

Cosa sia un linguaggio di programmazione è presto detto: una macchina viene schematizzata, secondo una diffusa notazione, come uno *stack* di macchine virtuali (Vm). Sul fondo risiede la “macchina fisica”, che è meramente l’insieme di circuiti elettronici di cui è composta. Gli strati per così dire superiori, invece, non esistono nella realtà, non sono palpabili e sono detti strati “logici”. Questi ultimi si introducono per poter astrarre dal funzionamento dei singoli transistor e poter vedere la macchina al livello di “quel che fa” più che di “come è fatta”. Il risultato è che l’utente finale, allorché col proprio mouse stuzzicherà le icone del suo *desktop*, non dovrà preoccuparsi del parapioggia di circuiti elettronici che sta attivando per vedere illuminarsi una stringa di testo. Ora, il linguaggio di programmazione non costituisce altro che uno dei tanti livelli di astrazione.

Più ci si pone in alto nella pila di astrazione, più comunicare con la mac-

china diventa intuitivo ed assistito da interfacce. Ma allora perché non lavorare sempre al più alto livello possibile? Semplice: perché più si sale nella gerarchia degli strati virtuali, più è ristretta la visione della macchina. Il che vuol dire che in un ambiente grafico ci si muove allegramente, ma si riduce notevolmente il grado di interazione con la macchina stessa.

Proprio a questa visione si riconduce il modo di catalogare i linguaggi di programmazione in linguaggi di “alto livello” o “basso livello”. “C” è sicuramente il linguaggio di più basso livello tra quelli comunemente utilizzati. Java, invece, astrae da qualsiasi cosa: dal disco alla memoria; dallo schermo al sistema operativo. Esso si pone di gran lunga più in alto di C nella gerarchia. C++ si trova per così dire in mezzo e costituisce un compromesso tra le potenzialità di C e il comportamento amichevole verso il programmatore tipico di Java.

Il tormento fra l’Uomo e

Tra gli innumerevoli linguaggi ormai affermati, quale è la vera e propria tecnologia premiera? O meglio, esiste un linguaggio che garantisce un buon prodotto software?

di Renato Mancuso



C è estremamente performante: il codice generato programmando in C non è sovraccaricato con tutti i fronzoli dei suoi discendenti orientati agli oggetti. È insomma una sorta di linguaggio macchina aggraziato che dispone di strumenti che permettono di trapassare gli strati di virtualizzazione e accedere direttamente all’hardware.

Di Java si può affermare tutto il contrario: con pochissime righe di codice si creano programmi in grado di essere eseguiti su qualsiasi macchina. Questa possibilità è garantita dall’inserimento di uno strato di virtualizzazione della piattaforma intermedio, detto *Java Virtual Machine* (Jvm). È semplicissimo costruire interfacce grafiche e applicazioni che sfruttino connessioni di rete, database e quant’altro. Il prezzo da pagare è sempre e comunque la per-

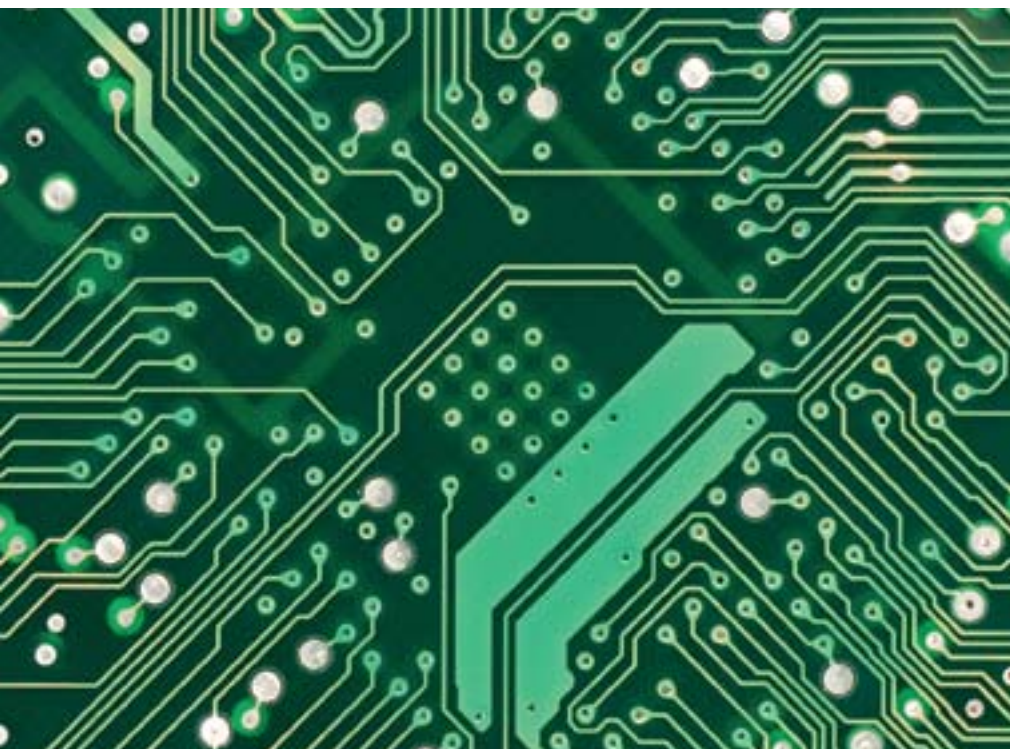
Foto: iStockphoto.com (enri-poloskun; eduenteg)

ato dialogo la Macchina

matismi, quale è quello che l'evoluzione
ha scelto sempre vincente per sviluppare

l'altro – il linguaggio di programmazione più opportuno rispetto alle esigenze di sviluppo.

Come è intuibile, tali esigenze sono di tipo sempre differente, perché in generale differenti sono le specifiche su cui si vuole che il software venga costruito. Questo implica che semplicemente non esiste una scelta ottima. In alcuni casi sviluppare velocemente un software che sia facilmente integrabile costituisce l'opzione vincente; altre volte, invece,



formance e sovente anche l'incapacità di comunicare con la piattaforma sottostante.

Allora, quale linguaggio è destinato ad affermarsi nel lungo periodo? Si potrà chiudere un occhio sulle performance e occuparsi solo della portabilità? O viceversa?

Quando si intende produrre un software, si utilizzano modelli di sviluppo articolati in fasi consecutive, durante le quali si definisce la struttura del prodotto in maniera iterativa e si procede per raffinamenti sempre successivi. La fase di stesura del codice corrisponde a una delle ultime di questo processo e va sotto il nome di "implementazione". Ora, quando il software è stato ben definito e il progetto dettagliato è pronto, si devono fare i conti con le tecnologie che si hanno a disposizione per scegliere – tra

il programma dovrà sfruttare appieno le potenzialità del sistema su cui viene installato.

Nel ristretto ambito di un dato contesto di sviluppo, quindi, le scelte dei linguaggi sono piuttosto chiare, alla luce dell'esperienza maturata e dei vantaggi osservati.

Linguaggi di alto livello come Java o Php o Asp sono notevolmente utilizzati nella produzione di applicazioni Web. Essi mettono a punto un buon layer di interazione tra applicativo e Web server, consentendo una rapida scrittura di codice che rispetti canoni di sicurezza, dinamicità dell'output prodotto e integrazione con altre tecnologie Web. Non solo: ora che le *virtual machines* in grado di eseguire codice Java hanno raggiunto un buon grado di ottimizzazione e riescono agevolmente a operare anche

su macchine dalle prestazioni medio-basse, quali i personal-computer, sta prendendo piede la tendenza a codificare in questo linguaggio anche software piuttosto corposi: dal *photo-editing* (Iris) agli ambienti di sviluppo integrati (NetBeans).

Scegliere un linguaggio di programmazione del livello più basso possibile è invece praticamente obbligatorio nel caso di sviluppo di software critico. Ancora adesso, la parte fondamentale di un sistema operativo anche evoluto viene scritta in C. In primo luogo perché sarebbe impossibile scrivere uno dei primi strati di virtualizzazione dell'hardware, quale è il sistema operativo, presupponendo l'esistenza di un'ulteriore strato intermedio. In secondo luogo perché, quando anche lo si facesse, il risultato finale sarebbe di qualità così scadente da non essere nemmeno lontanamente utilizzabile. Basti pensare che è addirittura sconsigliato l'impiego di C++ per la stesura di simili componenti software.

Per concludere, portiamo un esempio esplicativo: la scrittura di codice estremamente performante è importante nel caso si debba implementare un sistema per la gestione di basi di dati (Dbms, *Data Base Management System*). Tali software possiedono restrizioni molto forti sui tempi di servizio e il numero di richieste per unità di tempo da soddisfare. L'ottimizzazione in questi casi è praticamente vitale. MySQL, uno dei più celebri Dbms *open-source*, è infatti scritto in C e C++, ma se si va a vedere qual è l'interfaccia ai dati che esso offre, ci si imbatte in un linguaggio di alto livello come Sql (*Structured Query Language*), a sua volta utilizzato all'interno di altri linguaggi per realizzare la comunicazione con database remoti e non.

Quello che si può dire, in sostanza, è che l'esistenza di una molteplicità di linguaggi con le più diverse caratteristiche ha un ovvio motivo di esistere. Non solo, ma è inaccettabile l'idea che questa varietà sia destinata a scemare nel tempo. La tendenza più prevedibile è tutt'altra: che si creino mezzi di integrazione tra tecnologie di sviluppo sempre più evolute e che siano in grado di bilanciare pregi e svantaggi derivanti dall'uso dei numerosi linguaggi in gioco. ■