Uso della unit Toolbar

Due anni fa, dal gennaio al marzo 1992, avevamo messo a punto alcune unit che aiutavano a realizzare applicazioni MDI dotate di ribbon e riga di stato, sotto Windows 3.0. Quando giunse la versione 3.1, però, si dovette constatare che quelle unit non funzionavano più correttamente. Ritorniamo ora sull'argomento, sia per eliminare i problemi di allora, sia per giungere ad un'mplementazione più moderna e flessibile

di Sergio Polini

Sono poche le applicazioni Windows che non usano l'approccio MDI, mediante il quale è possibile far coesistere in una stessa applicazione diversi tipi di attività - scrittura in una finestra, grafica in un'altra, e così via - e diverse viste sull'oggetto di ognuna di tali attività. È ugualmente diffuso l'uso di strumenti quali «nastri» *(ribbon)*, «righelli» *(ruler)*, «barre di strumenti» *(toolbar)* e «righe di stato» *(status line)*, tutti tesi a facilitare l'interazione con l'utente. Questo è appunto il motivo per cui, due anni fa, avevamo cercato di approntare alcune unit che, estendendo la funzionalità di alcune classi già presenti in ObjectWindows ed aggiungendone una nuova, permettevano di realizzare facilmente applicazioni MDI con un nastro dotato di pulsanti o di altri controlli e con una riga di stato nella quale, tra l'altro, veniva mostrata una sintetica descrizione dei comandi associati alle opzioni dei menu.

Già in un demo del Resource Workshop, RWPDEMO, si poteva trovare anche allora un esempio di riga di stato realizzata mediante una bitmap; ne risultava un effetto estetico forse non esaltante, ma certo più interessante di quello offerto da una semplice linea di separazione tra la *client window* e il rettangolo dedicato alla comunicazione con l'utente; il meccanismo mediante il quale si proponevano all'utente informazioni sulle opzioni dei menu, tuttavia, lasciava un po' a desiderare: ne rimanevano escluse, ad esempio, le opzioni dei *system menu* sia della finestra principale che delle *child window*.

Tra i demo del Borland Pascal 7.0, per altro verso, si trova ora un'applicazione MFILEAPP in cui, grazie all'uso di una unit TOOLBAR, si propone all'utente un'interessante e più moderna variante del nostro vecchio *ribbon*. I meccanismi che presiedono all'orientamento ed alla visualizzazione del *toolbar* sono piuttosto flessibili e, soprattutto, si propongono esplicitamente come un approccio generale all'inserimento di righe, nastri e barre in un'applicazione MDI (basta leggere il commento al metodo *TToolbar.AMCalcParentClientRect*).

La riga di stato di RWPDEMO, oltre ai limiti già detti, non è compatibile con le barre strumenti come gestite dalla unit TOOLBAR; sembrerebbe ragionevole, quindi, affiancare a questa una unit STATLINE che ne condivida la logica, in modo da semplificare l'architettura generale dell'applicazione.

A questo scopo, è necessario procedere in primo luogo ad un esame della unit TOOLBAR. Scopriremo così che questa non è tanto flessibile quanto

```
unit ToolBar;

interface

{$R Toolbar.res}

uses Winprocs, Wintypes, Objects, OWindows, Strings, Win31;

const
  am_CalcParentClientRect = wm_User + 120;
  tbHorizontal   = $01;
  tbLeftVertical = $02;
  tbRightVertical= $04;
  DenyRepaint  = 0;
  AllowRepaint = 1;

type

  PTool = ^TTool;
  TTool = object(TObject)
    ...
  end;

  PToolbar = ^TToolbar;
  TToolbar = object(TWindow)
    ...
  end;

  PToolSpacer = ^TToolSpacer;
  TToolSpacer = object(TTool)
    ...
  end;

  PToolButton = ^TToolButton;
  TToolButton = object(TTool)
    ...
  end;
```

Figura 1 - L'interfaccia della unit TOOLBAR.

promette, ma anche che non è difficile trovare rimedio ai suoi limiti.

Le costanti e le classi

L'interfaccia della unit TOOLBAR comprende la dichiarazione di sei costanti e quattro classi (figura 1; prescindiamo, per brevità, dalla costante di tipo *TStreamRec*).

Tre costanti sono usate per precisare la posizione della barra strumenti: *tbHorizontal* per una barra orizzontale, posta subito sotto la barra del menu; *tbLeftVertical* e *tbRightVertical* per una barra verticale, posta, rispettivamente, a sinistra o a destra della *client window*. Ricordiamo che, in un'applicazione MDI, la finestra principale - *frame window* - serve solo da sfondo; tra essa e le finestre in cui viene mostrato l'output del programma - le *child window* - si interpone una *client window* che, normalmente, occupa tutta la *client area* della *frame window*, cioè quanto di essa rimane se si escludono il titolo, il menu e i bordi; le singole finestre - come i documenti di *Word* o i fogli e i grafici di *Excel* - sono «figlie» della *client window* e, quando ingrandite, non possono oltrepassare i suoi bordi; è possibile quindi, restringendo la *client window*, tenere aperte altre finestre che rimangano sempre visibili, dedicando ad esse zone della *frame window* non coperte dalla *client window*; per dirla in altro modo, la *client window* ed eventuali altre finestre, come una barra di strumenti o una riga di stato, sono «figlie» della *frame window*, mentre le finestre dedicate all'output del programma sono tutte «figlie» della *client window*.

La costante *am_CalcParentClientRect* denota appunto il messaggio che, in occasione di un cambiamento di dimensioni della finestra principale, chiede alle finestre «figlie» di questa, diverse dalla *client window*, di comunicare di quanto spazio hanno bisogno e, quindi, come e quanto va ridotta l'area da destinare alla *client window*.

Le costanti *DenyRepaint* e *AllowRepaint* vengono usate quando viene inviato il messaggio *am_CalcParentClientRect*, per distinguere tra una semplice richiesta di informazioni circa le dimensioni residue della *client window* ed una richiesta di ridisegnare tutto il contenuto della *frame window*; nel primo caso, finestre come la barra strumenti e la riga di stato si limitano a comunicare quanto spazio occupano, nel secondo provvedono anche a ridisegnare se stesse.

La classe *TTool* è una classe astratta, nel senso che l'implementazione di molti dei suoi metodi, ridotta ad un semplice **begin end**, è rimandata alle classi derivate, quali *TToolSpacer* e *TToolButton*.

La classe *TToolButton* viene usata per i pulsanti destinati a trovare collocazione nel *Toolbar*, ma potrebbero essere derivati da *TTool* altri tipi di «strumenti»; *TToolSpacer* viene usata tra uno strumento ed un altro, quando li si vuole mostrare distanziati invece che adiacenti.

La classe *TToolbar*, derivata da *TWindow*, è usata per la barra che deve ospitare gli strumenti. Vedremo che si possono creare, in una stessa applicazione, più istanze della stessa classe o di classi da questa derivata, anche se, per fare ciò, occorrono alcuni interventi sui sorgenti.

La classe TToolbar

Una barra di strumenti viene realizzata come una normale finestra, per la quale si crea, mediante il metodo *GetWindowClass*, una apposita classe caratterizzata da uno sfondo grigio (ciò si ottiene avvalorando il campo *hbrBackground* di *TWndClass* con l'handle ritornato da *GetStockObject(LtGray_Brush)*). Va sottolineata, tuttavia, un'importante accortezza.

Le finestre che si aprono in un'applicazione MDI, infatti, sono «figlie» della *client window*; in quanto tali, hanno settato il flag *wb_MDIChild* e, per i comportamenti non ridefiniti, ricorrono alla procedura *DefMDIChldProc*. Il constructor di *TToolbar*, quindi, provvede ad azzerare quel flag ed a riassegnare alla variabile *DefaultProc* l'indirizzo della procedura *DefWindowProc*, in modo da garantire, ad esempio, che la barra non obbedisca ai comandi tipici di un menu *Finestre*, come «affianca», «sovrapponi», «chiudi tutto», ecc.

Si attribuisce alla barra, tra gli altri, lo stile *ws_Border*, ma, al tempo stesso, si assegna il valore -1 ai campi X e Y della variabile d'istanza *Attr*; ne segue che, non potendo venire disegnato ciò che giace fuori della *frame window*, apparirà solo il bordo interno: una vera e propria linea di separazione tra la barra e altri ospiti della *frame window*, la *client window* in primo luogo.

Gli strumenti vengono implementati come elementi di una collezione *Tools*, creati dal metodo *ReadResource*. Questo cerca una risorsa di tipo TOOLBARDATA, che non appartiene al corredo standard e, quindi, può essere creata solo mediante un file di testo con estensione RC e con il formato illustra-

```
PToolbar = ^TToolbar;
TToolbar = object(TWindow)
  ResName: PChar;
  Tools: TCollection;
  Capture: PTool;
  Orientation: Word;
  constructor Init(AParent:PWindowsObject; AName:PChar; Orient:Word);
  destructor Done; virtual;
  constructor Load(var S: TStream);
  procedure Store(var S: TStream); virtual;
  function  CreateTool(Num: Integer; Origin: TPoint; Command: Word;
    BitmapName: PChar): PTool; virtual;
  procedure EnableTool(Command: Word; NewState: Boolean); virtual;
  procedure FreeResName;
  function  GetClassName: PChar; virtual;
  procedure GetWindowClass(var WC: TWndClass); virtual;
  procedure SetResName(NewName: PChar);
  procedure NextToolOrigin(Num: Integer; var Origin: TPoint;
    P: PTool); virtual;
  procedure Paint(DC: HDC; var PS: TPaintStruct); virtual;
  procedure ReadResource; virtual;
  function  GetOrientation: Word;  virtual;
  procedure SetOrientation(NewOrient: Word);  virtual;
  procedure SwitchTo(NewName: PChar);
  procedure AMCalcParentClientRect(var Msg: TMessage);
    virtual wm_First + AM_CalcParentClientRect;
  procedure WMLButtonDown(var Msg: TMessage);
    virtual wm_First + wm_LButtonDown;
  procedure WMMouseMove(var Msg: TMessage);
    virtual wm_First + wm_MouseMove;
  procedure WMLButtonUp(var Msg: TMessage);
    virtual wm_First + wm_LButtonUp;
end;
```

Figura 2 - L'interfaccia della classe TToolbar.

```
Toolbar_1 ToolBarData LOADONCALL MOVEABLE DISCARDABLE
BEGIN
      8
      tbHelp
      cm_Help
      0
      8
      tbFileOpen
      cm_MDIFileOpen
      tbFileSave
      cm_FileSave
      0
      8
      tbEditCut
      cm_EditCut
      tbEditCopy
      cm_EditCopy
      tbEditPaste
      cm_EditPaste
END
```

Figura 3 - La definizione di un insieme di pulsanti per un toolbar (tratta dal file MFILEAPP.RC). Il primo numero indica il numero degli strumenti, le successive coppie comprendono nomi simbolici per le bitmap dei pulsanti (iniziano con tb) e per i comandi ad essi associati (iniziano con cm_). Le coppie (0,8) indicano assenza di bitmap e dimensioni dello spazio da lasciare tra il pulsante precedente ed il successivo.

to nelle figure 3 e 4. Letta la prima *word*, che dà il numero dei pulsanti *(TToolButton)* che dovranno apparire, legge per ognuno di essi due *word*: la prima denota la bitmap con la quale il pulsante verrà rappresentato, la seconda il comando ad esso associato; se il primo numero è zero, si intende che, in luogo di un pulsante, va creata una istanza di *TToolSpacer*, con una dimensione pari al valore del secondo numero. Pulsanti e spazi vengono collocati l'uno a destra dell'altro, oppure l'uno sotto l'altro, secondo l'orientamento della barra. Orientamento che, peraltro,

```
TOOLBAR_1 TOOLBARDATA LOADONCALL MOVEABLE DISCARDABLE
BEGIN
     '08 00 F5 01 E6 03 00 00 08 00 F6 01 0C 5F'
     'F7 01 0D 5F 00 00 08 00 F8 01 00 5F F9 01 01 5F'
     'FA 01 02 5F'
END
```

Figura 4 - La risorsa della figura 3 come definita se non si ricorre a nomi simbolici (le coppie di numeri sono sottolineate).

può essere cambiato durante l'esecuzione mediante il metodo *SetOrientation*.

Il meccanismo, per quanto complesso, è efficace. Va solo sottolineato che, ai fini di una armonica rappresentazione dei pulsanti nella barra, si assume che le bitmap siano quadrate e, soprattutto, che abbiano una dimensione 20x20. Pulsanti più piccoli o più grandi apparirebbero decentrati rispetto all'asse orizzontale o verticale della barra (secondo il suo orientamento).

I comandi associati ai pulsanti possono essere abilitati o disabilitati con il metodo *EnableTool*, che chiama il metodo *Enable* del primo *tool* che risponde affermativamente al metodo *HasCommand* (ciò comporta che non vi possono essere più pulsanti con uno stesso comando: assunzione più che ragionevole).

La barra risponde al messaggio *WMLButtonDown* cercando l'eventuale pulsante su cui si trova il mouse per azionarlo con il metodo *BeginCapture*; al messaggio *WMMouseMove* per comunicare i movimenti del mouse con il messaggio *ContinueCapture*; al messaggio *WMLButtonUp* per segnalare l'evento al pulsante con *EndCapture*. Al resto, come vedremo, pensa la classe *TToolButton*.

È stata opportunamente prevista la possibilità di cambiare i pulsanti presenti sulla barra durante l'esecuzione. Provvede a ciò il metodo *SwitchTo*, che azzera e reinizializza la collezione *Tools*, per poi creare i nuovi pulsanti leggendone la descrizione in un'altra risorsa di tipo TOOLBARDATA.

La flessibilità dell'intero impianto è notevole, con un unico serio limite. Alcune applicazioni Windows (penso ora a *Word*, con cui sto scrivendo) hanno sia un nastro che una barra strumenti; entrambi potrebbero essere realizzati come istanze di *TToolbar*, eventualmente derivando nuove classi da *TTool*. La possibilità può essere verificata aggiungendo una variabile *Toolbar2* alla classe *TMDIFileWindow*, in MFILEAPP.PAS, e una riga al constructor di questa, come indicato nella figura 5.

L'effetto, tuttavia, non sarà pienamente conforme alle aspettative. Nel caso di una barra orizzontale ed una verticale, ad esempio, si noterà un imperfetto accostamento delle due barre che, invece di disporsi l'una adiacente all'altra, in parte si sovrapporranno. Ancora peggio se tentiamo di creare due barre orizzontali: ne vedremo una sola.

La causa è da ricercare nel metodo *AMCalcParentClientRect*. Questo, stando al lungo commento che lo accompagna, risponde al messaggio omonimo calcolando lo spazio occupato da barre di vario tipo quale che sia la loro posizione, e riducendo conseguentemente quello occupato dalla *client window*; se *Msg.WParam* è diverso da zero, inoltre, ogni barra viene ridisegnata per tener conto delle nuove dimensioni (la *frame window* invia il messaggio *am_CalcParentClientRect* quando deve a sua volta rispondere ad un messaggio *wm_Size*). L'implementazione è un po' contorta, in quanto si tende a ridisegnare solo quanto strettamente necessario (ad esempio, solo la parte destra di una barra orizzontale sita in una finestra che si sia allargata verso destra), in modo da ridurre al minimo il tremolio dello schermo. Va tutto bene, ma si assume che una barra orizzontale sia sempre posta nella parte superiore della *frame window*, subito sotto il menu. Ecco quindi che due barre orizzontali si trovano ad occupare lo stesso spazio.

Torneremo sull'argomento quando vedremo come una riga di stato dovrà rispondere a quel messaggio; per il momento potete trovare nella figura 6 una prima possibile variante: un'implementazione nettamente più semplice, che mostra sia gli inconvenienti di un codice non attento a ridurre al minimo il tremolio delle immagini sullo schermo, sia, per altro verso, la possibilità di visualizzare correttamente più di una barra strumenti (o una barra e un nastro, come preferite).

La classe TToolButton

Poco da dire sulla classe *TToolSpacer*: serve solo a tenere memoria dello spazio da interporre tra due pulsanti (il metodo *Paint*, ad esempio, non fa nulla).

Più interessante la classe *TToolButton*. Il constructor carica la bitmap indicata da *TToolbar.ReadResource*, ne registra le dimensioni e la posizione sulla barra, prende nota del comando associato.

L'abilitazione o la disabilitazione del comando, come anche i movimenti del mouse notificati da *TToolbar*, si traducono nell'avvaloramento delle variabili d'istanza *IsEnabled* e *IsPressed* e nel ridisegno del pulsante.

Questo avviene mediante il metodo *PaintState*, chiamato direttamente in caso di azione del mouse, oppure indirettamente, mediante invalidamento e conseguente *repaint* della zona occupata sulla barra, nel caso di abilitazione o disabilitazione di un comando. La bitmap, quando vi si passa sopra con il mouse, viene disegnata con offset diversi da zero rispetto ai lati superiore e sinistro del rettangolo occupato dal pulsante, in modo da rendere visivamente la pressione; il rettangolo viene patinato

```
constructor TMDIFileWindow.Init(ATitle: PChar; AMenu: HMenu);
begin
  ...
  ...
  Toolbar := New(PToolbar, Init(@Self, 'Toolbar_1', tbHorizontal));
  Toolbar2 := New(PToolbar, Init(@Self, 'Toolbar_2', tbLeftVertical));
end;
```

Figura 5 - Come modificare il constructor della classe TMDIFileWindow, *nel file MFILEAPP.PAS, per avere due barre strumenti, una orizzontale e l'altra verticale. Ma l'effetto non sarà del tutto conforme alle aspettative...*

di grigio - mediante un apposito *brush* - se il comando ad esso associato è disabilitato.

L'effetto è sicuramente accettabile; in particolare, nonostante le bitmap del demo (contenute in MFILEAPP.RES) siano colorate, la resa è soddisfacente anche su video monocromatico.

Quando viene rilasciato il pulsante sinistro del mouse, *TToolbar* chiama il metodo *EndCapture* passando come argomenti l'handle della *frame window* e le coordinate del cursore del mouse; se questo è ancora sulla bitmap, *EndCapture*, mediante *PostMessage*, invia alla *frame window* un messaggio uguale a quello che sarebbe stato prodotto dalla scelta di un'opzione di un menu. Nulla vieta, peraltro, che istanze di una classe derivata da *TToolButton* producano messaggi di altro tipo.

L'applicazione

Passata in rassegna la unit TOOLBAR, resta da vedere come questa va usata in un'applicazione che intenda servirsi di una barra di strumenti.

In primo luogo, occorre prevedere una variabile di tipo *PToolbar* nella classe che, derivando da *TMDIWindow*, fornirà la finestra principale; la variabile verrà inizializzata nel constructor, come abbiamo visto nella figura 5.

Sarà anche utile prevedere metodi come *HorizontalToolbar*, *LeftVerticalToolbar* e *RightVerticalToolbar*, per rispondere ai comandi dati dall'utente per cambiare l'orientamento della barra. Nel caso di due barre, potremmo volere che di una sola di queste si possa cambiare l'orientamento (la barra strumenti vera e propria), lasciando l'altra sempre orizzontale (il «nastro»); ciò si ottiene creando per prima la barra che dovrà mantenere sempre lo stesso orientamento. Per tornare alla figura 5, ciò comporta che, essendo i comandi riferiti alla variabile *Toolbar* - ed essendo quindi *Toolbar2* destinata a rimanere sempre orizzontale - la riga con cui viene creata *Toolbar2* dovrebbe essere posta prima di quella con cui viene creata *Toolbar*. Usare la modifica proposta nella figura 6 e provare per credere.

Una volta scelto l'orientamento iniziale della barra «mobile», sarà bene marcare la corrispondente opzione del menu (ad esempio: «Barra orizzontale» in un menu «Opzioni») mediante la funzione *CheckMenuItem* nel metodo *SetupWindow* della finestra principale. Analogamente, ad ogni abilitazione o disabilitazione di comandi si dovrà accompagnare la chiamata del metodo *Toolbar^.EnableTool*.

I cambiamenti nell'orientamento di una barra verranno realizzati nascondendo la barra, comunicandole la nuova disposizione, ricalcolando le dimensioni della *client window* e ridisegnando la barra. Il calcolo delle dimensioni della *client window* va inoltre effettuato, ovviamente, in risposta ad un messaggio *wm_Size*. Nel file MFILEAPP.PAS si usa un metodo *RedoClientRect* che, ottenute con *GetClientRect* le dimensioni della *client area* della finestra principale, le passa a tutte le finestre «figlie» di questa mediante il metodo *ForEach*; ogni «figlia» (e si tratta, ripetiamo, delle finestre figlie della finestra principale, non delle normali *child window* di un'applicazione MDI, che sono figlie della *client window*) si posiziona secondo il proprio orientamento e riduce di conseguenza le dimensioni che riceve; al termine dell'iterazione, quindi, le dimensioni ridotte vengono assegnate alla *client window*.

Rimando, per ulteriori dettagli, al demo MFILEAPP.PAS. La rassegna fin qui condotta aveva solo lo scopo di fornire una guida alla lettura dei file forniti con il compilatore, per apprezzarne gli aspetti più interessanti e quelli suscettibili di una diversa implementazione.

Tra trenta giorni vedremo come inserire nel meccanismo una «barra» per molti aspetti diversa ma, per altri, caratterizzata da comportamenti simili. MC

```
procedure TToolbar.AMCalcParentClientRect( var Msg: TMessage);
var
  PC, R, TB, NewTB: TRect;
  CX, CY: Integer;
begin
  PC := PRect(Msg.LParam)^;      (* client coord *)
  R := PC;
  ClientToScreen(Parent^.HWindow, PPoint(@PC)^);
  ClientToScreen(Parent^.HWindow, PPoint(@PC.Right)^);
  GetWindowRect(HWindow, TB);

  if Orientation = tbHorizontal then begin
    CX := R.Right - R.Left;
    CY := TB.Bottom - TB.Top;
    SetWindowPos(HWindow, 0,
      R.Left, R.Top, CX, CY,
      swp_NoZOrder or swp_NoRedraw);
    Attr.W := CX;
    InvalidateRect(HWindow, nil, True);
    Inc(R.Top, CY);
  end
  else if Orientation = tbLeftVertical then begin
    CX := TB.Right - TB.Left;
    SetWindowPos(HWindow, 0,
      R.Left, R.Top, CX, R.Bottom - R.Top,
      swp_NoZOrder or swp_NoRedraw);
    InvalidateRect(HWindow, nil, True);
    Inc(R.Left, CX);
  end
  else if Orientation = tbRightVertical then begin
    CX := TB.Right - TB.Left;
    SetWindowPos(HWindow, 0,
      R.Right - CX, R.Top, CX, R.Bottom - R.Top,
      swp_NoZOrder or swp_NoRedraw);
    InvalidateRect(HWindow, nil, True);
    Dec(R.Right, CX);
  end;
  SetRect(PRect(Msg.LParam)^, R.Left, R.Top, R.Right, R.Bottom);
end;
```

Figura 6 - Una prima modifica del metodo TToolbar.AMCalcParentClientRect, *per ottenere la corretta visualizzazione di più di una barra strumenti (o di una barra strumenti e un nastro).*

Sergio Polini è raggiungibile tramite MC-link alla casella MC1166 e tramite Internet all'indirizzo MC1166@mclink.it.