Appunti di programmazione del Macintosh

di Raffaello De Masi

La sezione dedicata al Macintosh, da questa puntata, cambia faccia.

Nel paio d'anni di vita di questa rubrica abbiamo sempre evitato, di entrare in merito alla programmazione della macchina nei suoi vari linguaggi; il motivo è presto detto; programmare una macchina Mac, come vedremo tra poco, non è semplice; inoltre, per uno di quegli strani fenomeni di mercato che talora è difficile interpretare, ben pochi sono stati in Italia i programmatori autonomi che hanno sposato la causa Macintosh. Così Mac è divenuto il computer «for last user», per l'utente super evoluto e superfinale che non si accontenterebbe mai di una applicazione non curatissima nella forma, non dotata di una interfaccia totalmente integrata all'ambiente, ecc. dopo i primi tentativi del 1984-'85, i linguaggi per questa macchina cominciarono a sparire inesorabilmente dal mercato; intendiamoci, parlo di linguaggi dedicati ai dilettanti, mentre i pochi sopravvissuti, «C» e Basic, soprattutto, ma anche Prolog, Lisp, Logo, Pascal, divennero sempre più specializzati e complessi. A contorno, certe applicazioni, come database e fogli elettronici, divenivano sempre più complesse, programmabili, ma al contempo facili da usare, riuscendo a coprire anche le richieste di chi desiderava piegare una applicazione generica alle sue esigenze.

A distanza di sette anni dalla comparsa del Macintosh, si sta lentamente verificando una inversione di tendenza: l'interfaccia Macintosh, grazie a linguaggi sempre più potenti e flessibili, è divenuta meno terrorifica; i linguaggi stessi sono ormai dotati di una serie di routine precostituite che consentono di bypassare la lunga, fastidiosa e complessa fase di programmazione delle window, dei pulsanti, dei menu, delle finestre d'errore; tanto per fare un esempio, un linguaggio, lo ZBasic della Zedcor, contiene una utility che, attraverso una interfaccia tipo MacDraw permette di disegnare finestre «context sensitive» e di generare il relativo codice sorgente, ampiamente commentato.

Perciò oggi ci troviamo a disporre di un ambiente di programmazione molto più amichevole, di ROM da 512 molto più elastiche ed efficienti delle vecchie 64 del 512, e programmare il «temibile» Mac non è più lo spauracchio di una volta.

Con questa rubrica ci riserviamo di aprire un colloquio con i lettori su tecniche, tipologie, esempi di programmazione dedicati essenzialmente alla gestione dell'interfaccia di base; in altre parole vedremo come è possibile creare quelle specifiche di gestione dei tool (bottoni, scroll-bar, menu, ecc.) che rendono unico Macintosh e la sua utilizzazione. Tutto questo, se mi si consente, senza aderire a questo o quel linguaggio, cosa che, specie nel Mac, assume come vedremo, un significato ridotto. Parleremo, perciò, di Logo (che su Mac gode di implementazioni eccellenti), di C, di Pascal, e, perché no?, anche di Basic. Ma parleremo soprattutto di routine, costrutti, blocchi di programma, da inserire nel codice che stiamo sviluppando, per ripassare, come dicevamo prima, la lunga routine di creazione di una finestra che risponda perfettamente ai nostri ordini.

Un ruolo particolarmente importante nella corretta redazione di un programma è la programmazione orientata all'oggetto. Si tratta di una tecnica che ha assunto oggi grande importanza, con lo sviluppo di programmi di ampio respiro e di grande potenza, che avrebbero scarsa possibilità di essere realizzati se non esistesse alle spalle il supporto di questa tecnica. Come avremo modo di dire più volte in seguito, questo tipo di programmazione non è retaggio specifico di Mac, ma risulta qui particolarmente utile semplificando in maniera impensabile la vita di chi scrive programmi.

La programmazione object-oriented su Macintosh

Le applicazioni Macintosh sono per antonomasia facili da usare; non a caso Mac è chiamato, in certe pubblicità, «Machine in ten minutes», la macchina che si impara a usare in poco tempo, minuti appena, e non c'è stato finora nessuno che sia mai venuto a smentire l'affermazione. Le applicazioni Mac, dal Finder al più complesso DB o foglio elettronico hanno un vantaggio; quello che ci si può cominciare a giocherellare anche senza aver mai aperto il manuale (non a caso l'utente Mac, all'acquisto di un nuovo pacchetto, usa prima il software e poi la manualistica). Ma poche persone pensano (anche se tutti immaginano) quanta fatica e lavoro ci sia alle spalle di questa «facilità», che a noi risparmia tanti mal di testa.

Fatto sta che programmare un'applicazione Mac, qualunque sia il linguaggio usato, è né più né meno facile della stessa applicazione su un'altra macchina, a patto di non voler uscire dalle tradizionali tecniche e forme di I/O e di corrispondenza tra macchina a utente; appena però si decide di lavorare di fino, vale a dire si cerca di inserire quelle caratteristiche che rendono tanto facile lavorare con questa macchina (caratteristiche, guarda caso, legate tutte o quasi tutte all'uso del mouse e dei tasti Comando-Opzione) cominciano i dolori e le faticacce vicino a un programma che improvvisamente diviene recalcitrante e ostico ai nostri voleri.

Alla base di tutto ciò ci sono una serie di fattori, che governano l'uso di questa macchina; il ToolBox con tutti i suoi controlli di evento, l'interfaccia QuickDraw, e la gestione delle risorse della macchina attraverso il corretto uso del linguaggio di programmazione; dal momento che sui primi due fattori c'è poco da intervenire, la fatica va rivolta alla corretta comprensione di queste (le risorse) attraverso il terzo fattore, il linguaggio. Ma, nell'ambito delle tecniche di programmazione, da qualche tempo è cominciata ad intervenire una serie di fattori, un punto di vista nuovo nel modo di organizzare e disegnare il programma stesso; questa nuova tecnica, che può essere a buona ragione definita una filosofia, va sotto il nome di programmazione object-oriented o, tout court, OO.

Il concetto di OO non è per la verità né nuovo, né originale; tracce di tale concezione «di vita», se così si può dire, si ritrova spesso tra le righe di manuali di linguaggi costruiti con una certa cura. In effetti l'idea e la prima implementazione di un linguaggio destinato a tali fini è addirittura del 1962, quando Kristen Nygaard e Ole Johan Dahl, dell'istituto di calcolo centrale dell'università di Oslo, si resero conto di avere la necessità di un linguaggio particolare capace di manipolare simulazioni molto complesse. In diversi casi molte simulazioni coinvolgono oggetti, intesi come categorie di strutture, entità, o altro, comunque ben definibili fisicamente, che, in maniera abbastanza autonoma, interagiscono tra di loro come entità a sé stanti.

Le idee di Nygaard e Dahl si realizzarono in un linguaggio, Simula, che fu concepito proprio per creare oggetti che interagissero tra di loro. Si trattava di un linguaggio dedicato, che, in altri tipi di applicazioni, non forniva risultati di pregio, ma faceva il suo compito con onestà e pulizia; proprio per questi limiti non raggiunse mai un elevato stadio di popolarità, come invece allora accadde con il Fortran, imperante a quei tempi, e oggi divenuto pezzo da museo. In seguito furono sviluppati linguaggi di gran qualità, decisamente orientati alla crea-

zione di oggetti (i cosiddetti object-oriented, di cui alla nostra trattazione), che si svincolarono dalla (sovente pretesa) universalità dei linguaggi generali, imboccando di nuovo la strada della specializzazione; nacquero così linguaggi ad hoc, come Planner, SmallTalk, Objective-C, Loops, Object Pascal e C++, e alcuni di questi divennero anche disponibili in ambiente Macintosh. Un luminoso esempio di linguaggio di tal tipo è stato Neon, della Kriya Systems, un idioma simile per certi aspetti al Forth, che di questo conserva la struttura interpretata; altra implementazione è l'Object Pascal, ben noto nell'ambiente di sviluppo Macintosh, ma probabilmente la migliore implementazione resta ancora l'Aztec C, che a buona ragione può essere considerato uno dei linguaggi più potente e completi disponibili su questa macchina.

Programmare in un linguaggio OO è più che un esercizio, una disciplina. Sembra, sotto certi aspetti, di costruire «un grattacielo usando piani e stanze prefabbricate, un po' come se si stessero usando i pezzi del Lego», per usare le parole stesse di Nygaard. In particolare, sempre nell'ambiente che ci interessa, esiste una vera e propria fortuna: quella di disporre di MacApp. Cosa è MacApp?

Intorno al 1985 Larry Rosenstein, Scott Wallace e Ken Doyle, assieme ad altri ricercatori della Apple, misero a punto un modello di applicazione tipica Macintosh strutturata in termini di «oggetti», come window, button, documenti, event manager. Il risultato di questa serie di routine bell'e pronte fu una libreria di «oggetti» appunto che prese il nome di MacApp; la libreria fu poi messa a disposizione degli utenti e più volte aggiornata. Questo cambiò in maniera pressoché istantanea il concetto di programmazione di Macintosh, fino ad allora considerato una macchina «difficile» da programmare, specie se si decideva di creare applicazioni ben integrate con l'interfaccia. Senza mezzi termini, l'uso di MacApp consentiva al programmatore di realizzare efficienti applicazioni con un risparmio di tempo pari anche all'80% rispetto alle tradizionali tecniche. Inoltre l'adozione di MacApp consentiva la creazione di applicazioni molto simili tra di loro nell'uso, conformi tra l'altro alle convenzioni cui l'utente era abituato nell'uso quotidiano della macchina. Inoltre veniva incontro alle aspettative dell'utente in termini di maneggio delle routine d'errore e di gestione degli oggetti fisici presenti sullo schermo (come DA, finestre, menu e altro). Al programmatore era solo richiesto di superare alcune abitudini programmatorie e di imparare le differenze tra lo stile dell'applicazione da lui realizzata e una tipica realizzazione secondo la tecnica MacApp. La cosa più eccitante in tutto questo è che un'applicazione MacApp, al di fuori dell'uso delle routine che vengono utilizzate tal quali, non è legata a un linguaggio particolare, ma può essere realizzata praticamente sia con la maggior parte di linguaggi di alto livello disponibili sul mercato, sia accedendo alle routine stesse attraverso l'Assembler.

Sebbene sia senz'altro la più famosa e articolata, MacApp non è la sola libreria OO presente sul mercato; fin dal 1970, addirittura, il Learning Research Group di Alan Kay, al Palo Alto Research Center progettò un sistema e un linguaggio chiamato SmallTalk, basato su una gestione di interfaccia utente window-oriented, corredata di una efficiente libreria di routine OO. La prima versione fu la SmallTalk-72 e (a riprova della facilità di uso di tale approccio) fu destinata ad un linguaggio di insegnamento per ragazzi; la casa si sviluppò verso orizzonti professionali, che diedero vita alla versione SallTalk-80. La libreria di tale blocco fu messa a disposizione anche separatamente e fu chiamata ModelViewController (MVC). Parallelamente fu realizzato, sempre in ambiente Mac, il Lisa Toolkit, destinato allo sfortunato predecessore del Mac, e comparvero linguaggi estremamente specializzati, come Neon, ExpertCommonLisp, ObjectLisp, ObjectLogo (ne ricordate la prova fatta in questa rubrica un anno e mezzo fa?) e Objective-C.

MacApp divenne disponibile nel 1985, appena dopo l'apparizione del Macintosh, come sistema di sviluppo incrociato Lisa-Macintosh, e a tutt'oggi diverse versioni successive hanno visto la luce. Oggi Apple fornisce un manuale, una serie di esempi di programmazione, e un blocco piuttosto cospicuo di materiale di riferimento. Ciononostante la programmazione orientata all'oggetto non è qualcosa di facile e intuitivo come imparare un linguaggio. Occorre predisporre la mente a realizzare nel modo migliore e ad accettare un modo di programmare un poco diverso dal solito.

La filosofia delle applicazioni dei programmi Mac

Alla comparsa, e prima dell'avvento delle tecniche della programmazione object-oriented, Macintosh era considerato, come dicevamo prima una macchina talvolta astrusa da programmare, almeno utilizzando le normali tecniche di costruzione di software, viste su altre macchine. Questo a causa, soprattutto, della interfaccia estremamente complessa della macchina, che imponeva che il programma «badasse» ad una serie piuttosto complessa ed articolata di eventi, come uso del mouse, del tasto di comando e di opzione, della barra di scorrimento (magari tenendo conto del refreshing dello schermo), degli alert box. La gestione di questi e di innumerevoli altri eventi doveva avvenire, oltre tutto, in maniera estremamente efficiente, senza conflitti e, soprattutto, in modo del tutto simile in tutte le applicazioni. Un programma che consente scelte attraverso la tastiera e non attraverso il mouse, un input non correggibile attraverso l'uso del mouse, una finestra che non si può restringere o allargare attraverso il resizing box, non avrebbe grande fortuna presso un utente Mac, troppo ben abituato ad avere uno schiavo servizievole ai suoi desideri e, perché no, capricci. Purtroppo quello che rappresenta una facility per l'utente è un lavoro in più per il programmatore, che, al contrario di quanto avviene in altri ambienti, è costretto a ragionare in termini di «A questo punto che potrebbe venire in mente di fare all'utente?», e a tutto ciò deve porre risposta e rimedio; tutto ciò si trasforma molto spesso in un inestricabile codice sorgente, estremamente complesso; per assurdo, abbiamo che il lavoro di base, realizzato attraverso schemi convenzionali, per rendere un programma Mac integrato con la sua interfaccia è addirittura maggiore di quello richiesto per lo sviluppo dell'applicazione stessa. Ma è possibile salvarsi da questa enorme massa di lavoro adottando maggiori o minori porzioni dell'Interfaccia Utente Mac.

Tutto quanto abbiamo precedentemente nominato per esemplificare la complicazione dell'interfaccia Macintosh ricade, nella maggior parte dei casi in questo ambiente. La standardizzazione di questa interfaccia, che fa felice l'utente e rende le applicazioni così semplici da usare, rappresenta una barriera temibile per lo scrittore del programma, specie se è alle prime armi e proviene da altri ambienti (MS-DOS in primis). A ciò, lo ripetiamo per la ennesima volta, supplisce l'uso della programmazione OO.

MC