Turbo Vision e ObjectWindows

di Sergio Polini (MC1166 su MC-link)

Sull'onda del successo di Windows 3.0, la Microsoft ha annunciato una versione 3.1, destinata ad essere seguita prima da un Windows a 32 bit e poi da un Windows NT (New Technology), una sorta di nucleo per PC e workstation su cui far girare applicazioni DOS, Windows, OS/2 e Posix. La IBM, dal canto suo, puntava e punta su OS/2: dovrebbe essere rilasciata entro quest'anno la versione 2.0 a 32 bit dedicata a macchine con 80386 o 80486. I due grandi però non vanno più d'accordo come un tempo; la IBM ha recentemente stretto alleanze con Apple, Borland, Novell, ecc., che sono state da molti interpretate come un vero e proprio assedio alla Microsoft, la quale sembra reagirà non supportando più OS/2. Si è così rotta quell'alleanza che aveva portato ad una sostanziale uniformità nel mondo dei personal computer basati sui processori Intel; probabilmente, entro un anno o poco più ogni utente sarà costretto a scegliere tra tre sistemi operativi (MS-DOS, Windows, OS/2), ognuno con qualche «apertura» verso l'altro ma ben diversi tra loro; su un altro versante, chi sviluppa software avrà ben più seri problemi

Veniamo dal monopolio MS-DOS, ci avviamo verso la concorrenza tra più ambienti diversi. Quale sceglieremo per lo sviluppo? Come porteremo le nostre applicazioni dall'uno all'altro?

Se è presto per rispondere alla prima domanda, la Borland ha qualcosa da dire circa la seconda; sostiene infatti che la programmazione orientata all'oggetto consente di «incapsulare» le parti di un programma strettamente legate al sistema operativo, e di realizzare quindi applicazioni facilmente portabili da un ambiente all'altro. Come notai già in occasione della prova del Turbo Pascal 6.0 (MC di febbraio), il Turbo Vision non va intenso come l'ennesima libreria di funzioni per un'interfaccia utente in modo carattere, ma come una gerarchia di classi per la realizzazione di programmi event-driven e con interfacce utente conformi allo standard SAA-CUA. Dal momento che programmazione per eventi e standard SAA-CUA sono elementi essenziali nella programmazione sotto Windows, ipotizzavo che la vera finalità del Turbo Vision fosse proprio quella di permettere lo sviluppo di applicazioni facilmente portabili sotto Windows. Poi abbiamo ottenuto anche un Turbo Pascal per Windows con un'analoga gerarchia di classi, ObjectWindows. Si tratta quindi ora di verificare quanto, e a che condizioni, tali strumenti mantengano quello che promettono.

Programmare per eventi

La programmazione per eventi costituisce la principale differenza tra Turbo Vision e altri prodotti analoghi che l'avevano preceduto. Si tratta di un modo di programmare che ha per buona parte le stesse radici della OOP. Ricordiamo che i concetti di classe e di oggetto nascono con il Simula 67, per l'esigenza di simulare in modo efficace una realtà in cui troviamo oggetti dotati ognuno di un proprio «stato» e di un proprio «comportamento» largamente autonomi, ma in continua interazione l'uno con l'altro. In una classe Simula, accanto ai campi-dato di un record tradizionale, abbiamo campi-procedura (chiamati poi «metodi» in Smalltalk) proprio per specificare il comportamento dei diversi oggetti; l'esecuzione di un programma è per il resto governata da un sequencing supervisor incaricato della gestione di un sequencing set (SQS), cioè di un insieme di eventi caratterizzati ognuno da un suo tempo. Un oggetto, durante una sua «fase attiva» (cioè durante l'esecuzione del suo codice), può «attivare» altri oggetti inserendo una segnalazione di evento nell'SQS con una istruzione activate. Il tutto è agevolato dall'ereditarietà: tali oggetti sono infatti istanze di una classe Process, derivata da una classe Simulation.

Il Simula attirò l'attenzione di Alan Kay quando, negli ultimi anni '60, si occupava di grafica. Il suo sogno del Dynabook, un computer tanto potente quanto facile da usare, trovò una prima materializzazione in una Flex machine costruita attorno ad un linguaggio Flex derivato proprio dal Simula. Nel giro di una decina d'anni dal suo lavoro nacque Smalltalk: un linguaggio orientato all'oggetto, ma anche un completo ambiente di sviluppo basato su grafica, finestre e mouse, in cui tutto è oggetto. Da Smalltalk hanno poi tratto ispirazione il Lisa e il Macintosh, Windows e Presentation Manager. È chiara l'utilità della OOP in tali ambienti: l'interazione con l'utente avviene mediante finestre delle più diverse fogge (menu, dialog box, list box, scroll bar, pulsanti, ecc.), che hanno tutte qualcosa in comune. Anche una scroll bar è una finestra, ogni finestra è un rettangolo, e così via. Gli elementi dell'interfaccia sono ben organizzabili in

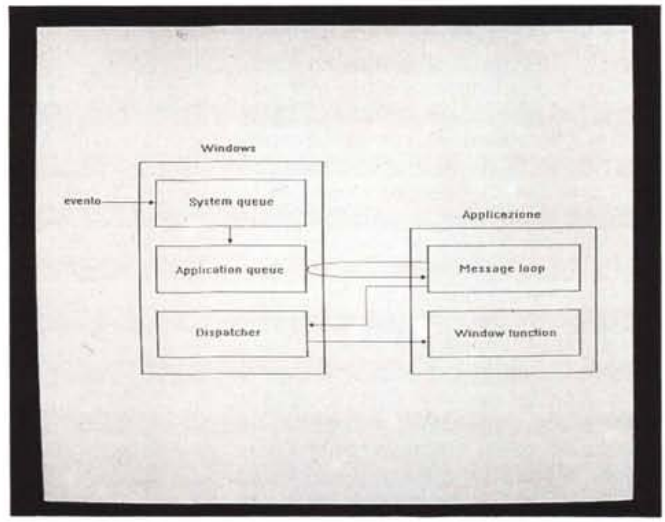

Figura 1 - Gestione dei messaggio in Windows. Un evento (ad esempio la pressione di un tasto) genera un messaggio che viene posto nella System queue; di qui il messaggio passa nella coda dell'applicazione cui il messaggio compete; il message loop dell'applicazione preleva i messaggi dalla coda, ma li ritrasmette a Windows perché vengano «tradotti» (ad esempio nel codice ASCII corrispondente al tasto premuto); Windows passa quindi il messaggio tradotto alla window function associata alla finestra dell'applicazione cui il messaggio è destinato.

una gerarchia di classi, mentre ereditarietà e polimorfismo consentono la facile estensione della gerarchia per creare finestre di tipo particolare secondo le esigenze delle diverse applicazioni.

Anche la programmazione per eventi svolge un ruolo importante, pur se in parte diverso da quello che aveva in Simula. Pensiamo ai menu come sono stati realizzati nel mondo MS-DOS. Una volta erano rappresentati mediante una schermata con l'indicazione delle diverse opzioni contrassegnate da un numero o una lettera, che l'utente poteva scegliere premendo il tasto corrispondente. Poi sono venuti i menu del Lotus, un albero di sottomenu percorribile premendo i tasti corrispondenti alle diverse opzioni. Poi i menu a tendina, usabili anche con il mouse. Infine siamo giunti ai menu di Windows. Premendo un tasto possiamo selezionare l'opzione di un menu; ne segue che il programma potrebbe intercettare la pressione dei tasti e riconoscere quelli che attivano il menu principale, un suo sottomenu, un'opzione. Analogamente con il mouse. Il mouse tuttavia, mentre rende più comode e immediate le scelta dell'utente, complica la vita a chi fa software. Pensate ad una schermata Windows in cui siano aperte più finestre: portandosi con il mouse sulle varie finestre, l'utente può «attivare» prima l'una poi l'altra con un semplice click; la finestra attiva non può gestire scelte operate dall'utente fuori dei suoi confini, ma può solo passare il controllo al sistema operativo perché questo a sua volta renda attiva un'altra finestra. Senza adeguato supporto, lo sviluppo di un'applicazione sconfinerebbe troppo spesso nel software di sistema. Dovrebbe essere chiara l'analogia tra un tale «supporto» e il sequencing supervisor del Simula, come anche la differenza tra gli eventi Simula e i più complessi eventi di un'interfaccia come quella di Windows: quelli si limitano ad associare un oggetto al tempo in cui dovrà essere attivato, questi devono contenere l'indicazione del tipo dell'evento (pressione di un tasto, spostamento del mouse, ecc.) e ulteriori informazioni variabili secondo il tipo (quale tasto è stato premuto, le coordinate del mouse, ecc.).

Nella OOP si usa ragionare in termini di «messaggi» inviati ad oggetti che «rispondono» ognuno secondo il proprio comportamento, con un proprio «metodo». Conformemente a tale terminologia, si dice che un evento genera un messaggio; al sequencing set di Simula corrisponde dunque la message queue (coda di messaggi) di Windows. In realtà vi sono più code di messaggi, secondo lo schema illustrato in figura 1; quello che ora voglio sottolineare, è che la programmazione per eventi non solo ben si sposa con quella per oggetti, ma nasconde al programmatore le difficoltà insite in interfacce utente tanto flessibili quanto complesse: non occorre occuparsi dell'intricato susseguirsi di eventi della più varia natura, in quanto è sufficiente definire gli oggetti del programma prevedendo i metodi corrispondenti ai messaggi ai quali si vuole che ognuno risponda; degli altri si occuperà qualcosa che rimane dietro le quinte.

Programmando sotto Windows, ciò che rimane dietro le quinte è Windows stesso, soprattutto se «incapsulato» da ObjectWindows; con il Turbo Vision, si occupa di tutto un event manager inizializzato dalla unit DRIVERS. Come se non bastasse, i diversi elementi dell'interfaccia utente sanno già come rispondere ad un gran numero di messaggi; non segue da questo un'improvvisa inusitata facilità del programmare, ma la possibilità di produrre, con la consueta fatica, programmi di migliore fattura.

Programmare per classi

Programmare in modo orientato all'oggetto è in realtà un programmare per classi. In luogo della tradizionale definizione di miriadi di variabili, in OOP i dati di un programma vengono organizzati in organiche gerarchie di classi; si usano le classi già disponibili, si derivano da queste classi più specifiche estendendo la gerarchia secondo le necessità di ogni applicazione.

Sarebbe certo possibile costruire ogni volta una nuova gerarchia partendo da zero, ma ciò richiederebbe molto tempo e vanificherebbe, soprattutto, quei benefici di «riusabilità del codice» che fanno della OOP lo stile di programmazione più produttivo. Ne segue che abbiamo bisogno di gerarchie già pronte per poter lavorare al meglio.

Ho già ricordato quanto sia difficile realizzare una gerarchia di classi, al punto che non esistono ancora principi standard che possano valere da guida, e che quindi non è possibile neppure individuare criteri univoci per giudicare se una data gerarchia è «buona» oppure no. Mi limiterò pertanto ad esporre le caratteristiche di Turbo Vision e ObjectWindows per quello che propongono: «schemi» di applicazioni praticamente identici sia per MS-DOS che per Windows, classi di oggetti d'interfaccia strutturate in modo analogo per i due

```
Turbo Vision

TObject
  TView
    TBackground
    TButton
    TCluster
      TCheckBoxes
      TRadioButtons
    TFrame
    TGroup
      TDeskTop
      TProgram
        TApplication
      TWindow
        TDialog
        THistoryWindow
    THistory
    TInputLine
    TListViewer
      TListBox
      THistoryViewer
    TMenuView
      TMenuBar
      TMenuBox
    TScroller
      TTextDevice
        TTerminal
    TScrollBar
    TStaticText
      TLabel
      TParamText
    TStatusLine
  TCollection
    TSortedCollection
      TStringCollection
        TResourceCollection
  TStream
    TDosStream
      TBufStream
    TEmsStream
  TResourceFile
  TStringList
  TStrListMaker
```

```
ObjectWindows

TObject
  TWindowsObject
    TWindow
      TControl
        TMDIClient
        TButton
        TScrollBar
        TStatic
          TEdit
        TListBox
        TComboBox
        TGroupBox
        TCheckBox
          TRadioButton
      TMDIWindow
      TEditWindow
      TFileWindow
    TDialog
      TDlgWindow
      TFileDialog
      TInputDialog
  TApplication
  TScroller
  TCollection
    TSortedCollection
      TStringCollection
      TStrCollection
  TStream
    TDosStream
      TBufStream
    TEmsStream
```

Figura 2 - Le gerarchie di classi di Turbo Vision e ObjectWindows.

ambienti, strutture di dati polimorfiche molto simili (le gerarchie complete sono riprodotte nella figura 2).

Cominciamo col notare che le attività di inizializzazione di un programma MS-DOS e di uno Windows sono molto diverse. Nel primo caso si tratta di determinare l'hardware su cui gira (processore, quantità di memoria, scheda video), preparare le operazioni con il mouse se presente, intercettare alcuni interrupt per controllare la pressione dei tasti Ctrl-C e Ctrl-Break o per non farsi sorprendere dai cosiddetti «errori critici» (un drive aperto, una stampante senza carta), inizializzare l'interfaccia a finestre, il sistema di menu, la riga di stato ecc. Nel secondo caso si tratta di definire una o più classi di finestre, registrare tali classi, creare la finestra principale, definire la window function ad essa associata, dare inizio al message loop, ecc. Le gerarchie di Turbo Vision e ObjectWindows sono piuttosto diverse quanto ad architettura generale, ma entrambe prevedono una classe TApplication che nasconde tutti i dettagli dell'inizializzazione: un programma «minimo» in Turbo Vision è praticamente identico ad un equivalente programma ObjectWindows (figura 3).

Naturalmente le differenze cominceranno non appena riempiremo lo schema. In primo luogo la gestione degli eventi in Turbo Vision è meno trasperente che in ObjectWindows; è spesso quindi necessario ridefinire il metodo HandleEvent che tutte le classi derivate da TView ereditano da questa. Vi è poi una diversa organizzazione dei messaggi: in Windows, ad esempio, la pressione del pulsante sinistro del mouse genera un messaggio WM_LBUTTONDOWN con due parametri, il primo relativo alla eventuale contemporanea pressione di tasti come Shift e Control, il secondo con le coordinate del mouse sullo schermo; in Turbo Vision viene generato un evento riempendo un record di tipo TEvent con campi come Buttons per il pulsante premuto, Double per indicare se si tratta di un doppio click, Where per le coordinate. Le classi inoltre non sono solo diversamente organizzate, ma a volte le classi dell'uno non trovano corrispondenti classi nella gerarchia dell'altro; non è un problema la mancanza di una classe TProgram in ObjectWindows, in quanto anche in Turbo Vision questa non viene praticamente mai usata direttamente (si usa solo la derivata TApplication); non crea eccessive difficoltà la mancanza in Turbo Vision della classe TGroupBox, in quanto radio button e check box vengono automaticamente organizzati in «gruppi» (con lievi differenze di comportamento); è fastidiosa, tuttavia, l'assenza in Turbo Vision di una classe come TComboBox, che può costringere ad un lavoro di programmazione che, per quanto agevolato dalla presenza di un «modello» nella unit STDDLG, sarebbe stato lecito aspettarsi non necessario.

Nonostante tali differenze, si dispone comunque di un ampio insieme di classi comuni (TApplication, TWindow, TDialog, TScroller, TScrollBar, TButton, TCheckBox, TRadioButton, TListBox) che consentono comunque di portare avanti con efficacia il progetto di un'applicazione portabile. Sarà spesso necessario intervenire su questo o quel dettaglio dell'implementazione nei due ambienti, ma almeno la «logica» promette di essere identica. Può valere la pena di provare. Provare vuol dire imparare a conoscere i due ambienti e tenere presenti analogie e differenze mentre si lavora su un ambiente, per facilitare il porting sull'altro; vuol dire anche individuare analogie magari poco visibili, come quella tra il desktop del Turbo Vision con le sue molteplici finestre affiancate o sovrapposte e un'applicazione MDI (Multiple Document Interface) di Windows. Ma di questo riparleremo.

Dobbiamo anche considerare che l'interfaccia è solo la parte esteriore di un programma. Quando si lavora sulla «sostanza», buona parte del lavoro se ne va nella messa a punto delle idonee strutture di dati, soprattutto quelle destinate a fungere da «contenitori» di oggetti: una lista di righe per un editor, una coda FIFO per una simulazione, un array di stringhe per i messaggi d'errore, un albero binario per una tabella ordinata di nomi, un grafo per problemi di cammino

minimo o di massimo flusso in una rete, ecc. Già sappiamo che la OOP incoraggia l'utilizzo di strutture di dati come quelle che abbiamo discusso nei mesi precedenti; guardando all'ampia gerarchia di «contenitori» di Smalltalk (che ho preso ad esempio nella realizzazione di quella che vi ho proposto su queste pagine), si rileva che si può scegliere la classe più adatta alle proprie esigenze badando esclusivamente a quello che ognuna fa, prescindendo completamente dai dettagli dell'implementazione. Se occorre un elenco ordinato di oggetti, non occorre valutare se può convenire un array, una lista o un albero binario: basta usare una SortedCollection. Per il resto, l'ereditarietà consente di adattare eventualmente la classe a situazioni particolari, il polimorfismo consente di usare la classe con oggetti dei tipi più diversi. Ricorderete la semplicità con cui, nel numero di aprile, abbiamo usato nel programma PREMAKE una classe TSet per realizzare l'equivalente di quella complicata lista di liste che avevamo approntato per il MINIMAKE di due anni fa.

Turbo Vision e ObjectWindows comprendono ognuno una gerarchia di contenitori semplificata e meno flessibile di quella di Smalltalk. La minore flessibilità è dovuta al fatto che, mentre in Smalltalk tutto è oggetto, in Turbo Pascal si continuano ad utilizzare tipi tradizionali come interi, reali e stringhe. Una collezione ordinata di stringhe non può stabilire l'ordine mandando ad ogni stringa un messaggio del tipo «confrontati con quest'altra stringa»; ne segue che TStringCollection deriva da TSortedCollection ridefinendo quattro metodi, tra cui un Compare che sa come paragonare le stringhe, ma solo quelle. Analogamente occorrerà derivare da TSortedCollection apposite collezioni per ogni tipo di oggetto che vorremo tenere in ordine (ne trovate un esempio nella TFileCollection della unit STDDLG).

Le gerarchie di contenitori di Turbo Vision e ObjectWindows sono comunque utili, e soprattutto sono praticamente identiche nei due ambienti. C'è una sola differenza: in ObjectWindows, accanto a una TStringCollection per le normali stringhe del Turbo Pascal, compare anche una TStrCollection per il nuovo tipo di stringhe riconosciuto dalla versione per Windows, quelle con zero finale e senza byte di lunghezza iniziale, introdotte per compatibilità con le stringhe come volute dalla API di Windows.

Turbo Vision

```
Program Schema;

uses App;

type
  TSchema = object(TApplication)
  end;

var
  S: TSchema;

begin
  S.Init;
  S.Run;
  S.Done;
end.
```

ObjectWindows

```
Program Schema;

uses WObjects;

type
  TSchema = object(TApplication)
  end;

var
  S: TSchema;

begin
  S.Init('Schema');
  S.Run;
  S.Done;
end.
```

Figura 3 - Due programmi «minimi» in Turbo Vision e ObjectWindows; come si può notare, sono praticamente identici nonostante la diversa articolazione delle rispettive gerarchie di classi.

Programmare con le risorse

Programmando sotto Windows, non è necessario scrivere decine e decine di istruzioni per definire la struttura di un menu o il contenuto di una dialog box; si utilizzano invece strumenti come il Whitewater Resource Toolkit per creare interattivamente tali «risorse», che possono poi essere semplicemente aggiunte al programma. Anche nel Turbo Vision possiamo creare risorse fuori del programma per poi aggiungerle col comando COPY del DOS, ma ciò avviene scrivendo appositi programmi, non interattivamente su video.

A San Francisco ho avuto modo di vedere un prodotto della Blaise Computing che colma la lacuna. Si tratta di un Turbo Vision Development Toolkit che comprende anche RESEDIT.EXE, un resource editor analogo a quelli disponibili per Windows: si può descrivere e modificare interattivamente la struttura di un menu, verificandone in ogni momento l'apparenza sullo schermo; si può disegnare sullo schermo una dialog box aggiungengo i vari tipi di «controlli» (campi di input, check box, pulsanti, ecc.), specificando anche per quali di essi non si userà la classe standard ma una classe specializzata da questa derivata; si può definire una lista di stringhe da utilizzare per i messaggi d'errore o per quelli che dovranno apparire sulla riga di stato. L'ho subito comprato e ho già avuto modo di utilizzarlo con profitto.

Uno degli aspetti più interessanti del Toolkit della Blaise è la presenza di un programma RC-1.EXE con il quale, a partire da un file di risorse generato da RESEDIT, è possibile generare un file utilizzabile sotto Windows con il Resource Compiler della Microsoft (fornito insieme al Turbo Pascal per Windows). La conversione non è priva di problemi, come avremo modo di vedere e come ho già illustrato nella conferenza Pascal di MC-link, ma può comunque offrire una base di partenza per accelerare il porting di un'applicazione da DOS a Windows.

Tanto si sente la necessità di prodotti come questo, che mi è giunta notizia di programmatori che hanno iniziato a farsi il proprio resource editor in casa. Facile immaginare quanto tempo richieda una tale impresa. Mi sembra più saggio acquistare qualcosa di già pronto (ci sono anche i sorgenti), soprattutto tenendo conto del prezzo contenuto in 150 dollari e delle politiche di vendita della Blaise; queste sono tali che organizzazioni di vendita per corrispondenza, come ve ne sono ormai molte anche in Italia, sono sicuramente in grado di praticare prezzi paragonabili a quello in dollari.

Conclusione

Cercando di sintetizzare quanto detto finora, potrei dire che Turbo Vision e ObjectWindows consentono di programmare per eventi, per classi e con risorse in modo analogo, rendendo praticabile, anche se non immediato, il porting di un'applicazione da MS-DOS a Windows.

Data la prevedibile prossima frantumazione del monopolio dell'MS-DOS, può valere la pena di verificare in concreto fino a che puno le due gerarchie di classi, pur con le loro differenze, riescono a nascondere la profonda diversità tra i due ambienti cui sono destinate, facilitando lo sviluppo di applicazioni nel dopo-monopolio,

È quanto cominceremo a fare a partire dal mese prossimo.

MC