ISDNLINK - Uma biblioteca de classes para RDIS

Osvaldo A. Santos, Fernando M. S. Ramos

***Resumo*- Um programa genérico que utilize RDIS tem à sua disposição uma API que implementa apenas os protocolos de baixo nível da camada OSI. Este artigo descreve uma biblioteca de classes para Windows 3.x que implementa os serviços de alto nível geralmente utilizados em aplicações finais, tais como transferência de ficheiros e blocos de dados. Foi dada especial atenção à interface entre este módulo e o programa cliente, utilizando a estratégia método-evento, facilitada pelas potencialidades da linguagem C++, nomeadamente derivação de classes e métodos virtuais.**

A performance e flexibilidade foram outros aspectos tidos em conta, bem como aspectos de *multitasking* e operação em *background*.

***Abstract*- A generic program which uses ISDN is confronted with an API that implements the OSI layer lower protocols only. This paper describes a Windows 3.1x class library, that implements the high level services usually required in final applications, such as file and memory block transfer. A special attention was taken concerning the aspects related to the interface between this library and the client application, using mainly the event-method strategie, made easy by using C++ programming language, namely inheritance and virtual functions.**

Performance and flexibility were another focused aspects, as well as multitasking and background operation aspects.

I. INTRODUÇÃO

O acesso básico RDIS[1] 2B+D disponibiliza dois canais B de 64 Kbit/s e um canal D de 16 Kbit/s, tornando-o atraente para aplicações de Televigilância, Telemedicina, Videoconferência, acesso a redes locais, etc [1].

A carta PCBIT, desenvolvida pelo INESC, permite aceder aos recursos deste tipo de acesso, incluindo os serviços de voz. A comunicação entre as aplicações e a carta PCBIT é feita através de uma API[2] fornecida pelo fabricante, que na prática é uma DLL[3] chamada ISDNBIOS.DLL, que implementa as primitivas básicas de comunicação. Estas primitivas são: REGISTER, RELEASE, PUTMESSAGE, GETMESSAGE, SETSIGNAL, DEINSTALL, INISDNBIOS, GETMANUFACTURER, GETVERSION, GETSERIALNUMBER e MANUFACTURER. Todas elas têm associadas as suas respectivas estruturas de dados, que permitem a troca da informação necessária a cada tipo de primitiva.

Uma vez que a API implementa apenas os protocolos de baixo nível do modelo OSI (até à camada 3), é implicitamente pouco aconselhável o seu uso directo em aplicações finais, uma vez que o programador teria que implementar os protocolos de alto nível do módulo de comunicações.

II. SERVIÇOS DE ALTO NÍVEL QUE UMA APLICAÇÃO GENÉRICA NECESSITA.

Básicamente, os serviços que uma aplicação genérica necessita são os seguintes: estabelecimento de ligações no canal B, terminação de ligação, transmissão e recepção de mensagens de controlo, transmissão e recepção de blocos de dados de tamanho variável e transmissão e recepção de ficheiros.

Na construção desta biblioteca de classes foi procurado um equilíbrio entre flexibilidade, simplicidade, independência e performance, vectores estes que nem sempre se podem conjugar facilmente:

-Flexibilidade, para permitir um conjunto diverso de serviços, que eventualmente serão usados concorrentemente e em *background*.

-Simplicidade, traduzida na escrita do menor número possível de linhas de código no programa cliente, para usar determinado serviço RDIS.

-Independência, de maneira a tornar o seu uso geral, independentemente do programa cliente.

-Performance, para aproveitar ao máximo a largura de banda oferecida pela RDIS e ocupar ao mínimo o CPU da máquina, permitindo assim ao programa cliente usufruir do máximo de recursos da máquina.

Nos próximos pontos será mostrado como cada um destes objectivos foi atingido, respeitando sempre os outros vectores.

III. *MULTITASKING* COOPERATIVO.

O sistema operativo Windows 3.1x permite às aplicações correrem concorrentemente apenas em *multitasking* cooperativo (fig. 1): quando uma aplicação tem o controlo do CPU as outras aplicações param, a menos que tenham

[1] Rede Digital com Integração de Serviços

[2] *Aplication Programming Interface*

[3] *Dynamic Link Library*

rotinas de serviço a interrupção. Assim, uma aplicação só tem acesso ao CPU quando as outras aplicações deixarem.

Uma aplicação típica recebe mensagens do sistema operativo, processa cada uma dessas mensagens o mais rapidamente possível e devolve em seguida o controlo ao sistema operativo. Na maior parte do tempo só a aplicação activa recebe mensagens, que correspondem geralmente a eventos gerados pelo utilizador através da interface gráfica.

Assim, uma aplicação em *background* arrisca-se a não ter tempo de CPU de maneira a poder efectuar uma tarefa em tempo-real. Uma maneira de obter regularmente tempo de CPU é utilizar um *timer*. Esta facilidade permite que o *kernel* do sistema envie regularmente uma mensagem para a aplicação, mesmo quando esta se encontra em *background*, sendo o intervalo de tempo programável. No entanto, isto só é possível se as outras aplicações forem bem comportadas, ou seja, processarem rápidamente as

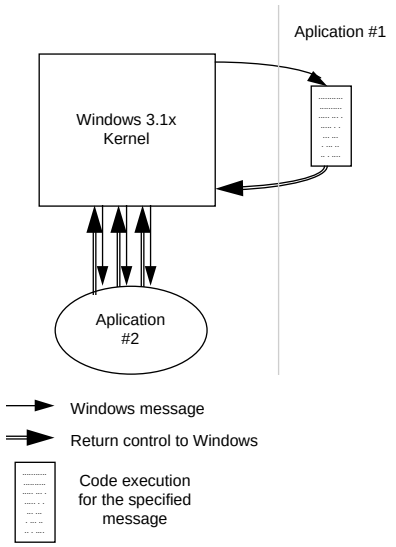

Fig. 1 - *multitasking* cooperativo no sistema operativo Windows 3.x..

mensagens e devolverem o controlo ao sistema.

IV. A ESTRATÉGIA MÉTODO-EVENTO.

Uma das estratégias mais simples de interagir com um módulo de software é a política método-evento (fig 2), utilizada com grande sucesso, por exemplo, no Microsoft Visual Basic. Através desta política, além de ser possível utilizar os serviços de determinado objecto, esse mesmo objecto pode ter capacidades de notificação de eventos, que permitem chamar funções do programa cliente sempre que seja necessário.

Assim, o programa cliente apenas tem que construir o código adequado para cada um dos eventos gerados pelo objecto, tal como no Visual Basic. Esta estratégia revelou-se de uma extrema simplicidade e flexibilidade, tal como será demonstrado.

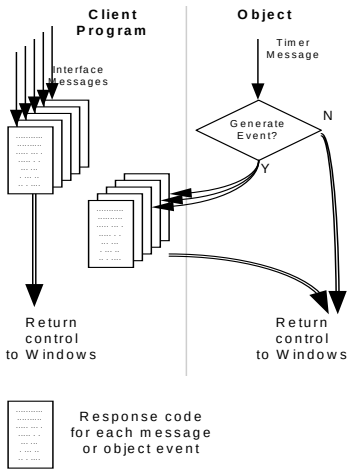

Fig. 2 - estratégia método-evento

V. VANTAGENS DA UTILIZAÇÃO DE C++.

Além das vantagens naturais de organização de software, tais como encapsulamento de dados, maior facilidade de manutenção ou capacidade de herança, a grande vantagem aplicada neste módulo é a capacidade de redefinir métodos virtuais da classe base. Esta capacidade é utilizada para permitir à classe base notificar as suas classes derivadas da existência de eventos. Como os métodos da classe derivada são definidos na aplicação cliente, o código a executar para cada evento é especifico de cada aplicação final que use a biblioteca.

VI. DIAGRAMA DE CLASSES.

A fig. 3 mostra a estrutura de classes da biblioteca, onde se podem ver as diferentes relações entre classes: herança ou uso. As classes virtuais ISDNLINK e LINK servem básicamente para permitir a interface entre as classes base PBITLINK e DATALINK e as suas respectivas classes derivadas.

As classes actuam como processos virtuais, uma vez que

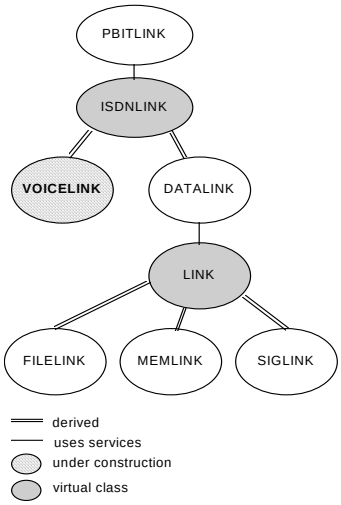

Fig. 3- diagrama de classes

obtêm regularmente tempo de CPU. Essa atribuição de tempo de CPU é feita chamando regularmente o método TimeSlice() de cada classe.

Para evitar que seja a aplicação cliente a fazer pooling a esses objectos, foi implementado um timer que chama o método TimeSlice() da classe PBITLINK em cada 20 ms (fig. 4). Os objectos da classe ISDNLINK, ou de classes derivadas desta, registam-se na classe PBITLINK quando são criados, ou seja, a classe PBITLINK sabe o endereço desses objectos. Então, basta chamar o método TimeSlice() de cada um destes objectos para garantir que todos eles têm tempo de CPU regularmente, comportando-se assim como processos virtuais, permitindo a operação em background.

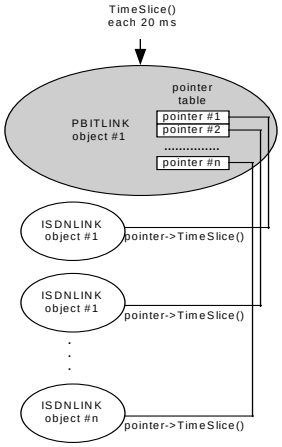

Fig. 4 - distribuição de tempo de CPU

VII. A CLASSE PCBITLINK.

Esta classe tem duas grandes funções: fazer a interface com a API e distribuir o tempo de CPU pelos objectos ISDNLINK. A distribuição de tempo de CPU já foi analisada anteriormente. Quanto à interface com a API, resume-se à distribuição de mensagens no sentido API -> objectos ISDNLINK, uma vez que estes objectos têm capacidade de usar directamente a API.

As mensagens podem ser caracterizadas em dois tipos de acordo com o seu destino: mensagens destinadas a um determinado objecto ISDNLINK, por exemplo as mensagens utilizadas durante a transferência de ficheiros, e mensagens com destino a um novo objecto ISDNLINK, por exemplo as mensagens recebidas durante a fase de estabelecimento de ligação. As mensagens são encaminhadas de acordo com o seu identificador de ligação, que é processado pela API. Quando as mensagens não têm identificador, isso significa que são mensagens de estabelecimento de ligação, e neste caso são direccionadas para o primeiro objecto ISDNLINK livre. A fig. 5 descreve o algoritmo de encaminhamento de mensagens.

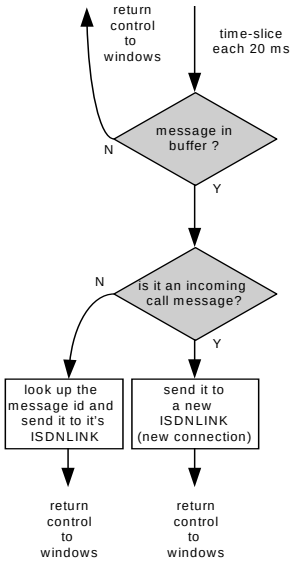

Fig. 5 -distribuição de mensagens na classe PBITLINK

VIII. A CLASSE BASE ISDNLINK.

A classe ISDNLINK tem como função a interface entre a classe PBITLINK que distribui as mensagens vindas da API e as classes DATALINK e VOICELINK que são as responsáveis respectivamente pelas ligações de voz e dados.

Segue-se a declaração da classe:

```
#ifdef ISDN_DLL
 #define TIPO_CLASSE _export
#else
 #define TIPO_CLASSE _import
#endif

class TIPO_CLASSE ISDNLINK
{
private:
   WORD ID;
   DWORD Father;
   WORD wRC; //API connection identifier.
   BYTE Uso; //Object state: FREE ou BUSY
   BYTE Estado;

public:

   far pascal _export ISDNLINK();
   far pascal _export ~ISDNLINK();
   SetLowerLevelLink(DWORD Ilink,WORD Id);

   virtual WORD TimeSlice(void);
   virtual LowerLayerIsDisconnecting();
   virtual GetEstado(void);
   virtual SetLowerLevelLink();
   virtual RemoveLowerLevelLink();
```

```
    virtual IL_ConnectInd(LPBYTE ptr);
    virtual IL_EstablishLapDInd(LPBYTE ptr);
    virtual IL_DataLapDInd(LPBYTE ptr);
    virtual IL_DisconnectInd(LPBYTE ptr);
    virtual IL_InfoInd(LPBYTE ptr);
    virtual IL_ConnectActiveInd(LPBYTE ptr);
    virtual IL_ConnectActiveConf(LPBYTE ptr);
    virtual IL_SelectProtocolConf(LPBYTE ptr);
    virtual IL_ReleaseLapDConf(LPBYTE ptr);
    virtual IL_DisconnectConf(LPBYTE ptr);
    virtual IL_ConnectConf(LPBYTE ptr);
    virtual IL_EstablishLapDConf(LPBYTE ptr);
    virtual IL_DataLapDConf(LPBYTE ptr);
};
```

Analisando o protótipo da classe, é muito importante referir que TIPO_CLASSE é <_export> ou <_import> conforme se está a compilar a biblioteca ou o programa cliente, permitindo assim exportar ou importar a classe. Esta particularidade é implementada em todas as classes. De salientar que nem todos os compiladores permitem importar classes de DLL's; foi usado o compilador Borland C++ 4.0 que é um dos que permitem esta facilidade. Os métodos SetLowerLevelLink(), LowerLayerIsDisconnecting() e RemoveLowerLevelLink() têm como função a ligação à classe de nível inferior, PBITLINK, e permitem a essa classe gerir a tabela de ISDNLINKs. O método TimeSlice(), presente em todas as classes, é utilizado para obter tempo de CPU, sendo chamado regularmente pela classe hierarquicamente inferior.

Os métodos virtuais IL_xxxxx correspondem a mensagens vindas da API, e são chamados pela classe PBITLINK quando faz a distribuição de mensagens. Esta distribuição tem em conta o identificador de ligação, a variável wRC. Como estes métodos são virtuais, são as classes derivadas de ISDNLINK que vão processar cada uma destas mensagens.

IX. A CLASSE DATALINK.

Esta classe faz a gestão de chamadas de dados sobre o canal B (fig. 6). As suas responsabilidades são: estabelecimento e terminação de ligações de dados incluindo a montagem dos protocolos adequados das camadas 2 e 3 do modelo OSI e suporte e gestão de objectos de nível superior, que correspondem às classes do tipo LINK. Os serviços que esta classe fornece aos objectos LINK são basicamente transmissão e recepção de pacotes.

O estabelecimento de uma ligação de dados em modo circuito é feito através da troca de mensagens apropriadas (figs.6, 7 e 8).

Os métodos e variáveis mais importantes desta classe são:

```
class TIPO_CLASSE DATALINK:public ISDNLINK
{
  private:

    //Callbacks
 virtual   ConnectActiveIndCallback(char   far
*address,char          far          *subaddress,
IEUserToUserSignalling UTUS);
 virtual ConnectActiveConfCallback(void);
 virtual DisconnectIndCallback(void);
 virtual DisconnectConfCallback(void);
 virtual ConnectConfCallback(void);
 virtual ConnectActiveProtocolsCallback(void);

  public:

 //Gestao de Links de nivel superior
 LINK *LinkTable[MAXIMO_LINKS];
 WORD LinksUsed;

 ConnectRequest(char   far   *Number,char   far
*SubAddress=""); //Vai para a API
 SendData(LPBYTE Data,WORD Len);
```

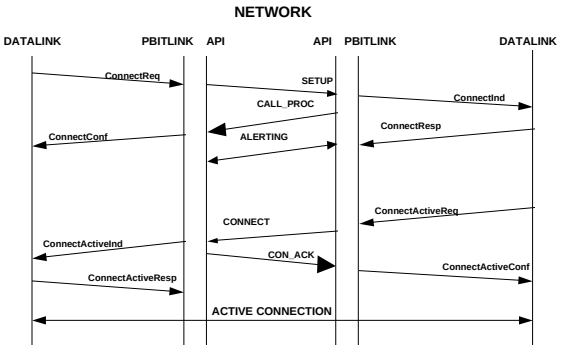

Fig. 6 - Sinalização no canal D necessária para o estabelecimento de uma ligação em comutação de circuitos no canal B.

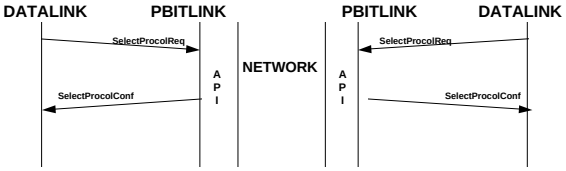

Fig. 7 - Selecção do protocolo do nível 2 da pilha de protocolos do utilizador.

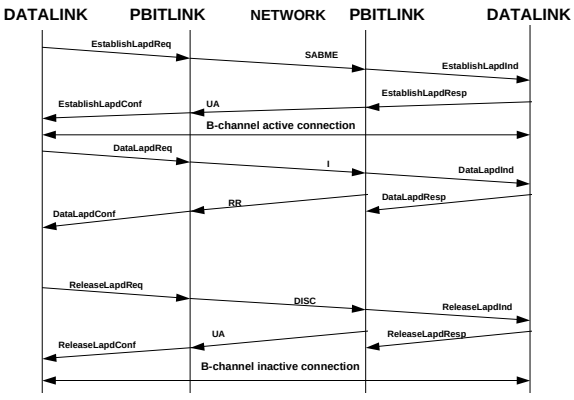

Fig. 8 - Mensagens (canal B) necessárias para o estabelecimento de uma ligação LAPD no canal B.

```
IL_ConnectInd(LPBYTE ptr);
IL_EstablishLapDInd(LPBYTE ptr);
IL_DataLapDInd(LPBYTE ptr);
IL_DisconnectInd(LPBYTE ptr);
IL_InfoInd(LPBYTE ptr);
IL_ConnectActiveInd(LPBYTE ptr);

IL_ConnectActiveConf(LPBYTE ptr);
IL_SelectProtocolConf(LPBYTE ptr);
IL_ReleaseLapDConf(LPBYTE ptr);
IL_DisconnectConf(LPBYTE ptr);
IL_ConnectConf(LPBYTE ptr);
IL_EstablishLapDConf(LPBYTE ptr);
IL_DataLapDConf(LPBYTE ptr);

SetLink(LINK *HLink,WORD Id);
TimeSlice();
} ;
```

Os métodos virtuais xxxxCallback() são usados para notificar a aplicação final do progresso da fase de estabelecimento e terminação de ligação. De modo a usar esta facilidade, a aplicação final terá que construir uma classe própria, derivada de DATALINK.

A tabela de ponteiros de LINKS, LinkTable, serve para fazer a gestão dos LINKs de nível superior, que vão usar esta classe como servidor de pacotes. Como foi utilizado um array de ponteiros para uma classe virtual, facilmente se implementam classes de alto nível que possam usar a classe DATALINK de um modo transparente. Um bom exemplo disto é a classe SIGLINK, que foi desenvolvida já depois desta biblioteca estar terminada. O tempo de desenvolvimento e integração foi de cerca de uma hora !

Os métodos IL_xxxxx são a implementação dos métodos virtuais da classe base ISDNLINK, e como cada um deles corresponde a uma mensagem trocada com a API, estes contêm o código resposta a essa mensagem. Estes métodos são chamados pela classe PBITLINK, de acordo com a política de distribuição de mensagens explicada anteriormente.

Assim, uma aplicação final terá que incluir a biblioteca de classes, e conter o seguinte código:

Exemplo 1: sem notificação.

```
DATALINK Channel1;

void ExemploEstabelecerLigacaoDados(void)
 {
   Channel1.ConnectRequest(“370563”,”0”);
 }

void ExemploTerminarLigacoes(void)
 {
   Channel1.DisconnectRequest();
 }
```

Exemplo 2: com notificação da aplicação cliente.

```
//prototyping

//criação de uma nova classe derivada de
DATALINK.
class B1:public DATALINK
{
  public:
      //métodos redefinidos
      WORD FAR PASCAL _export
   DisconnectIndCallback();
      WORD FAR PASCAL _export
   ConnectActiveProtocolsCallback(void);
   };

//main program

B1 Channel1;

void ExemploEstabelecerLigacaoDados(void)
 {
   Channel1.ConnectRequest(“370563”,”0”);
 }

void ExemploTerminarLigacoes(void)
 {
   Channel1.DisconnectRequest();
 }

//redefinição do método virtual
//callback de ligação activa, é chamada quando
os protocolos estão montados.
B1::ConnectActiveProtocolsCallback(void)
 {
  Interface(“Ligação activa”);

}

//redefinição do método virtual
//callback de terminação de ligação, é chamada
quando a ligação é terminada
B1::DisconnectIndCallback()
 {
  Interface("Ligação terminada");
}
```

X. A CLASSE BASE LINK.

Esta classe virtual tem como função a interface entre o DATALINK e as classes de nível superior FILELINK, MEMLINK, SIGLINK e outras que poderão ser implementadas fácilmente sobre esta estrutura.

Cada LINK (fig. 9) comporta-se como um canal de dados virtual, de largura de banda variável, que pode chegar até muito próximo dos 64 Kbit/s se, por exemplo, todos os outros LINK’s estiverem inactivos. Se vários LINKs estiverem activos, a largura de banda é distribuída

equitativamente, permitindo assim vários canais de transmissão de dados paralelos e independentes. Esta facilidade é importante, por exemplo, para a troca de mensagens de controlo em simultâneo com a transmissão de um ou mais ficheiros.

Analisando o protótipo da classe, facilmente é verificado o carácter de servidor de pacotes da classe DATALINK. De facto, o método PacketReception() da classe LINK é chamado da classe DATALINK sempre que a esta chega um pacote com esse destino. Além de servidora de pacotes, a classe DATALINK também atribui tempo de CPU para que cada objecto LINK envie pacotes, caso necessite. Esta atribuição de tempo de CPU permite, por exemplo, enviar ficheiros em *background.*

```
class TIPO_CLASSE LINK
{
  private:
   WORD ID;
   DWORD Father;

  public:
   far pascal _export LINK();
   far pascal _export ~LINK();

SetLowerLevelLink(DWORD Ilink,WORD Id);
   virtual    WORD    far    pascal    _export
PacketReception(LPBYTE Data,WORD Len);
   virtual    WORD    far    pascal    _export
TimeSlice(void);
   virtual    WORD    far    pascal    _export
LowerLayerIsDisconnecting();
};
```

Os LINKs são reconhecidos pelo seu identificador que não passa de um simples número. Assim, as mensagens vindas do LINK com o identificador 2 numa aplicação são enviadas pela classe DATALINK para o LINK com o identificador 2 na outra aplicação. O número máximo de LINKs é de 256 por cada DATALINK, permitindo assim, por exemplo, a transferência de 256 ficheiros simultâneamente, a uma taxa média de 250 bits/s. O número do LINK é atribuído no programa cliente quando é registado num DATALINK. Exemplo:

```
DATALINK Channel1;
FILELINK FileService;

//Aqui regista-se um LINK de transmissão de
ficheiros sobre o DATALINK Channel1 com o
identificador 5.

FileService.Init(&Channel1,5);
```

Para a classe DATALINK poder gerir o destino dos pacotes, foi introduzido o identificador do LINK no primeiro byte de cada pacote de dados. Os bytes seguintes são bytes de controlo específicos de cada tipo de LINK.

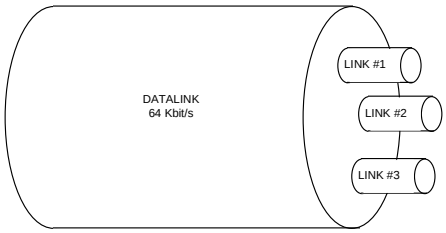

Fig. 9 - canais de dados virtuais sobre um DATALINK

XI. A CLASSE FILELINK.

Esta classe, derivada da classe LINK, é responsável pelo envio e recepção de ficheiros. Ambas as operações são efectuadas em *background*, deixando o processador livre para quaisquer outras operações. Os métodos principais desta classe são:

```
class TIPO_CLASSE FILELINK:LINK
 {
   ...
   public:

     WORD Init(DATALINK *LowLayer,WORD Id);
     WORD SendFile(char far *Name);

     ...
  virtual TxFileStart(char *Name);
  virtual TxFileEnd(WORD flag);
  virtual TxFileProgress(DWORD Bytes,DWORD
 Total);
   virtual RxFileStart(char *Name);
virtual RxFileEnd(WORD flag);
virtual  RxFileProgress(DWORD  Bytes,DWORD
Total);
 };
```

Para poder utilizar um objecto desta classe, é necessária a sua correcta inicialização sobre um objecto DATALINK. Para enviar um ficheiro basta chamar o método SendFile() com o nome do ficheiro como parâmetro. A recepção de ficheiros é feita automáticamente, bastando inicializar correctamente o objecto FILELINK.

Os métodos virtuais TxFile___ e RxFile___ têm como função a notificação da aplicação cliente do progresso da transmissão e recepção de ficheiros. Para utilizar esta facilidade basta criar uma nova classe na aplicação cliente, derivada de FILELINK, e redefinir estes métodos, escrevendo o código resposta a cada um deles.

Exemplo de utilização, pressupondo ligação estabelecida:

```
********* programa transmissor ********

DATALINK Channel1;
FILELINK FileService;
```

```
//Vai utilizar o identificador
//de link número 3 por exemplo

FileService.Init(&Channel1,3);
FileService.SendFile("teste.bmp");


********* programa receptor ********

DATALINK Channel1;
FILELINK FileService;

//Vai utilizar o identificador
//de link número 3

FileService.Init(&Channel1,3);

*********  programa  receptor  com  notificação
********

   //prototyping

   class FILET:FILELINK
    {
        RxFileStart(char *Name);
        RxFileEnd(WORD flag);
        RxFileProgress(DWORD Bytes,DWORD
   Total);
  };

//main program

DATALINK Channel1;
FILET FileService;

//Vai utilizar o identificador
//de link número 3

FileService.Init(&Channel1,3);


//redefinição do método virtual
FILET::RxFileStart(char *Name)
{
  Interface("Recebendo o ficheiro %s",Name);
}

//redefinição do método virtual
FILET::RxFileEnd(char *Name)
{
  Interface("Ficheiro %s transferido",Name);
}

//redefinição do método virtual
FILET::RxFileProgress(DWORD Bytes,DWORD Total)
{
  Interface("Recebendo %ld bytes de
%ld",Bytes,Total);
 }
```

XII. A CLASSE MEMLINK.

Esta classe é similar à classe FILELINK, excepto no tipo de informação transmitida: blocos de memória. Existem dois métodos de envio destes blocos:

```
class TIPO_CLASSE MEMLINK:LINK
 {
   ...
   public:
    SendData(HGLOBAL Hmem);
  SendData(HPBYTE Data,DWORD Size,WORD
Mode=MEM_NORMAL);
   ...
 }
```

O primeiro método é passar simplesmente ao objecto o *handle*[4] do bloco de memória desejado. O segundo método requer o ponteiro para o bloco e o tamanho do bloco. Este segundo método permite dois modos de operação: normal e modo cópia. No modo cópia o bloco é copiado, garantindo assim que os dados não são alterados durante a transmissão.

Na parte do receptor é alocado um bloco de memória com o tamanho necessário, sendo o *handle* desse bloco passado à aplicação cliente quando a transferência chega ao fim.

XIII. A CLASSE SIGLINK.

Esta classe fornece serviços de troca de mensagens de controlo, tipicamente com apenas alguns bytes de comprimento. O tamanho da mensagem foi limitado ao tamanho máximo de um pacote, que é 1920 bytes. Exemplo de utilização:

```
********* programa transmissor ********

DATALINK Channel1;
SIGLINK ControloCamara;

//Vai utilizar o identificador
//de link número 7 por exemplo

ControloCamara.Init(&Channel1,7);
...
ControloCamara.SendMessage(MOVE_RIGHT);
...
ControloCamara.SendMessage(STOP);


********* programa receptor ********
```

[4] identificador de um bloco de memóra protegida, alocada e gerida pelo sistema operativo.

```
//prototyping

class CAMCONTROL:SIGLINK
 {
   public:
    ReceiveMessage(LPBYTE Data,WORD Len);
 };

//main program

DATALINK Channel1;
CAMCONTROL ControloCamara;

//Vai utilizar o identificador
//de link número 7 por exemplo

ControloCamara.Init(&Channel1,7);

//redefinição do método virtual
CAMCONTROL::ReceiveMessage(LPBYTE Data,WORD Len)
 {
   ...
    //processamento das mensagens
   ...

 };
```

XIV. ANÁLISE DE DESEMPENHO.

Para avaliar o desempenho da biblioteca foi feito um conjunto de testes, envolvendo medida de tempos de estabelecimento e terminação de ligação, e transferência de ficheiros e blocos de memória.

O primeiro teste consistiu em medir os tempos de estabelecimento e terminação de ligação. Os testes foram feitos entre dois PC's 486DX2 66MHz, usando a rede pública. Num dos PC's foi medido o tempo gasto, utilizando o relógio interno da máquina.

Connect Time (D channel)	Connect Time (D+B channels)	Disconnect Time (B+D channels)
3.07 s	4.12 s	220 ms

Tabela 1 - tempos de estabelecimento e terminação de ligação, usando a rede pública.

O segundo teste foi a transferência de um único ficheiro, medindo o tempo gasto nessa transferência. Analisando a tabela 2 conclui-se que a taxa de transferência foi 62.3 Kbit/s, com uma eficiência de 62.3/64.00 = 97.3%. Não foi atingida uma eficácia de 100% devido aos tempos de leitura e escrita no disco duro, bem como ao *overhead* de controlo presente em todos os pacotes enviados para a rede.

	Size (bytes)	Time spent (s)	Transfer rate (Kbit/s)	Eficiency (%)
File	2,698,537	346.3	62.3	97.3
Memory Block	384,000	48,3	63.6	99.4

Tabela 2 - desempenho da transferência de ficheiros e blocos de memória

O terceiro teste consistiu em transferir um bloco de memória, medindo o tempo de transmissão. Analisando a tabela 2, verifica-se um aumento da eficácia para quase 100%, devido à inexistência de operações de leitura ou escrita em disco, sendo o *overhead* de controlo dos pacotes o único responsável pelos 0,6% de desperdício da largura de banda disponível no canal B.

XV. CONCLUSÕES.

Os objectivos iniciais foram cumpridos: foi construída uma biblioteca de classes que permite integrar facilmente comunicações RDIS em aplicações finais, de uma forma rápida e flexível. Apesar de ser implementada numa linguagem orientada a objectos, não se registaram efeitos negativos a nível de velocidade de execução.

De momento esta biblioteca está a ser usada no projecto Televigilância para Windows e Sistema de Videotelefonia, estando prevista a sua extensão aos projectos Teleradiologia e Teletrabalho.

A evolução desta biblioteca prevê um módulo de compressão de dados em tempo real, o que permitirá um aumento da largura de banda virtual média para cerca de 128 Kbit/s, se considerarmos uma taxa de compressão média de 2:1. Um outro aspecto importante será a possibilidade de atribuição de prioridades aos LINK's, permitindo à aplicação final definir quais as operações mais importantes.

REFERÊNCIAS

[1] Mário Serafim Nunes, Augusto Júlio Casaca, "Redes Digitais com Integração de Serviços", Editorial Presença, 1992.