    *(pbuf + 40 + size + 3) = ck;
                    fwrite(pbuf, 1, 40 + size + 4, outFile1);


                    //encapsulamento TS

                    //TS
                    char ts1 = 0x47;  //sync byte
                    char ts2 = 0x11;  //first PID bits (to do OR with
Payload unit start indicator)
                    char ts3 = 0xF8;  //last PID bits
```

```
                  char ts4 = 0x00;  //TSC (to do OR with Adaptation
field control and continuity counter)

                  char bufTS[188];              //buffer para colocar
os pacotes TS
                  char *pbufTS = bufTS;         //ponteiro ao buffer
                  int writedSize = 0;
                  int firstTSFlag = 0;
                  int outFlag = 0;

                  do{
                        *(pbufTS) = ts1;  //sync byte

                        if(firstTSFlag == 0)    //unit start = 1
                        {
                              *(pbufTS + 1) = (0x40 | ts2);
                              firstTSFlag = 1;
                        }
                        else                            //unit start = 0
                              *(pbufTS + 1) = ts2;

                        *(pbufTS + 2) = ts3;    //PID

                        if(count == 16)               //contador ate
15
                              count = 0;
                        if((40 + size + 4 - writedSize) >= 184)
      //adaptation 01 (only payload)
                        {
                              *(pbufTS + 3) = (ts4 | 0x10) | count;
                              for(int i = 0; i< 184; i++)
                              {
                                    *(pbufTS + 4 + i) = *(pbuf +
writedSize + i);
                              }
                              if((40 + size + 4 - writedSize) == 184)
                              {
                                    outFlag = 1;
                              }

                              fwrite(pbufTS, 1, 188, outFile2);
                              writedSize += 184;
                              count ++;
                        }
                        else
            //adaptation 11 (adaptation & payload)
                        {
                              *(pbufTS + 3) = (ts4 | 0x30) | count;
                              *(pbufTS + 4) = (184 - (40 + size + 4 -
writedSize)) - 1;
                              *(pbufTS + 5) = 0x00;
                              if(((184 - (40 + size + 4 - writedSize))
- 1) < 0x01)      // necessario mais um ts de modo a haver espaço para
o adaptation field
                              {
                                    *(pbufTS + 4) = 0x01;
                                    //for(int i = 0; i < 6; i++)
                                    //{
                                    //    *(pbufTS + 6 + i) = 0xFF;
                                    //}
                                    for(int i = 0; i < 182; i++)
```

```
                                        {
                                              *(pbufTS + 6 + i) = *(pbuf +
writedSize + i);
                                        }
                                        fwrite(pbufTS, 1, 188, outFile2);
                                        writedSize += 182;
                                        count ++;

                                        //last TS packet
                                        *(pbufTS) = ts1;        //sync
byte
                                        *(pbufTS + 1) = ts2;    //unit
start = 0
                                        *(pbufTS + 2) = ts3;    //last PID
bits
                                        if(count == 16)
      //contador ate 15
                                              count = 0;
                                        *(pbufTS + 3) = (ts4 | 0x30) |
count;
                                        *(pbufTS + 4) = (184 - (40 + size
+ 4 - writedSize)) - 1;
                                        *(pbufTS + 5) = 0x00;
                                        for(int i = 0; i < ((184 - (40 +
size + 4 - writedSize)) - 2); i++)
                                        {
                                              *(pbufTS + 6 + i) = 0xFF;
                                        }
                                        for(int i = 0; i < (40 + size + 4
- writedSize); i++)
                                        {
                                              *(pbufTS + 6 + ((184 - (40 +
size + 4 - writedSize)) - 2) + i) = *(pbuf + writedSize + i);
                                        }
                                  }
                                  else  //espaço para dados e adaptation
field
                                  {
                                        for(int i = 0; i < ((184 - (40 +
size + 4 - writedSize)) - 2); i++)
                                        {
                                              *(pbufTS + 6 + i) = 0xFF;
                                        }
                                        for(int i = 0; i < (40 + size + 4
- writedSize); i++)
                                        {
                                              *(pbufTS + 6 + ((184 - (40 +
size + 4 - writedSize)) - 2) + i) = *(pbuf + writedSize + i);
                                        }
                                  }

                                  fwrite(pbufTS, 1, 188, outFile2);
                                  count ++;
                                  outFlag = 1;
                            }

                      }while(outFlag == 0);

                }
                t++;
          }while(a != EOF);
```

```
        fclose(sizesFile);
        fclose(inFile);
        fclose(outFile1);
        fclose(outFile2);
        }
        }
        }
        }
        return 0;
}
```

Anexo 2

Dump_m4u.cpp

```
//------------------------------------------------------------------------
----------
// File: Dump_m4u.cpp
//
//     Mauricio Andrade nº 26039
//------------------------------------------------------------------------
----------


#include <windows.h>
#include <commdlg.h>
#include <streams.h>
#include <initguid.h>
#include <strsafe.h>


#include "dumpuids_m4u.h"
#include "dump_m4u.h"


//////////////////////////////////////////////////////////////////////////
//////////

unsigned char buf[10000];
unsigned char *pbuf = buf;
unsigned long resto = 0;


////////////////////////////variaveis do
desencapsulador//////////////////////////
unsigned char bufTS[188];           //buffer para TS
unsigned char *pbufTS = bufTS;
unsigned char bufMPE[2044];         //buffer para MPE
unsigned char *pbufMPE = bufMPE;
unsigned char bufRTP[2000];         //buffer para RTP
unsigned char *pbufRTP = bufRTP;

unsigned char c, d, b, mask;


int tsErrorFlag = 0;
int errorCountFlag = 0;
int contMpeFlag = 0;
int nullPacketFlag = 0;
unsigned char tsCount = 0;

long sizeMPE = 0;       //tamanho dos pacotes MPE
long writedSize = 0;    //tamanho já colocado no buffer


//////////////////////////////////////////////////////////////////////////
///////////

// Setup data
```

```
const AMOVIESETUP_MEDIATYPE sudPinTypes =
{
    &MEDIATYPE_Stream,                    // Major type
    &MEDIASUBTYPE_MPEG2_TRANSPORT   // Minor type
};

const AMOVIESETUP_PIN sudPins =
{
    L"Input",                   // Pin string name
    FALSE,                      // Is it rendered
    FALSE,                      // Is it an output
    FALSE,                      // Allowed none
    FALSE,                      // Likewise many
    &CLSID_NULL,                // Connects to filter
    L"Output",                  // Connects to pin
    1,                          // Number of types
    &sudPinTypes                // Pin information
};

const AMOVIESETUP_FILTER sudDump_m4u =
{
    &CLSID_Dump_m4u,                // Filter CLSID
    L"Dump_TS",                   // String name
    MERIT_DO_NOT_USE,           // Filter merit
    1,                          // Number pins
    &sudPins                    // Pin details
};


//
//  Object creation stuff
//
CFactoryTemplate g_Templates[]= {
    L"Dump_TS", &CLSID_Dump_m4u, CDump_m4u::CreateInstance, NULL,
&sudDump_m4u
};
int g_cTemplates = 1;


// Constructor

CDump_m4uFilter::CDump_m4uFilter(CDump_m4u *pDump,
                         LPUNKNOWN pUnk,
                         CCritSec *pLock,
                         HRESULT *phr) :
    CBaseFilter(NAME("CDump_m4uFilter"), pUnk, pLock, CLSID_Dump_m4u),
    m_pDump(pDump)
{
}


//
// GetPin
//
CBasePin * CDump_m4uFilter::GetPin(int n)
{
    if (n == 0) {
        return m_pDump->m_pPin;
    } else {
        return NULL;
    }
```

```
}


//
// GetPinCount
//
int CDump_m4uFilter::GetPinCount()
{
    return 1;
}


//
// Stop
//
// Overriden to close the dump_m4u file
//
STDMETHODIMP CDump_m4uFilter::Stop()
{
    CAutoLock cObjectLock(m_pLock);

    if (m_pDump)
        m_pDump->CloseFile();

    return CBaseFilter::Stop();
}


//
// Pause
//
// Overriden to open the dump_m4u file
//
STDMETHODIMP CDump_m4uFilter::Pause()
{
    CAutoLock cObjectLock(m_pLock);

    if (m_pDump)
    {
        // GraphEdit calls Pause() before calling Stop() for this
filter.
        // If we have encountered a write error (such as disk full),
        // then stopping the graph could cause our log to be deleted
        // (because the current log file handle would be invalid).
        //
        // To preserve the log, don't open/create the log file on
pause
        // if we have previously encountered an error.  The write
error
        // flag gets cleared when setting a new log file name or
        // when restarting the graph with Run().
        if (!m_pDump->m_fWriteError)
        {
            m_pDump->OpenFile();
        }
    }

    return CBaseFilter::Pause();
}
```

```
//
// Run
//
// Overriden to open the dump file
//
STDMETHODIMP CDump_m4uFilter::Run(REFERENCE_TIME tStart)
{
    CAutoLock cObjectLock(m_pLock);

    // Clear the global 'write error' flag that would be set
    // if we had encountered a problem writing the previous dump file.
    // (eg. running out of disk space).
    //
    // Since we are restarting the graph, a new file will be created.
    m_pDump->m_fWriteError = FALSE;

    if (m_pDump)
        m_pDump->OpenFile();

    return CBaseFilter::Run(tStart);
}


//
//  Definition of CDumpInputPin
//
CDump_m4uInputPin::CDump_m4uInputPin(CDump_m4u *pDump,
                             LPUNKNOWN pUnk,
                             CBaseFilter *pFilter,
                             CCritSec *pLock,
                             CCritSec *pReceiveLock,
                             HRESULT *phr) :

    CRenderedInputPin(NAME("CDump_m4uInputPin"),
                  pFilter,                   // Filter
                  pLock,                     // Locking
                  phr,                       // Return code
                  L"Input"),                 // Pin name
    m_pReceiveLock(pReceiveLock),
    m_pDump(pDump),
    m_tLast(0)
{
}


//
// CheckMediaType
//
// Check if the pin can support this specific proposed type and format
//
HRESULT CDump_m4uInputPin::CheckMediaType(const CMediaType *pmt)
{
    if((pmt->majortype == MEDIATYPE_Stream) && (pmt->subtype ==
MEDIASUBTYPE_MPEG2_TRANSPORT))
      {
            return S_OK;
    }
    return S_FALSE;
    //return S_OK;
}
```

```
//
// BreakConnect
//
// Break a connection
//
HRESULT CDump_m4uInputPin::BreakConnect()
{
    if (m_pDump->m_pPosition != NULL) {
        m_pDump->m_pPosition->ForceRefresh();
    }

    return CRenderedInputPin::BreakConnect();
}


//
// ReceiveCanBlock
//
// We don't hold up source threads on Receive
//
STDMETHODIMP CDump_m4uInputPin::ReceiveCanBlock()
{
    return S_FALSE;
}


//
// Receive
//
// Do something with this media sample
//
STDMETHODIMP CDump_m4uInputPin::Receive(IMediaSample *pSample)
{
    CheckPointer(pSample,E_POINTER);

    CAutoLock lock(m_pReceiveLock);
    PBYTE pbData;

    // Has the filter been stopped yet?
    if (m_pDump->m_hFile == INVALID_HANDLE_VALUE) {
        return NOERROR;
    }

    REFERENCE_TIME tStart, tStop;
    pSample->GetTime(&tStart, &tStop);

    DbgLog((LOG_TRACE, 1, TEXT("tStart(%s), tStop(%s), Diff(%d ms),
Bytes(%d)"),
           (LPCTSTR) CDisp(tStart),
           (LPCTSTR) CDisp(tStop),
           (LONG)((tStart - m_tLast) / 10000),
           pSample->GetActualDataLength()));

    m_tLast = tStart;

    // Copy the data to the file

    HRESULT hr = pSample->GetPointer(&pbData);
    if (FAILED(hr)) {
        return hr;
```

```
    }

      ////////////////////////////////////////////////////////////////
//////////////////
      unsigned long totalLength = pSample->GetActualDataLength() +
resto;
      unsigned long t = 0;

      memcpy(pbuf + resto, pbData, pSample->GetActualDataLength());

      if((totalLength / 188) == 0)
      {
            resto = totalLength;
      }
      else
      {
            do{

                  //copia um pacote TS para o bufTS
                  memset(pbufTS, 0, sizeof(bufTS)); //limpa o buffer
de alocaçao do TS
                  pbufTS = bufTS;
                  memcpy(pbufTS, pbuf + (t * 188), 188);

                  c = *(pbufTS);    //verifica o 0x47
                  if(c != 0x47)
                  {
                        FILE *logFile;
                        logFile = fopen("logFile.txt", "w");
                        fprintf(logFile, "ERROR: TS Sync Byte
error.");
                        fclose(logFile);
                        return 0;
                  }
                  c = *(pbufTS + 1);      //le o sugundo e terceiro
byte e analisa
                  d = *(pbufTS + 2);
                  //transport error indicator = 1
                  b = c;
                  mask = 0x80;
                  b = b & mask;
                  if( b == mask )
                        tsErrorFlag = 1;
                  else
                        tsErrorFlag = 0;

                  if(tsErrorFlag == 1)//detectados erros, descarta
toda a informaçao
                  {
                        contMpeFlag = 0;
                        writedSize = 0;
                        errorCountFlag = 0;
                        memset(pbufMPE, 0, sizeof(bufMPE));
                        pbufMPE = bufMPE;
                  }
                  else
                  {
                        if((c == 0x1F) && (d == 0xFF))
      //null packet
                        {
                              //nullPacketFlag = 1;
```

```
                        }
                        else if((c == 0x51) && (d == 0xF8))
      //unit start = 1
                        {
                              c = *(pbufTS + 3);      //le o TSC, AFC
e CC
                              b = c;
                              //verifica o Adaptation field control
                              mask = 0x30;
                              c = c & mask;
                              if(c == mask)     //adaptation field and
payload
                              {
                                    //le o offset aos dados
                                    long offset = *(pbufTS + 4);
                                    if((*(pbufTS + 5 + offset)) !=
0x3E) //verifica o 0x3E
                                    {
                                          FILE *logFile;
                                          logFile =
fopen("logFile.txt", "w");
                                          fprintf(logFile, "ERROR: MPE
Sync Byte error.");
                                          fclose(logFile);
                                          return 0;
                                    }
                                    for(long i = 0; i < (184 - offset
- 1); i++)
                                    {
                                          *(pbufMPE + i) = *(pbufTS +
5 + offset + i);
                                    }
                                    writedSize = (184 - offset - 1);

                                    //le o size
                                    mask = 0x0F;
                                    b = *(pbufTS + offset + 1 + 5);
                                    b = b & mask;
                                    d = *(pbufTS + offset + 1 + 6);
                                    sizeMPE = (b*256) + d + 3;

                                    if(writedSize < sizeMPE) //para
detectar os casos de haver 2 pacotes com adaptation field a completar
o MPE
                                    {
                                          mask = 0x0F;      //inicia o
contador
                                          b = *(pbufTS + 3);
                                          tsCount = b & mask;
                                          contMpeFlag = 1;

                                    }
                                    else
                                    {

                                          for(int i = 0; i <
writedSize - 44; i++)
                                          {
                                                *(pbufRTP + i) =
*(pbufMPE + 40 + i);
                                          }
```

```
                                        m_pDump->Write(pbufRTP,
writedSize - 44);
                                        writedSize = 0;
                                        contMpeFlag = 0;
                                  }
                            }
                            else if(c == 0x10)      //payload only
                            {
                                  mask = 0x0F;
                                  tsCount = b & mask;     //inicia o
contador

                                  if((*(pbufTS + 4)) != 0x3E)
      //verifica o 0x3E
                                  {
                                        FILE *logFile;
                                        logFile =
fopen("logFile.txt", "w");
                                        fprintf(logFile, "ERROR: MPE
Sync Byte error.");
                                        fclose(logFile);
                                        return 0;
                                  }

                                  //le o size
                                  mask = 0x0F;
                                  b = *(pbufTS + 5);
                                  b = b & mask;
                                  d = *(pbufTS + 6);
                                  sizeMPE = (b*256) + d + 3;

                                  for(long i = 0; i < 184; i ++)
                                  {
                                        *(pbufMPE + i) = *(pbufTS +
4 + i);
                                  }
                                  writedSize = 184;
                                  contMpeFlag = 1;
                                  if(writedSize == sizeMPE)
                                  {
                                        for(int i = 0; i <
writedSize - 44; i++)
                                        {
                                              *(pbufRTP + i) =
*(pbufMPE + 40 + i);
                                        }
                                        m_pDump->Write(pbufRTP,
writedSize - 44);
                                        contMpeFlag = 0;
                                        writedSize = 0;
                                  }
                            }
                            else  //erro nos bits de adaptation
field control
                            {
                                  FILE *logFile;
                                  logFile = fopen("logFile.txt",
"w");
                                  fprintf(logFile, "ERROR: Invalid
Adaptation field control bits.");
                                  fclose(logFile);
```

```
                                    return 0;
                              }
                        }
                        else if((c == 0x11) && (d == 0xF8)) //unit
start = 0
                        {
                              if(contMpeFlag == 1)
                              {
                                    //le bits de contador
                                    mask = 0x0F;
                                    c = *(pbufTS + 3);
                                    c = c & mask;

                                    //contador = 0x0F - retorna a zero
                                    if(tsCount == 0x0F)
                                    {
                                          if(c == 0x00)     //OK
                                          {
                                                errorCountFlag = 0;
                                                tsCount = c;

                                          }
                                          else              //count
error
                                                errorCountFlag = 1;

                                    }
                                    else
                                    {
                                          if(c == tsCount + 1)
                                          {
                                                errorCountFlag = 0;
      //OK
                                                tsCount = c;

                                          }
                                          else
                                                errorCountFlag = 1;
      //count error
                                    }

                                    if(errorCountFlag == 1)
      //detectado erro de contador
                                    {

                                          contMpeFlag = 0;
                                          writedSize = 0;
                                          errorCountFlag = 0;
                                          memset(pbufMPE, 0,
sizeof(bufMPE));
                                          pbufMPE = bufMPE;
                                    }
                                    else
                                    {

                                          c = *(pbufTS + 3);      //le
o TSC, AFC e CC
                                          //verifica o Adaptation
field control
                                          mask = 0x30;
                                          c = c & mask;
```

```
                                                if(c == mask)
      //adaptation field and payload
                                                {
                                                      //le o offset aos
dados
                                                      long offset = *(pbufTS
+ 4);
                                                      for(long i = 0; i <
(184 - offset - 1); i++)
                                                      {
                                                            *(pbufMPE +
writedSize + i) = *(pbufTS + 5 + offset + i);
                                                      }
                                                      writedSize += (184 -
offset - 1);

                                                      if(writedSize <
sizeMPE) //para detectar os casos de haver 2 pacotes com adaptation
field a completar o MPE
                                                            contMpeFlag = 1;
                                                      else
                                                      {
                                                            for(int i = 0; i
< writedSize - 44; i++)
                                                            {
                                                                  *(pbufRTP
+ i) = *(pbufMPE + 40 + i);
                                                            }
                                                            m_pDump-
>Write(pbufRTP, writedSize - 44);
                                                            writedSize = 0;
                                                            contMpeFlag = 0;

                                                      }

                                                }
                                                else if(c == 0x10)
      //payload only
                                                {
                                                      for(long i = 0; i <
184; i ++)
                                                      {
                                                            *(pbufMPE +
writedSize + i) = *(pbufTS + 4 + i);
                                                      }
                                                      writedSize += 184;
                                                      contMpeFlag = 1;



                                                      if(writedSize ==
sizeMPE)
                                                      {
                                                            for(int i = 0; i
< writedSize - 44; i++)
                                                            {
                                                                  *(pbufRTP
+ i) = *(pbufMPE + 40 + i);
                                                            }
                                                            m_pDump-
>Write(pbufRTP, writedSize - 44);
                                                            contMpeFlag = 0;
                                                            writedSize = 0;
```

```
                                                }
                                          }

                                          else  //erro nos bits de
adaptation field control
                                          {

                                                FILE *logFile;
                                                logFile =
fopen("logFile.txt", "w");
                                                fprintf(logFile,
"ERROR: Invalid Adaptation field control bits.");
                                                fclose(logFile);
                                                return 0;
                                          }
                                    }
                              }
                        }
                        else
                        {
                              FILE *logFile;
                              logFile = fopen("logFile.txt", "w");
                              fprintf(logFile, "ERROR: Invalid PID.");
                              fclose(logFile);
                              return 0;
                        }
                  }

                  t++;
            }while(t < (totalLength / 188));

            resto = totalLength - ((totalLength / 188) * 188);
            memcpy(pbuf, pbuf + (totalLength - resto), resto);
      }

      ////////////////////////////////////////////////////////////////
//////////////////////

      //fclose(flog);

      //m_pDump->Write(pbuf, pSample->GetActualDataLength());

    return S_OK;
}


//
// WriteStringInfo
//
// Write to the file as text form
//
HRESULT CDump_m4uInputPin::WriteStringInfo(IMediaSample *pSample)
{
    CheckPointer(pSample,E_POINTER);

    TCHAR TempString[256],FileString[256];
    PBYTE pbData;

    // Retrieve the time stamps from this sample

    REFERENCE_TIME tStart, tStop;
```

```
    pSample->GetTime(&tStart, &tStop);
    m_tLast = tStart;

    // Write the sample time stamps out

    HRESULT hr = StringCchPrintf(FileString,256,TEXT("\r\nRenderer
received sample (%dms)\0"),timeGetTime());
    m_pDump->WriteString(FileString);

    hr = StringCchPrintf(FileString,256,TEXT("    Start time
(%s)\0"),(LPCTSTR)CDisp(tStart));
    m_pDump->WriteString(FileString);

    hr = StringCchPrintf(FileString,256,TEXT("    End time
(%s)\0"),(LPCTSTR)CDisp(tStop));
    m_pDump->WriteString(FileString);

    // Display the media times for this sample

    hr = pSample->GetMediaTime(&tStart, &tStop);
    if (hr == NOERROR)
    {
        hr = StringCchPrintf(FileString,256,TEXT("    Start media time
(%s)\0"),(LPCTSTR)CDisp(tStart));
        m_pDump->WriteString(FileString);
        hr = StringCchPrintf(FileString,256,TEXT("    End media time
(%s)\0"),(LPCTSTR)CDisp(tStop));
        m_pDump->WriteString(FileString);
    }

    // Is this a sync point sample

    hr = pSample->IsSyncPoint();
    hr = StringCchPrintf(FileString,256,TEXT("    Sync point
(%d)\0"),(hr == S_OK));
    m_pDump->WriteString(FileString);

    // Is this a preroll sample

    hr = pSample->IsPreroll();
    hr = StringCchPrintf(FileString,256,TEXT("    Preroll (%d)\0"),(hr
== S_OK));
    m_pDump->WriteString(FileString);

    // Is this a discontinuity sample

    hr = pSample->IsDiscontinuity();
    hr = StringCchPrintf(FileString,256,TEXT("    Discontinuity
(%d)\0"),(hr == S_OK));
    m_pDump->WriteString(FileString);

    // Write the actual data length

    LONG DataLength = pSample->GetActualDataLength();
    hr = StringCchPrintf(FileString,256,TEXT("    Actual data length
(%d)\0"),DataLength);
    m_pDump->WriteString(FileString);

    // Does the sample have a type change aboard

    AM_MEDIA_TYPE *pMediaType;
```

```
    pSample->GetMediaType(&pMediaType);
    hr = StringCchPrintf(FileString,256,TEXT("   Type changed
(%d)\0"),
         (pMediaType ? TRUE : FALSE));

    m_pDump->WriteString(FileString);
    DeleteMediaType(pMediaType);

    // Copy the data to the file

    hr = pSample->GetPointer(&pbData);
    if (FAILED(hr)) {
        return hr;
    }

    // Write each complete line out in BYTES_PER_LINES groups

    for (int Loop = 0;Loop < (DataLength / BYTES_PER_LINE);Loop++)
    {
        hr = StringCchPrintf(FileString,256,FIRST_HALF_LINE,
                 pbData[0],pbData[1],pbData[2],
                 pbData[3],pbData[4],pbData[5],pbData[6],
                 pbData[7],pbData[8],pbData[9]);

        hr = StringCchPrintf(TempString,256, SECOND_HALF_LINE,
                 pbData[10],pbData[11],pbData[12],
                 pbData[13],pbData[14],pbData[15],pbData[16],
                 pbData[17],pbData[18],pbData[19]);

        hr = StringCchCat(FileString,256,TempString);
        m_pDump->WriteString(FileString);
        pbData += BYTES_PER_LINE;
    }

    // Write the last few bytes out afterwards

    hr = StringCchPrintf(FileString,256,TEXT("   \0"));
    for (int Loop = 0;Loop < (DataLength % BYTES_PER_LINE);Loop++)
    {
        hr = StringCchPrintf(FileString,256,TEXT("%x
\0"),pbData[Loop]);
        hr = StringCchCat(FileString,256,TempString);
    }

    m_pDump->WriteString(FileString);
    return NOERROR;
}


//
// EndOfStream
//
STDMETHODIMP CDump_m4uInputPin::EndOfStream(void)
{
    CAutoLock lock(m_pReceiveLock);
    return CRenderedInputPin::EndOfStream();

} // EndOfStream


//
```

```
// NewSegment
//
// Called when we are seeked
//
STDMETHODIMP CDump_m4uInputPin::NewSegment(REFERENCE_TIME tStart,
                                           REFERENCE_TIME tStop,
                                           double dRate)
{
    m_tLast = 0;
    return S_OK;

} // NewSegment


//
//  CDump class
//
CDump_m4u::CDump_m4u(LPUNKNOWN pUnk, HRESULT *phr) :
    CUnknown(NAME("CDump_m4u"), pUnk),
    m_pFilter(NULL),
    m_pPin(NULL),
    m_pPosition(NULL),
    m_hFile(INVALID_HANDLE_VALUE),
    m_pFileName(0),
    m_fWriteError(0)
{
    ASSERT(phr);

    m_pFilter = new CDump_m4uFilter(this, GetOwner(), &m_Lock, phr);
    if (m_pFilter == NULL) {
        if (phr)
            *phr = E_OUTOFMEMORY;
        return;
    }

    m_pPin = new CDump_m4uInputPin(this,GetOwner(),
                                   m_pFilter,
                                   &m_Lock,
                                   &m_ReceiveLock,
                                   phr);
    if (m_pPin == NULL) {
        if (phr)
            *phr = E_OUTOFMEMORY;
        return;
    }
}


//
// SetFileName
//
// Implemented for IFileSinkFilter support
//
STDMETHODIMP CDump_m4u::SetFileName(LPCOLESTR pszFileName,const
AM_MEDIA_TYPE *pmt)
{
    // Is this a valid filename supplied

    CheckPointer(pszFileName,E_POINTER);
    if(wcslen(pszFileName) > MAX_PATH)
        return ERROR_FILENAME_EXCED_RANGE;
```

```
    // Take a copy of the filename

    size_t len = 1+lstrlenW(pszFileName);
    m_pFileName = new WCHAR[len];
    if (m_pFileName == 0)
        return E_OUTOFMEMORY;

    HRESULT hr = StringCchCopyW(m_pFileName, len, pszFileName);

    // Clear the global 'write error' flag that would be set
    // if we had encountered a problem writing the previous dump file.
    // (eg. running out of disk space).
    m_fWriteError = FALSE;

    // Create the file then close it

    hr = OpenFile();
    CloseFile();

    return hr;

} // SetFileName


//
// GetCurFile
//
// Implemented for IFileSinkFilter support
//
STDMETHODIMP CDump_m4u::GetCurFile(LPOLESTR *
ppszFileName,AM_MEDIA_TYPE *pmt)
{
    CheckPointer(ppszFileName, E_POINTER);
    *ppszFileName = NULL;

    if (m_pFileName != NULL)
    {
        size_t len = 1+lstrlenW(m_pFileName);
        *ppszFileName = (LPOLESTR)
        QzTaskMemAlloc(sizeof(WCHAR) * (len));

        if (*ppszFileName != NULL)
        {
            HRESULT hr = StringCchCopyW(*ppszFileName, len,
m_pFileName);
        }
    }

    if(pmt)
    {
        ZeroMemory(pmt, sizeof(*pmt));
        pmt->majortype = MEDIATYPE_NULL;
        pmt->subtype = MEDIASUBTYPE_NULL;
    }

    return S_OK;

} // GetCurFile
```

```
// Destructor

CDump_m4u::~CDump_m4u()
{

      CloseFile();
    delete m_pPin;
    delete m_pFilter;
    delete m_pPosition;
    delete m_pFileName;
}


//
// CreateInstance
//
// Provide the way for COM to create a dump filter
//
CUnknown * WINAPI CDump_m4u::CreateInstance(LPUNKNOWN punk, HRESULT
*phr)
{
    ASSERT(phr);

    CDump_m4u *pNewObject = new CDump_m4u(punk, phr);
    if (pNewObject == NULL) {
        if (phr)
            *phr = E_OUTOFMEMORY;
    }

    return pNewObject;

} // CreateInstance


//
// NonDelegatingQueryInterface
//
// Override this to say what interfaces we support where
//
STDMETHODIMP CDump_m4u::NonDelegatingQueryInterface(REFIID riid, void
** ppv)
{
    CheckPointer(ppv,E_POINTER);
    CAutoLock lock(&m_Lock);

    // Do we have this interface

    if (riid == IID_IFileSinkFilter) {
        return GetInterface((IFileSinkFilter *) this, ppv);
    }
    else if (riid == IID_IBaseFilter || riid == IID_IMediaFilter ||
riid == IID_IPersist) {
        return m_pFilter->NonDelegatingQueryInterface(riid, ppv);
    }
    else if (riid == IID_IMediaPosition || riid == IID_IMediaSeeking)
{
        if (m_pPosition == NULL)
        {

            HRESULT hr = S_OK;
```

```
            m_pPosition = new CPosPassThru(NAME("Dump_m4u Pass
Through"),
                                           (IUnknown *) GetOwner(),
                                           (HRESULT *) &hr, m_pPin);
            if (m_pPosition == NULL)
                return E_OUTOFMEMORY;

            if (FAILED(hr))
            {
                delete m_pPosition;
                m_pPosition = NULL;
                return hr;
            }
        }

        return m_pPosition->NonDelegatingQueryInterface(riid, ppv);
    }

    return CUnknown::NonDelegatingQueryInterface(riid, ppv);

} // NonDelegatingQueryInterface


//
// OpenFile
//
// Opens the file ready for dumping
//
HRESULT CDump_m4u::OpenFile()
{
    TCHAR *pFileName = NULL;

    // Is the file already opened
    if (m_hFile != INVALID_HANDLE_VALUE) {
        return NOERROR;
    }

    // Has a filename been set yet
    if (m_pFileName == NULL) {
        return ERROR_INVALID_NAME;
    }

    // Convert the UNICODE filename if necessary

#if defined(WIN32) && !defined(UNICODE)
    char convert[MAX_PATH];

    if(!WideCharToMultiByte(CP_ACP,0,m_pFileName,-
1,convert,MAX_PATH,0,0))
        return ERROR_INVALID_NAME;

    pFileName = convert;
#else
    pFileName = m_pFileName;
#endif

    // Try to open the file

    m_hFile = CreateFile((LPCTSTR) pFileName,    // The filename
                         GENERIC_WRITE,          // File access
                         FILE_SHARE_READ,        // Share access
```

```
                            NULL,                   // Security
                            CREATE_ALWAYS,          // Open flags
                            (DWORD) 0,              // More flags
                            NULL);                  // Template

    if (m_hFile == INVALID_HANDLE_VALUE)
    {
        DWORD dwErr = GetLastError();
        return HRESULT_FROM_WIN32(dwErr);
    }

    return S_OK;

} // Open


//
// CloseFile
//
// Closes any dump file we have opened
//
HRESULT CDump_m4u::CloseFile()
{
    // Must lock this section to prevent problems related to
    // closing the file while still receiving data in Receive()
    CAutoLock lock(&m_Lock);

    if (m_hFile == INVALID_HANDLE_VALUE) {
        return NOERROR;
    }

    CloseHandle(m_hFile);
    m_hFile = INVALID_HANDLE_VALUE; // Invalidate the file

    return NOERROR;

} // Open


//
// Write
//
// Write raw data to the file
//
HRESULT CDump_m4u::Write(PBYTE pbData, LONG lDataLength)
{
    DWORD dwWritten;

    // If the file has already been closed, don't continue
    if (m_hFile == INVALID_HANDLE_VALUE) {
        return S_FALSE;
    }

    if (!WriteFile(m_hFile, (PVOID)pbData, (DWORD)lDataLength,
                    &dwWritten, NULL))
    {
        return (HandleWriteFailure());
    }

    return S_OK;
}
```

```
HRESULT CDump_m4u::HandleWriteFailure(void)
{
    DWORD dwErr = GetLastError();

    if (dwErr == ERROR_DISK_FULL)
    {
        // Close the dump file and stop the filter,
        // which will prevent further write attempts
        m_pFilter->Stop();

        // Set a global flag to prevent accidental deletion of the
dump file
        m_fWriteError = TRUE;

        // Display a message box to inform the developer of the write
failure
        TCHAR szMsg[MAX_PATH + 80];
        HRESULT hr = StringCchPrintf(szMsg, MAX_PATH + 80, TEXT("The
disk containing dump file has run out of space, ")
                  TEXT("so the dump filter has been stopped.\r\n\r\n")
                  TEXT("You must set a new dump file name or restart
the graph ")
                  TEXT("to clear this filter error."));
        MessageBox(NULL, szMsg, TEXT("Dump Filter failure"),
MB_ICONEXCLAMATION);
    }

    return HRESULT_FROM_WIN32(dwErr);
}

//
// WriteString
//
// Writes the given string into the file
//
void CDump_m4u::WriteString(TCHAR *pString)
{
    ASSERT(pString);

    // If the file has already been closed, don't continue
    if (m_hFile == INVALID_HANDLE_VALUE) {
        return;
    }

    BOOL bSuccess;
    DWORD dwWritten = 0;
    DWORD dwToWrite = lstrlen(pString);

    // Write the requested data to the dump file
    bSuccess = WriteFile((HANDLE) m_hFile,
                         (PVOID) pString, (DWORD) dwToWrite,
                         &dwWritten, NULL);

    if (bSuccess == TRUE)
    {
        // Append a carriage-return and newline to the file
        const TCHAR *pEndOfLine = TEXT("\r\n\0");
        dwWritten = 0;
        dwToWrite = lstrlen(pEndOfLine);
```

```
        bSuccess = WriteFile((HANDLE) m_hFile,
                             (PVOID) pEndOfLine, (DWORD) dwToWrite,
                             &dwWritten, NULL);
    }

    // If either of the writes failed, stop receiving data
    if (!bSuccess || (dwWritten < dwToWrite))
    {
        HandleWriteFailure();
    }

} // WriteString


////////////////////////////////////////////////////////////////////////
//
//
// Exported entry points for registration and unregistration
// (in this case they only call through to default implementations).
//
////////////////////////////////////////////////////////////////////////
//

//
// DllRegisterSever
//
// Handle the registration of this filter
//
STDAPI DllRegisterServer()
{
    return AMovieDllRegisterServer2( TRUE );

} // DllRegisterServer


//
// DllUnregisterServer
//
STDAPI DllUnregisterServer()
{
    return AMovieDllRegisterServer2( FALSE );

} // DllUnregisterServer


//
// DllEntryPoint
//
extern "C" BOOL WINAPI DllEntryPoint(HINSTANCE, ULONG, LPVOID);

BOOL APIENTRY DllMain(HANDLE hModule,
                      DWORD  dwReason,
                      LPVOID lpReserved)
{
      return DllEntryPoint((HINSTANCE)(hModule), dwReason,
lpReserved);
}
```

Anexo 3

FilterGraphTools.cs

```
using System;
using System.Collections;
using System.Runtime.InteropServices;
using System.Security.Permissions;

using DirectShowLib;

#if !USING_NET11
using System.Runtime.InteropServices.ComTypes;
#endif

namespace DirectShowLib.Utils
{
    public sealed class FilterGraphTools
    {
        private FilterGraphTools(){}

        /// <summary>
        /// Adiciona um filtro atraves do seu identificador
        /// </summary>

        [SecurityPermission(SecurityAction.LinkDemand,
UnmanagedCode=true)]
        public static IBaseFilter AddFilterFromClsid(IGraphBuilder
graphBuilder, Guid clsid, string name)
        {
            int hr = 0;
            IBaseFilter filter = null;

            if (graphBuilder == null)
                throw new ArgumentNullException("graphBuilder");

            try
            {
                Type type = Type.GetTypeFromCLSID(clsid);
                filter = (IBaseFilter) Activator.CreateInstance(type);

                hr = graphBuilder.AddFilter(filter, name);
                DsError.ThrowExceptionForHR(hr);
            }
            catch
            {
                if (filter != null)
                {
                    Marshal.ReleaseComObject(filter);
                    filter = null;
                }
            }

            return filter;
        }
```

```
        /// <summary>
        /// Remove e liberta todos os filtros utilizados
        /// </summary>


        [SecurityPermission(SecurityAction.LinkDemand,
UnmanagedCode=true)]
        public static void RemoveAllFilters(IGraphBuilder
graphBuilder)
        {
            int hr = 0;
            IEnumFilters enumFilters;
            ArrayList filtersArray = new ArrayList();

            if (graphBuilder == null)
                throw new ArgumentNullException("graphBuilder");

            hr = graphBuilder.EnumFilters(out enumFilters);
            DsError.ThrowExceptionForHR(hr);

            try
            {
                IBaseFilter[] filters = new IBaseFilter[1];

                while (enumFilters.Next(filters.Length, filters,
IntPtr.Zero) == 0)
                {
                    filtersArray.Add(filters[0]);
                }
            }
            finally
            {
                Marshal.ReleaseComObject(enumFilters);
            }

            foreach(IBaseFilter filter in filtersArray)
            {
                hr = graphBuilder.RemoveFilter(filter);
                Marshal.ReleaseComObject(filter);
            }
        }

        /// <summary>
        /// Salva o grafo de filtros gerado
        /// </summary>

        [SecurityPermission(SecurityAction.LinkDemand,
UnmanagedCode=true)]
        public static void SaveGraphFile(IGraphBuilder graphBuilder,
string fileName)
        {
            int hr = 0;
            IStorage storage = null;
#if USING_NET11
            UCOMIStream stream = null;
#else
            IStream stream = null;
#endif

            if (graphBuilder == null)
                throw new ArgumentNullException("graphBuilder");
```

```
            try
            {
                hr = NativeMethods.StgCreateDocfile(
                    fileName,
                    STGM.Create | STGM.Transacted | STGM.ReadWrite |
STGM.ShareExclusive,
                    0,
                    out storage
                    );

                Marshal.ThrowExceptionForHR(hr);

                hr = storage.CreateStream(
                    @"ActiveMovieGraph",
                    STGM.Write | STGM.Create | STGM.ShareExclusive,
                    0,
                    0,
                    out stream
                    );

                Marshal.ThrowExceptionForHR(hr);

                hr = (graphBuilder as IPersistStream).Save(stream,
true);
                Marshal.ThrowExceptionForHR(hr);

                hr = storage.Commit(STGC.Default);
                Marshal.ThrowExceptionForHR(hr);
            }
            finally
            {
                if (stream != null)
                    Marshal.ReleaseComObject(stream);
                if (storage != null)
                    Marshal.ReleaseComObject(storage);
            }
        }

        /// <summary>
        /// Verifica se um determinado objecto COM está
disponibilizado
        /// </summary>

        [SecurityPermission(SecurityAction.LinkDemand,
UnmanagedCode=true)]
        public static bool IsThisComObjectInstalled(Guid clsid)
        {
            bool retval = false;

            try
            {
                Type type = Type.GetTypeFromCLSID(clsid);
                object o = Activator.CreateInstance(type);
                retval = true;
                Marshal.ReleaseComObject(o);
            }
            catch{}

            return retval;}
```

Anexo 4

BDAGraphBuilder.cs

```
using System;
using System.Runtime.InteropServices;
using System.Windows.Forms;

using Microsoft.Win32;

using DirectShowLib;
using DirectShowLib.BDA;
using DirectShowLib.Utils;

namespace DirectShowLib.Sample
{
      public class BDAGraphBuilder : IDisposable
      {
    Control hostingControl = null;

    IFilterGraph2 graphBuilder = null;
    DsROTEntry rot = null;

    IBaseFilter networkProvider = null;
    IBaseFilter mpeg2Demux = null;
    IBaseFilter tuner = null;
    IBaseFilter demodulator = null;
    IBaseFilter capture = null;


    IBaseFilter customFilter = null;

    Guid CLSID_customFilter = new Guid {FE922858-E8FC-4df0-9907-
85846C0E859C}");

    IFileSinkFilter fs1 = null;

    IMpeg2Demultiplexer imux = null;


      public BDAGraphBuilder(Control renderingControl)
      {
      this.hostingControl = renderingControl;
      }

    public void BuildGraph(ITuningSpace tuningSpace)
    {
      this.graphBuilder = (IFilterGraph2) new FilterGraph();
      rot = new DsROTEntry(this.graphBuilder);

      // Method names should be self explanatory
      AddNetworkProviderFilter(tuningSpace);
      AddMPEG2DemuxFilter();
      AddAndConnectBDABoardFilters();
      AddAndConnectCustomFilter();
    }

    public void SubmitTuneRequest(ITuneRequest tuneRequest)
```

```
    {
      int hr = 0;

      hr = (this.networkProvider as
ITuner).put_TuneRequest(tuneRequest);
      DsError.ThrowExceptionForHR(hr);
    }

    public void RunGraph()
    {
      int hr = 0;

      hr = (this.graphBuilder as IMediaControl).Run();
      DsError.ThrowExceptionForHR(hr);
    }

    public void SaveGraph(string filepath)
    {
      // Nothing to do with a DTV viewer but can be useful
      FilterGraphTools.SaveGraphFile(this.graphBuilder, filepath);
    }


    private void AddNetworkProviderFilter(ITuningSpace tuningSpace)
    {
      int hr = 0;
      Guid genProviderClsId = new Guid("{B2F3A67C-29DA-4C78-8831-
091ED509A475}");
      Guid networkProviderClsId;

      // First test if the Generic Network Provider is available (only
on MCE 2005 + Update Rollup 2)
      if (FilterGraphTools.IsThisComObjectInstalled(genProviderClsId))
      {
        this.networkProvider =
FilterGraphTools.AddFilterFromClsid(this.graphBuilder,
genProviderClsId, "Generic Network Provider");

        hr = (this.networkProvider as
ITuner).put_TuningSpace(tuningSpace);
        return;
      }

      // Get the network type of the requested Tuning Space
      hr = tuningSpace.get__NetworkType(out networkProviderClsId);

      // Get the network type of the requested Tuning Space
      if (networkProviderClsId == typeof(DVBTNetworkProvider).GUID)
      {
        this.networkProvider =
FilterGraphTools.AddFilterFromClsid(this.graphBuilder,
networkProviderClsId, "DVBT Network Provider");
      }
      else
        throw new ArgumentException("This application doesn't support
this Tuning Space");

      hr = (this.networkProvider as
ITuner).put_TuningSpace(tuningSpace);
    }
```

```
    private void AddMPEG2DemuxFilter()
    {
        int hr = 0;

        this.mpeg2Demux = (IBaseFilter)new MPEG2Demultiplexer();
        hr = this.graphBuilder.AddFilter(this.mpeg2Demux, "MPEG-2
Demultiplexer");
        DsError.ThrowExceptionForHR(hr);

        this.imux = (IMpeg2Demultiplexer)this.mpeg2Demux;
        //create pinOut for PID 120
        IPin pinOut1;
        AMMediaType mt1 = new AMMediaType();
        mt1.majorType = new Guid("{E436EB83-524F-11CE-9F53-
0020AF0BA770}"); // MEDIATYPE_Stream
        mt1.subType = new Guid("{e06d8023-db46-11cf-b4d1-
00805f6cbbea}"); //MEDIASUBTYPE_MPEG2_TRANSPORT
        mt1.sampleSize = 188;
        hr = this.imux.CreateOutputPin(mt1, "pid 0x11F8", out
pinOut1);
        DsError.ThrowExceptionForHR(hr);

        IMPEG2PIDMap map1 = (IMPEG2PIDMap)pinOut1;
        int[] PID1 = { 0x11F8 };
        map1.MapPID(1, PID1, MediaSampleContent.TransportPacket);

        Marshal.ReleaseComObject(pinOut1);
        Marshal.ReleaseComObject(map1);
    }

    private void AddAndConnectCustomFilter()
    {
        int hr = 0;

        this.customFilter =
FilterGraphTools.AddFilterFromClsid(this.graphBuilder,
CLSID_customFilter, "Custom Filter");

        hr = this.graphBuilder.AddFilter(this.customFilter, " Custom
Filter ");
        DsError.ThrowExceptionForHR(hr);

        this.fs1 = (IFileSinkFilter)this.dump1;
        this.fs1.SetFileName("/data.dat", null);

        IPin pinOut, pinIn;
        pinOut = DsFindPin.ByName(this.mpeg2Demux, "pid 0x11F8");
        pinIn = DsFindPin.ByName(this.customFilter, "Input");
        this.graphBuilder.Connect(pinOut, pinIn);

        Marshal.ReleaseComObject(pinOut);
        Marshal.ReleaseComObject(pinIn);
    }



    private void AddAndConnectBDABoardFilters()
    {
      int hr = 0;
      DsDevice[] devices;
```

```
      ICaptureGraphBuilder2 capBuilder = (ICaptureGraphBuilder2) new
CaptureGraphBuilder2();
      capBuilder.SetFiltergraph(this.graphBuilder);

      try
      {
        // Enumerate BDA Source filters category and found one that
can connect to the network provider
        devices =
DsDevice.GetDevicesOfCat(FilterCategory.BDASourceFiltersCategory);
        for(int i = 0; i < devices.Length; i++)
        {
          IBaseFilter tmp;

          hr = graphBuilder.AddSourceFilterForMoniker(devices[i].Mon,
null, devices[i].Name, out tmp);
          DsError.ThrowExceptionForHR(hr);

          hr = capBuilder.RenderStream(null, null,
this.networkProvider, null, tmp);
          if (hr == 0)
          {
            // Got it !
            this.tuner = tmp;
            break;
          }
          else
          {
            // Try another...
            hr = graphBuilder.RemoveFilter(tmp);
            Marshal.ReleaseComObject(tmp);
          }
        }

        if (this.tuner == null)
          throw new ApplicationException("Can't find a valid BDA
tuner");

      hr = capBuilder.RenderStream(null, null, tuner, null, infTee1);
      DsError.ThrowExceptionForHR(hr);
      hr = capBuilder.RenderStream(null, null, infTee1, null, dump1);
      DsError.ThrowExceptionForHR(hr);

        // trying to connect this filter to the MPEG-2 Demux
      hr = capBuilder.RenderStream(null, null, infTee1, null,
mpeg2Demux);
        if (hr >= 0)
        {
          // this is a one filter model
          this.demodulator = null;
          this.capture = null;
          return;
        }
        else
        {
          // Then enumerate BDA Receiver Components category to found
a filter connecting
          // to the tuner and the MPEG2 Demux
          devices =
DsDevice.GetDevicesOfCat(FilterCategory.BDAReceiverComponentsCategory)
;
```

```
          for (int i = 0; i < devices.Length; i++)
          {
            IBaseFilter tmp;

            hr =
graphBuilder.AddSourceFilterForMoniker(devices[i].Mon, null,
devices[i].Name, out tmp);
            DsError.ThrowExceptionForHR(hr);

            hr = capBuilder.RenderStream(null, null, this.tuner, null,
tmp);
            if (hr == 0)
            {
              // Got it !
              this.capture = tmp;

              // Connect it to the MPEG-2 Demux
              hr = capBuilder.RenderStream(null, null, this.capture,
null, this.mpeg2Demux);
              if (hr >= 0)
              {
                // This second filter connect both with the tuner and
the demux.
                // This is a capture filter...
                return;
              }
              else
              {
                // This second filter connect with the tuner but not
with the demux.
                // This is in fact a demodulator filter. We now must
find the true capture filter...

                this.demodulator = this.capture;
                this.capture = null;

                // saving the Demodulator's DevicePath to avoid
creating it twice.
                string demodulatorDevicePath = devices[i].DevicePath;

                for (int j = 0; i < devices.Length; j++)
                {
                  if
(devices[j].DevicePath.Equals(demodulatorDevicePath))
                    continue;

                  hr =
graphBuilder.AddSourceFilterForMoniker(devices[i].Mon, null,
devices[i].Name, out tmp);
                  DsError.ThrowExceptionForHR(hr);

                  hr = capBuilder.RenderStream(null, null,
this.demodulator, null, tmp);
                  if (hr == 0)
                  {
                    // Got it !
                    this.capture = tmp;

                    // Connect it to the MPEG-2 Demux
```

```
                    hr = capBuilder.RenderStream(null, null,
this.capture, null, this.mpeg2Demux);
                    if (hr >= 0)
                    {
                      // This second filter connect both with the
demodulator and the demux.
                      // This is a true capture filter...
                      return;
                    }
                  }
                  else
                  {
                    // Try another...
                    hr = graphBuilder.RemoveFilter(tmp);
                    Marshal.ReleaseComObject(tmp);
                  }
                } // for j

                // We have a tuner and a capture/demodulator that
don't connect with the demux
                // and we found no additionals filters to build a
working filters chain.
                throw new ApplicationException("Can't find a valid BDA
filter chain");
              }
            }
            else
            {
              // Try another...
              hr = graphBuilder.RemoveFilter(tmp);
              Marshal.ReleaseComObject(tmp);
            }
          } // for i

          // We have a tuner that connect to the Network Provider BUT
not with the demux
          // and we found no additionals filters to build a working
filters chain.
          throw new ApplicationException("Can't find a valid BDA
filter chain");
        }
      }
      finally
      {
        Marshal.ReleaseComObject(capBuilder);
      }
    }

      public void Decompose()
    {
      int hr = 0;

      // Decompose the graph
      hr = (this.graphBuilder as IMediaControl).StopWhenReady();
      hr = (this.graphBuilder as IMediaControl).Stop();

      RemoveHandlers();

      FilterGraphTools.RemoveAllFilters(this.graphBuilder);
```

```
      Marshal.ReleaseComObject(this.networkProvider);
this.networkProvider = null;
      Marshal.ReleaseComObject(this.mpeg2Demux); this.mpeg2Demux =
null;
      Marshal.ReleaseComObject(this.tuner); this.tuner = null;
      //Marshal.ReleaseComObject(this.capture); this.capture = null;
      Marshal.ReleaseComObject(this.bdaTIF); this.bdaTIF = null;
      Marshal.ReleaseComObject(this.bdaSecTab); this.bdaSecTab = null;
      Marshal.ReleaseComObject(this.audioRenderer); this.audioRenderer
= null;
      Marshal.ReleaseComObject(this.videoRenderer); this.videoRenderer
= null;

      rot.Dispose();
      Marshal.ReleaseComObject(this.graphBuilder); this.graphBuilder =
null;
    }
```