Code reuse: deriving new visual components to suit special needs

Lect.drd. Cristian USCATU
Catedra de Informatică Economică, ASE Bucureşti

All Windows programming environments come with a built-in hierarchy of classes that suit common needs for Windows applications (buttons, edit fields etc.) They may be combined in any way to suit the needs of particular applications. What if a certain combination is needed repeatedly, in several applications? One way is to create a library that contains the needed functionality (classes, subprograms etc.) and include it in projects. Since visual and RAD environments are en vogue, it would be nice to have these special functionalities readily available inside the environment. Borland environments make use of a Visual Component Library (Delphi name - name may vary among products) quickly available in a customizable palette. Beside the original components, third party components and user created components can be included in this palette for future use.
Keywords: *Delphi, RAD, components, code reuse, object, class.*

Introducere

Din punctul de vedere utilizării claselor se disting două momente: cel al dezvoltării aplicaţiei şi cel al execuţiei. O clasă obişnuită este manipulată strict prin scrierea de cod şi instanţiată doar în momentul execuţiei aplicaţiei; la dezvoltarea ei nu există un feed-back vizual. Orice nepotrivire (detectabilă doar la execuţie) înseamnă reluarea procesului de dezvoltare. În mediile vizuale, elementele cu impact vizual în aplicaţie (şi nu numai ele) sînt create şi manipulate chiar în timpul dezvoltării aplicaţiei, reducînd mult timpul şi efortul de dezvoltare (aceste medii sînt numite RAD – Rapid Application Development. Multe medii se pretind a fi de tip vizual sau RAD, dar puţine au de fapt această caracteristică.). Elementele care pot fi folosite astfel sînt numite *componente*. O componentă poate fi aproape orice element de program care se doreşte a fi manipulat în timpul dezvoltării aplicaţiei (design-time).

În fond o componentă este o clasă instalată în biblioteca de componente vizuale, fiind astfel recunoscută de mediul de programare care poate să o manipuleze în timpul dezvoltării aplicaţiei. Toate componentele sînt clase derivate (pe diferite niveluri) din clasa generică *TComponent*, parte a ierarhiei bibliotecii de componente vizuale.

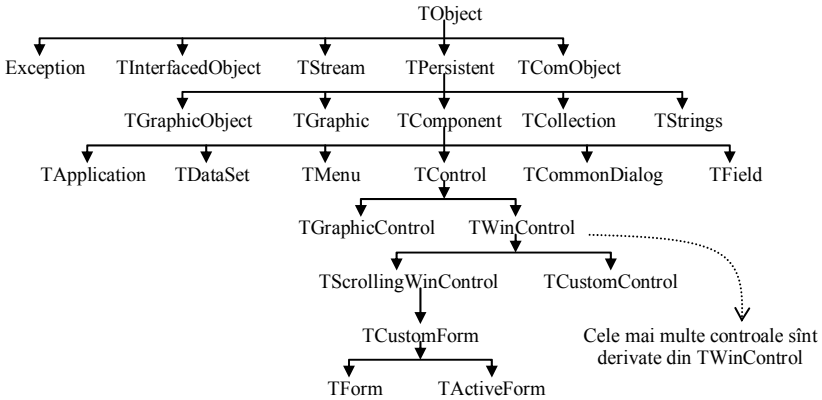

Fig.1. Ierarhia de la baza Bibliotecii de Componente Vizuale

Figura nu prezintă decît partea de bază a ierarhiei deoarece partea superioară este pe de o parte foarte stufoasă, pe de altă parte poate sa varieze foarte mult de la un calculator la altul, in funcţie de nevoile dezvoltatorului respectiv.

Pentru a crea o componentă nouă se derivează o clasă nouă pornind de la una din cla-

sele ierarhiei. Crearea unei noi componente presupune lucrul cu clasele la un alt nivel decît în cazul dezvoltării unei aplicații. În principiu se lucrează cu părți ale claselor inaccesibile programatorilor de aplicații (partea privată) şi se adaugă noi elemente claselor (de exemplu proprietăți). O atenție specială trebuie acordată respectării unor convenții şi imaginării modului în care programatorii de aplicații vor utiliza componenta respectivă.

Crearea unei noi componente

Pentru crearea unei noi componente pot fi urmate mai multe căi, în funcție de nevoi:

- Modificarea unei componente existente (în principiu se modifică valorile implicite ale unor proprietăți). În acest caz se porneşte de la o componentă existentă sau o componentă abstractă de pe un nivel inferior al ierarhiei.
- Crearea unei componente-fereastră. În acest caz se porneşte de la clasa *TWinControl*.
- Crearea unui control grafic. În acest caz se porneşte de la clasa *TGraphicControl*.
- Derivarea unui control existent. În acest caz se porneşte de la orice control existent.
- Crearea unei componente nevizuale. În acest caz se porneşte de la clasa *TComponent*.

În crearea unei componente trebuie realizate următoarele elemente:

- Eliminarea dependențelor. O componentă poate fi inclusă într-o mare varietate de aplicații, în multiple combinații, de aceea ea trebuie să fie cît mai generală posibil. Restricțiile generale (implementate în componentă) trebuie reduse la minim. Restricțiile particulare (specifice unei aplicații) trebuie implementate la dezvoltarea aplicației. Timpul necesar pentru crearea unei componente fără dependențe este timp cîştigat de dezvoltatorii de aplicații, care sînt scutiți de sarcini repetitive, de rutină.
- Stabilirea proprietăților, metodelor, evenimentelor. Proprietățile constituie o formă de acces controlat la atributele clasei. Controlul accesului elimină sursa multor erori şi face mai uşoară viața dezvoltatorilor de aplicații. Metodele descriu funcționalitatea clasei iar evenimentele descriu modul de reacție la evenimente externe.
- Încapsularea elementelor grafice. Delphi uşurează modul de lucru cu elemente grafice prin includerea unei proprietăți numite *Canvas* (pînză pentru pictură) – obiect din clasa *TCanvas* – care încapsulează toate elementele necesare (creion, pensulă, font etc.) şi gestionează automat resursele grafice necesare.
- Înregistrarea componentei în paleta de componente prin apelul procedurii *RegisterComponent.*

Crearea unei noi componente se poate face manual sau prin intermediul unui expert inclus în Delphi, care automatizează procesul prin crearea scheletului noii componente. Paşii necesari sînt:

1. Crearea unui unit pentru noua clasă.
2. Derivarea clasei din una existentă.
3. Adăugarea proprietăților, metodelor şi evenimentelor.
4. Înregistrarea componentei în paleta de componente.
5. Crearea unei pictograme care să reprezinte componenta în paletă.
6. Crearea unui pachet (o bibliotecă cu legare dinamică specială) care să instaleze componenta în paletă.
7. Crearea unui *Help* care să ajute utilizatorii în folosirea componentei (opțional).

Componenta TMySpinEdit

Pentru exemplificare, se cere crearea unei componente care să aibă următoarea funcționalitate: să conțină un cîmp pentru editarea unei valori numerice de tip real, cu posibilitatea incrementării/decrementării ei cu un pas fix, comandată prin folosirea mouse-ului sau a tastelor săgeți sus/jos. Delphi conține o componentă pentru editare (*TEdit*) şi o componentă formată din două butoane marcate cu săgeți în sensuri opuse (*TUpDown*). Prin atașarea lor se poate obține funcționalitatea dorită, dar pentru numere de tip întreg. De remarcat faptul că prin atașarea lor nu se obține o componentă nouă, ci două componente care lucrează împreună.

Pentru atingerea scopului dorit trebuie creată o componentă nouă, derivată din *TEdit* şi care să conțină un obiect de tip *TUpDown*. În plus, sînt necesare atribute de tip real pentru valoare, pas, valoare minimă şi valoare ma-

ximă. Trebuie adăugate metodele care să codifice funcționalitatea dorită și care sa răspundă la apăsarea săgeților (cu mouse-ul sau tastele).

Componentă de tipul *TEdit*

Componentă de tipul *TUpDown*

Componentă de tipul *TUpDown* atașată de o componentă *TEdit*

Fig.2. Componente de tipul *TEdit* și *TUpDown*, la dezvoltarea aplicației

În figura 3 se observă expertul pentru crearea scheletului unei noi componente (accesibil prin intermediul meniului *Component->New Component*) și informațiile pe care acesta le solicită.

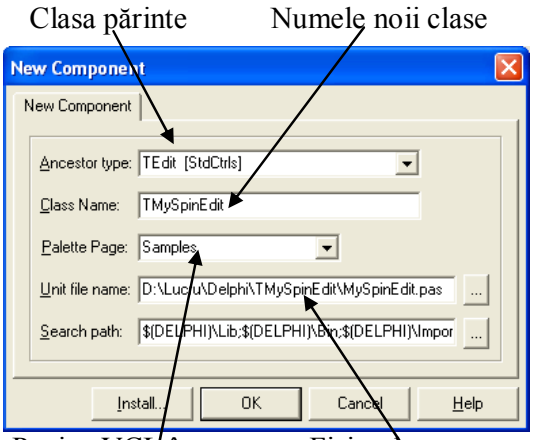

Fig.3. Expertul pentru crearea unei noi componente

După completarea informațiilor se obține următorul schelet:

```
unit MySpinEdit;
interface
uses
  SysUtils, Classes, Controls, StdCtrls;
type
  TMySpinEdit = class(TEdit)
  private
    { Private declarations }
  protected
    { Protected declarations }
  public
    { Public declarations }
  published
    { Published declarations }
  end;
procedure Register;
implementation
procedure Register;
begin RegisterComponents('Samples',
        [TMySpinEdit]);
end;
end.
```

Expertul a rezolvat pașii 1, 2 și 4 din crearea noii componente. Se observă crearea automată a procedurii *Register* care rezolvă înregistrarea noii componente.

Pentru folosirea clasei *TUpDown* trebuie inclus unitul *ComCtrls* (în care este definită), pentru lucrul cu mesaje Windows și unele constante specifice trebuie incluse unit-urile *Messages* și *Windows*:

```
uses
  SysUtils, Classes, Controls, StdCtrls,
  ComCtrls, Messages, Windows;
```

Pentru generalitate și în vederea unor schimbări ulterioare am definit un sinonim pentru tipul *Real*: `TFieldType=Real;`

Am adăugat atributele necesare în secțiunea privată a clasei:

```
FSpin:TUpDown;
FValoare:TFieldType;
FMinim:TFieldType;
FMaxim:TFieldType;
FCircular:Boolean;
FPas:TFieldType;
```

Atributul *FSpin* reprezintă butonul de tip *TUpDown*, pentru incrementarea și decrementarea valorii. Atributul *FCircular* arată dacă la depășirea valorii maxime se revine circular la valoarea minimă și invers (valoarea *True*) sau dacă valoarea este limitată la intervalul *FMinim..FMaxim*. Ca funcționalitate, dacă nu se dorește impunerea unui interval de acest tip, valorile minimă și maximă trebuie să fie egale, indiferent de valoarea lor. Celelalte atribute reprezintă valoarea curentă (*FValoare*), minimă (*FMinim*), maximă (*FMaxim*) și pasul de variație (*FPas*).

Aceste atribute sînt publicate prin intermediul secțiunii *published* pentru a fi accesibile utilizatorului atît în timpul dezvoltării aplicației (prin intermediul inspectorului de obiecte) cît și în timpul execuției programului

(prin intermediul codului).

```
property Valoare:TFieldType read GetValoare
      write SetValoare;
property Minim:TFieldType read FMinim write
      FMinim;
property Maxim:TFieldType read FMaxim write
      FMaxim;
property Circular:boolean read FCircular
      write FCircular;
property Pas:TFieldType read FPas write FPas;
```

Doar pentru atributul *Valoare* este necesar controlul accesului, prin intermediul metodelor private *GetValoare* şi *SetValoare*:

```
function GetValoare:TFieldType;
procedure SetValoare(ValoareNoua:TFieldType);
```

Acestea sînt implementate astfel:

```
function TMySpinEdit.GetValoare:TFieldType;
begin try Result:= StrToFloat(Text);
      except Result:= FMinim;
      end;
end;

procedure  TMySpinEdit.SetValoare(ValoareNoua:
           TFieldType);
begin Text:=FloatToStr(
             VerificaValoare(ValoareNoua) );
end;
```

Pentru celelalte atribute accesul este liber. Metoda privată *VerificaValoare* are rolul de a verifica depăşirea intervalului impus, dacă este cazul:

```
function TMySpinEdit.VerificaValoare(
         ValoareNoua:TFieldType):TFieldType;
begin Result:=ValoareNoua;
      if FMinim<>FMaxim then
         if ValoareNoua<FMinim then
            if FCircular then Result:=FMaxim
            else Result:=FMinim
         else
            if ValoareNoua>FMaxim then
             if FCircular then Result:=FMinim
             else Result:=FMaxim;
end;
```

Componenta trebuie să creeze în mod dinamic butoanele de incrementare/decrementare şi să seteze/modifice unele atribute, ca urmare este necesar un constructor şi un destructor special, care să adauge funcţionalităţi noi celor moşteniţi. Constructorul şi destructorul sînt publici şi folosesc metodele moştenite, adăugînd doar lucrurile specifice noii componente:

```
constructor TMySpinEdit.Create(
                AOwner:TComponent);
begin inherited create(AOwner);
      Parent:=(AOwner as TWinControl);
      FSpin:=TUpDown.Create(Self);
      FSpin.Parent:=Self;
      FSpin.Height:=Self.ClientHeight;
      FSpin.Top:=0;
      FSpin.Left:=Self.ClientWidth-
FSpin.Width;
      FSpin.OnClick:=UpDownClick;
      FSpin.Min:=-32768;
      FSpin.Max:=32767;
      Self.Text:='0';
      FPas:=1;
      FMinim:=0;
      FMaxim:=0;
      FCircular:=False;
      FValoare:=0;
end;

destructor TMySpinEdit.Destroy;
begin FSpin.Free;
      inherited destroy;
end;
```

Clasa *TUpDown* limitează variaţia într-un interval dat prin intermediul atributelor *Min* şi *Max*. Pentru a permite o variaţie cît mai liberă pentru noua componentă, am setat la dimensiunea maximă intervalul permis de *TUpDown* (care are pasul 1). Efectul este că se permite variaţia valorii din noua componentă cu 32767 de paşi în sus sau în jos faţă de valoarea iniţială. Este o limitare care contrazice principiul eliminării dependenţelor, dar care nu deranjează în majoritatea aplicaţiilor, limitele fiind destul de largi. Pentru eliminarea acestei limite, ar fi fost necesar să folosim clasa părinte a lui *TUpDown*, *TCustomUpDown*.

Pentru a răspunde la apăsarea săgeţilor, sînt necesare metode care să răspundă evenimentelor de tip *click* pe butoane şi apăsării tastelor săgeţi. Aceste metode sînt definite în secţiunea *protected*, ceea ce le face disponibile în eventualii descendenţi ai clasei:

```
procedure TMySpinEdit.KeyDown(var Key: Word;
             Shift: TShiftState);
begin if Key = VK_UP then
                  UpDownClick(Self,btNext)
      else if Key = VK_DOWN then
                  UpDownClick (Self,btPrev);
      inherited KeyDown(Key, Shift);
end;

procedure TMySpinEdit.UpDownClick(Sender:
                TObject; Button: TUDBtnType);
begin if Button=btPrev then
          Valoare:=Valoare-FPas
      else Valoare:=Valoare+FPas;
end;
```

Metoda *KeyDown* înlocuieşte metoda moştenită de la clasa părinte pentru a răspunde apăsării tastelor săgeţi. Metoda *UpDownClick* răspunde click-urilor pe butoanele de incrementare/decrementare şi este un *event-handler* pus în legătură cu evenimentul *OnClick* al butoanelor de către constructorul noii componente:

```
FSpin.OnClick:=UpDownClick;
```

Întrucît se doreşte lucrul cu valori reale, nu

orice caracter poate să apară în cîmpul de editare. Pentru a preveni apariţia caracterelor nedorite, trebuie înlocuită metoda care răspunde apăsării tastelor cu una care să ignore caracterele invalide din punctul de vedere al funcţionalităţii dorite. Se foloseşte o metodă privată care decide ce caractere sînt valide:

```
procedure TMySpinEdit.KeyPress(var Key:
          Char);
begin if not CaracterValid(Key) then
         begin
              Key := #0;
              MessageBeep(0)
         end;
      if Key <> #0 then
            inherited KeyPress(Key);
      FSpin.Invalidate
end;

function TMySpinEdit.CaracterValid(
                          Key:Char):Boolean;
begin
  Result:=(Key in ['.','+','-','0'..'9'])
   or ((Key<#32) and (Key<>Chr(VK_RETURN)));
  if not Enabled and Result and ((Key>=#32)
     or (Key= Char( VK_BACK)) or (Key=
     Char(VK_DELETE))) then Result:=False;
  if Result and (Key in ['.','+','-']) and
     (pos(Key,Text)<>0) then Result:=False;
  if Result and (Key in ['+','-']) and
     (length(Text)<>0) and (Selstart>1)
      then Result:=False;
end;
```

După completarea unitului, înainte de instalare, trebuie creată o pictogramă adecvată, care să reprezinte componenta în paleta de componente. Aceasta va fi inclusă într-un fişier de resurse cu extensia *.dcr* şi cu acelaşi nume cu al unitului. Resursa trebuie să aibă acelaşi nume ca şi noua clasă. Fişierul de resurse se editează cu instrumentul inclus *ImageEdit*:

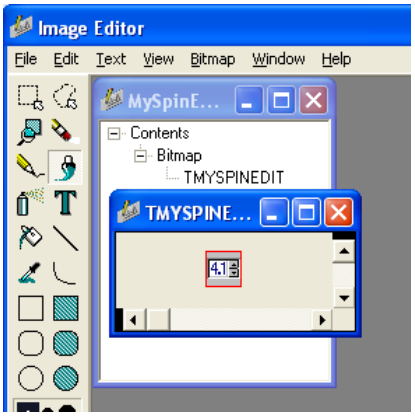

Fig.4. Editorul de resurse

Componenta este gata pentru instalare. Ea poate fi instalată într-un pachet existent sau unul nou. Delphi pune la dispoziţie un pachet gol numit *Borland User Components* în care se pot instala componente noi. Se selectează meniul *Component->Install Component* şi se introduc informaţiile necesare:

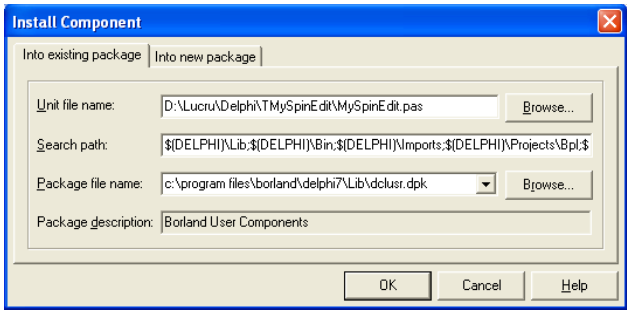

Fig.5. Instalarea noii componente

Apăsarea butonului *Ok* va produce recompilarea pachetului *dclusr.dpk* (Borland User Components). Ca urmare, noua componentă apare în paletă, în pagina stabilită prin procedura de înregistrare şi e gata de utilizare.

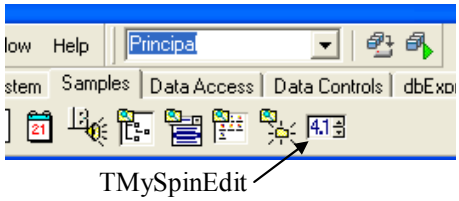

Fig.6. Componenta e gata de utilizare în aplicaţii

Bibliografie

*** Borland Software Corporation – Borland Delphi 7 Help
*** Borland Software Corporation – Borland Delphi 7 Component writers guide