Sistema de Videotelefonia para Windows suportado em comunicações RDIS*

João Paulo N. Firmeza, Valter José G. Bouça, Fernando M. S. Ramos, Osvaldo A. Santos

Resumo- **Um sistema de video telefonia para a plataforma MS-Windows é uma aplicação de *software* que nos permite estabelecer uma comunicação de voz com outra pessoa e simultaneamente visualizar a sua imagem. Esta é uma das novas aplicações que é possível realizar utilizando as facilidades da RDIS (Rede Digital com Integração de Serviços). Neste Sistema foram utilizadas placas RDIS e de aquisição/codificação de vídeo disponiveis para o PC.**

Abstract- **A Video Phone application for the MS-Windows platform is a package of software that allow us to talk to another person while viewing his image. This is one of the new applications that can be done by using the facilities of the ISDN (Integrated Services Digital Network). Using ISDN and Video Capture cards currently available for PC *'Video Telephony for Windows'* gained form.**

I. INTRODUÇÃO

1.1 Um sistema de Vídeo Telefonia para PC... Porquê?

Hoje em dia cada vez mais se consegue aliar um computador pessoal a sistemas de telecomunicações avançadas. Com o aumento da capacidade de processamento dos PC's actuais, e com o surgimento de bons periféricos de telecomunicações e codificação de dados, já é possível criar sistemas de elevado desempenho sem necessidade de utilização de *hardware* dedicado. Para complementar esta ideia surge ainda a possibilidade de utilizarmos em qualquer sistema deste tipo uma interface de interacção perfeitamente standardizada, como é o caso do Microsoft Windows. Com a utilização deste ambiente gráfico, muitas das operações tornam-se intuitivas para quem trabalha regularmente com PCs.

Foi com base nestas ideias que surgiu o *Sistema de VideoTelefonia para Windows suportado em RDIS*. Pretendeu-se com ele criar uma aplicação de *software* que formasse ao mesmo tempo um programa robusto e eficiente cujos módulos fossem facilmente integráveis em outros programas, permitindo exportar muito do código para outras aplicações.

Fazendo uma descrição mais pormenorizada podemos dizer que o *VideoTelefonia para Windows* é uma aplicação que permite o diálogo entre dois utilizadores, em que é possível em simultâneo com a conversação visualizar a imagem dos interlocutores e efectuar transferência de dados (ficheiros ou mensagens). Existe também uma base de dados que faz a gestão de uma agenda telefónica semelhante a uma agenda convencional. A ideia consiste em automatizar o processo de estabelecimento de chamadas, podendo-se inclusivamente criar *queues* de chamadas. Além disso é possível ter documentos relacionados com determinada ficha, aos quais se pode aceder em qualquer altura. Todas as comunicações no *VideoTelefonia* operam sobre uma linha RDIS em acesso básico (2B+D).

1.2 Utilização da RDIS

surgimento da RDIS em Portugal permite fornecer vários novos serviços aos utilizadores pois, mesmo em acesso básico, é possível associar as particularidades das ligações telefónicas com ligações de dados semelhantes às permitidas aos utilizadores de MODEMs, mas dispondo de uma largura de banda bastante superior. Com isto é agora possível satisfazer o velho desejo do utilizador clássico do telefone de poder ver a imagem do seu interlocutor durante a conversação. Claro que isso já era possível se utilizássemos simples MODEM's, mas aí teríamos um grave problema relacionado com a largura de banda disponível numa linha telefónica convencional. A RDIS é, portanto, dentro dos modernos serviços de telecomunicações aquele que melhor se adequa a uma aplicação de vídeotelefonia: custos de comunicação reduzidos, 2 canais para transmissão de informação dispondo de boa largura de banda (ideal para transportar audio num e vídeo no outro), possibilidade de usufruir de serviços extra (custos *on-line*, etc.)

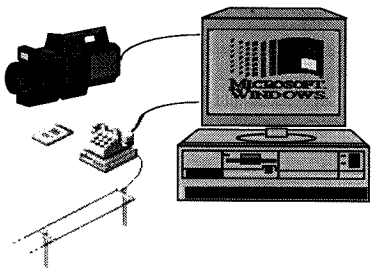

Fig. 1: Um sistema de Video telefoina para PC

* Trabalho realizado no âmbito da disciplina de Projecto.

II. CONSIDERAÇÕES SOBRE O *HARDWARE* UTILIZADO

2.1 Introdução

Os equipamentos onde a aplicação irá correr, deverão possuir os requisitos necessários, ou seja:

⇒ Computador PC compatível, com os seguintes requisitos mínimos:
- CPU 486 DX2/66Mhz (Pentium recomendado)
- 8Mb de RAM
- Placa gráfica aceleradora de elevado desempenho[1] capaz de suportar resoluções iguais ou superiores a 800x600 com profundidade de cores de 16bits (65536 cores).
- Microsoft Windows versão 3.1 ou superior, configurado para um modo gráfico que suporte 64Kcores no mínimo, e resolução superior ou igual a 800x600. Compatível com o Windows95 em MS-DOS *compability mode*[2].

⇒ Placa RDIS PCBIT, versão voz/dados.
⇒ placas de aquisição/codificação/descodificação de vídeo C30.
⇒ Placa *multiplexer* para *interface* camaras-placa C30
⇒ Telefone RDIS
⇒ Câmara de vídeo a cores com saídas RGB+Sync ou câmara a preto-e-branco

2.2 A Placa PCBIT

A placa PCBIT é uma placa RDIS para PC que utiliza a interface ISA, permitindo a aplicações de *software* operando sobre ambientes DOS ou Windows o acesso à rede RDIS (acesso básico 2B+D). Juntamente com a placa, foram criados um conjunto de rotinas (*Application Program Interface - API*) para lhe ser possível aceder de uma forma normalizada, e que torna as aplicações a desenvolver independentes da placa RDIS utilizada.

2.2.1 Descrição/configuração da PCBIT

A PCBIT é constituída por 2 processadores um dos quais controla as ligações de voz, enquanto o outro controla as ligações de dados. Quanto a ligações externas, existem 2 *sockets* para esse efeito. Um deles permite ligar um telefone à placa, enquanto o outro permite efectuar a ligação à RDIS, através de um NT (*Network Terminator*).

Configuração da PCBIT

Antes de instalar a PCBIT num *slot* livre do PC, é necessário configurá-la. Os parâmetros configuráveis são o IRQ, o *interrupt de software* e a zona de memória partilhada com o PC. As configurações recomendadas são: IRQ 5 ou 7, zona de memória EFC0, DFC0 ou D400 e obrigatoriamente *interrupt de software* 80.

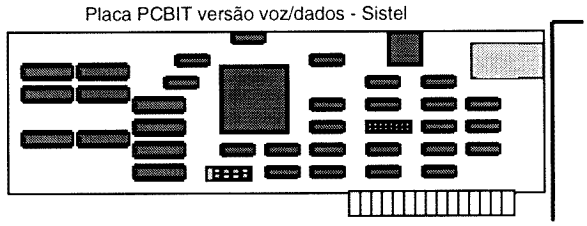

Fig. 2: A Placa PCBIT

Relativamente á zona de memória é necessário ter um certo cuidado, pois esta pode estar a ser usada por outra placa, ou por um gestor de memória do DOS (usualmente o EMM386). Caso isto aconteça, deve-se excluir do gestor de memória a área correspondente aquela que vamos utilizar para a PCBIT. Para assegurarmos que não existem problemas podemos utilizar um dos muitos utilitários DOS que permitem analisar o estado da memória alta do PC.

A configuração dos parâmetros da PCBIT além de ser feita através dos *jumpers* da placa, tem também de ser feita no ficheiro de arranque do DOS: o *autoexec.bat*. O IRQ é definido através do parâmetro -I, a zona de memória através do parâmetro -A e o *interrupt de software* através do parâmetro -S. Por exemplo

SET PCBIT=C:\PCBIT\BIN -I7 -ADFC00 -S80

configura a PCBIT para o IRQ 7, zona de memória DFC0, e *interrupt de software* 80.

2.3 As Placas de processamento de imagem TMS320C30

2.3.1 Introdução

As placas de processamento de imagem utilizadas no *Video Telefonia para Windows são* baseadas no DSP TMS320C30 da *Texas Instruments, e foram* desenvolvidas no INESC-Porto.

O TMS320C30 possui um espaço de endereçamento de 16 Mwords. Cada 'word' ou palavra de dados possui 32 bits.

De acordo com as tarefas a executar por este sistema, é necessário que se organize a memória em duas zonas distintas. Uma destas zonas é destinada ao armazenamento de imagens. Outra zona é reservada para as rotinas a serem executadas.

Como o C30 opera com largura de dados de 32 bits, a memória reservada a programas e respectivos dados terá que ser de 32 bits. Assim a zona de memória da placa

[1] Placas com BUS PCI ou VESA Local Bus altamente recomendáveis.

[2] Este é um modo que o Windows95 possui para assegurar compatibilidade com aplicações DOS que operam em modo real, como é o caso do *device driver* da PCBIT: ISDNBIOS. Este *driver* obriga o Windows95 a operar em modo de compatibilidade.

utilizada para programas e dados tem a dimensão de 32Kx32, e está localizada no início do mapa de memória do C30. A zona seguinte é destinada ao armazenamento de imagens, onde cada pixel é representado por um número de 8 bits, no caso de imagens com 256 níveis de cinzento.

A placa possui dois métodos de codificação de imagem: O primeiro destina-se a codificar imagens em níveis de cinzento, com qualidade razoável, mas com elevado nível de compressão. Este método é usado essencialmente para codificação rápida de várias imagens por segundo (cerca de 8), dando a ideia de imagem móvel.

O segundo método destina-se a codificar imagens coloridas em *True Color* utilizando o algoritmo de codificação JPEG. Neste caso as imagens adquiridas são de elevada qualidade, mas o tempo de processamento para cada imagem é bastante grande, o que não permite a utilização deste método para imagem móvel. No entanto é possível implementar um processo em que continuamente são adquiridas imagens deste tipo, obtendo-se uma sequência de imagens coloridas a uma taxa de aproximadamente uma imagem de 2 em 2 segundos.

A mesma placa configurada de outra forma é capaz de descodificar as imagens geradas pelos dois algoritmos, devolvendo uma imagem descomprimida do tipo *raster*. Portanto, nesta configuração a placa C30 comporta-se como uma placa de descompressão por *hardware* de JPEG + Algoritmo proprietário.

2.3.2 As placas C30 no 'VideoTelefonia for Windows'

No 'VideoTelefonia' temos vídeo bidireccional, ou seja, quando temos uma conversação com vídeo entre dois utilizadores, estamos simultaneamente a codificar-> enviar por RDIS e a receber de RDIS -> descodificar. Para isso podemos ter duas hipóteses: Ou descodificamos o vídeo por *hardware* utilizando uma C30 configurada para descodificação, ou implementamos os algoritmos de descodificação por *software*. Optámos pela primeira solução, dado ser esta a mais eficiente em termos de desempenho global da aplicação.

Perante isto torna-se evidente que no nosso PC, terão que existir duas placas C30, encarregando-se uma da aquisição e compressão do vídeo, enquanto outra de encarrega unicamente da descompressão.

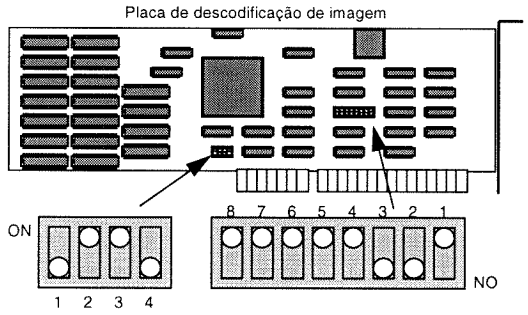

Fig. 3: Placa C30 (versão s/ Codec) para descompressão de imagem. Pormenor dos *jumpers* de configuração.

Para ligar a(s) câmara(s) ao sistema é ainda necessário uma placa de *buffers*. A figura seguinte exemplifica o modo são as ligações entre a(s) câmara(s), placa de *buffers* e placa C30 de aquisição:

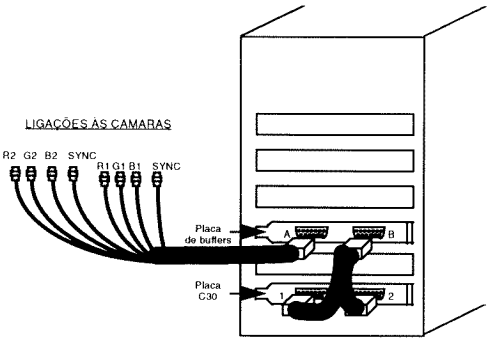

Fig. 4: Ligações câmaras -> placa de *buffers* -> placa C30 de aquisição

A configuração dos *jumpers* da placa de *buffers* depende do tipo de câmaras que se coloquem nas entradas de vídeo, de acordo com a tabela que se segue:

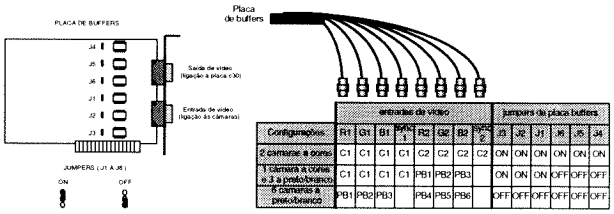

Fig. 5: Configuração da placa de *buffers* de acordo com o tipo e nº de câmaras

Além destas placas de *hardware* o *Video Telefonia para Windows* necessita ainda de um telefone RDIS, através do qual são efectuadas as ligações de voz.

III. A INTERFACE DO VÍDEO TELEFONIA PARA WINDOWS

Na fase de *Design* do Vídeo Telefonia para Windows pretendeu-se criar uma interface *user-friendly* de forma a tornar o mais evidente possível a função dos vários botões, janelas e *Dialog Boxes* e a utilização dos mesmos. Quando o rato se encontra sobre um dos botões é apresentada uma nota sumária sobre o mesmo em rodapé.

A janela principal da aplicação pode ser observada na figura seguinte, sendo de seguida descritos os vários elementos que a constituem.

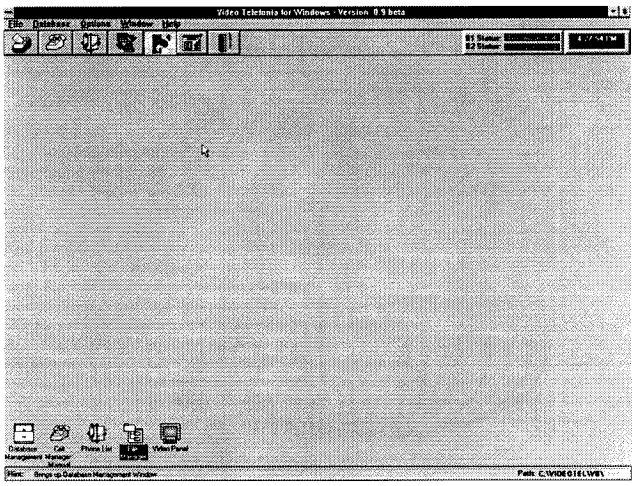

Fig. 6: Janela principal da aplicação

Os botões existentes na *Toolbar* permitem invocar as *MDI Child Windows* mais utilizadas assim como efectuar a execução das tarefas mais comuns. Temos os seguintes botões disponíveis:

1. **Base de dados** - invoca o menu Database Manager
2. **Dialer** - invoca o menu Call Manager
3. **Ficha** - invoca o menu Phone List
4. **Acesso Remoto** - invoca o menu File Manager
5. **Video Panel**- invoca o menu Video Panel
6. **Configuração** - invoca o menu configuração
7. **Exit** - Termina a aplicação, libertando todos os recursos por esta utilizados.
8. **Status da Rede** - indica a existência ou não de ligação sobre cada um dos canais B e o seu tipo (voz ou dados).

Vamos agora descrever as várias janelas e caixas de dialogo existentes na aplicação:

3.1 Database Manager

Permite alterar ou apagar as fichas da base de dados bem como os ficheiros associados a cada ficha ou criar novas fichas.

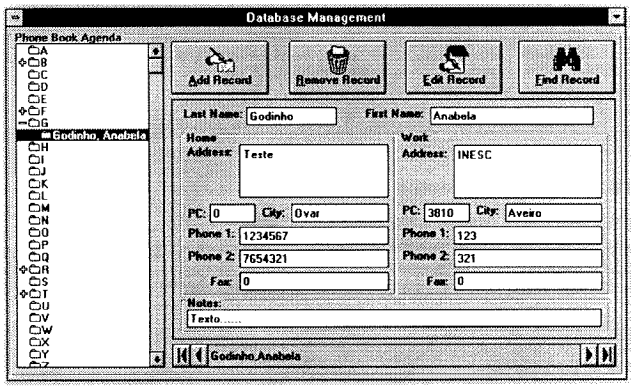

Fig. 7: Janela de Edição da Base de Dados

1. **Estrutura de fichas** - apresenta a estrutura das fichas de dados existente. O sinal + antes de cada letra indica a existência de fichas. O sinal - antes da letra indica que temos a pasta correspondente a essa letra aberta e podemos visualizar as fichas aí existentes.
2. **Nova ficha** - permite adicionar uma nova ficha à base de dados.
3. **Apagar ficha** - apaga uma ficha da base de dados.
4. **Editar ficha** - permite modificar os dados de ficha.
5. **Encontrar ficha** - permite procurar o texto indicado pelo utilizador no campo indicado das fichas existentes (sendo o campo *Last Name*, *First Name*, *City* ou *Address*). Se o texto for encontrado a ficha torna-se automaticamente activa.
6. **Próxima ficha** - muda a ficha activa para a próxima por ordem alfabética.
7. **Ficha anterior** - muda a ficha activa para a ficha anterior por ordem alfabética.
8. **Dados** - campos de dados da ficha.

3.2 Call Manager

Permite estabelecer uma ligação de voz. O destino dessa chamada pode ser preenchido pelo utilizador ou um dos valores guardados na *quick-dial list*, a lista dos endereços aos quais se ligou mais recentemente. Esta lista está guardada num ficheiro na raiz do directório onde se instalou a aplicação.

A ligação pode ser só de voz ou de voz e dados. Durante a chamada é possível alterar o tipo de ligação.

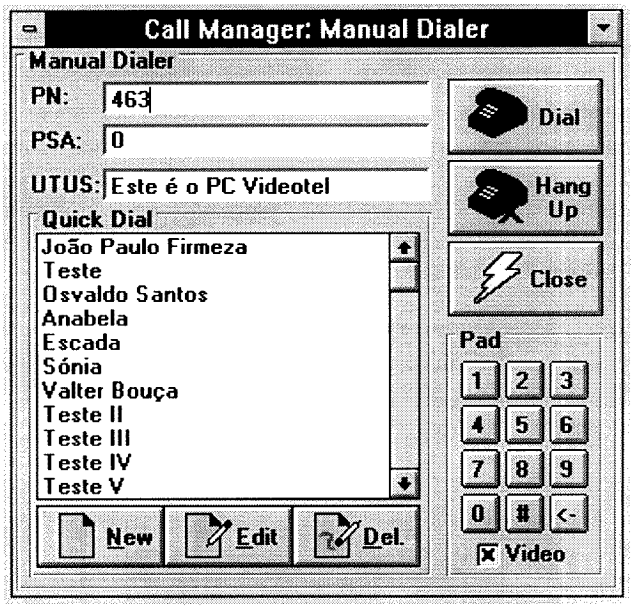

Fig. 8: Janela *Call Manager*: Permite efectuar ligações rápidas

1. **Informação** - Apresenta informação sobre o destino da chamada.
2. **Quick Dial List** - Apresenta a quick dial list. Para estabelecer uma chamada com um destes destino basta seleccioná-lo, o que irá preencher

automaticamente os vários campos necessários para estabelecer a chamada.

3. **Nova ficha -** Adiciona uma nova ficha ao quick dial list.
4. **Editar ficha** - Permite modificar os dados de uma ficha já existente.
5. **Apagar ficha** - Remove uma ficha do quick dial list
6. **Video** - Indica se a chamada que vai ser estabelecida terá imagem.
7. **Teclado** - O número do destinatário da chamada pode ser introduzido quer através do teclado numérico do PC quer por meio dos botões aqui existentes.
8. **Close** - Sai do menu. Se estiver alguma ligação activa, esta é automaticamente desfeita.
9. **Hang Up** - Interrompe a ligação.
10. **Dial** - Estabelece uma ligação com o destino indicado.

3.3 Phone List

Apresenta ao utilizador a base de dados, não permitindo a sua alteração. Esta opção permite percorrer e visualizar a base de dados, procurar texto dentro de alguns dos campos das fichas, visualizar os documentos associados ou utilizar o endereço de uma das fichas para estabelecer uma ligação (invoca o menu Call Manager com esse endereço).

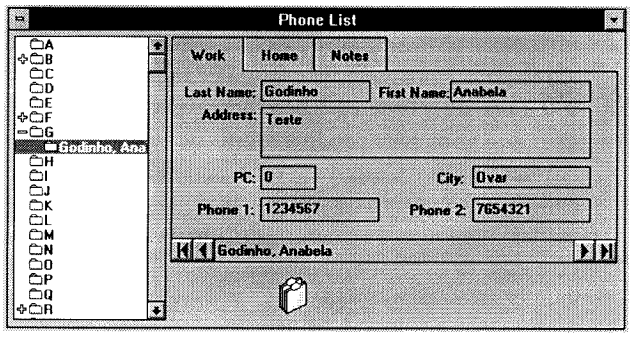

Fig. 8: Janela *Phone List*: Permite efectuar *queues* de chamadas

3.4 File Manager

Apresenta a estrutura do disco do utilizador. Para estabelecer a ligação com a máquina do outro utilizador é invocado um menu que pede a password definida pelo seu interlocutor, sendo permitidas três tentativas. Ao estabelecer a ligação passará a ter disponível a estrutura do disco do seu interlocutor. Pode navegar dentro da estrutura dos directórios tanto do seu disco como do remoto (as suas acções aparecem reflectidas no menu do outro utilizador). Depois pode seleccionar um ficheiro da sua máquina e usar o botão de envio, ou seleccionar um da máquina remota e copiá-lo para o seu disco.

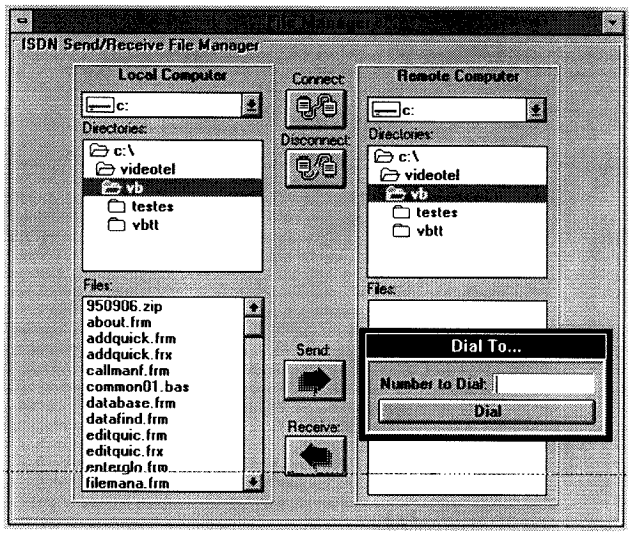

Fig. 9: Janela *File Manager*: Permite a transferência de ficheiros durante uma ligação (conversação)

1. **Drive local** - Permite mudar de drive na máquina local.
2. **Directório Local** - Apresenta a lista dos directórios locais.
3. **Ficheiros Locais** - Apresenta a lista de todos os ficheiros do directório escolhido.
4. **Send** - Copia o ficheiro seleccionado para o directório escolhido na máquina remota.
5. **Receive** - Copia o ficheiro remoto seleccionado para o directório local.
6. **Dial to** - Ao estabelecer-se uma chamada só de dados é necessário indicar o endereço do destinatário.
7. **Ficheiros Remotos** - Apresenta a lista de todos os ficheiros do directório remoto escolhido.
8. **Directório Remoto** - Apresenta a lista dos directórios remotos.
9. **Drive Remoto** - Permite mudar de drive na máquina remota.

3.5 Video Panel

Apresenta a janela que permite visualizar a imagem recebida. Permite modificar o modo de imagem e ainda fazer alguns ajustes relativos ao contraste, brilho e saturação da mesma. Quanto ao modo podemos seleccionar imagem móvel em tons de cinzento, ou sequências de imagem fixa a cores. Podemos ainda efectuar *snapshots* a cores de imagens num determinado instante. Existe ainda um botão que permite fazer o *Zoom* da imagem quando em modo de imagem móvel (que por defeito aparece num ecrã de menores dimensões).

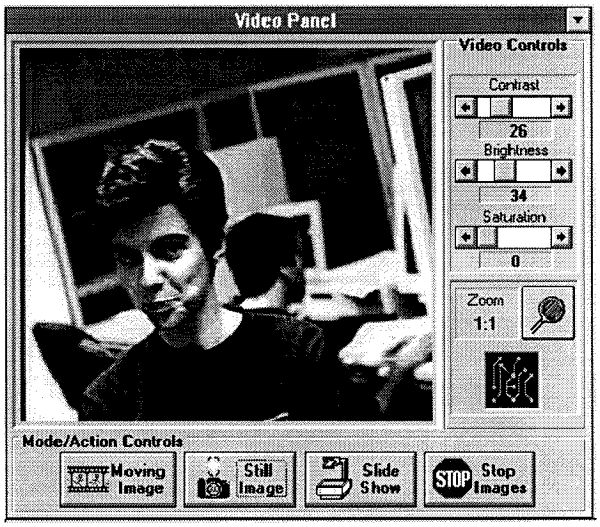

Fig. 10: Janela *Video Panel*: Permite a visualização do interlocutor remoto, assim como ajustar *settings* relaccionados com a imagem

1. **Imagem** - é aqui que vai aparecer a imagem. Devido às características do hardware utilizado a resolução máxima disponível é de 352*288 pixels com 16Milhões de cores mas esta resolução torna o processo de captura muito lento. Como forma de versatilizar o programa permitiu-se ao utilizador seleccionar o modo de imagem que pretende, de entre três modos disponíveis que passamos a descrever.
2. **Moving picture** - um dos três modos de visualização da imagem. Neste a imagem é actualizada cerca de 9 vezes por segundo com uma resolução de 176*144 *pixels* e 256 níveis de cinzento. Este modo é o que permite acompanhar melhor o movimento do utilizador à custa da qualidade da imagem.
3. **Still picture** - neste modo, a imagem é actualizada sempre que o utilizador carrega neste botão, tendo a figura apresentada a resolução máxima permitida pelo hardware (352*288*16M). Tem o inconveniente de não ser em tempo real.
4. **Slide Show** - solução de compromisso entre os dois modos anteriores. A imagem é apresentada à resolução máxima permitida e actualizada sem necessidade de intervenção do utilizador. O processo de actualização dá-se sempre que uma nova imagem completa é recebida.
5. **Stop Images** - pára o modo de imagem
6. **Zoom** - permite comutar a função de *zoom* da câmara entre o modo 1:1 e 2:1 permitindo melhor detalhe de imagem.
7. **Saturação** - permite o ajuste do nível de saturação das cores.
8. **Brilho** - permite o ajuste do brilho da imagem.
9. **Contraste** - regula o contraste da imagem.

3.6 Menu de configuração

Permite alterar os *settings* das placas C30. Estes são guardados num ficheiro do sistema e serão utilizados aquando da próxima vez que a aplicação for invocada.

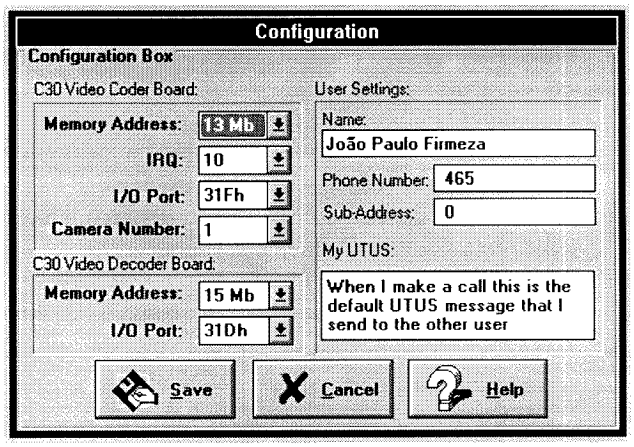

Fig. 11: Janela *Configuration*: Permite configurar os endereços das C30 e a configuração pessoal do utilizador local

1. **Endereço da placa Codificadora** - permite modificar o endereço inicial da memória RAM que a placa codificadora vai utilizar para armazenar a imagem.
2. **IRQ da placa Codificadora** - permite modificar o *interrupt* utilizado pela placa codificadora para temporizar o processo de captura de imagem.
3. **I/O *Port* da placa codificadora** - permite alterar o *Port* de I/O da placa codificadora.
4. **Número da Câmara** - a entrada da placa codificadora consiste num *multiplexer* de oito entradas aos quais podem estar ligadas outras tantas câmaras. Este campo identifica qual destas câmaras é a utilizada pela aplicação.
5. **Endereço da placa Descodificadora** - permite modificar o endereço inicial da memória RAM que a placa descodificadora vai utilizar para armazenar a imagem.
6. **I/O *Port* da placa Descodificadora** - permite alterar o *Port* de I/O da placa descodificadora.
7. **Save** - grava os *settings* actuais no ficheiro de configuração e sai do menu de configuração.
8. **Cancel** - sai do menu de configuração sem actualizar os *settings* das placas.
9. **Help** - apresenta uma explicação sumária dos vários campos.
10. **Campos de configuração local** - Nestes campos é possível identificar o utilizador, o número e o sub-endereço locais, e a mensagem de UTUS usada por defeito.

IV. ANÁLISE RESUMIDA DO *SOFTWARE* E SUA ORGANIZAÇÃO

A aplicação foi desenvolvida utilizando essencialmente duas ferramentas: O *Borland C++ 4.0* e o *Microsoft Visual Basic 3.0 Professional Edition.*

Por detrás da concepção desta aplicação está a ideia de modularidade. Assim o *software* desenvolvido irá assentar-se nos seguintes módulos essenciais, que constituem os seus blocos lógicos de acordo com o tipo de funções realizadas :

- Módulo de comunicações RDIS
- Módulos de vídeo: Codec C30 e Decoder C30
- Rotinas de interface: DLL's <=> Visual Basic
- Rotinas de interface: Visual Basic <=> User/GDI
- Rotinas de gestão de Bases de Dados
- Rotinas de controlo da aplicação: Timers, Máquinas de estados e *delays.*

O diagrama seguinte mostra a organização da aplicação em termos funcionais, podendo ser observadas as várias DLL's[3] que compõem a aplicação, assim como as relações entre estas, o *hardware,* e os restantes módulos existentes:

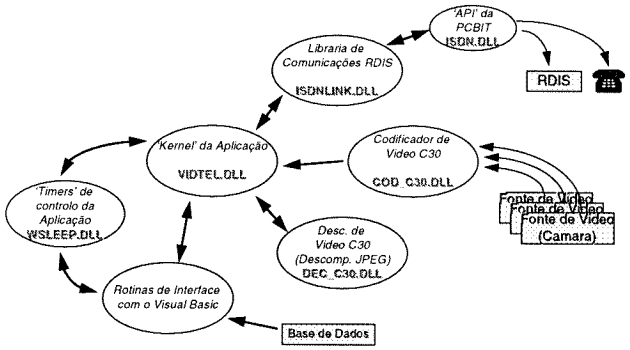

Fig. 12: Diagrama da organização dos módulos da aplicação

4.1 Módulo de comunicações RDIS

A API[4] da carta PCBIT permite aceder a uma ligação RDIS em acesso básico, disponibilizando as primitivas básicas de comunicação por meio de uma DLL (ISDNBIOS.DLL). No entanto os serviços de estabelecimento de ligações, sua terminação, envio e recepção de mensagens de sinalização ou de pacotes de dados têm que ser desenvolvidos tendo em vista a aplicação final. A biblioteca ISDNLINK foi desenvolvida tendo em conta estes requisitos bem como uma boa eficiência e performance, que são requisitos de qualquer aplicação. No caso do Windows 3.x essa procura de eficiência e performance levou ao desenvolvimento de código que permite vários serviços que podem ser usados concorrentemente sobre a ligação física e em *background* por forma a rentabilizar a largura de banda do canal e aproveitar ao máximo as potencialidades da máquina usada com um mínimo de tempo de utilização do CPU.

A biblioteca de classes ISDNLINK.DLL[5] é a responsável pelo estabelecimento e gestão das ligações lógicas sobre o canal RDIS. Este módulo foi construído em C++, logo tem uma estrutura orientada ao objecto, que permite uma maior versatilidade na construção do programa e uma maior eficiência do mesmo. Um simples diagrama de classes, tal como é gerado pelo Borland C++, pode ser observado na figura seguinte:

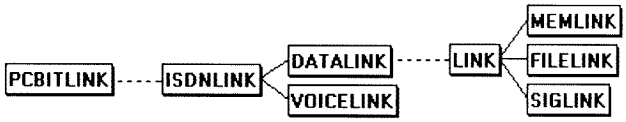

Fig. 13: Diagrama de classes da biblioteca ISDNLINK.DLL. As linhas a tracejado indicam que a classe na hierarquia inferior utiliza serviços da classe hierarquicamente superior.

Por análise do diagrama das classes e suas relações podemos ver que a classe base de todo o processo é a PCBITLINK. Esta classe é responsável pela inicialização da placa PCBIT, o registo da aplicação e encaminhamento das mensagens provenientes da API para os objectos ISDNLINK. O objecto PCBITLINK mantêm o registo dos objectos ISDNLINK existentes. Quando surge uma mensagem para um objecto que não existe ainda (por exemplo quando se está a estabelecer uma ligação), este é criado. Mensagens que não têm destinatário são encaminhadas para o primeiro objecto ISDNLINK disponível.

Utilizando os serviços desta, temos a classe virtual ISDNLINK. Esta classe é responsável pelo interface entre a classe base PCBITLINK e as classes DATALINK e VOICELINK. Os métodos nela definidos são métodos gerais de resposta às mensagens provenientes da API.

A classe VOICELINK é responsável pela gestão das ligações de voz sobre um canal B. É responsável ainda pelo configuração dos métodos utilizados para a codificação da voz ao nível do *hardware* da PCBIT.

A classe DATALINK é responsável pelo estabelecimento e gestão das chamadas de dados sobre o canal B. É ainda responsável pelo estabelecimento das camadas 2 e 3 do modelo ISO, fornecendo aos objectos LINK a possibilidade de transmissão e recepção de pacotes.

Utilizando serviços desta temos a classe LINK, cuja função é a de garantir a interface entre a DATALINK e as classes que dela são derivadas- FILELINK, MEMLINK e SIGLINK. Cada LINK comporta-se como um canal virtual e vai utilizar o objecto DATALINK como um servidor de pacotes. Podem existir vários LINKs ao mesmo tempo (até 256) que partilham entre si equitativamente a largura de banda do canal B, permitindo assim vários canais de transmissão independentes, possibilitando em simultâneo a transmissão de imagem, ficheiros e mensagens de sinalização sobre a mesma ligação física. Cada LINK tem

[3] DLL - *Dynamic Link Library*

[4] API - Application programmable interface.

[5] Ver Revista do DETUA Vol. 1 Nº 4: 'ISDNLINK - Uma biblioteca de classes para RDIS'.

um identificador que lhe é atribuído quando é registado num DATALINK (e é igual no outro extremo do canal) de forma a garantir que os pacotes de informação são bem encaminhados.

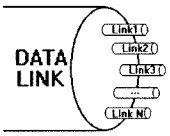

Fig. 14 Vários LINK's lógicos sobre um DATALINK

A classe FILELINK trata da transferência de ficheiros. Para iniciar a transmissão de um ficheiro basta registar um objecto deste tipo num objecto DATALINK e invocar o método responsável pelo envio de um ficheiro. No outro extremo a recepção é automática. Quando o tamanho do ficheiro a transmitir é superior ao tamanho máximo do pacote, esta é fragmentada em vários pacotes que depois são reconstruídos na recepção.

A classe MEMLINK trata da transferência de blocos de memória, identificados por um *handle* desse bloco. Na recepção é alocado um bloco de memória com o tamanho indicado, e o seu *handle* é fornecido à aplicação quando a transmissão finaliza. Tal como na FILELINK também aqui o programa trata de fragmentar os blocos de tamanho superior ao limite e de os reconstruir na recepção.

A classe SIGLINK é semelhante à MEMLINK, mas para blocos de memória mais pequenos. O bloco de memória a enviar pode ter no máximo 1920 bits, que é o tamanho máximo de um pacote. È útil para o envio de pequenas mensagens de sinalização e controlo entre os dois utilizadores.

É sobre esta biblioteca, que assenta todo o processo de comunicações da aplicação, desde a transferência de mensagens de controlo, transferência de imagens ou mesmo transferência de ficheiros. Este modo permite ao utilizador aceder à outra máquina como se fosse uma máquina remota e copiar ficheiros bidireccionalmente entre as duas. Foi introduzido um sistema de segurança por *password* de forma a impedir acessos indesejados. Quando se tenta estabelecer o modo de transferência de ficheiros é pedida a *password* que o outro utilizador definiu para o seu computador e só se responder correctamente (em três possibilidades) é que tem acesso à estrutura de directórios do seu interlocutor (e ele à sua).

Ao navegar ao longo da estrutura de directórios os seus movimentos são reflectidos no outro extremo. Dessa forma é possível ao utilizador controlar o processo de transferência pois ao verificar que estão a aceder a áreas de informação que se pretendem manter privadas pode interromper a ligação.

É possível transferir vários ficheiros ao mesmo tempo, mas esta solução não é eficiente porque o processo de transferência demora o mesmo tempo do que transferindo-os em sequência (pois ambos estarão a disputar os recursos do canal de transmissão). As várias ligações lógicas FILELINK vão concorrer pelo mesmo canal de comunicação, virtualmente representado pela classe DATALINK, por isso não há ganho de velocidade.

O programa prevê a possibilidade de estabelecer chamadas apenas para dados.

4.2 Módulo de processamento de Imagem

Este módulo é composto por duas bibliotecas dinâmicas (DLL's), sendo uma delas responsável pela aquisição e codificação de imagens vídeo, e outra pela descompressão/descodificação. Estas bibliotecas foram desenvolvidas em C++, tendo-se criado uma classe que representa a placa de aquisição/codificação na libraria COD_C30.DLL e uma classe que representa a placa de descodificação na libraria DEC_C30.DLL.

Quando a aplicação é lançada, as duas placas C30 são inicializadas, uma para aquisição/compressão e a outra para descompressão de imagem.

A imagem é adquirida por intermédio de uma placa TMS320C30 à qual está ligada uma câmara de vídeo. A placa é responsável pela compressão da imagem para um formato específico no caso de imagens móveis, ou para o formato JPEG no caso de imagem fixa. É esta imagem comprimida que será recebida no outro extremo do canal de comunicação, sendo então descomprimida por uma segunda placa C30, que fornece à aplicação a imagem que será então apresentada ao utilizador. A configuração das placas é feita por software e pode ser alterada em *run-time*.

A apresentação da imagem pode funcionar de três modos: imagem fixa, móvel e por *steps*. A imagem fixa é uma imagem colorida de resolução 352x288x16*Milhões de cores* que é actualizada cada vez que o utilizador pede uma nova imagem. A imagem móvel é a preto e branco e é adquirida ao ritmo máximo que o equipamento permite, com uma resolução de 176x144x256*Niveis de cinzento*. O modo step é uma solução de compromisso entre os dois modos, tendo em conta as limitações da placa e do canal. Neste modo a placa vai adquirindo imagens semelhantes à do modo fixo que vão sendo actualizadas logo que uma nova imagem esteja disponível. As imagens surgem por isso a um ritmo muito mais lento, mas com uma resolução bastante boa. O processo de captura de imagens móveis baseia-se num algoritmo de diferenças - primeiro é captada uma imagem com um detalhe bastante baixo que vai progressivamente sendo melhorada, pela recepção de imagens diferença entre esta e as anteriores. No caso de termos alterações bruscas na imagem (movimentos muito rápidos) o detalhe da imagem nos instantes seguintes será bastante baixo, mas como em princípio as imagens captadas não vão ser de variação rápida este problema não é muito relevante. O processo de *steps* foge a este problema, porque todas as imagens são totalmente codificadas e transmitidas na integra, não havendo qualquer relação entre uma imagem e as anteriores.

Em modo móvel são apresentadas cerca de 9 imagens por segundo e em modo step uma imagem (em média) a cada 2 segundos.

4.3 Interface com o utilizador

O módulo responsável pela interface com o utilizador foi escrito em *Visual Basic 3.0* aproveitando a simplicidade na construção da mesma que esta linguagem garante com a vantagem de permitir uma abordagem fácil da aplicação graças à estratégia evento-método com que está desenvolvida. Utilizando esta ferramenta foram criados todos os elementos da interface, assim como a resposta a todos os eventos por ela gerados.

Uma das questões que surgiram foi a seguinte: Será que valeria a pena efectuar algumas das rotinas de processamento no *Visual Basic*, ou seria preferível utilizar uma DLL construída em C, e deixar para o Basic apenas as questões relativas à interface?

A opção tomada foi a segunda, pois foi aquela que se mostrou melhor em termos de desempenho do sistema. Assim, quase todas as rotinas de processamento (e mesmo algumas das funções vitais em desempenho da interface) foram desenvolvidas numa DLL construída em C++: A VIDTEL.DLL. A única excepção de realce, foram as rotinas de gestão de uma base de dados, que apesar de incluírem algum processamento, foram realizadas em Visual Basic tirando partido da facilidade com que o é possível fazer.

4.4 Base de dados

Usando as ferramentas '*Database Access*' do Visual Basic, foi implementada uma agenda telefónica convencional com a possibilidade de associar a cada ficha vários documentos relacionados que podem ser chamados e visualizados a qualquer momento. Esta base de dados pode ser modificada livremente pelo utilizador. Ao estabelecer uma chamada ou ao recebê-la, o programa lê o campo de sinalização que identifica o número do interlocutor e usa-o como campo de pesquisa na base de dados. Se o encontrar invoca a base de dados para permitir ao utilizador visualizar os dados disponíveis e os documentos relacionados com o essa ficha.

Além da base de dados geral, existe uma *quick-dial list*, ou seja a lista dos números mais utilizados, numa perspectiva de poder fazer rapidamente uma ligação, sem perder muito tempo a pesquisar na base de dados geral.

Todas as rotinas de acesso a bases de dados foram escritas em *Visual Basic*, tirando partido das respectivas funções que a versão *3.0 Professional* possui. Assim, foi utilizado o formato de base de dados do *Microsoft Access*, tendo-se criado uma tabela com os campos necessários. O acesso à base de dados é feita essencialmente utilizando o *Custom Control - DataControl* que se encontra numa VBX da versão *Professional* do VB. Este controlo permite praticamente todas as funções essenciais a uma base de dados: Adicionar, remover, editar *Records* de uma base de dados, para alem de todas as funções da linguagem SQL.

4.5 O Kernel *da aplicação: VIDTEL.DLL*

É esta DLL a principal entidade de interacção com as outras librarias desenvolvidas, e onde estão definidas classes de alto nível derivadas de classes presentes nas bibliotecas desenvolvidas.

Este método foi apenas possível porque o compilador utilizado - *Borland C++ 4.0*, tem a funcionalidade de permitir derivar classes de librarias pré-compiladas, incluído nos projectos em questão ficheiros *.lib* desde que desenvolvidos em C++, utilizando a técnica de programação orientada ao objecto. Note-se que este processo não era possível utilizando outros compiladores de C++, como por exemplo a versão anterior do *Borland C++* : a versão 3.1. Nestes, não nos era possível derivar classes de classes definidas em librarias pré-compiladas.

A vantagem deste processo consiste na facilidade em distribuir diferentes conjuntos de classes contendo serviços específicos por diferentes librarias dinâmicas, que, em *run-time*, são carregadas, quando uma libraria cliente solicita um ou vários dos seus serviços.

- processo descrito encontra-se esquematizado nas figuras seguintes:

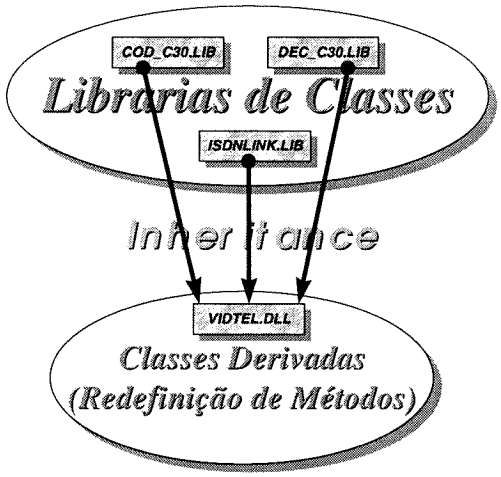

Fig. 15: Classes derivadas das classes oferecidas nas librarias. 'Herança' de métodos e dados.

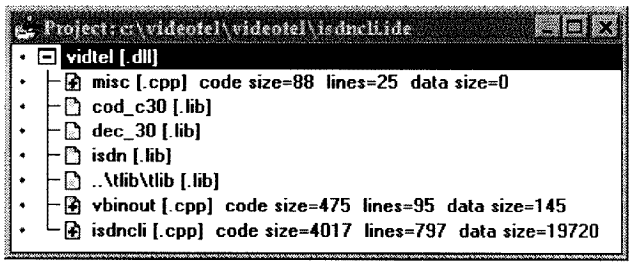

Fig. 16: Projecto da libraria VIDTEL.DLL, onde se pode ver a inclusão no mesmo das librarias de *cod_c30.dll, dec_30.dll, isdn.dll* e *tlib.dll*.

4.5.1 Classes de VIDTEL.DLL derivadas de classes definidas noutras bibliotecas

Na biblioteca VIDTEL.DLL são derivadas uma série de classes de classes definidas noutras DLL's de nível inferior, e são ainda definidas algumas novas classes. O diagrama seguinte (obtido no *Borland C++ 4.0*) representa a estrutura de classes existentes na VIDTEL.DLL:

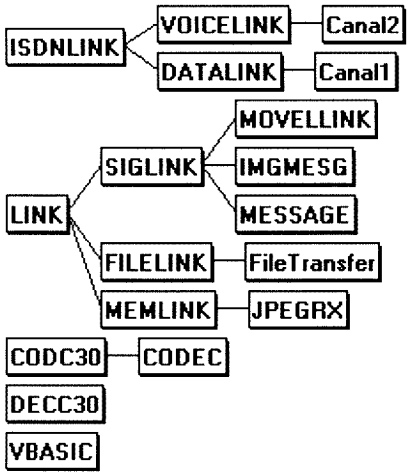

Fig. 17: A estrutura global de classes da libraria VIDTEL.DLL

I) A classe *Canal1*

Esta classe é derivada da classe *DATALINK* pertencente à libraria ISDNLINK.DLL.

A sua função é permitir a criação de objectos que representem um canal físico ISDN de 64Kbits para transferência de dados. O número máximo de objectos que podemos instanciar desta classe, no caso de um acesso básico, são 2. Na aplicação em questão, apenas queremos utilizar um canal B para transferência de dados, logo apenas foi declarado um objecto que usa esse canal.

```
class _export Canal1:public DATALINK
{
  public:
            WORD FAR PASCAL _export ConnectActiveIndCallback(char far
*address,char far *subaddress,IEUserToUserSignalling UTUS);
             WORD FAR PASCAL _export ConnectActiveConfCallback(void);
             WORD FAR PASCAL _export ConnectActiveProtocolsCallback(void);
             WORD FAR PASCAL _export DisconnectIndCallback();
             WORD FAR PASCAL _export DisconnectConfCallback();
};
```

Código 1: Classe 'Canal1' derivada da classe DATALINK

É sobre um objecto desta classe que serão definidos vários objectos que representam ligações lógicas de dados sobre o canal mapeado pelo objecto instanciado da classe 'canal1'.

As funções de *callback* que são virtuais em DATALINK são agora redefinidas nesta classe. Estas funções servem essencialmente para sinalizar a aplicação cliente da libraria VIDTEL.DLL do estado da ligação física ISDN: Estado Connected/Not Connected, estado dos protocolos para transferência de dados, etc.

```
Canal1 B1;  //Objecto instanciado de Canal1 para transferencia de dados
                    //sobre um canal B de 64Kbits
```

Código 2: Objecto instanciado de 'Canal1'

II) A classe *Canal2*

Esta classe é derivada da classe *VOICELINK* pertencente à libraria ISDNLINK.DLL.

A sua função é permitir a criação de objectos que representem um canal físico ISDN de 64Kbits para voz. O número máximo de objectos que podemos instanciar desta classe, no caso de um acesso básico, são 2, ou seja, virtualmente poderíamos ter duas ligações de voz em simultâneo sobre um acesso básico de ISDN. No caso da nossa aplicação, apenas vamos ter um canal B para ligações de voz, o que implica que iremos ter apenas um objecto instanciado da classe *Canal2*.

```
class _export Canal2:public VOICELINK
{
  public:
            WORD FAR PASCAL _export ConnectActiveIndCallback(char far
                    *address,char far *subaddress,
IEUserToUserSignalling UTUS);
            WORD FAR PASCAL _export ConnectActiveConfCallback(void);
            WORD FAR PASCAL _export ConnectActiveProtocolsCallback(void);
            WORD FAR PASCAL _export DisconnectIndCallback();
            WORD FAR PASCAL _export DisconnectConfCallback();
};
```

Código 3: Classe 'Canal2' derivada da classe VOICELINK

As funções de *callback* que são virtuais em VOICELINK são agora redefinidas nesta classe. Estas funções servem essencialmente para sinalizar a aplicação cliente da libraria VIDTEL.DLL do estado da ligação física de voz sobre ISDN: Estado Connected/Not Connected, etc.

```
Canal2 B2;  //Objecto instanciado de Canal2 para ligações de voz
                    //sobre um canal B de 64Kbits
```

Código 4: Objecto instanciado de 'Canal2'

III) A classe *MESSAGE*

Esta classe é derivada da classe SIGLINK da libraria ISDNLINK.DLL. É responsável pelo tratamento das mensagens gerais de controlo da aplicação. Para isso a classe contem a redefinição da função virtual ReceiveMessage que é a função de *callback* quando uma mensagem chega da API à libraria ISDNLINK.

```
class _export MESSAGE:public SIGLINK
{
  public:
   WORD far pascal _export ReceiveMessage(LPBYTE Data,WORD Len);
};
```

Código 5: Classe 'MESSAGE' derivada da classe SIGLINK

Por forma a tornar esta classe polivalente no que diz respeito a tratamento de mensagens gerais, as mensagens seguem um formato pré-estabelecido: Possuem um primeiro campo denominada MessageID, que não é mais do que uma palavra de 16bits. Este campo identifica o tipo de cada uma das mensagens de uma forma unívoca. O segundo campo da mensagem representa a informação propriamente dita que pretendemos enviar. Como exemplo duma mensagem geral da aplicação podemos ter a

mensagem que transporta as *passwords* dos utilizadores. - Essa mensagem é composta por um ID, que é definido como uma constante inteira, e pela *password* propriamente dita, que é uma *string* encriptada.

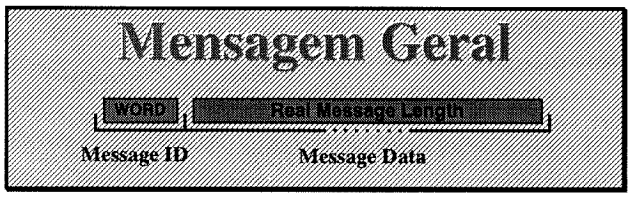

Fig. 18: Estrutura de uma mensagem geral da aplicação

Com esta abordagem, basta-nos instanciar um objecto da classe MESSAGE, para transmitir e receber mensagens gerais da aplicação, sobre o canal mapeado para dados. A função ReceiveMessage não passa de uma simples máquina de estados, que consoante o valor do campo *MessageID* executa uma determinada operação. Como exemplo ilustrativo podemos observar o seguinte código:

```
WORD far pascal _export MESSAGE::ReceiveMessage(LPBYTE Data,WORD Len)
{
 WORD Code;

 Code=MAKEWORD(Data[0],Data[1]);   //The first WORD of the Message is the
MessageID

switch (Code)
 {
  case SEND_PASS:        // Do something
            ...
            break;

  case PASS_OK:          // Do something else...
            ...
            break;

  case PASS_WRONG:       // Do another thing...
            ...
            break;

            .
            .
            .
            .
  case STOP              // Do some action
            ...
            break;

 }
```

Código 6: Exemplo da função *ReceiveMessage* para uma classe de mensagens gerais

IV) A Classe IMGMESG

Esta classe é outra classe derivada de SIGLINK. A diferença desta, relativamente à classe MESSAGE é que neste caso as mensagens a transmitir são apenas pedidos de mudança de modo de codificação da placa C30. Neste caso a função virtual *ReceiveMessage* da libraria ISDNLINK.DLL, vai ser redefinida para responder a ordens dadas pela aplicação remota quando esta pretende que se altere o modo de aquisição de vídeo.

```
class _export IMGMESG:public SIGLINK
{
 public:
  WORD far pascal _export ReceiveMessage(LPBYTE Data,WORD Len);
};
```

Código 7: A classe IMGMESG

```
IMGMESG IR1;     //objecto instanciado de SIGLINK para transmissao
                 // de imagens móveis
```

Código 8: O objecto IR1 instanciado de IMGMESG

Neste caso, vamos querer que sempre que chegue uma mensagem deste tipo, que não é mais nem menos que um pedido de alteração do modo de codificação da placa C30, ordenar à placa C30 que assim o faça. O código da função *ReceiveMessage* que é apresentado na figura seguinte mostra como é processada uma mensagem deste tipo. Mais uma vez estamos perante uma pequena máquina de estados.

```
/*************************************************************************\
* FUNCTION: IMGMESG::ReceiveMessage                                       *
*                                                                         *
* PURPOSE: Respond to messages requesting mode changement on C30 board    *
\*************************************************************************/
WORD far pascal _export IMGMESG::ReceiveMessage(LPBYTE Data,WORD Len)
{
  BYTE TipoImagem,Camara;

  TipoImagem=MAKEWORD(Data[0],Data[1]);
  Camara=MAKEWORD(Data[2],Data[3]);


 switch (TipoImagem)
   {
     case MOVEL:
            Coder.SelMovel(Camara); //selects the Codec for moving frames
     break;

     case FIXA:
Coder.SelFixa(Camara); //selects the Codec for still image
     break;

     case FILME:
Coder.SelScan(Camara); //selects the Codec for Step-by-Step
     break;

    case STOP:
Coder.Stop(); //stops the Codec
    break;

   }
  return 0;
}
```

Código 9: Código da redefinição da função virtual *ReceiveMessage* de SIGLINK em IMGMESG

V) A Classe MOVELLINK

Esta classe é mais uma vez uma classe derivada de SIGLINK. Neste caso pretende-se usar esta classe para transmitir imagens móveis através da RDIS, criando-se assim mais um *link* lógico sobre o canal físico representado pelo objecto B1. Á primeira visto a isto pode parecer estranho: Enviar *frames* sobre um link destinado essencialmente a pequenas mensagens cujo tamanho não exceda o tamanho máximo de um pacote ?...

De facto não existe qualquer problema e isto é possível, porque de acordo com o algoritmo de codificação de imagem móvel da C30, as *frames* móveis nunca excedem o tamanho máximo de um pacote ISDN. Perante isto o *overhead* é muito menor se for usada uma classe derivada de SIGLINK em vez de derivar uma de MEMLINK[6].

```
class _export MOVELLINK:public SIGLINK
{
 public:
  WORD far pascal _export ReceiveMessage(LPBYTE Data,WORD Len);
};
```

Código 10: A classe MOVELLINK

[6] Na classe MEMLINK o overhead é maior, porque virtualmente sobre objectos de MEMLINK podem ser transmitidos *buffers* de memória de qualquer dimensão, o que implica existir necessáriamente código que segmente o blocos, (para além de outras questões). Na classe SIGLINK, que foi criada a pensar únicamente na trasnmissão de mensagens, o tratamento da mensagem a enviar é praticamente nulo.

Declarando um objecto de MOVELLINK, vamos ter o nosso *link* virtual criado para transmissão de imagem móvel.

```
MOVELLINK ML;          //objecto instanciado de SIGLINK para transmissao
                  //imagem móvel
```

Código 11: O objecto ML instanciado de MOVELLINK

A função *ReceiveMessage* neste caso, vai ter como função, encaminhar a mensagens que chegam da rede, enviadas pela aplicação remota, que não são mais do que *frames* moveis para a placa C30 configurada para descodificação de vídeo. O código da função é apresentado na figura seguinte.

```
/*****************************************************************\
* FUNCTION: MOVELLINK::ReceiveMessage                           *
*                                                               *
* PURPOSE:  Responds to moving image messages                   *
\*****************************************************************/

WORD far pascal _export MOVELLINK::ReceiveMessage(LPBYTE Data,WORD Len)
{
  BYTE TipoImagem,Camara;

  TipoImagem=MAKEWORD(Data[0],Data[1]);

  switch (TipoImagem)
   {
     case MOVEL: Decoder.DecompressMovingFrame(&Data[2],VBInOut.P_hWnd);

     default: Decoder.DecompressMovingFrame(&Data[2],VBInOut.P_hWnd);

     break;
  }
  return 0;
}
```

Código 12: Código da função ReceiveMessage de MOVELLINK

VI) A Classe JPEGRX

Esta classe é derivada de MEMLINK e a sua função é permitir a transmissão de imagens fixas codificadas pela placa C30 no formato JPEG. É sobre um objecto instanciado desta classe que virtualmente se cria um canal lógico para transmissão de JPEG's. A única função da classe original MEMLINK que foi aqui redefinida foi a função *HandleReception* que é a função responsável pelo processamento das mensagens que chegam da rede a um objecto de MEMLINK (ou de JPEGRX neste caso).

```
class _export JPEGRX:public MEMLINK
{
  public:
   WORD far pascal _export HandleReception(HGLOBAL Object);
};
```

Código 13: A classe JPEGRX

```
JPEGRX CanalImgFixa; // Canal lógico para transmissão de imagens JPEG
```

Código 14: O objecto CanalImgFixa instanciado da classe JPEGRX

Relativamente á função *HandleReception*, a sua função neste caso será encaminhar os dados recebidos da rede (que não são mais do que imagens JPEG) para a placa C30 configurada para descodificação de imagem. Podemos observar o corpo da função na figura seguinte:

```
/*****************************************************************\
* FUNCTION: JPEGRX::HandleReception                             *
*                                                               *
* PURPOSE: Handles the Reception of buffers containing JPEG images      *
\*****************************************************************/
WORD far pascal _export JPEGRX::HandleReception(HGLOBAL Object)
{
  LPBYTE Ptr;

  Ptr=GlobalLock(Object);
  if (!Ptr)
   {
     MessageBox(GetFocus(),"","GlobalLock error",MB_OK);
     return 1;
   }

  // Vou descomprimir o JPEG com a C30
  Decoder.DecompressJPEG(Ptr,GlobalSize(Object),VBInOut.P_hWnd);

  GlobalUnlock(Object);
  GlobalFree(Object);

  return 0;
};
```

Código 15: Código da função *HandleReception* de JPEGRX

A Classe FileTransfer

Esta classe é derivada de FILELINK e a sua função é permitir a transmissão de ficheiros. É sobre um objecto instanciado desta classe que virtualmente se cria um canal lógico para transmissão de ficheiros.

```
class _export FileTransfer:public FILELINK
{
          public:
                    WORD ChannelState;  // Estado do canal de transmissão
de ficheiros


                    WORD FAR PASCAL _export RxFileStart(char far *Name);
                    WORD FAR PASCAL _export RxFileEnd(WORD flag);
                    WORD FAR PASCAL _export TxFileStart(char far *Name);
                    WORD FAR PASCAL _export TxFileEnd(WORD flag);
                    WORD FAR PASCAL _export TxFileProgress(DWORD Bytes,
DWORD Total);
                    WORD FAR PASCAL _export RxFileProgress(DWORD Bytes,
DWORD Total);

};
```

Código 16: A classe *FileTransfer*

Como se pode observar no código da declaração da classe, são redefinidas aqui as funções virtuais da classe FILELINK pertencente à libraria ISDNLINK.DLL. Estas funções são funções de *callback* que são chamadas automaticamente pela DLL. As funções *RxFilexxx* são chamadas aquando da recepção de um ficheiro, enquanto *TxFilexxx* são chamadas na transmissão. Como exemplo do uso destas funções, podemos observar o código da função *TxFileProgress* que permite actualizar o valor de um controlo da aplicação Visual Basic (normalmente uma barra de percentagem) indicando o progresso da transmissão de um ficheiro.

```
WORD FAR PASCAL _export FileTransfer::TxFileProgress(DWORD Bytes, DWORD Total)
{
          char aux1[8];
          DWORD aux2;

          aux2=Bytes*100;
          aux2=aux2/Total;
          wsprintf(aux1,"%d",aux2);

EscreveNoControloDoVisualBasic(aux1,VBGlobalHandle.HandleFromVb4);

          // A percentagem esta em aux2

          Debug("Transmitindo %ld bytes de %ld ",Bytes,Total);

          return(0);
}
```

Código 17: A definição da função *TxFileProgress* da classe *FileTransfer*

Para utilizar esta classe, temos mais uma vez que instanciar um objecto dela:

```
FileTransfer F1;   // O objecto que permite a transmissão de ficheiros sobre
um canal B
```

Código 18: O objecto que simboliza a transferência de ficheiros

VIII) A Classe CODEC

Esta classe é derivada da classe CODC30 definida em COD_C30.DLL. A ideia neste caso é redefinir as funções virtuais *FrameMovel* e *JpegReady* de CODC30.

```
class _export CODEC:public CODC30
{
   public:
   far pascal _export CODEC(long a,int b,int c):CODC30(a,b,c) {};

   WORD far pascal _export FrameMovel(LPBYTE Frame,WORD Comp);
   WORD far pascal _export JpegReady(LPBYTE Data,WORD Comp);
};
```

Código 19: A classe CODEC derivada de CODC30

Estas funções são funções de *callback* e são chamadas sempre que a placa C30 de codificação, codifica completamente uma *frame* móvel ou uma imagem JPEG respectivamente. Como no caso da nossa aplicação queremos enviar estas imagens imediatamente por RDIS nada mais fácil do que redefinir as funções em questão para esse efeito. Assim, temos para a função *FrameMovel,*

```
/*************************************************************************\
* FUNCTION: CODEC::FrameMovel                                             *
* PURPOSE:  Sends moving Frame thru ISDN                                  *
\*************************************************************************/
WORD far pascal _export CODEC::FrameMovel(LPBYTE Frame,WORD Comp)
{
 // para a RDIS

 ML.SendMessage(MOVEL,Frame,Comp);  //Utilizando o objecto ML para mandar a
frame por RDIS

 Debug("Frame mandada por RDIS");

 return 0;
}
```

Código 20: A função *FrameMovel*

e para a função *JpegReady*,

```
/*************************************************************************\
* FUNCTION: CODEC::JpegReady                                              *
* PURPOSE:  Sends JPEG Image thru ISDN                                    *
\*************************************************************************/
WORD far pascal _export CODEC::JpegReady(LPBYTE Data,WORD Comp)
{

 WORD r=CanalImgFixa.SendData(Data,Comp,MEM_COPY);

 return 0;
}
```

Código 21: A função *JpegReady*

V. A APLICAÇÃO FINAL - RESULTADOS

5.1 - Atraso na transmissão da imagem

Verificou-se que existe um atraso de aproximadamente 2 segundos entre a captura da imagem e a recepção da mesma no outro extremo do canal de comunicação. Embora este facto tenha um efeito negativo no desempenho da aplicação é justificado e inevitável devido às características da aplicação e do *hardware* utilizado. Este atraso é devido a duas causas distintas: atraso quando da transmissão da imagem via RDIS e atraso devido ao processo de captura e compressão de imagem e respectiva descompressão no receptor. No modo de imagem móvel o atraso devido ao processo de compressão e descompressão é pouco significativo, sendo a transmissão da imagem a grande responsável por esta demora. No modo step o tempo de processamento na compressão e descompressão também já é significativo.

5.1.1 Imagem móvel

O atraso na transmissão da imagem é fundamentalmente devido ao tempo necessário à transmissão das imagens sobre o canal B, que tem uma largura de banda relativamente baixa. Verificou-se que o processo de transmissão leva cerca de 2 segundos enquanto que a transmissão da voz é imediata, o que leva ao desfasamento entre as duas.

5.1.2 Modo Step

A resolução da imagem neste modo é de 352*288 pixels por 16 milhões de cores, resultando num bloco de dados bastante grande. Mesmo com a compressão a imagem terá em média cerca de 20Kbytes (depende bastante do tipo de imagem), levando a tempos de transmissão de ordem dos 2,5 a 3 segundos. Além disso o processo de compressão é bastante demorado, tendo-se verificado ser de cerca de 2 segundos tanto para compressão como para descompressão. Somando tudo obtemos atrasos de 6 a 7 segundos entre a captura da imagem e a sua descompressão no receptor. O tempo que cada nova imagem demora a aparecer é igual ao maior destes atrasos, logo de cerca de 2 a 3 segundos.

5.2 - Resolução da imagem transmitida

Como já foi referido a aplicação trabalha com o *hardware* no seu limite máximo de resolução, 176*144 pixels com 256 tons de cinzento para imagem móvel e 352*288 pixels por 16 milhões de cores no modo fixo. É fácil de perceber que para trabalhar com imagens de resolução superior é necessário equipamento com características diferentes do utilizado e consequente encarecimento do mesmo. Além disso as imagens de resolução superior iriam ocupar maior largura de banda (admitindo um factor de compressão uniforme) o que iria limitar o número de imagens transmitidos por segundo que já assim é reduzido. Uma maior taxa de compressão talvez fosse possível, mas à custa da complexidade da placa e de maior demora no processo de compressão e descompressão levando ao agravamento do problema apresentado no ponto anterior e à custa da qualidade da imagem.

5.3 - Eficiência da transmissão

Além dos bits de controlo enviados em cada pacote, que percentualmente são pouco significativos quando comparados com o tamanho do mesmo, existe um outro motivo que leva a que a taxa de ocupação efectiva do canal de transmissão de dados seja inferior à unidade, mesmo que exista um LINK a tentar ocupá-lo totalmente. Este motivo é facilmente perceptível se tivermos em conta

a forma como os objectos DATALINK fazem a gestão do canal e dos vários objectos LINK.

Qualquer objecto LINK deve registar-se no objecto DATALINK existente para o canal de dados por forma a obter um identificador que lhe vai permitir partilhar o canal de comunicação de uma forma transparente-qualquer mensagem com este identificador recebida é-lhe encaminhada e qualquer mensagem enviada é recebida pelo outro extremo da ligação lógica. Este processo resulta, contudo, num *overhead* para o processo de recepção de mensagens, pois sempre que o objecto DATALINK recebe uma mensagem necessita de fazer o *polling* na tabela de LINKS para identificar o destino dessa mensagem, por forma a poder invocar o método TimeSlice() que vai permitir ao objecto LINK responder a essa mensagem e processar os respectivos dados associados. O *overhead* será tanto maior quanto maior o número de LINKS existentes, ocupando o CPU durante o processo de atribuição da mensagem o que faz com que o canal não seja ocupado durante esse tempo. Este intervalo de tempo é pouco significativo a menos que existam um grande número de LINKs simultaneamente.

BIBLIOGRAFIA:

Common-ISDN-API, Perfil Português, ver. 3.0

The Integrated Service Digital Network, Peter Bocker - 1988

Especificações da placa de processamento de imagem C30 - INESC-Porto

Sistema Integrado de Televigilancia, Manual de Instalação, Osvaldo Santos

ISDNLINK - Uma biblioteca de classes para RDIS, Osvaldo Santos

Redes Digitais com Integração de Serviços, Mário Nunes, Augusto Casaca