PROGRAMMATION

DESTRUCTIVE

(programmation destructive)

(J-C Armici janvier 2003 www.unvrai.com)

Ce document constitue une introduction à un cours de Delphi; c'est pourquoi il est beaucoup question de Delphi. Les concepts sous-jacents restent bien entendu valables quel que soit l'environnement RAD.

PROGRAMMATION DESTRUCTIVE

Introduction

Il y a quelques années, c'est-à-dire au moment de l'apparition de Visual Basic 1.0, le développement d'applications dans l'environnement Windows a subit de profondes modifications. On est passé de la programmation constructive à la programmation destructive:

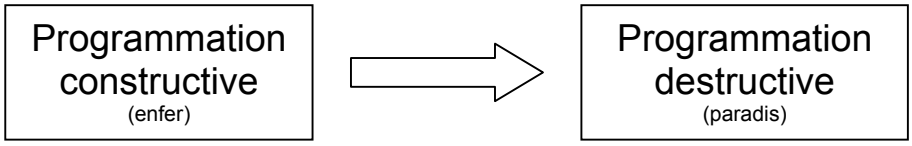

Fig. 1

Voyons quel était le calvaire réservé au programmeur avant l'apparition des outils magiques qui ont pour nom Visual Basic, Delphi, C++ Builder, Visual Age, etc.

L'enfer

Quelques faits caractérisant l'enfer:

- Windows est un système d'exploitation complexe (plus complexe que DOS)
- Seuls les programmeurs chevronnés et maîtrisant le langage C peuvent développer des applications Windows
- Ecrire un programme qui affiche "bonjour" dans une fenêtre nécessite plusieurs dizaines de lignes
- Très peu de programmeurs Windows sont capables d'écrire un programme qui ne fait rien (affichage d'une fenêtre uniquement). Car un programme qui ne fait rien est déjà très long
- La programmation dans un environnement de type "objet" à l'aide d'outils qui ne le sont pas tient du paradoxe, que Microsoft à exploité pendant plusieurs années
- Le fonctionnement de Windows est de type multitâches et il est entièrement régi par des événements. La programmation doit être événementielle
- Pour écrire un programme il faut obligatoirement passer par une longue phase d'apprentissage (plusieurs mois dans les meilleurs cas).
- Dès que le programmeur se heurte à un problème, il se peut qu'il soit obligé de compléter sa formation au moyen de documentations adéquates (help, livres, internet, etc)
- Pour programmer dans Windows il faut utiliser les fonctions que Microsoft met à disposition: plusieurs centaines de fonctions ayant une

multitude de paramètres, plusieurs centaines de messages, de constantes prédéfinies, de cas particuliers…

- De temps à autre (vers 1985 au rythme d'environ tous les ans) Microsoft offre à ses chers développeurs de nouveaux outils ou de nouvelles technologies (Windows SDK, MAPI, TAPI, FPSDK, MPC, MIDI, WING, GameSDK, RDO, RDC, COM, DCOM, ADO, DDE, OLE, OLE2, ActiveX, Direct 2D, Direct 3D, IIS, IEADK, ISAPI, J++, VBS, et des **dizaines d'autres**)
- Actuellement, le rythme auquel apparaissent ces nouveauté est infernal: quasiment **hebdomadaire**
- Chacune de ces nouveautés peut demander des mois de familiarisation à un programmeur. Mais entre temps d'autres nouveautés apparaissent.
- Le système ne pousse pas le programmeur à être curieux et inventif
- Le programmeur maîtrise généralement un sous ensemble des outils qui lui sont proposés. Lorsqu'il a besoin d'une implémentation qu'il ignore, sont seul recours est la lecture et/ou la recherche pénible d'informations. Il **construit** son savoir
- Le développement tient plus de l'esclavage que de la création
- Les choix des programmeurs sont:
  - Le doute
  - La dépression
  - Le suicide
  - Le changement de profession
  - L'acharnement dû à un optimisme démesuré
  - Le repentir
- La situation n'est vraiment plus vivable. La programmation n'est plus supportable pour un être humain. Seules certaines grandes sociétés peuvent résister

Le paradis

Mais voilà la programmation destructive qui pointe son nez. Une vraie bouée de sauvetage pour le programmeur "normal". Grâce à elle, un débutant peut au bout de quelques minutes, créer un programme avec des caractéristiques qu'un programmeur confirmé mettrait des jours à inclure.

Mais voyons pourquoi et comment ces environnements dits "**visuels**" ou **RAD** (Rapid Application Development) permettent de quitter un enfer insupportable et de rejoindre un paradis virtuel, visuel, doré. Nous verrons plus loin que ce paradis a tout de même quelques nuages (Windows oblige).

Un exemple valant mieux que 18 discours, voici deux listings. Le premier, presque une centaine de lignes, programmation constructive, représente un programme qui ne fait rien d'autre qu'afficher une fenêtre. Il est écrit avec Delphi et utilise uniquement des moyens traditionels. Ce programme est tiré du livre "Delphi 2 secrets d'experts" de Charles Calvert:

```
program Window1;
{ Standard Windows API application written in Object Pascal.
  No VCL code included. This is all done on the Windows API
  level.
  Note that you need to include both Windows and Messages!}
uses Windows, Messages;

const AppName = 'Window1';

function WindowProc(Window: HWnd; AMessage, WParam,
                    LParam: Longint): Longint; stdcall; export;
begin
  WindowProc := 0;
  case AMessage of
    wm_Destroy: begin
      PostQuitMessage(0);
      Exit;
    end;
  end;
  WindowProc := DefWindowProc(Window, AMessage, WParam, LParam);
end;

{ Register the Window Class }
function WinRegister: Boolean;
var WindowClass: TWndClass;
begin
  WindowClass.Style := cs_hRedraw or cs_vRedraw;
  WindowClass.lpfnWndProc := @WindowProc;
  WindowClass.cbClsExtra := 0;
  WindowClass.cbWndExtra := 0;
  WindowClass.hInstance := HInstance;
  WindowClass.hIcon := LoadIcon(0, idi_Application);
  WindowClass.hCursor := LoadCursor(0, idc_Arrow);
  WindowClass.hbrBackground := HBrush(Color_Window);
  WindowClass.lpszMenuName := nil;
  WindowClass.lpszClassName := AppName;

  Result := RegisterClass(WindowClass) <> 0;
end;

{ Create the Window Class }
function WinCreate: HWnd;
var hWindow: HWnd;
begin
  hWindow := CreateWindow(AppName, 'Object Pascal Window',
              ws_OverlappedWindow, cw_UseDefault, cw_UseDefault,
              cw_UseDefault, cw_UseDefault, 0, 0, HInstance, nil);
  if hWindow <> 0 then begin
    ShowWindow(hWindow, CmdShow);
    UpdateWindow(hWindow);
  end;
  Result := hWindow;
end;

var AMessage: TMsg;
    hWindow: HWnd;
begin
  if not WinRegister then begin
    MessageBox(0, 'Register failed', nil, mb_Ok);
    Exit;
  end;
  hWindow := WinCreate;
  if hWindow = 0 then begin
    MessageBox(0, 'WinCreate failed', nil, mb_Ok);
    Exit;
  end;

  while GetMessage(AMessage, 0, 0, 0) do begin
    TranslateMessage(AMessage);
    DispatchMessage(AMessage);
  end;
  Halt(AMessage.wParam);
end.
```

Fig. 2

Voici maintenant l'équivalent en Delphi version RAD. Pour mieux saisir la différence, le listing est encadré:

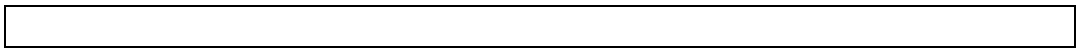

Fig. 3

Oui, vous avez bien vu!! Il n'y a rien. Et ce rien fait la même chose que le premier listing. Si, de plus, on voulait écrire "Bonjour" en fonte Arial 12 pt rouge au centre de la fenêtre et avoir une fenêtre avec fond bleu, sans bord, le premier listing verrait son volume doubler ou tripler, alors qu'en Delphi RAD on n'aurait toujours **aucune ligne de code**.

Y a-t-il mystère ou miracle ? S'agit-il d'une tromperie ? La complexité est-elle cachée ailleurs ?

En réalité il n'y a aucun miracle. La difficulté de la programmation constructive est déplacée, encapsulée. L'équivalent du code du premier listing est bien entendu contenu dans l'exécutable lié au second listing; il est simplement masqué.

Dans un outil comme Delphi, le gagnant est le développeur qui peut se concentrer plus efficacement sur l'algorithmique du développement. Toutefois, si le besoin s'en fait sentir, il est toujours possible de programmer, localement ou globalement, à un niveau plus bas.

Dans un programme Delphi, ce qui caractérise un programme (par exemple la couleur de la fenêtre) et qui n'apparaît pas sous forme de code Pascal est en réalité dans un fichier décrivant la fenêtre. Ce fichier est mis à jour lorsque l'une ou plusieurs des propriétés de la fenêtre sont modifiées. Dans la phase de génération du programme exécutable ces caractéristiques sont intégrées au code Pascal.

C'est là l'essence même de la programmation destructive:

- Par défaut chaque objet fonctionne de manière standard, prévue par l'environnement de développement
- Si une caractéristique ne convient pas au programmeur, il peut la changer. Il détruit donc ce qui lui est proposé et le remplace par ce don't il a besoin
- Le programmeur peut enfin devenir créatif, curieux et acteur
- Généralement les propriétés et les méthodes associées aux différents objets sont visibles. Il suffit de regarder autour de soi; tout est disponible. Fini le temps où il fallait des heures pour trouver comment modifier telle ou telle propriété

En conclusion on peut dire que la programmation destructive met fin à une absurdité et rétablit la réalité et le bon sens, à savoir que:

Il est impossible de rechercher quelque

chose dont on ignore l'existence.

Les nuages

La facilité apparente d'utilisation de la programmation destructive à un revers, un prix à payer. Par exemple:

- Un programme en RAD est généralement plus volumineux à fonctionnalités égales. L'exécutable du premier listing fait 9 Koctets, alors que celui du second fait 153 Koctets
- Certains considèrent qu'il peut y avoir perte d'efficacité et de précision
- Le programmeur peut avoir l'impression que le contrôle de son programme lui échappe quelque peu

Conclusion

Chacun fait comme il l'entend, mais la programmation destructive est en marche et ne s'arrêtera que lorsqu'un nouveau concept simplificateur prendra le relais. Il y aura toujours des programmeurs qui regrettent le temps des cartes perforées, car le changement demande évidemment plus d'efforts que l'immobilisme.