Controlli ActiveX

Come sviluppare un componente ActiveX in Visual Basic, per incapsulare le funzionalità di un orologio analogico

di Francesco Sblendorio

Francesco Sblendorio *è studente di Informatica presso l'Università degli Studi di Bari. Si occupa di programmazione in C/C++, Visual Basic e di problematiche legate al Web. Può essere contattato tramite e-mail all'indirizzo sblendorio@infomedia.it*

Riuso, riuso, riuso. Questa è la parola chiave alla base della creazione di controlli ActiveX, ossia componenti software riutilizzabili in qualunque ambiente di programmazione che ne supporti l'utilizzo. Oggi è infatti possibile scrivere componenti in Visual Basic, per poi usarli in Delphi, C++ Builder, MFC e così via. Da un punto di vista pratico, un ActiveX è simile ad un qualunque altro controllo disponibile nella palette di quelli classici Visual Basic e pertanto esporterà al programmatore un insieme di proprietà, metodi ed eventi. L'anima di un ActiveX è il modello COM (Component Object Model), un protocollo la cui specifica è descritta accuratamente in diversi articoli di MSDN (si vedano [1] e [2]).

LA CREAZIONE DI UN OROLOGIO ACTIVEX

In questo articolo vogliamo creare un "controllo orologio" che consenta di monitorare costantemente l'ora, visualizzando un quadrante analogico con tre lancette (si osservi l'icona del controllo in basso a destra nella palette, in **Figura 1**). Dopo aver aperto il familiare IDE del VB, creiamo un nuovo progetto di tipo *Componente ActiveX*. Dal menu *File* scegliamo poi *"Aggiungi progetto"* di tipo *"EXE Standard"*: si desidera, infatti, poter usufruire di un progetto standard EXE su cui effettuare il "collaudo" del nuovo componente in fase di creazione. Occupiamoci adesso del progetto vero e proprio del componente. Notiamo che la finestra di progettazione contiene un oggetto che somiglia ad un Form senza bordo. Questo oggetto, di tipo *UserControl*, si chiama *UserControl1* ed è la tavolozza dove disegneremo l'aspetto esteriore del componente.

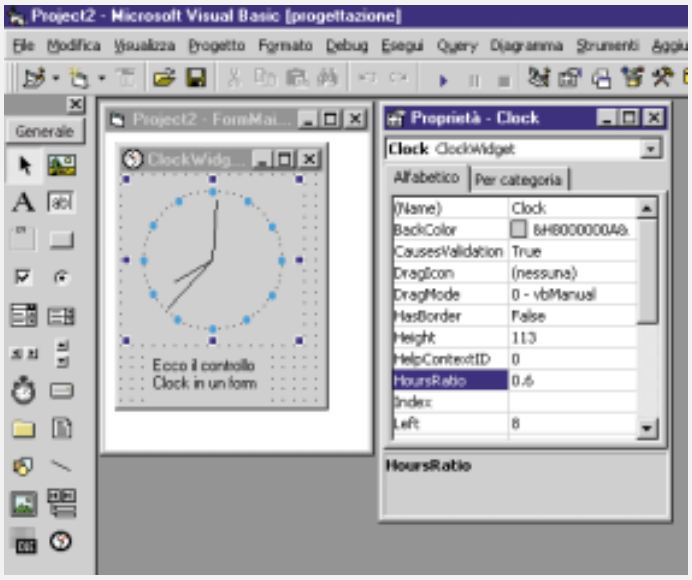
IL DISEGNO DELL'INTERFACCIA

Cominciamo subito con il disegno dell'interfaccia grafica del controllo, per poi interessarci della stesura del codice. Per avere un'idea su quali controlli posizionare su *UserControl* e sui nomi da assegnare si osservi la **Figura 2**: diamo a *UserControl* il nome *ClockWidget*, posizioniamo tre controlli *Line* chiamandoli *LineHours, Line Minutes, LineSeconds* ed infine un controllo *Timer* a cui diamo il nome *TimeScanner*, che servirà per scandire il tempo.

Come si può notare, le lancette sono posizionate in maniera del tutto casuale e non vi è traccia del quadrante. Ciò è normale, in quanto quest'ultimo verrà disegnato tramite codice ad ogni ridimensionamento (evento *Resize*) dello *UserControl* e le lancette verranno posizionate a *run-time* durante la scansione dei secondi (evento *Timer* del controllo *TimeScanner*).

Per terminare il "disegno" dell'interfaccia dobbiamo settare alcune proprietà dei controlli appena posizionati: lo *UserControl* (*ClockWidget*) deve avere la proprietà *AutoRedraw* impostata a *True*: questo serve a rendere "permanente" il disegno del quadrante che verrà effettuato da codice; un'altra proprietà di UserControl importante da impostare è *ScaleMode*, che va settata a

Figura 1 Il controllo ClockWidget terminato



Pixel: serve ad indicare l'unità di misura in cui si indicano le dimensioni dei controlli in esso contenuti.

La classe PropertyBag ha due metodi: WriteProperty e ReadProperty

Per personalizzare l'icona che dovrà apparire nella palette dei controlli, è sufficiente impostare la proprietà *ToolboxBitmap* con l'immagine bmp preferita. Il controllo *TimeScanner* deve avere la proprietà *Interval* impostata a 500; questo valore indica il numero di millesimi di secondo che separano due eventi *Timer*: Ci si potrebbe chiedere perché non è stata impostata a 1000; il motivo è che il controllo *Timer* non è preciso e si potrebbe vedere che ogni tanto la lancetta dei secondi si sposterebbe ogni due secondi anziché uno. Impostando *Interval* a 500 si migliorano un po' le cose; per avere una scansione precisa dei secondi bisognerebbe ricorrere a tecniche che esulano dagli scopi di questo articolo.

IL CODICE

Ora possiamo finalmente scrivere il codice che si occuperà di "dar vita" al nostro controllo. Descriveremo:

- i tipi di dati
- le costanti e le variabili private che verranno utilizzate all'interno delle varie routine
- le routine di utilità da utilizzare all'interno di quelle per la gestione degli eventi
- le routine *Property* e le routine di gestione degli eventi.

I TIPI DI DATI, LE COSTANTI, LE VARIABILI PRIVATE

Prima di cominciare a scrivere le definizioni dei tipi, iniziamo il codice con *Option Explicit* che ci eviterà molti problemi dovuti all'utilizzo di variabili non definite.

Poi cominciamo a descrivere i seguenti tipi:

```
' TIPI PRIVATI
Private Type Point ' Modella un punto dello spazio
    X As Long
    Y As Long
End Type

Private Type MetaInstant ' Modella CARATTERISTICHE
    Hours As Double     ' di ore, minuti secondi
    Minutes As Double
    Seconds As Double
End Type
```

Il tipo *Point* è un *record* che contiene le coordinate *X* ed *Y* di un punto dello spazio bidimensionale, mentre il tipo *MetaInstant* serve per *descrivere* delle caratteristiche di un *orario*; una variabile di tipo *MetaInstant* può contenere ad esempio:

- l'ora attuale (ore, minuti, secondi)
- la dimensione in pixel delle tre lancette
- la dimensione percentuale delle tre lancette

Vediamo ora quali sono le costanti che saranno utilizzate:

```
' VALORI PRECALCOLATI
Private Const pi As Double = 3.14159265358979     'pigreco
Private Const pi_2 As Double = 1.5707963267949    'pigreco/2
Private Const pi_6 As Double = 0.523598775598299 'pigreco/6
Private Const pi_30 As Double = 0.10471975511966 'pigreco/30

' IMPOSTAZIONI DI DEFAULT
Private Const default_ratioHours As Double = 3 / 5    'dim.
Private Const default_ratioMinutes As Double = 8 / 10 ' %
Private Const default_ratioSeconds As Double = 8 / 10 'default
```

Infine vediamo quali sono le variabili private:

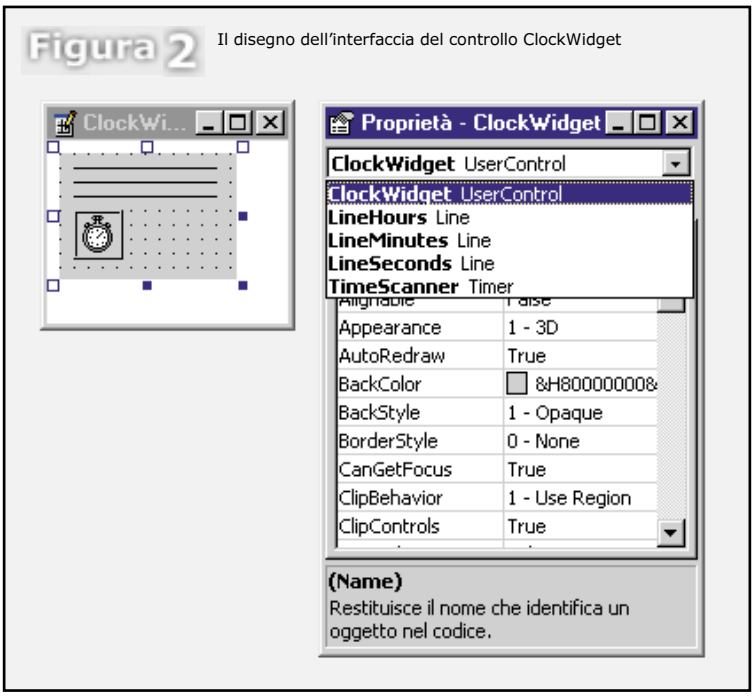
```
' Centro del controllo
Private center As Point

' Dimensione percentuale delle lancette
Private hand_ratio As MetaInstant

' Dimensione in pixel delle lancette (calcolata)
Private hand_size As MetaInstant
```

Guardando *hand_ratio* e *hand_size* sarà più chiaro l'utilizzo del tipo di dati *MetaInstant*.

LE ROUTINE DI USO GENERALE

Verranno riportati qui i segmenti di codice più significativi oltre a tutte le interfacce delle routine, e non il codice integrale (esso è disponibile su internet all'indirizzo *ftp://ftp.infomedia.it/DEV./listati)*. Le due routine "generali" (ovvero quelle che non sono né di gestione di eventi né *property*) sono la funzione *Minimo* e la *sub Com-*

Figura 2 Il disegno dell'interfaccia del controllo ClockWidget



Listato 1 Il codice del controllo ClockWidget

```
Option Explicit

' TIPI PRIVATI
Private Type Point  ' Modella un punto dello spazio
   X As Long
   Y As Long
End Type

Private Type MetaInstant  ' Modella delle CARATTERISTICHE di
                                        ore, minuti, secondi
   Hours As Double
   Minutes As Double
   Seconds As Double
End Type

' COSTANTI PRIVATE
Private Const pi As Double = 3.14159265358979     ' pigreco
Private Const pi_2 As Double = 1.5707963267949    ' (1/2)  *
                                                    pigreco
Private Const pi_6 As Double = 0.523598775598299  ' (1/6)  *
                                                    pigreco
Private Const pi_30 As Double = 0.10471975511966  ' (1/30) *
                                                    pigreco
Private Const default_ratioHours As Double = 3 / 5
                                                  ' Dimensione
Private Const default_ratioMinutes As Double = 8 / 10
                                                  ' percentuale
Private Const default_ratioSeconds As Double = 8 / 10
                                                  ' di default

' VARIABILI PRIVATE
Private center As Point              ' Centro del controllo
Private hand_ratio As MetaInstant    ' Dimensione percentuale
                                                delle lancette
Private hand_size As MetaInstant     ' Dimensione in pixel
                                     delle lancette (calcolata)

' METODI PRIVATI

' Restituisce il minimo tra "x" ed "y"
Private Function Minimo(X As Long, Y As Long) As Long
   If X < Y Then
      Minimo = X
   Else
      Minimo = Y
   End If
End Function

' Parametri di input: hand_size as MetaInstant
'       Rappresenta le dimensioni delle varie lancette
                                        (ore, minuti, secondi)
' Parametri di output: NewHour, NewMinute, NewSecond    as
'                                                      Point
' Sono le coordinate che devono avere i vertici (x,y) delle
'                                                     lancette
'       in un sistema cartesiano con origine (0,0) per indicare
'      l'orario corrente di sistema
Private Sub ComputeCoords(hand_size As MetaInstant, _
        ByRef NewHour As Point, ByRef NewMinute As Point, _
                                    ByRef NewSecond As Point)
   Dim now As MetaInstant

   now.Hours = Hour(Time) Mod 12
   now.Minutes = Minute(Time)
   now.Seconds = Second(Time)

   now.Hours = now.Hours + (now.Minutes / 60#) +
                                       (now.Seconds / 3600#)
   now.Minutes = now.Minutes + (now.Seconds / 60#)

   NewHour.X = -hand_size.Hours * Cos(pi_2 +
                                          now.Hours * pi_6)
   NewHour.Y = -hand_size.Hours * Sin(pi_2 +
                                          now.Hours * pi_6)

   NewMinute.X = -hand_size.Minutes * Cos(pi_2 +
                                        now.Minutes * pi_30)
   NewMinute.Y = -hand_size.Minutes * Sin(pi_2 +
                                        now.Minutes * pi_30)

   NewSecond.X = -hand_size.Seconds * Cos(pi_2 +
                                       now.Seconds * pi_30)
   NewSecond.Y = -hand_size.Seconds * Sin(pi_2 +
                                       now.Seconds * pi_30)
End Sub

' Riposiziona le lancette
Private Sub TimeScanner_Timer()
   Dim Hours As Point, Mins As Point, Secs As Point

   ComputeCoords hand_size, Hours, Mins, Secs

   LineHours.X2 = center.X + Hours.X
   LineHours.Y2 = center.Y + Hours.Y
   LineMinutes.X2 = center.X + Mins.X
   LineMinutes.Y2 = center.Y + Mins.Y
   LineSeconds.X2 = center.X + Secs.X
   LineSeconds.Y2 = center.Y + Secs.Y

End Sub

Private Sub UserControl_Initialize()
   hand_ratio.Hours = default_ratioHours
   hand_ratio.Minutes = default_ratioMinutes
   hand_ratio.Seconds = default_ratioSeconds

   UserControl_Resize
End Sub

Private Sub UserControl_ReadProperties(PropBag As
                                              PropertyBag)
   Dim border As Boolean

   hand_ratio.Hours = PropBag.ReadProperty
                                          ("HoursRatio")
   hand_ratio.Minutes = PropBag.ReadProperty
                                        ("MinutesRatio")
   hand_ratio.Seconds = PropBag.ReadProperty
                                        ("SecondsRatio")
   UserControl.BackColor = PropBag.ReadProperty
                ("BackColor", UserControl.BackColor)
   border = PropBag.ReadProperty("HasBorder", False)
   If border Then
      UserControl.BorderStyle = 1
   Else
      UserControl.BorderStyle = 0
   End If

   UserControl_Resize
End Sub

Private Sub UserControl_Resize()
   Dim sizeHalfMin As Long

   ' Ricalcolo: Le coordinate del centro del controllo
   '                                          (halfX,halfY)
   '          Il raggio della circonferenza inscritta nel
   '                                  controllo (sizeHalfMin)
   '          Le dimensioni delle lancette (sizeHours,
   '                                sizeMinutes, sizeSeconds)
   center.X = UserControl.ScaleWidth / 2
   center.Y = UserControl.ScaleHeight / 2
   sizeHalfMin = Minimo(center.X, center.Y)
   hand_size.Hours = (sizeHalfMin * hand_ratio.Hours)
   hand_size.Minutes = (sizeHalfMin *
                                        hand_ratio.Minutes)
   hand_size.Seconds = (sizeHalfMin *
                                       hand_ratio.Seconds)

   ' Disegna il quadrante
   Dim i As Integer
   Dim gradi As Double
   Dim final As Point
   Dim radius As Long  'radius = raggio del quadrante

   radius = sizeHalfMin - 5
   UserControl.Cls
   For i = 1 To 60
      gradi = i * pi_30
      final.X = center.X + (radius * Cos(gradi))
      final.Y = center.Y + (radius * Sin(gradi))
      If (i Mod 5 = 0) Then
         UserControl.DrawWidth = 5
         UserControl.PSet (final.X, final.Y),
                                        RGB(50, 150, 200)
      Else
         UserControl.DrawWidth = 1
         UserControl.PSet (final.X, final.Y), RGB(0, 0, 0)
```

(continua)

Un controllo ActiveX creato con Visual Basic può essere usato anche in Delphi, C++ Builder, ecc.

Listato 1 ...prosecuzione Listato 1

```
    End If
  Next

  ' Riposiziona il centro delle lancette
  LineHours.X1 = center.X
  LineHours.Y1 = center.Y
  LineMinutes.X1 = center.X
  LineMinutes.Y1 = center.Y
  LineSeconds.X1 = center.X
  LineSeconds.Y1 = center.Y

  ' Ridimensiona e riposiziona le lancette
  ' (richiamando il codice associato al timer)
  TimeScanner_Timer
End Sub

' Proprietà dell'oggetto
Public Property Get HoursRatio() As Double
  HoursRatio = hand_ratio.Hours
End Property

Public Property Get MinutesRatio() As Double
  MinutesRatio = hand_ratio.Minutes
End Property

Public Property Get SecondsRatio() As Double
  SecondsRatio = hand_ratio.Seconds
End Property

Public Property Let HoursRatio(H_Ratio As Double)
  If H_Ratio > 0 And H_Ratio <= 1 Then
    hand_ratio.Hours = H_Ratio
    UserControl_Resize
  End If
End Property

Public Property Let MinutesRatio(M_Ratio As Double)
  If M_Ratio > 0 And M_Ratio <= 1 Then
    hand_ratio.Minutes = M_Ratio
    UserControl_Resize
  End If
End Property
Public Property Let SecondsRatio(S_Ratio As Double)
  If S_Ratio > 0 And S_Ratio <= 1 Then
    hand_ratio.Seconds = S_Ratio
    UserControl_Resize
  End If
End Property

Public Property Get HasBorder() As Boolean
  If UserControl.BorderStyle = 1 Then
    HasBorder = True
  Else
    HasBorder = False
  End If
End Property

Public Property Let HasBorder(b As Boolean)
  If b Then
    UserControl.BorderStyle = 1
  Else
    UserControl.BorderStyle = 0
  End If
End Property

Private Sub UserControl_WriteProperties(PropBag
                                        As PropertyBag)
  PropBag.WriteProperty "HoursRatio", hand_ratio.Hours
  PropBag.WriteProperty "MinutesRatio", hand_ratio.Minutes
  PropBag.WriteProperty "SecondsRatio", hand_ratio.Seconds
  PropBag.WriteProperty "BackColor", UserControl.BackColor
  If UserControl.BorderStyle = 1 Then
    PropBag.WriteProperty "HasBorder", True
  Else
    PropBag.WriteProperty "HasBorder", False
  End If
End Sub

Public Property Get BackColor() As OLE_COLOR
  BackColor = UserControl.BackColor
End Property

Public Property Let BackColor(newcolor As OLE_COLOR)
  UserControl.BackColor = newcolor
  UserControl_Resize
End Property
```

puteCoords le cui interfacce sono le seguenti:

```
Private Function Minimo(X As Long, _
                        Y As Long) As Long
Private Sub ComputeCoords(hand_size
  As MetaInstant, ByRef NewHour As Point, _
                  ByRef NewMinute As Point, _
                  ByRef NewSecond As Point)
```

Tralasciamo la prima, il cui significato è ovvio; la *ComputeCoords* è una *sub* che ha un parametro di input, *hand_size* di tipo *MetaInstant*, e tre parametri di output, *NewHour, NewMinute, NewSecond* di tipo *Point*.

In questo caso *hand_size* contiene le dimensioni in *pixel* delle tre lancette.

L'unità di misura è il *pixel* in virtù del valore della proprietà *ScaleMode* dello *UserControl*. Come output della routine otterremo le coordinate delle estremità delle tre lancette, che verranno calcolate (1) in base alle dimensioni contenute in *hand_size* e (2) in base all'ora corrente, ricavata con la variabile di sola lettura *Time*.

LE ROUTINE PROPERTY

Questo tipo di routine può risultare nuovo a chi non ha mai scritto una classe od un controllo *ActiveX* in Visual Basic.

Da un punto di vista astratto un controllo ActiveX può essere visto come una classe

Esse sono routine che, quando richiamate, sembrano in realtà degli attributi di un oggetto. Mi spiego meglio con un esempio. Possiamo posizionare su di un *form* vuoto (che ad esempio si chiama FormMain) un *CommandButton* che nell'evento *Click* abbia:

```
FormMain.Width = FormMain.Width - 10
```

Sembra che *Width* sia un attributo dell'oggetto *FormMain*, mentre esistono *due* routine property per gestire "Width", una per la lettura ed una per la scrittura. In *Visual Basic* le routine property (o routine di proprietà) si scrivono così (facciamo l'esempio di *Width*):

```
Public Property Let Width(newWidth
                  as Long) ' per scrivere
  ...
  ... ' viene usata newWidth per
                        "fare qualcosa"
  ...
End Property

Public Property Get Width() as Long
                          ' per leggere
  ...
  ...
  Width = ... ' viene restituito il valore
End Property
```

Se il tipo del valore da scrivere

è una *classe,* allora va usato un terzo tipo di routine, *Property Set* anziché *Property Let*; per la lettura invece non cambia niente. Di solito una proprietà nasconde una variabile privata, ma questo non sempre è vero. Per ulteriori informazioni sull'uso delle routine *Property* si veda [1].

Ora vediamo quali sono le proprietà definite nel controllo. Cominciamo con le proprietà per impostare la lunghezza delle lancette: queste sono *Hours Ratio*, *MinutesRatio* e *SecondsRatio* che indicano la dimensione *percentuale* delle lancette rispetto alla dimensione del controllo, e quindi devono essere valori compresi tra 0 ed 1.

La loro implementazione è molto semplice in quanto non fanno altro che nascondere i singoli campi della variabile privata *hand_ratio;* le routine *Property Get* non fanno altro che restituire i singoli campi del record, mentre le *Property Let*, prima di assegnare i nuovi valori, controllano che questi siano compresi tra 0 ed 1. Il nostro controllo ha altre due proprietà: *HasBorder* e *BackColor*. La prima è di tipo booleano: se esso è *True*, il controllo avrà un bordo tridimensionale; la seconda proprietà imposta il colore di sfondo.

Queste due proprietà *non* nascondono variabili private, ma le proprietà dello stesso *UserControl* che non sarebbero altrimenti accessibili. La proprietà *HasBorder* nasconde la *BorderStyle* di *UserControl*: la nostra è di tipo booleano, mentre *Border Style* è di tipo intero (può contenere i valori 0, 1, 2); dunque, quando impostiamo *HasBorder* a *True* si assegna 1 a *BorderStyle* (che vale 0 altrimenti).

In lettura vale il viceversa.

BackColor nasconde la proprietà omonima dello UserControl: non c'è alcuna ambiguità in quanto ci si riferisce a due oggetti diversi; il tipo di questa proprietà, *OLE_COLOR* sembra un po' strano: internamente è rappresentato come un *long* ma permette di far visualizzare la palette dei colori quando, in fase di utilizzo del controllo, voglio impostarne il valore. Quando questa proprietà viene letta (*Get*) viene semplicemente restituito il valore di *UserControl. BackColor*, mentre quando viene impostata (*Let*), oltre a settare la corrispondente proprietà di *UserControl*, viene forzato il redraw del controllo, tramite la routine di evento *UserControl_Resize*.

LE ROUTINE DI GESTIONE DEGLI EVENTI

Prima di procedere con l'esame delle routine di gestione di eventi del nostro controllo, vediamo quali sono gli eventi più importanti da gestire in un generico controllo *ActiveX*:

UserControl_Inizialize
: Effettua l'inizializzazione del controllo: imposta variabili private e proprietà a valori di default ed altre "amenità" del genere. Viene generato *la prima volta* che il controllo viene creato (il significato di questa frase verrà spiegato tra poco).

UserControl_ReadProperties
: Viene richiamato quando il controllo viene *ricreato*, cioè quando viene creato dopo la prima volta; contiene codice che serve per reimpo-

stare le proprietà del controllo, che vengono perse quando il controllo viene *distrutto*.

UserControl_WriteProperties
Viene richiamato quando il controllo viene distrutto; contiene codice che serve a memorizzare in una apposita struttura dati (*PropertyBag*) i valori delle proprietà, che verranno ripristinati da *UserControl_ReadProperties*.

Spieghiamo ora cosa significano le espressioni "*la prima volta che il controllo viene creato*" e "*dopo la prima volta*".

Le routine property, quando sono richiamate, sembrano in realtà degli attributi di un oggetto

Prendiamo ad esempio il noto controllo *FlexGrid*; viene creato la prima volta quando lo si seleziona dalla palette e lo si posiziona sul form: le sue proprietà vengono impostate a valori di default (nel caso della *FlexGrid* sono il numero di righe, di colonne, ecc.): questa operazione è effettuata dalla routine di gestione dell'evento *Initialize*. Quando noi chiudiamo la finestra di un form contenente il controllo in questione (ad esempio per ridurre il caos sullo schermo), il controllo viene *distrutto*: è in questo momento che viene generato l'evento *WriteProperties*, la cui routine di gestione si occupa di *salvare* le proprietà del controllo, che altrimenti verrebbero perse.

Quando poi decidiamo di riaprire la finestra del form suddetto, il controllo viene *ricreato*, e la routine di gestione dell'evento *ReadProperties* si occupa di ripristinare i valori delle proprietà precedentemente impostati. Vediamo come funzionano le routine di gestione degli eventi *WriteProperties* e *ReadProperties*. Entrambe hanno un argomento, *PropBag* di classe *PropertyBag*; i metodi di questa classe sono *WriteProperty* e *ReadProperty*.

Vediamo come funzionano:

```
PropBag.WriteProperty "nome proprietà", valore
```

Memorizza il valore indicato nella proprietà che ha il nome corrispondente alla stringa specificata.

```
val = PropBag.ReadProperty ("nome proprietà", default)
```

Restituisce il valore della proprietà specificata dalla stringa; se la proprietà indicata non è stata memorizzata in *PropBag* viene restituito il valore di default specificato.

Ora siamo in grado di esaminare il codice del nostro controllo *ClockWidget*. Nelle routine di gestione di *WriteProperties* semplicemente salviamo i valori delle dimensioni percentuali delle lancette (*HoursRatio*, *MinutesRatio* e *SecondsRatio*), il colore di sfondo (*BackColor*) ed il valore booleano HasBorder, per ottenere il quale bisogna però prima leggere la proprietà *BorderStyle* di *UserControl*. Nella routine di gestione di *ReadProperties* viene fatta l'operazione inversa; in virtù di quanto descritto prima, non credo sia necessario dilungarsi ancora sulla gestione di questi due eventi. La routine *UserControl_Initialize* si occupa di impostare le dimensioni delle lancette ai valori di default visti prima nelle costanti ed a richiamare la routine *UserControl_Resize*, che si occupa del redraw del controllo.

Vediamo ora le due routine più importanti del nostro controllo *ClockWidget*, *TimeScanner_Timer* e *UserControl_Resize*: la prima, molto semplice, viene richiamata circa ogni mezzo secondo, e tramite la *ComputeCoords* vista in precedenza calcola la nuova posizione di ogni lancetta e le sposta in modo appropriato. La seconda invece si occupa del ridimensionamento del controllo; attenzione: stiamo parlando del ridimensionamento del controllo che avviene in fase di progettazione, *non* di quello che avviene durante l'esecuzione come potrebbe essere per un form. In particolare vengono ricalcolati: il centro del controllo, il raggio della circonferenza inscritta nel controllo, la dimensione *in pixel* delle lancette; in seguito viene ridisegnato il quadrante, tramite i metodi *Cls*, *PSet* e la proprietà *DrawWidth* di *UserControl*; viene riposizionato il centro delle lancette ed infine, tramite la routine *TimeScanner_Timer* vengono riposizionate le lancette.

CONCLUSIONI

I controlli *ActiveX* sono importanti perché indipendenti dal linguaggio utilizzato e sono semplici da usare: ci sono altri modi di creare componenti software, ad esempio le DLL (Dynamically Linked Library), il cui uso non è però immediato come quello dei controlli *ActiveX*.

Questi ultimi hanno comunque anche un lato negativo: la loro richiesta di risorse è superiore a quella di una DLL; uno dei motivi che può spingere all'uso di controlli *ActiveX* è la necessità di un'interfaccia grafica per il componente o la possibilità di generare eventi (che in questo articolo abbiamo tralasciato). Ovviamente, quando vogliamo creare un componente software dobbiamo analizzare le nostre necessità e poi scegliere l'approccio da adottare.

BIBLIOGRAFIA

[1] Nicola Martella, *"Programmi di Classe in Visual Basic"*, DEV n. 74.
[2] Peter Aitken, *"Programmare con Visual Basic 5"*, Tecniche Nuove, 1997, ISBN 88-481-0702-80.