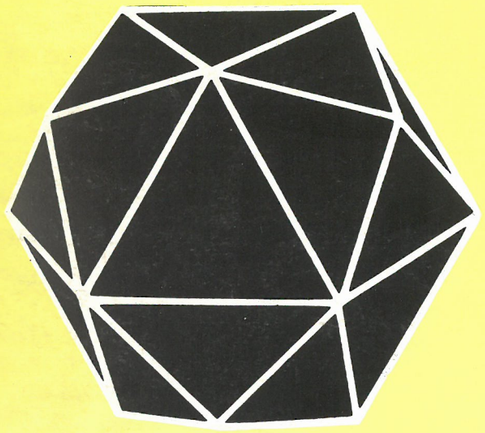
Spedizione in abbonamento postale, Gruppo IV, 1° semestre.
Pubblicità Inferiore al 70%.
N. 41/42, Gennaio - Luglio 1991

TAXE PERÇUE
P.P. Tassa riscossa PT Piacenza F. Italia

Quaderni di informatica

Rassegna di tecnologia ed applicazioni degli elaboratori elettronici

a cura del Centro Studi Informatica e Automazione Bull Italia



41/42
Anno XVII - Numero 1/2 - 1991

FPDM-48

Worldwide
Information
Systems

Bull

Quaderni di informatica

Rassegna di tecnologia ed applicazioni degli elaboratori elettronici
Anno XVII - Numero 1/2 - 1991

Sommario

Direttore Responsabile
Franco Filippazzi

Comitato di Redazione
Antonio Cicu, Carlo Falcetti, Paolo Lupo,
Mario Nobile, Giulio Occhini,
Fulvia Sala

Segreteria di Redazione
Elena Pighin

Redazione
Bull Italia
Centro Studi Informatica e Automazione
Via G.M. Vida, 11 - 20127 Milano
Tel. (02) 6779 - 3783

Stampa
Caleidograf

Autorizzazione del Tribunale di Milano
n. 93 del 20 Marzo 1974

Paradigmi di interazione con realtà artificiali

ROBERTO POLILLO

Università di Milano,
Dipart. di Scienze dell'Informazione
e Etnoteam SpA Milano

1. INTRODUZIONE

La interazione con le complesse *realtà artificiali* realizzabili con le potenti workstation di oggi (e, più ancora, di domani) pone nuovi e stimolanti problemi al progettista di interfacce uomo-macchina.
Vari *paradigmi di interazione* sono infatti possibili, alcuni di tipo tradizionale, altri - suggeriti da nuovi device di interazione - tutti da esplorare. Sebbene questa disciplina sia ancora ben lontana dall'essere consolidata e la letteratura sistematica su questi problemi pressochè inesistente, molti spunti possono essere suggeriti dalle soluzioni adottate nei computer game, nei sistemi ipermediali attualmente oggetto di numerosi sviluppi prototipali, e dalle sperimentazioni in corso presso vari laboratori di ricerca.
Questo articolo presenta una prima discussione su questi temi, e propone vari esempi tratti da prodotti software reperibili sul mercato (principalmente, computer game).

2. Realtà artificiali e realtà virtuali

Realtà artificiale

"Realta' artificiale" significa *un mondo dentro il computer, con il quale noi possiamo interagire.*
È possibile realizzare tipi di mondi molto diversi, dentro un computer: mondi puramente matematici, mondi che rispecchiano la natura fisica in cui viviamo, o anche *mondi immaginari,* con loro peculiari leggi della fisica, o senza legge alcuna ("caos").
Con la potenza di calcolo delle attuali workstation, e grazie alle tecniche della computer graphics, siamo oggi in grado di visualizzare sullo schermo di un computer mondi artificiali complessi, con grande dettaglio ed impatto visivo. Inoltre, possiamo far si' che queste visualizzazioni cambino rapidamente e in modo continuo mentre noi cambiamo il nostro punto di vista sulla scena.
Questi mondi possono essere esplorati e manipolati mediante opportuni device di interazione. Possiamo *viaggiare* in questi mondi, *modificarli, interagire* con essi, ed abituarci alle loro *leggi naturali,* che non necessariamente riflettono le leggi della natura "reale":
" 11 computer è diventato lo specchio in cui il reale rivive una sua nuova realtà al di fuori dello spazio e del tempo: la realtà artificiale." (DEGL89).
Esempi tradizionali di applicazioni di questo tipo sono i *simulatori di volo* professionali, e vari tipi di computer game, dai simulatori di guida "giocattolo" (aerei, automobili, sottomarini, ecc.) a molti *arcade game* (labirinti, invasori spaziali, ecc.) fino al celebre *Dragon's Lair,* un *cartone animato interattivo* realizzato alcuni anni fa utilizzando un videodisco Laservision.
Quest'ultimo permetteva al giocatore di controllare, mediante un *joystick,* le azioni del protagonista dell'avventura, che doveva superare ostacoli di ogni sorta all'interno di un castello in un mondo di fantasia.
La Fig.1 mostra una immagine video tratta dal sistema *CT5-A,* un sofisticato simulatore di volo disponibile sul mercato nei primi anni ottanta, della Evans & Sutherland. La Fig.11 mostra, invece, il gioco best-seller *Flight Simulator* della Microsoft (MICR86), che permette di "guidare" un velivolo Chessna, e che ha ispirato un gran numero di fortunati prodotti software per il mercato degli home computer (ad esempio, il più recente *Falcon,* un simulatore di caccia F-16 prodotto dalla Spectrum HoloByte, SPEC87).
Tutti questi esempi possono in realta' essere fuorvianti, in quanto potrebbero suggerire che una realtà artificiale, per definizione, rifletta il mondo reale in cui viviamo (anche se in modo molto semplificato, come accade nei computer game).
Questo non è necessariamente vero. Possiamo, infatti, concepire mondi artificiali completamente *diversi* dal mondo a cui siamo abituati: mondi fatti di entità matematiche, mondi che rappresentano una realtà fisica, ma su *scala differente* (macroscopica o

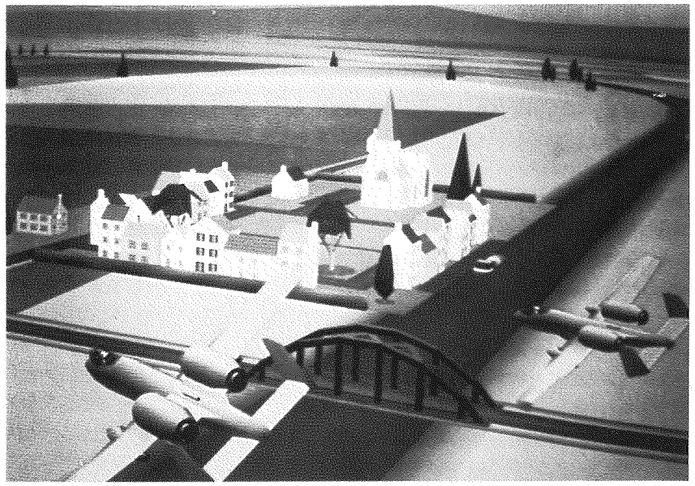


Fig. 1 - Il simulatore di volo CT5-A.

microscopica), mondi di fantasia come in molti computer game del tipo *shoot'em up* (ad esempio, il classico *Space Invaders)* o come nel film *Tron* realizzato da Stephen Lisberger per la Walt Disney, o perfino mondi fatti di documenti e informazioni, in cui possiamo *navigare,* come nella metafora usata nei sistemi *ipertestuali* o *ipermediali* (si veda, ad esempio, FAIR89, HAMM88). Si rimanda il lettore a BR0088 per esempi tratti dal settore della modellazione scientifica.

Realtà virtuale

La terminologia in uso non è affatto uniforme: nella letteratura, termini come "realtà artificiale", realtà virtuale", "mondi sintetici, "mondi virtuali", ... sono usati in modo più o meno intercambiabile.
In questo articolo, seguendo una distinzione proposta da G. Degli Antoni, useremo il termine *realtà virtuale* (in contrapposizione a quello di "realtà artificiale") in tutti quei casi in cui *esiste qualche tipo di interazione fra il mondo artificiale e il mondo reale, in modo tale che interagendo con il primo sia possibile interagire con il secondo, e viceversa.*
Ciò permette di controllare il reale, attraverso il controllo di una sua rappresentazione (e viceversa), come avviene nei tradizionali sistemi di controllo di processo, di building automation, e così via.
Ma possiamo immaginare esempi di realtà virtuali molto più complessi, in cui la interazione fra "reale" e "artificiale" sia molto più stretta che in un semplice controllo di processo.

Per esempio, in (DEGL89) G. Degli Antoni descrive una applicazione immaginaria nell'area dei servizi pubblici. Un Ufficio Pubblico (per esempio, un Ministero) è visualizzato sullo schermo di un computer, con tutti i suoi corridoi, porte, impiegati ed archivi. L'utente può "entrare" e "muoversi" al suo interno, aprire porte con un semplice click del mouse, puo' consultare gli archivi di suo interesse e porre domande agli "impiegati artificiali" che incontra lungo il suo percorso.
Quando l'impiegato artificiale non è in grado di soddisfare le richieste del visitatore, si metterà in contatto con *l'impiegato reale presso il Ministero reale*, che in qualche modo apparirà anch'egli sullo schermo, e parlerà direttamente a *entrambi gli interlocutori.*

In questo esempio, pertanto, la interazione è *triplice:* esiste una interazione fra utente e mondo artificiale, fra mondo artificiale e mondo reale, e fra utente e mondo reale (anche se, in qualche modo, con la mediazione del computer).
La distinzione fra realtà artificiale e reale è schematizzata in Fig.2.

3. Realtà artificiale e interfaccia utente

Per interagire con le realtà artificiali (o virtuali), abbiamo bisogno di una opportuna *interfaccia utente,* che ci permetta di *visualizzare, esplorare, modificare* tali mondi.
Per questo, vari *paradigmi di interazione* possono essere concepiti, a seconda dei *device di interazione* che utilizziamo, e delle *metafore* alle quali ci ispiriamo nella progettazione della interfaccia utente.
Lo studio e la sperimentazione di queste possibilità rappresenta una nuova affascinante disciplina, tuttora agli inizi, che ha prodotto per ora poca letteratura. Tuttavia numerosi spunti possono essere tratti da ambiti applicativi diversi, alcuni già ben consolidati: computer game, sistemi ipermediali, nuovi device di interazione uomo-macchina, simulatori, ecc.
Esaminando questi esempi, è possibile proporre una prima classificazione dei paradigmi di interazione con realtà artificiali in cinque classi:
- *linguaggi di comandi*
- *menù*
- *manipolazione diretta*
- *navigazione*
- *presenza virtuale*.

Questi paradigmi sono schematizzati in Fig.3, e verranno brevemente esaminati nei paragrafi seguenti.

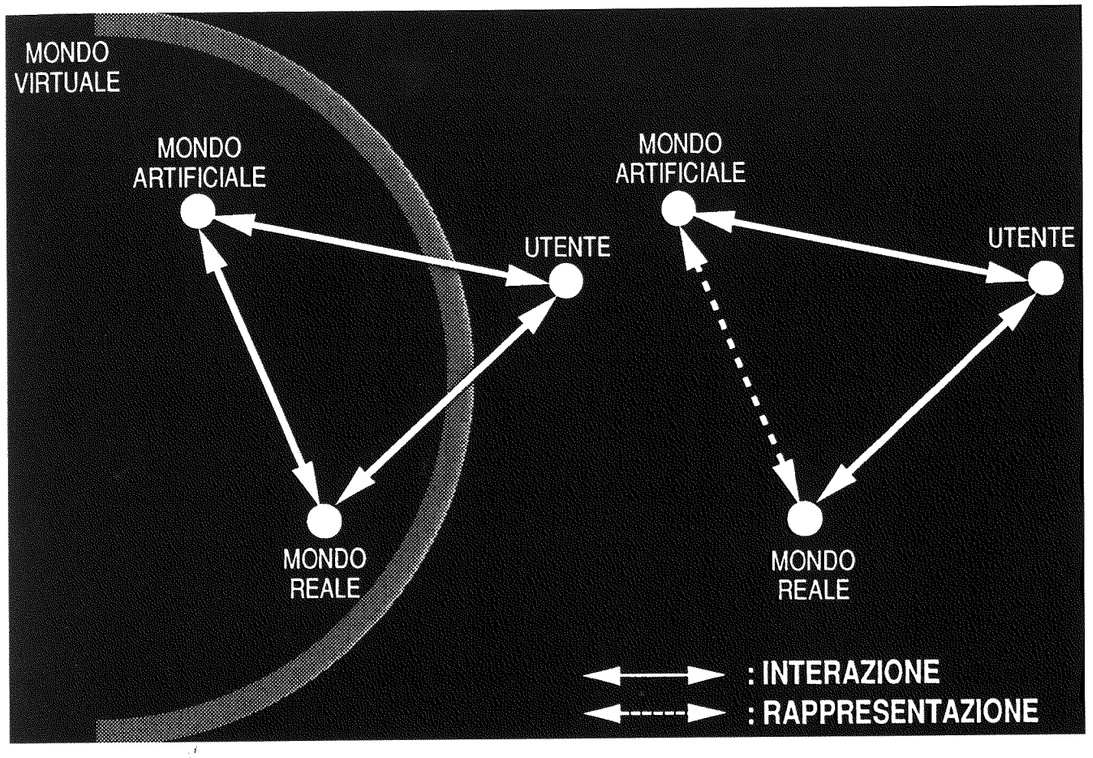


Fig. 2 - Mondi artificiali e reali.

2. Linguaggi di comandi.

I *linguaggi di comandi* costituivano il tipico paradigma di interazione uomo-calcolatore, quando il dialogo con il computer veniva effettuato mediante *terminali scriventi (teletype).*
In applicazioni di realtà artificiale, questo e' il paradigma tipico utilizzato nei *text adventure game* classici (dal primo *Adventure,* scritto da W.Crowther per il PDP-11 nel 1975, fino ai best-seller della Infocom, da *Zork 1: The Great Underground Empire* di M.Blank e D. Lebling ai molti giochi successivi, cfr. MCGA84).
In questi sistemi (Fig.3a), la realtà artificiale (cioè il mondo dell'avventura) non è visibile, ma "immaginata" dal giocatore, che controlla un proprio *alter ego* che vive, si muove, combatte (e a volte muore) dentro di essa.

L'interazione avviene per mezzo di un *linguaggio pseudonaturale,* di solito dotato di una struttura grammaticale molto semplice e ispirato all' inglese. Di contro, la struttura lessicale - cioè il *dizionario* del gioco - è spesso piuttosto ricca.
In molti casi, il *parser* di questi giochi si limita a riconoscere, nella frase digitata dal giocatore, alcune *parole chiave,* secondo la tecnica suggerita per la prima volta da *Elisa,* un programma ormai classico, che "finge di essere uno psicanalista" scritto da Weizenbaum piu' di 25 anni fa (WEIZ66).
Come esempio tipico dei dialoghi realizzati da questi giochi di avventure qui di seguito è riportato uno dei possibili inizi di *Hitchhiker's Guide to the Galaxy,* un altro gioco piuttosto noto della Infocom, INF088a (il testo digitato dal giocatore è scritto in caratteri maiuscoli):

>SWITCH THE LIGHT ON
Good start to the day. Pity it's going to be the worst of your life. The light is now on. Bedroom, in the bed. The bedroom is a mess. It is a small bedroom with a faded carpet and old wallpaper. There is a washbasin, a chair with a tatty dressing gown s/ung over it, and a window with the curtain drawn. Near the exit leading south is a phone. There is a flathead screwdriver here. (outside the bed) There is a toothbrush here. (outside the bed)

>GET OUT
(bed) Very difficult, but you manage it. The room is still spinning. It dips and sways a little.

>GO TO THE DOOR
It's here!

>OPEN THE DOOR
It is already open.

>GO THROUGH THE DOOR
Did you have any particular direction in mind?

>NORTH
You can't go that way

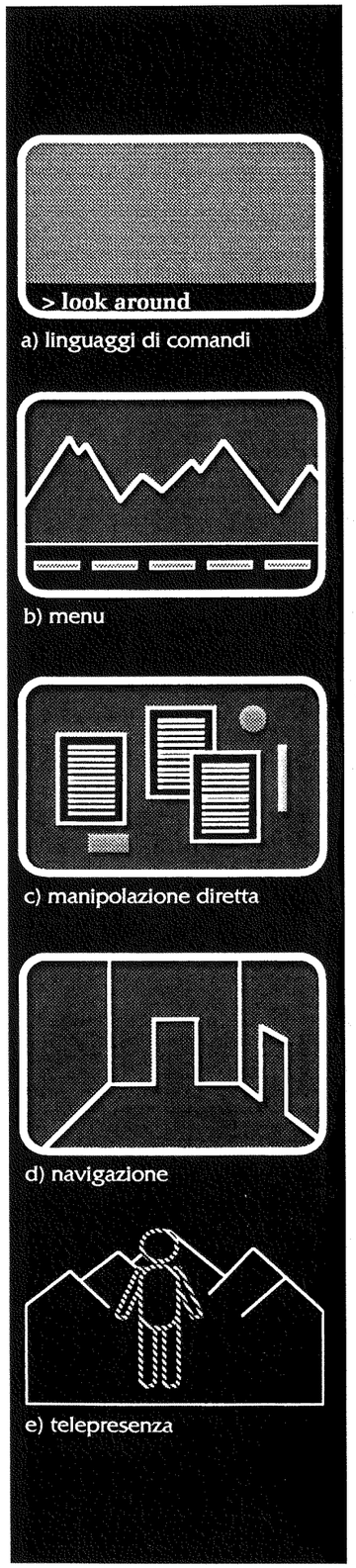


Fig. 3 - I cinque principali paradigmi di interazione con realtà artificiali.

>SOUTH
You miss the doorway by a good eighteen inches. The wall jostles you rather rudely.
. . . . . .

Il problema principale con questo tipo di interfaccia (come è evidente dall'esempio di cui sopra) è che, per procedere nell'avventura, il giocatore è continuamente costretto a risolvere dei *puzzle di tipo linguistico* ("come dovrò esprimere il comando che intendo dare?"), che nulla hanno a che fare con il dominio dell'avventura che sta vivendo.
Per questo motivo, Ted Nelson chiamo' una volta questi giochi "*guess my command games*".
Da un lato, questo può essere considerato una sfida alle capacità intellettuali del giocatore (e può aver contribuito in modo significativo al successo commerciale di questi giochi). Dall'altro, però, questo tipo di dialogo viola uno dei principi fondamentali della progettazione di buone interfacce utente, cioè il *principio di trasparenza,* che è ben formulato nella seguente citazione da Winograd e Flores (WIN86):
"Lavorando con le persone, noi stabiliamo dei domini di conversazione nei quali la nostra comune pre-comprensione (pre-understanding) *ci permette di comunicare con un minimo di parole e di sforzo conscio. Noi diventiamo consapevoli della struttura della conversazione soltanto quando c'è un qualche tipo di rottura* (breakdown), *che richiede un'azione correttiva. Se le macchine potessero comprendere nello stesso modo in cui comprendono le persone, la interazione con i computer sarebbe ugualmente trasparente* (...). *Una cattiva progettazione forza l'utente ad avere a che fare con complessità che appartengono al dominio sbagliato." (WIN086)*

3. Menù

I *menù* costituiscono uno dei piu' diffusi paradigmi di comunicazione uomo-calcolatore, fin dai tempi in cui il device di interazione standard era il *terminale alfanumerico non programmabile, o VDU (Video Display Unit).*
Tipicamente, le applicazioni utilizzavano *menù a pieno schermo.* Il video del terminale mostrava un elenco di azioni possibili, e all'utente veniva chiesto di sceglierne una. Questa scelta, tipicamente, causava la presentazione di un *menù di livello inferiore,* oppure di una *maschera (form)* per immettere (o richiedere) dati. (Questo paradigma di interazione è stato di recente standardizzato dalla IBM, nella *Entry Model Interface* del Common User Access dell'architettura SAA, cfr. IBM89a).
Questo tradizionale paradigma ha avuto in anni recenti una nuova evoluzione con la diffusione dei personal computer, nei quali le capacità di elaborazione *locale* della macchina hanno permesso, fra utente e applicazione, una interazione molto più *rapida* di quanto non potesse avvenire con i terminali tradizionali. Su un PC, tipicamente, la maggior parte dello spazio sullo schermo viene riservata ai dati dell'applicazione (ad esempio, uno spreadsheet), e i menù vengono confinati a una ridotta zona dello schermo, molto spesso una singola linea *(menù a singola linea).*
Nei sistemi con menù a singola linea, si sono consolidati, in effetti, due paradigmi principali:
- il paradigma della *barra dei menù (menu bar)*, reso popolare dallo spreadsheet Lotus 1-2-3;
- il paradigma dei *menù a tendina (pull-down menus)*, reso popolare dal Macintosh della Apple, e in seguito adottato da un grande numero di applicazioni su PC, e anche su terminali alfanumerici tradizionali (come, ad esempio, nel Text Subset of the Graphical Model dello standard Common User Access della IBM, cfr.lBM89a).

In applicazioni di realtà artificiale, entrambi i tipi di menù a singola linea sono stati utilizzati, a partire dalla metà degli anni ottanta, in numerosi giochi di avventure che sfruttavano le capacità grafiche degli home computer.
Nei primi giochi di questo genere, i comandi disponibili venivano selezionati da un menù a barra visualizzato (per esempio) nella riga più bassa del video. Nella restante porzione di video, veniva visualizzato il mondo dell'avventura (anche se con grafica piuttosto rudimentale) mostrando una *immagine statica* della scena corrente (Fig.3b) .
In altri casi, i comandi disponibili vengono selezionati da menù a tendina. Questo è, per esempio, il caso di *Quarterstaff*, pubblicato dalla Infocom per il Macintosh (INF088b). In questo gioco, le capacità grafiche del Macintosh vengono utilizzate per visualizzare una mappa del mondo, riportante la posizione corrente dei personaggi, oppure il ritratto di un nuovo personaggio o, ancora, dei pittogrammi che rendano più immediatamente comprensibili le voci dei menù (Fig.4a e 4b).

4. Manipolazione diretta

Secondo Shneiderman, la idea centrale delle interfacce a manipolazione diretta è"*la visibilità dell'oggetto di interesse; azioni rapide, reversibili, incrementali; e la sostituzione della complessa sintassi dei linguaggi a comandi con la manipolazione diretta dell'oggetto di interesse*" (SHNE83).
Questo paradigma ha probabilmente le sue radici nei primi editor a pieno schermo, in cui il testo, alla posizione indicata dal *cursore,* veniva manipolato con facilità mediante una varietà di comandi realizzati con *tasti funzione* (EMACS, vi, Wordstar, ...).
Oggi, i piu' tipici esempi di manipolazione diretta si trovano nelle interfacce basate su *mouse* utilizzate sulle workstation o su personal computer (Fig.3c).

"Flatlands"
Gli esempi piu' noti sono i sistemi basati sulla cosiddetta *metafora della scrivania (desktop metaphor),* dallo *Star* della Xerox (SMIT82) al *Finder* del Macintosh della Apple (APPL87), al *Presentation Manager* per OS/2 (IBM89b), al *Windows* 3 per MS-

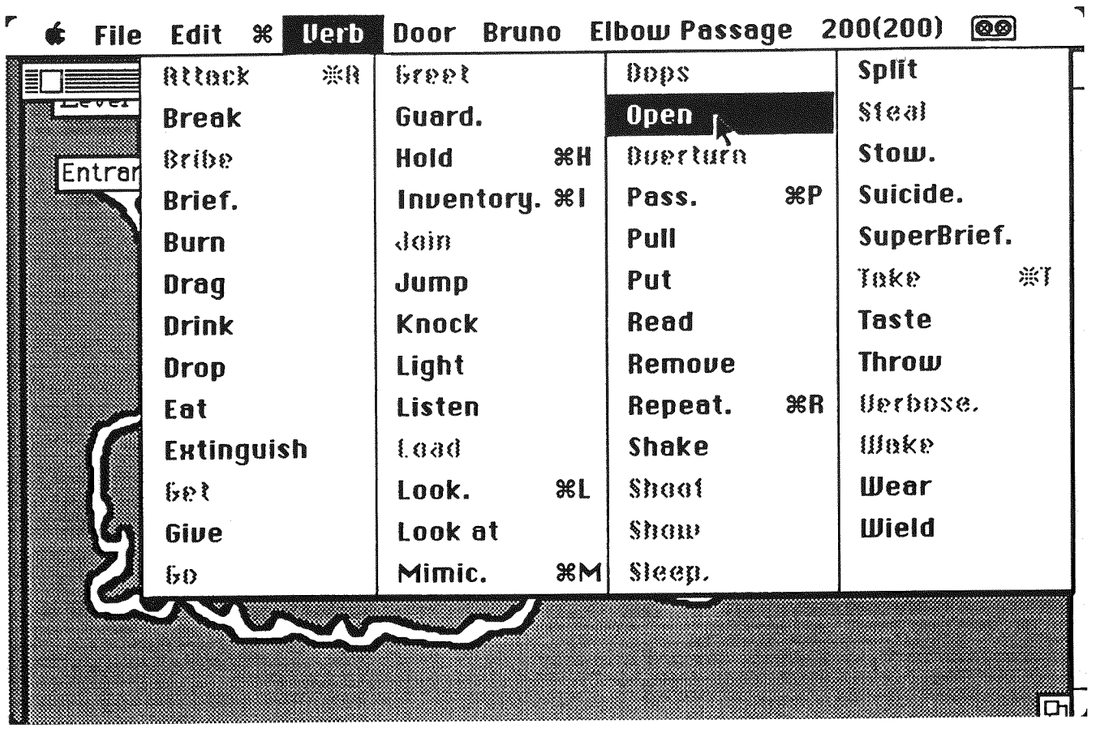


Fig. 4a - Quarterstaff: un menù a tendina.

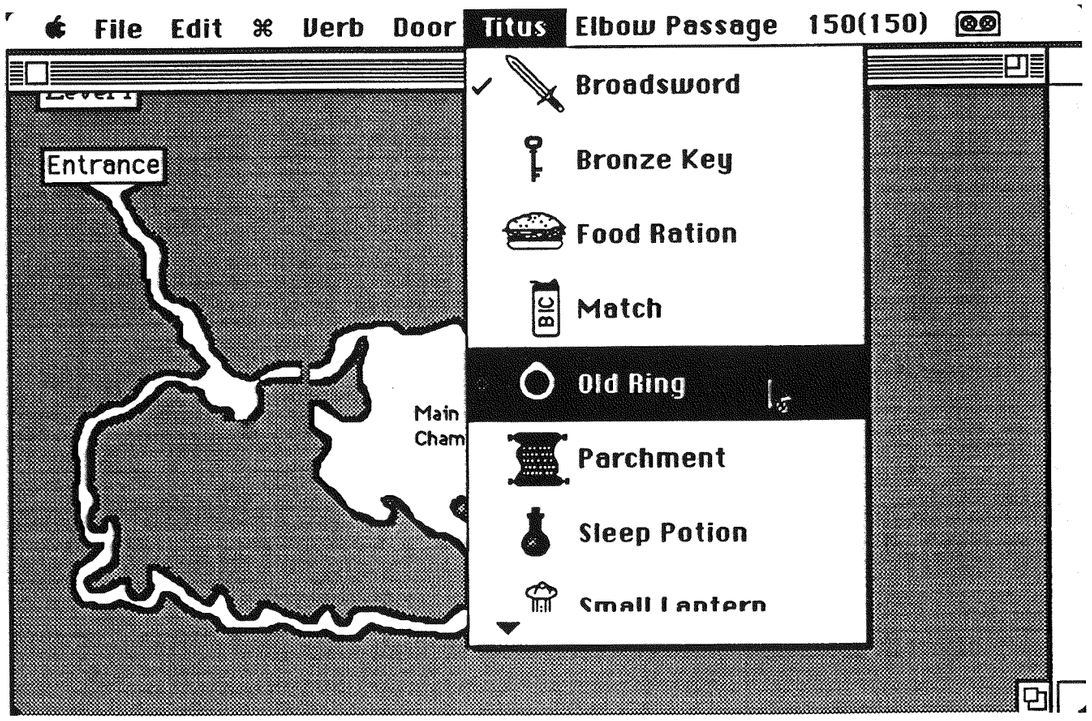


Fig. 4b - Quarterstaff: un menù iconico.

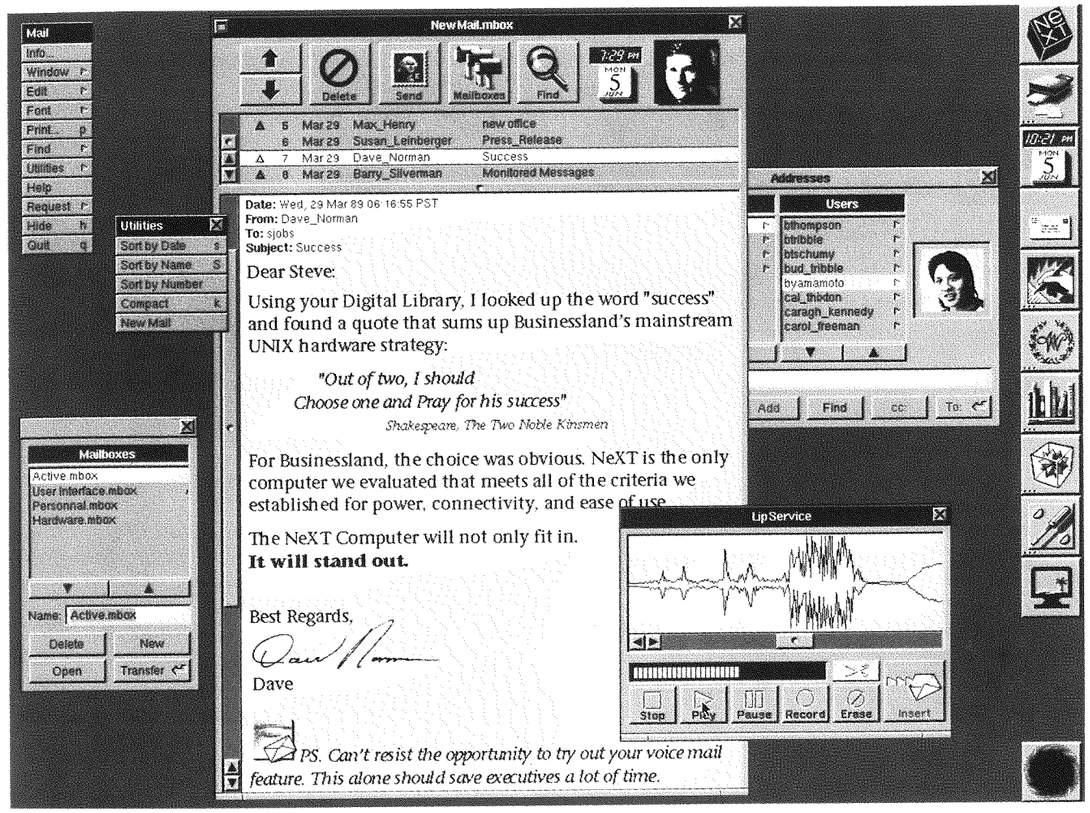


Fig. 5 - Il "workspace" del sistema Next.

DOS della Microsoft, all' *OpenLook* della Sun Microsystems (SUN89), al *Motif* di OSF (OSF89), fino alla sofisticata interfaccia del sistema Next (NEXT89). In questi sistemi, con le crescenti capacità grafiche delle workstation, la originaria rappresentazione "piatta", bidimensionale, degli oggetti della scrivania acquisisce, progressivamente, una sorta di stile a "bassorilievo". Questo è particolarmente evidente nella interfaccia della Next, per la quale si potrebbe più propriamente parlare di *metafora del pannello di controllo* (Fig.5).
Tutti i sistemi citati sono stati progettati, principalmente, per applicazioni d'ufficio: pertanto, i mondi artificiali che essi rappresentano sullo schermo sono costituiti, essenzialmente, di *documenti, cartelle, cestini per la carta straccia,* e cosi' via.
Un esempio di natura diversa è mostrato in Fig.6. Esso è tratto da *SimCity,* un gioco per Macintosh pubblicato di recente dalla Broderbund (BROD89), che permette di creare, modificare e simulare il comportamento di una città. La figura mostra la visione dall'alto di una città, durante la simulazione. Il giocatore può modificare la città in molti modi (ad esempio, costruendo strade o distruggendo ponti), utilizzando svariati appositi strumenti che possono essere selezionati, col mouse, da un *tooltray* visualizzato sulla sinistra dello schermo.
Nonostante l'apparenza a "bassorilievo" di molti di questi esempi, essi rappresentano essenzialmente mondi bidimensionali, o "flatlands", come potremmo chiamarli ispirandoci alle fantasie di E. Abbott e C.Hinton (ABB082, HINT88).
Con tali "mondi piatti" noi interagiamo direttamente *con la nostra mano,* rappresentata dal *puntatore del mouse,* e mediante l'uso di un elementare *linguaggio gestuale:* gesti semplici e standardizzati denotano le azioni che desideriamo effettuare (ad esempio, *click* significa "seleziona questo, *doppio-click* significa "apri questo, eccetera).

Dalla metafora del cursore alla metafora della mano.
A partire dal tradizionale mouse, ormai vecchio di venticinque anni (ENGL67) - che può avere uno, due o tre bottoni - è possibile progettare nuovi device per *manipolazione diretta.* Alcuni esempi sono i *touch sensitive screen,* la *trackball,* il *data-glove* (FISH86), il *force and position sensitive screen* (M I N S84) .
A questo proposito, è possibile osservare che i sistemi a manipolazione diretta mostrano una lenta, progressiva evoluzione dalla *me-*

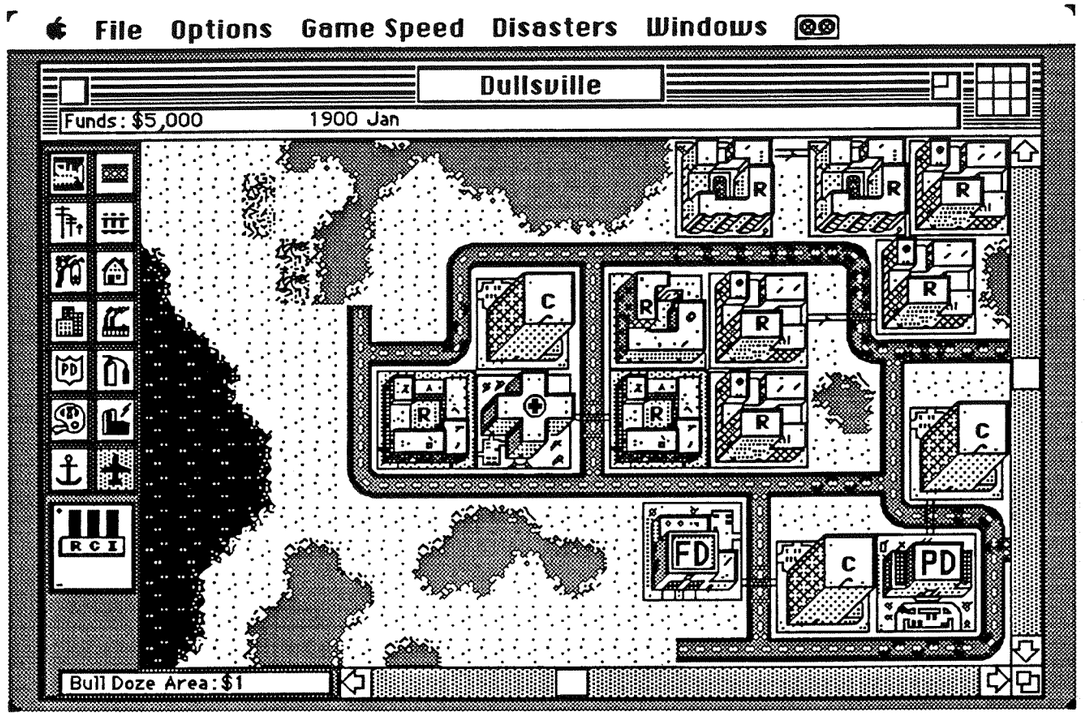


Fig. 6 - SimCity.

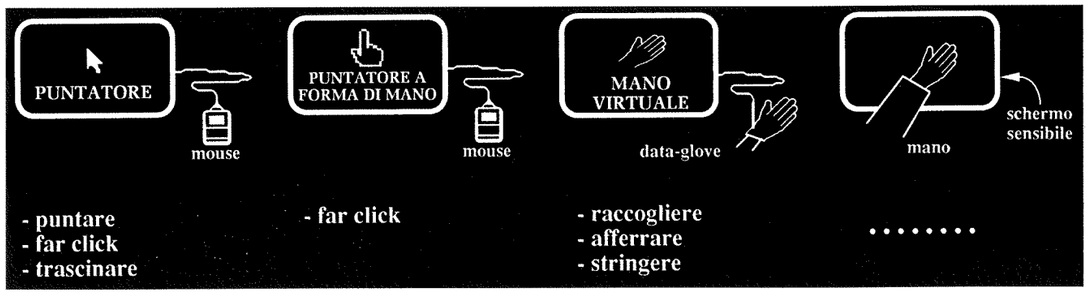


Fig. 7 - Dalla "metafora del cursore" alla "metafora della mano".

tafora del cursore alla *metafora della mano,* come si può verificare dagli esempi qui di seguito descritti (Fig.7).

Mouse e "piccole mani".
Un primo passo, in questa progressiva evoluzione, si può vedere in *Hypercard* (APPL87b), dove il tradizionale puntatore a forma di freccia del Macintosh (Fig.7a) si trasforma in una "piccola mano", o *hand-shaped pointer* (Fig.7b). Il linguaggio gestuale, già semplice, si semplifica ulteriormente: la piccola mano di Hypercard può soltanto *indicare* e *premere* (click): ogni altra azione, come il *trascinare* e il *doppio click,* è eliminata.
Con queste semplificazioni, la interfaccia utente può basarsi su una *singola* idea elementare: la *metafora dei bottoni.* I bottoni possono essere visibili o invisibili, possono avere forme e etichette differenti, possono rispondere in maniera differente (con un feedback visivo) al click dell'utente, e possono essere associati a *script* che vengono eseguiti (interpretati) quando il bottone viene premuto.
Su questi concetti si basano, ora, le numerosissime applicazioni in commercio basate su Hypercard *(stackware).*

Data-glove e mani virtuali.
Un device molto interessante, già disponibile sul mercato ma dalle

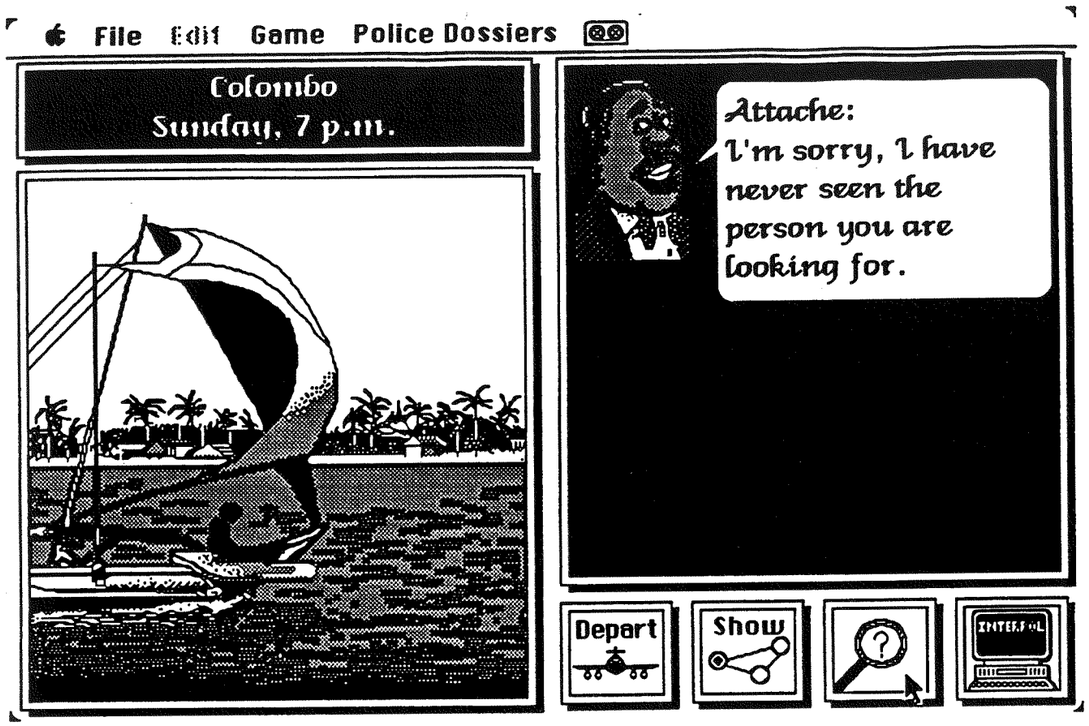


Fig. 8 - Where in the world is Carmen Sandiego.

potenzialità ancora poco esplorate, è il *data-glove.* Si tratta di un guanto sottile dotato di sensori a fibre ottiche che possono "sentire" la curvatura delle dita della mano che lo indossa, sviluppato dalla VPL Research Inc., di Palo Alto, California (cfr. ad esempio FISH86, TELL88, FOLE87).
Varie esperienze d'uso di questo device innovativo sono ora in corso. Ad esempio, in una interfaccia utente sperimentale sviluppata ai Bell Labs (WEIM89), il tradizionale puntatore del mouse si trasforma (anche nella forma) in una "mano virtuale", che viene controllata dalla "mano reale" dell'utente, che indossa il data-glove e si sposta sul piano "reale" della scrivania.
In questa applicazione, le azioni da eseguire sugli oggetti artificiali (bidimensionali) rappresentati sul video vengono espresse mediante un semplice linguaggio gestuale, comprendente gesti quali "raccogliere", "afferrare", "stringere", e così via (Fig .7c).

Schermi sensibili e mani reali.
Nelle ricerche di Margaret Minsky al Media Lab del MIT (MINS84), la mano virtuale viene sostituita da una mano "reale": uno schermo sensibile di tipo sperimentale *(force and position sensitive screen)* è in grado di rilevare le azioni effettuate dal dito dell'utente sullo schermo, in termini di posizione, pressione e forza trasversale (Fig.7d).
Con questa tecnologia, sono state sviluppate applicazioni per *disegnare con le dita sul* video *(fingerpainting),* per muovere, o addirittura "lanciare" in un altro punto del video, con il movimento e la pressione delle dita, degli oggetti virtuali a forma di bottone, e così via.
Scrive Margaret Minsky:
"Noi vogliamo creare, dentro il computer, mondi che possano essere manipolati in un modo concreto e naturale usando il gesto come modo di interazione. L'effetto desiderato è quello di una sorta di "telepresenza", nel senso che, per l'utente, la distinzione fra oggetti fisici reali e oggetti simulati sullo schermo risulti confusa, per il fatto che egli può toccare, spingere e muovere qua e là con i movimenti delle dita, sullo schermo, gli oggetti simulati". (MINS84)

Tre esempi
Anche restando nell'ambito tradizionale dei sistemi che utilizzano il mouse, le modalità di interazione con mondi artificiali sono, con il paradigma della manipolazione diretta, significativamente piu' articolate di quelle possibili con i linguaggi di comandi o i menu, e vari approcci possono essere adottati.
Qui di seguito, descriveremo brevemente tre esempi di crescente complessità, nei quali l'utente può "viaggiare" in un mondo artificiale (visualizzato mediante *immagini statiche),* ed eventualmente compiervi delle azioni. Tutti questi esempi sono tratti da giochi di successo pubblicati di recente per il Macintosh.
L'esempio più semplice è costituito dal gioco educativo *Where in*

the world is Carmen Sandiego?, della Broderbund (BROD85). In esso, il giocatore-poliziotto deve viaggiare in diversi Paesi del mondo, alla ricerca di Carmen Sandiego e dei suoi complici, per arrestarla. Per recarsi, in volo, in una certa città, il giocatore farà, semplicemente, "click" con il mouse sul posto desiderato, su una carta geografica dei cinque continenti. Otterrà cosi', sul video, una semplice immagine (quasi una *cartolina illustrata)* della città di destinazione. Qui, l'utente potrà effettuare alcune semplici azioni, premendo un bottone in un semplice "menù di bottoni" visualizzato sul video (Fig.8).

Il secondo esempio è costituito da *The Manhole,* una storia interattiva per bambini sviluppata con Hypercard e pubblicata dalla Activision. Qui, la realtà artificiale è visualizzata per mezzo di un gran numero di immagini statiche (cioè schede Hypercard), sulle quali sono disposti numerosi *bottoni invisibili.*

Premendo tali bottoni col mouse, la scena visualizzata sullo schermo cambia, in modo spesso sorprendente (Fig.9, ACT188). L'utente può così esplorare una mondo di fantasia, popolato da personaggi bonari (parlanti). In questo esempio, che ha aperto la strada a tutta una serie di *iperstorie,* la realtà artificiale e' pertanto realizzata mediante uno stack di *immagini sensibili,* collegate le une alle altre in un complesso *ipergrafo.*

Ciò permette di realizzare mondi con una topologia complessa, oppure dotati di caratteristiche "ricorsive". Ad esempio, in *The Manhole, si* può salire su una barca, percorrere con essa un fiume, e sfociare in una tazzina da caffe' tenuta in mano da un personaggio accanto al fiume. Ancora, si può entrare, dopo aver scelto il programma opportuno, *dentro lo schermo di un televisore,* per ritrovarsi, infine, nella stanza in cui si trova il televisore stesso.

Le stesse idee di base e la stessa tecnologia, con l'aggiunta dell'*animazione* (anche se molto semplice) sono utilizzate in *Cosmic Osmo,* un'altra recente storia interattiva ambientata nello spazio, realizzata dagli stessi autori. Qui, alcune schede mostrano oggetti (o personaggi) che si animano, per esempio, quando ad essi "ci si rivolge" con un click col mouse (ACT189).

Un terzo, più sofisticato esempio in cui ci si muove e si agisce in un mondo artificiale è costituito dal gioco di avventure di ambiente poliziesco *Deja Vu* (MIND85), della Mindscape, o dai suoi successori *Uninvited* (MIND86) e *Shadowgate* (MIND87), realizzati sulla stessa falsariga.

In questi giochi (Fig.10), la parte alta dello schermo presenta un menu di bottoni corrispondenti alle azioni elementari possibili, che possono essere premuti col mouse. In una finestra sottostante, invece, viene mostrata una immagine statica rappresentante l'ambiente in cui ci si trova. Viene adottato il paradigma *azione/oggetto:* per aprire

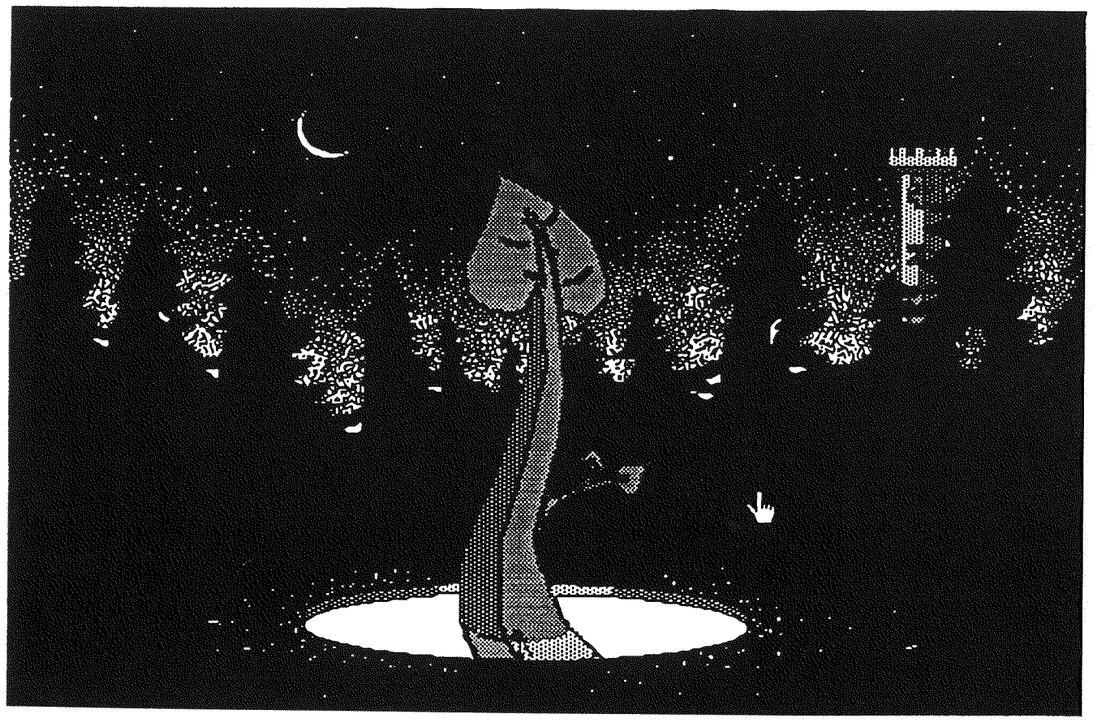


Fig. 9 - The Manhole.

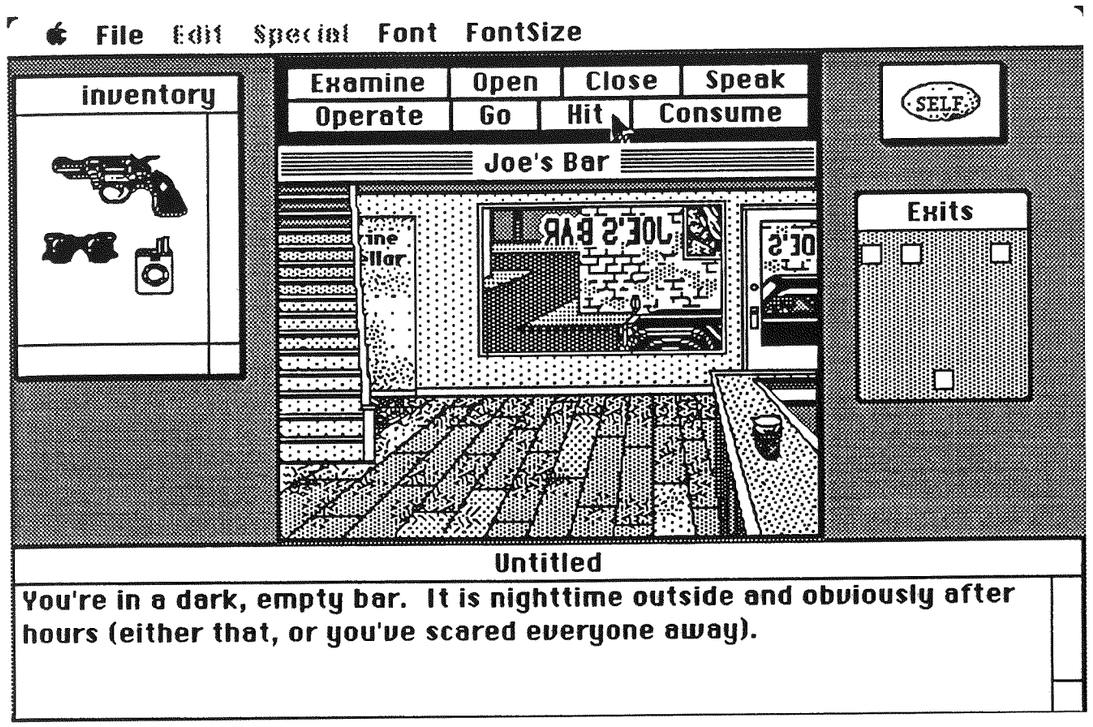


Fig. 10 - Deja Vu.

una porta, ad esempio, bisogna prima far click sul bottone denominato "OPEN", e quindi sulla porta in figura (oppure, più semplicemente, far click due volte sulla porta, senza premere alcun bottone).

Le immagini del mondo sono statiche, ma la maggior parte degli oggetti in esse rappresentati sono *mobili,* e possono essere *trascinati* col mouse per lo schermo, ad esempio per collocarli nella "finestra inventario" (dove vengono mostrati tutti gli oggetti portati dal giocatore).

5. Navigazione

Secondo questo paradigma (al quale ci si riferisce abitualmente quando si parla di realtà artificiale), il video visualizza una realtà, solitamente tridimensionale, nella quale ci si muove con continuità, usando un opportuno sistema di guida *(viaggio virtuale).*

Sistemi di guida

Possiamo usare vari device per controllare il nostro viaggio virtuale in un mondo artificiale: ad esempio un *joystick*, un *bat* (una sorta di mouse che può essere spostato nello spazio, e non soltanto su un piano), o anche un tradizionale mouse.

Varie metafore possono suggerire il paradigma di guida:

- nella *metafora dell'elicottero*, l'"occhio dell'utente" può muoversi, ruotare o fermarsi liberamente in uno spazio tridimensionale, come se l'utente fosse ai comandi di un elicottero all'interno della scena.
- Nella *metafora dell'aereo*, invece, l'utente "vola" come se stesse conducendo un aeroplano.

Normalmente, egli guarderà avanti, ma potrà, se lo desidera, "girare la testa" per guardare a sinistra o a destra (od anche indietro). Un esempio tipico, e ben noto, e' il Flight Simulator della Microsoft (MICR86, Fig.11).

- Nella *metafora dell'automobile*, l'utente è vincolato al piano xy, come se stesse guidando un'auto, e può, se lo desidera, guardare a sinistra e a destra (e, a volte, indietro).
- Nella *metafora del carrello*, l'utente è ancora vincolato al piano xy, ma può solo guardare avanti, e non di lato (si pensi ai vincoli di movimento dei carrelli dei supermercati).

Un caso interessante è il gioco di avventure The *Colony,* pubblicato dalla Mindscape (MIND88, Fig.12), che ha un sistema di guida particolarmente semplice.

In esso, la metafora utilizzata è quella del *casco spaziale.* Il giocatore vede il mondo tridimensionale attraverso un casco immaginario, sulla cui visiera una sorta di *mirino di riferimento* indica il centro della scena (Fig.12). Il giocatore si sposta muovendo il puntatore del mouse nel piano cartesiano di cui tale mirino costituisce l'origine: l'asse y costituisce il controllo della velocità nel movimento in avanti

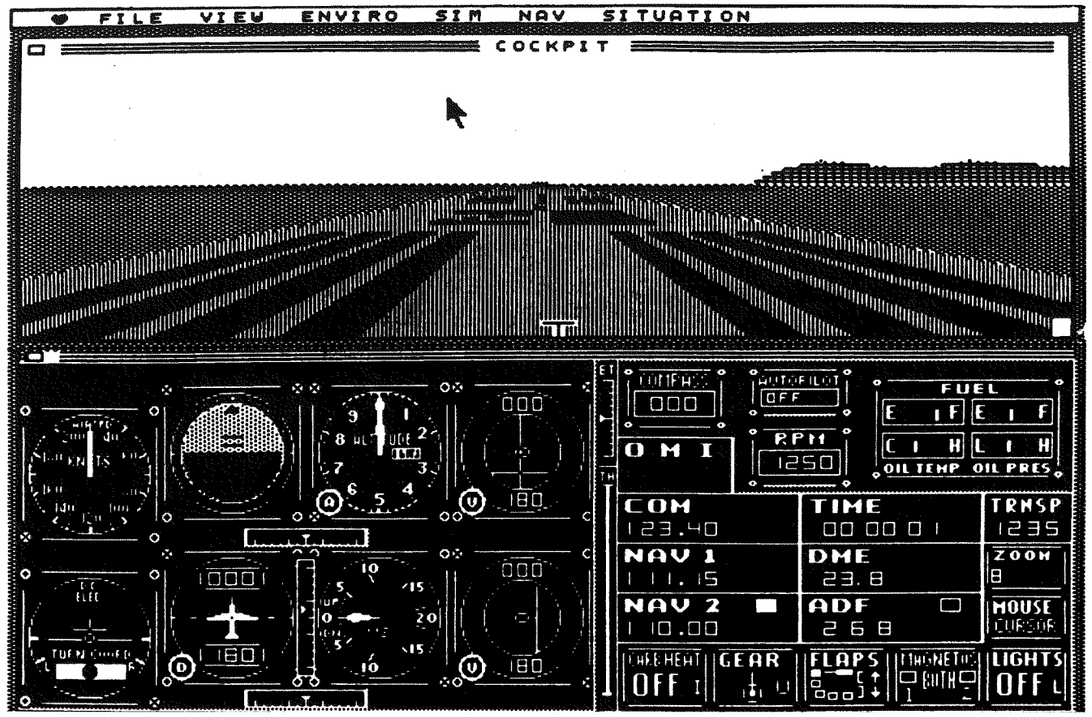


Fig. 11 - Flight Simulator.

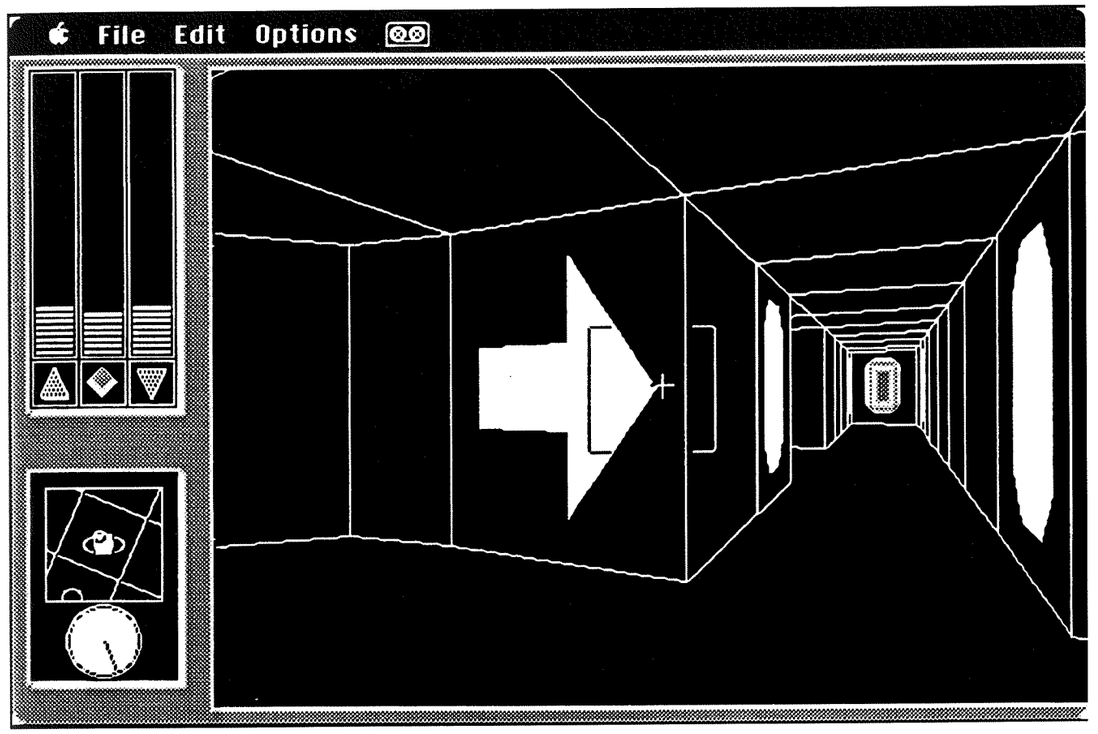


Fig. 12 - The Colony.

e all'indietro, mentre l'asse x permette di controllare la velocità angolare (nella rotazione verso destra e verso sinistra).
In sostanza: si muova il puntatore al di sopra del mirino per avanzare, lo si muova sotto per arretrare; lo si muova a sinistra per girare a sinistra, e a destra per girare a destra. Se il puntatore è vicino al mirino, ci si muove lentamente; se è lontano, ci si muove in fretta.
In molti dei sistemi citati, l'orientamento dell'utente è facilitato dalla presenza di una *mappa bidimensionale* del mondo artificiale, o strumenti analoghi (in Fig.12, ad esempio, in basso a sinistra si vedono una *bussola* e un piccolo riquadro in cui è visualizzata una *vista dall'alto* della posizione del giocatore).

Sistemi a "snapshots"
In alcuni esempi (meno recenti), il viaggio nel mondo artificiale è visualizzato mostrando sul video in rapida sequenza (come in un cartone animato) una serie di immagini statiche e "contigue" *(snapshots))* del mondo virtuale, come ad esempio nel classico prototipo *Movie Map* sviluppato alcuni anni fa all'Architecture Machine Group del MIT (LIPP80).

Per realizzare questo sistema, inizialmente venne effettuata una serie di fotografie della città di Aspen, Colorado, facendo percorrere, a macchine da ripresa montate su carrelli, tutte le strade della città (una immagine ogni 9 piedi di percorso). Questa serie di fotografie venne quindi trasferita su videodisco.

Con il supporto di un software appositamente sviluppato per controllare la presentazione delle immagini dal videodisco, l'utente poteva, così, "viaggiare" per le strade di Aspen, al livello stradale. Il sistema di guida era realizzato per mezzo di un touch sensitive screen montato su un monitor televisivo a colori: raggiunto un incrocio, si poteva svoltare a sinistra o a destra toccando, rispettivamente, una freccia verso sinistra o verso destra sullo schermo (Fig.13).

Una piccola mappa inserita nella parte superiore dello schermo visualizzava l'itinerario percorso, rispetto alle due strade principali di Aspen. Inoltre, un secondo monitor montato orizzontalmente forniva una vista dall'alto degli incroci stradali di Aspen, come in una mappa convenzionale, e mostrava in rosso l'itinerario percorso dall'utente (BOLT84).

6. Presenza virtuale.

L'ultimo paradigma che vogliamo considerare in questo articolo è ancora oggetto di ricerca avanzata: lo chiameremo *presenza virtuale.*
L'esempio più noto è costituito da un sistema sperimentale sviluppato alla Aerospace Human Factors Research Division dell'Ames Research Center della NASA a Mountain View, California (FISH86, cfr. anche TELL88 e FOLE87).

Questo sistema, denominato *Virtual Interactive Environment Workstation* (VIEWS), utilizza diverse tecnologie avanzate in modo integrato: un display *head-tracked* e *head-mounted*, un *data-glove* per riconoscimento di gesti, input e output vocale.
In sostanza, l'utente indossa uno speciale casco che regge due piccoli display, uno per ciascun occhio, in modo da mostrare una immagine tridimensionale. Il casco impedisce a chi lo indossa di vedere altro, immergendo l'utente, per cosi' dire, nella scena simulata visualizzata dai due display.
Un sensore montato sul casco tiene traccia della posizione e dell'orientamento della testa dell'utente.
Come questi ruota la testa, la scena computerizzata, visualizzata sui due display, ruota di conseguenza.
Il sistema permette di esplorare un campo visivo di 360 gradi (che può presentare sia un ambiente artificiale, simulato dal computer, che un ambiente reale remoto, ripreso mediante telecamere), e di avere una interazione *diretta* con tutti i suoi elementi .
Per interagire con gli oggetti del mondo simulato, l'utente indossa un data-glove: con questo, *egli può afferrare e muovere gli oggetti virtuali con le proprie mani:* un sensore, simile a quello posto sul casco, permette di determinare la posizione della mano nello spazio.
All'utente viene fornito un feedback appropriato mostrando una mano virtuale nella scena simulata.
VIEWS incorpora anche un sistema (commerciale) di riproduzione sonora: mediante degli auricolari, l'utente può udire suoni, provenienti sia dall'interno che dall'esterno del suo campo visivo.
Inoltre, un sistema di riconoscimento del parlato permette all'utente di interagire con l'ambiente mediante comandi vocali.
Con un sistema di questo tipo, un essere umano può, ad esempio, controllare il braccio di un robot collocato in un ambiente pericoloso: ad ogni movimento della testa dell'uomo, apposite telecamere montate sul robot inviano ai display montati sul casco la vista del robot sulla scena *(tele-robotica).*
La NASA prevede di utilizzare questa tecnologia per permettere a un astronauta all'interno di una stazione spaziale di controllare un robot che effettui delle operazioni di manutenzione all'esterno *(telepresenza,* Fig .14) .

7. Cinestetica, realtà artificiale, realismo.

Qualunque sia il paradigma adottato, il progettista dell'interfaccia uomo-realtà artificiale dovrà ben comprendere e applicare i *principi di cinestetica* relativi a tale paradigma.
Il problema principale è quello di evitare i fenomeni di *disorientamento* che possono verificarsi quando ci troviamo in ambienti non familiari, o differenti, o troppo complessi o, semplicemente, per qualche ragione, *sconcertanti* per l'essere umano.

La comprensione di questi principi è ancora molto limitata (ad esempio, BR0088 elenca alcune "regole di buon senso" relative alle interfacce per ambienti 3D). Essi dovranno essere sistematizzati in una nuova disciplina, che potremmo chiamare *cinestetica artificiale ("la percezione di sè nella realtà artificiale").*

Questa nuova disciplina dovrà esplorare anche le possibilità e i problemi aperti dalle *interfacce multisensoriali* che possono essere realizzate con i *sistemi multimediali* (AMBR88, HUG89) che stanno inziando a diffondersi, o da insoliti *responsive environments* del tipo, ad esempio, di quelli descritti in KRUE83.

Il problema della cinestetica e dell'orientamento in ambienti "strani" è tanto più importante, in quanto i mondi artificiali che ci presenteranno le workstation future non necessariamente dovranno mimare la realtà naturale secondo i principi del *realismo e le leggi della prospettiva.*

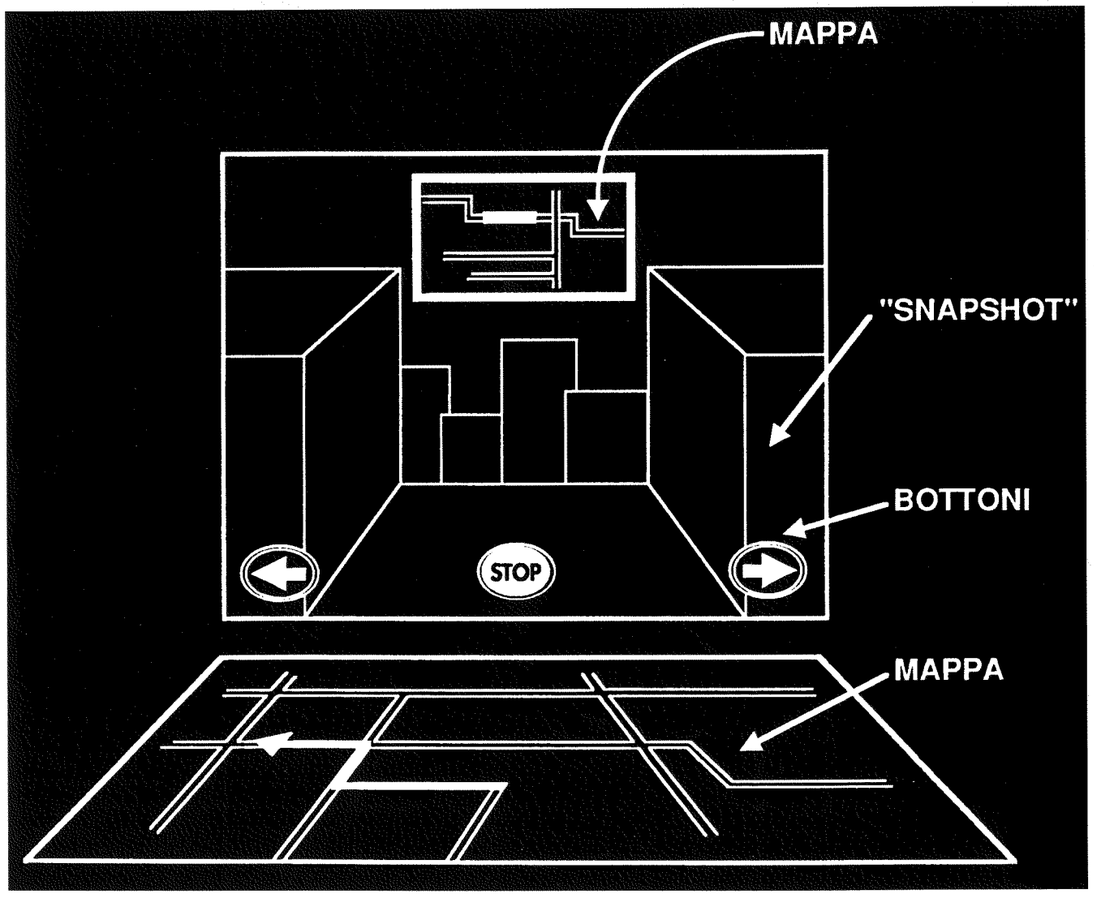
Nulla, infatti, ci impedisce di pensare di implementare mondi "impossibili", quali ad esempio quelli visualizzati nelle opere di Escher (Fig.15).

Citando ancora L. Sutherland (che costrui' il primo *head mounted display,* più di vent'anni fa):

"Un display connesso a un computer ci dà l'opportunità di prendere familiarità con concetti non realizzati nel mondo fisico. È uno specchio in un paese delle meraviglie matematico."

Questa possibilità di creare, rappresentare, esplorare e modificare interattivamente, al calcolatore, *"worlds that are not and never can* be" (BR0088) suggerisce alla fantasia nuovi e stimolanti scenari: dal *cinema interattivo,* a nuove forme di didattica *computer-based,* alla *computer art,* (cfr., ad esempio, KRUE83), all'uso del calcolatore per l'analisi interattiva di fenomeni scientifici.

Fig. 13 - La "Movie Map" di Aspen, Colorado.

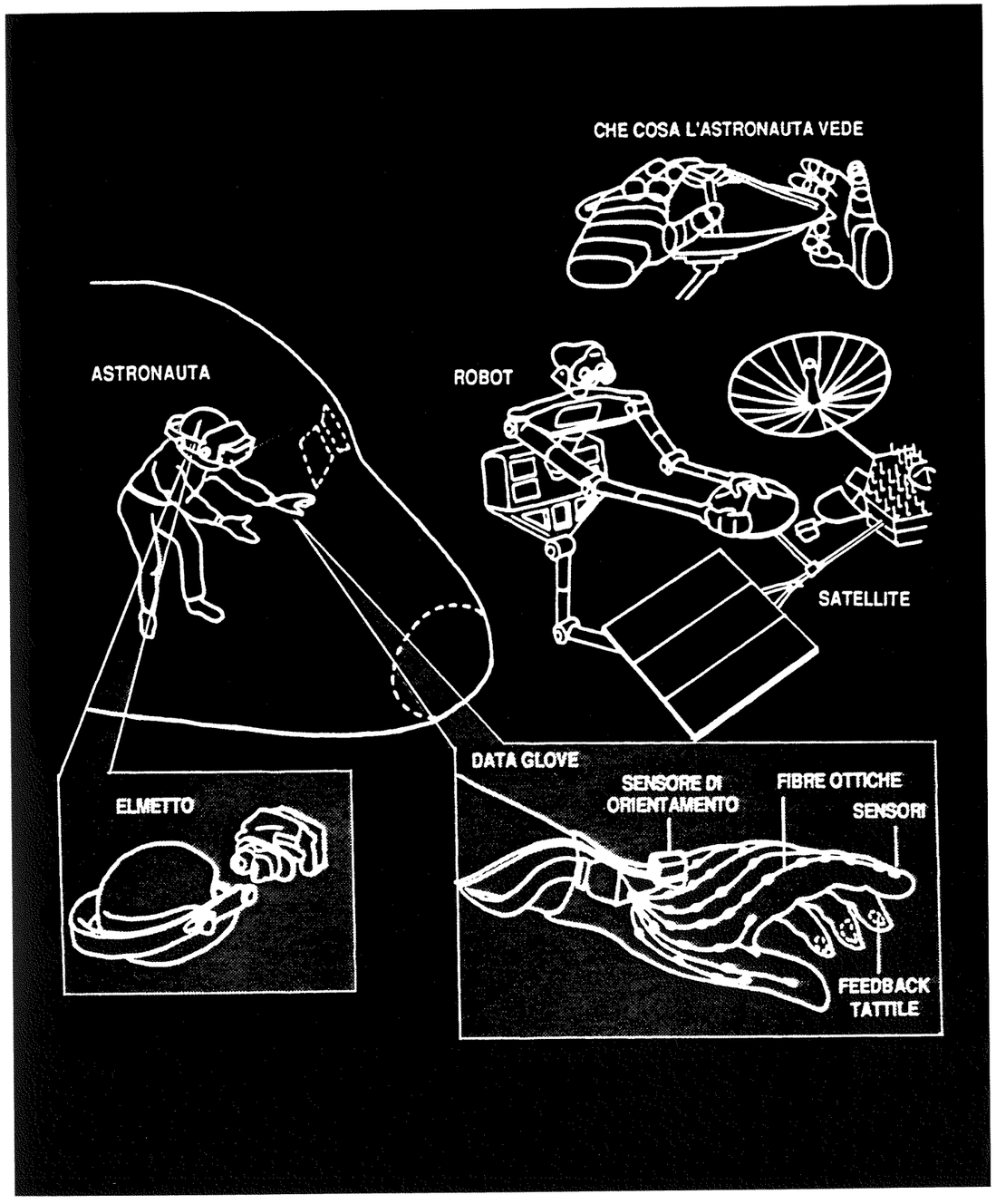


La disciplina emergente della *vita artificiale* (LANG89a, LANG89b) ci aiuterà, in futuro, a popolare questi mondi con creature viventi simulate.

Ciò apre nuovi e vasti orizzonti alle applicazioni dei computer, e non solo per la realizzazione di nuovi e più interattivi apparati per *il controllo dei sistemi* (controllo di processo, automazione industriale, building automation, ...) ma anche per la progettazione di *sistemi informativi di* nuova concezione.

"Se la realtà artificiale si può osservare allora si può entrare nella realtà artificiale. Se si può entrare nella realtà artificiale allora si può disporre di tutto il reale reso reale/artificiale vicino a noi, ovunque ci troviamo" (DEGL88).

Fig. 14 - Telepresenza.



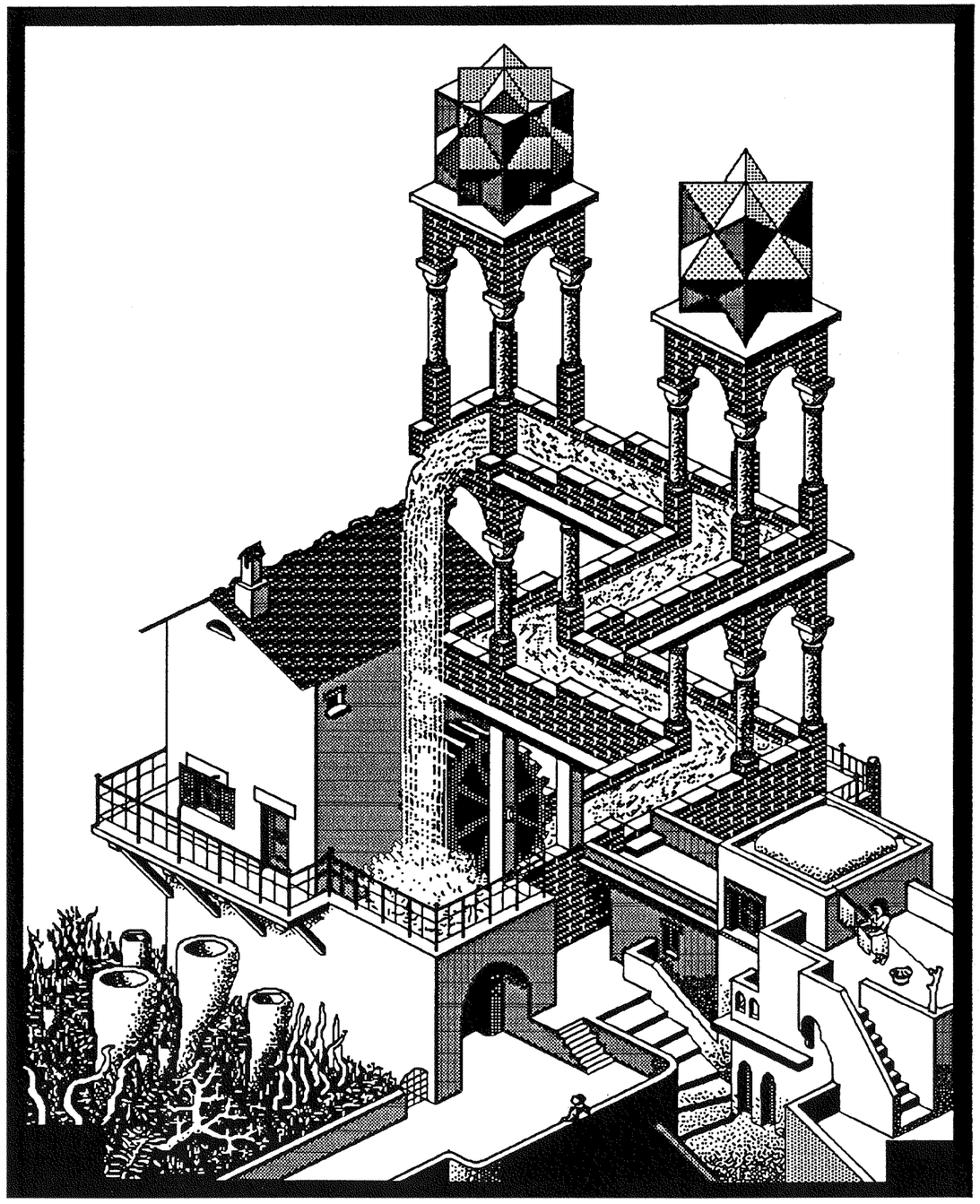


Fig. 15 - M. C. Escher, "Moto perpetuo", 1961.

8. Ringraziamenti.

L'Autore ringrazia Gianni Degli Antoni, che negli ultimi anni ha promosso i concetti della realtà artificiale in un gran numero di convegni, seminari, attività di ricerca e prototipazione, e al quale sono dovute molte delle idee presentate in questo articolo.
Ringrazia inoltre Roberta Macchi, per la sua preziosa attività di supporto nell'ambito dello HUG (Hypertext User Group) del Dipartimento di Scienze dell'Informazione dell'Università di Milano.
Il presente articolo e' una rielaborazione estesa di un intervento presentato al Workshop CNR "Conoscenza per Immagini", Roma, 27-28 novembre 1989, ed è stato pubblicato, in lingua inglese, sul numero 48/49 di Note di Software (giugno-ottobre 1990).

9. Riferimenti bibliografici

(ABB082) Abbott, E.A., *"Flatland - A Romance of Many Dimensions"*, Prima Edizione: 1882.

(AMBR88) Ambron, S., Hooper, K. (editors), "*Interactive Multimedia*", Microsoft Press, pp.xi339, 1988.

(ACT188) Activision, Inc., *"The Manhole"*, software per Macintosh, di Robin e Rand Miller, 1988.

(ACT189) Activision, Inc., "*Cosmic Osmo*", software per Macintosh, di Robin e Rand Miller, 1989.

(APPL87a) Apple Computer Co., *"Human Interface Guidelines: the Apple Desktop Interface"*, Addison-Wesley Publishing Co., pp.xii 144, 1987.

(APPL87b) Apple Computer, "*Hypercard*", software per Macintosh, di B.Atkinson, 1987.

(BOLT84) Bolt., R.A., *"The Human Interface - Where People and Computers Meet"*, Lifetime Learning Publications, Belmont, CA, pp.xvii113, 1984.

(BROD85) Broderbund Software, *"Where in the world is Carmen Sandiego*", software per Macintosh, Apple IIGS, Amiga, di G. Portwood and L. Elliott, 1985-1989.

(BROD89) Broderbund Software, "*SimCity*", software per Macintosh di Maxis Software, 1989.

(BR0088) Brooks, F.P.jr., *"Grasping Reality Through Illusion - Interactive Graphics Serving Science"*, in CHI '88 Conference Proceedings, Special Issue of the SIGCHI Bulletin, ACM Press, pp. 1-10.

(DEGL88) Degli Antoni, G., *"Il computer, il reale, l'artificiale"*, in Note di Software n. 41, ottobre 1988.

(DEGL89) Degli Antoni, G., *"From Reality to Hypermedia through Artificial Reality"*, in HUG89, pp.10-15.

(DEGL89b) Degli Antoni, G., "*Artificial Worlds*", in HUG89, pp.8-9.

(ENGL67) English, W.K., Engelbart, D.C.,Berman,M.A., "*Display selection techniques for text manipulation*", IEEE Transactions on Human Factors in Electronics, HFE-8, 1967, pp.5-15.

(FAIR89) Fairchild, K., Meredith, G., Wexelblet, A., *"The Tourist Artificial Reality"*, in CHI '89 Conference Proceedings, Special Issue of the SIGCHI Bulletin, ACM Press, pp.299-304.

(FISH86) Fisher, S.S., *"Telepresence master glove controller for dexterous robotic end-effectors"*, in Proceedings SPIE - vol. 726, "Intelligent Robots and Computer Visionn, 1986, pp.396-399.

(FOLE87) Foley, J.D., "*Interfaces for Advanced Computing*", in Scientific American, ottobre 1987, pp. 127-135.

(GARD89) Gardin, F., *"Lenti, Leve e Specchi sul Reale e l'Artificiale"*, in Proceedings of Conference "Verso la comunicazione elettronica", Grande Fiera d'Aprile, Milano, 20-21 aprile 1989, pp.21-24.

(HAMM88) Hammond, N., Allinson, L., *"Travels around a Learning Support Environment: Rambling, Orienteering or Touring?"*, in CHI '88 Conference Proceedings, Special Issue of the SIGCHI Bulletin ACM Press, pp. 269-273.

(HINT88) Hinton, C.H., "A *plane world"*, in "Scientific Romances", Prima Edizione: 1888.

(HUG89) *"Sistemi Informativi e Multimedialità"*, Università di Milano, Hypermedia User Group, Atti di un seminario organizzato da SOI Informatica, Milano, 9 novembre 1989.

(KRUE83) Krueger, M.W., *"Artificial Reality"*, Addison-Wesley Publishing Co., pp.xviii+312, 1983.

(IBM89a) IBM, *"System Application Architecture, Common User Access: Basic Interface Design Guide"*, doc.no.SC26-4583-0, pp.xx 268, First Edition, dicembre 1989.

(IBM89b) IBM, *"System Application Architecture, Common User Access: Advanced Interface Design Guide"*, doc.no.SC26-4582-0, pp.xvi 195, First Edition, giugno 1989.

(INF088a) Infocom, Inc., *"The Hitchhiker's Guide to the Ga/axy"*, software per Apple 11, PC, Macintosh e Commodore 64/128, di D.Adams e S.Meretzky, 1988.

(INF088b) Infocom, Inc., *"Quarterstaff - The Tomb of Setmoth"*, software per Macintosh di S. Schmitz e K.Updike, 1988.

(LANG89a)Langton, C.G. (ed.), *"Artificial Life - Proceedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems"* - Los Alamos, New Mexico, September 1987, Addison-Wesley Publishing Co., pp.xxix 655, 1989.

(LANG89b) Langton, C.G., *"Artificial Life"*, in LANG89a, pp.1-47.

(LIPP80) Lippman, A., *"Movie-Maps: An Application of the Optical Videodisc to Computer Graphics"*, Computer Graphics 14 (3), pp.32-42, 1 980.

(MCGA84) McGath, *G., "Compute! Guide to Adventure Games"*, Compute! Publications Inc., Greensboro, North Carolina, pp.viii 203, 1984.

(MICR86) Microsoft Corp., "*Flight Simulator*", software per PC, Macintosh e altri computer, di B.A.Artwick, 1986.

(MIND88) Mindscape, Inc., "*The Colony*", software per Macintosh di A.D.Smith, 1 988.

(MIND85) Mindscape, Inc., "*Deja Vu, A Nightmare Comes True"*, software per Apple 11, Amiga, Atari ST, Commodere 64/128, Macintosh e PC di ICOM Simulations, 1985.

(MIND86) Mindscape, Inc., *Uninvited*", software per Apple 11, Amiga, Atari ST, Commodere 64/128, Macintosh e PC di ICOM Simulations, 1986.

(MIND87) Mindscape, Inc., *"Shadowgate"*, software per Apple 11, Amiga, Atari ST, Commodere 64/128, Macintosh e PC di ICOM Simulations, 1 987.

(MINS84) Minsky, M.R., *"Manipulating Simulated Objects with Real-World Gestures using a Force and Position Sensitive Screen"*, in ACM Computer Graphics, vol.18, n.3, luglio 1984, pp.195-203.

(NEXT89) NeXT Inc., *"The NeXT User's Reference Manual"*, pp. 450, 1989.

(OSF89) Open Software Foundation, *"OSF/Motif Style Guide", 1989.*

(PIZZ89) Pizzi, R., *"Through the looking glass: una metafora della realtà artificiale"*, Atti del convegno "Verso la comunicazione elettronica", Grande Fiera d'Aprile, Milano, 20-21 aprile1989, pp.7-16.

(POL189) Polillo, R., *"Paradigmi di interazione con realtà artificiali*", Atti della giornata CNR "Conoscenza per Immagini", Roma, 27-28 novembre 1989.

(SHNE83) Shneiderman, B., *"Direct Manipulation: A Step Beyond Programming Languages"*, in IEEE Computer, agosto 1983, pp. 57-69.

(SMIT82) Smith,C., Irby, C., Kimball, R., Verplank, B., Harslem, E., *"Designing the Star User Interface"*, in BYTE, Vol.7, n.4, aprile 1982, pp. 242-282.

(SPEC87) Spectrum HoloByte, *"Falcon"*, software per Macintosh e altri computer, di G.Louie e M.Johnson, 1987.

(SUN89) Sun Microsystems, *"OPEN LOOK Graphical User Interface Specification"*, 1989.

(SUTH65) Sutherland, 1., *"The Ultimate Display"*, Proceedings of the IFIP Congress, vol.2, 1965, pp.506-509.

(TELL88) Tello, E.R., *"Between Man and Machine"*, BYTE, settembre 1988, pp. 288-293.

(WEIM89) Weimer, D., Ganapathy, S.K., "A *Synthetic Visual Environment with Hand Gesturing and Voice Input"*, in CHI '89 Conference Proceedings, Special Issue of the SIGCHI Bulletin, ACM Press, pp.235-240.

(WEIZ66) Weizenbaum, J., "*ELIZA*", Communications of the ACM, 9:1, gennaio 1966, pp.36-45.

(WIN086) Winograd, T., Flores, F., *"Understanding Computers and Cognition"*, Ablex Publishing Co., pp. xiv 207, 1986.

Il mondo del CASE: stato dell'arte e prospettive(*)

ALFONSO FUGGETTA

CEFRIEL
Centro per la Ricerca e la Formazione in Ingegneria dell'Informazione
Milano

1. Introduzione

Nel corso degli ultimi anni è stato sviluppato, sia in ambito accademico che in ambito industriale, un notevole numero di nuovi prodotti che mirano a rendere sempre più agevole il compito di chi deve sviluppare e manutenere un'applicazione informatica.
Questa grande espansione dell'offerta è dovuta ad una molteplice serie di cause sia tecnologiche che culturali ed organizzative, che possono essere sintetizzate, seppur in modo schematico, dalle seguenti considerazioni:

1. La diminuzione dei costi dell'hardware ha fatto crescere l'attenzione verso quegli aspetti, legati alla produzione e gestione del software, che si sono rivelati come maggiormente critici per ciò che riguarda l'aumento dei costi complessivi di sviluppo e manutenzione del prodotto finale.
2. Lo sviluppo dei personal computer e delle workstation grafiche avanzate ha enormemente aumentato la massa potenziale di utenti, in quanto il costo relativamente basso di queste nuove apparecchiature ha reso possibile la loro diffusione anche in settori che precedentemente non avevano beneficiato della introduzione delle tecnologie informatiche.
3. L'avvento delle interfacce grafiche, di dispositivi di interazione uomo macchina avanzati (mouse, penne ottiche, tavolette, dials, ...) e delle reti locali di computer ha reso possibile la creazione di nuove tipologie di strumenti e di architetture HW/SW, che hanno in alcuni casi radicalmente cambiato il modo attraverso il quale i problemi applicativi vengono affrontati e risolti (si consideri ad esempio il caso di HyperCard, il sistema per la gestione di ipertesti sviluppato dalla Apple).

Per identificare in modo agevole questa classe di prodotti è stata creata l'espressione *Computer Aided Software Engineering* - o *CASE* - che si è rapidamente diffusa all'interno sia del mondo accademico che di quello industriale (si veda [Chi88]). Gli strumenti di questa classe sono chiamati *CASE tools*, mentre per indicare un ambiente di sviluppo spesso si utilizza anche la sigla *SEE* (*Software Engineering Environment*). Si noti che, almeno inizialmente, con l'espressione *CASE Tools* venivano indicati solo gli strumenti per la specifica e il progetto delle applicazioni: ultimamente si tende ad utilizzare tale espressione in modo più generale per identificare tutti i prodotti che costituiscono di per se stessi o concorrono alla creazione di un moderno ambiente di supporto alle attività di ingegneria del software (SEE). (Per i prodotti originariamente chiamati strumenti CASE è stata coniata una nuova espressione - *Upper CASE* - che vuole indicare il fatto che tali strumenti sono utilizzati nelle fasi più alte del ciclo di vita).

Le caratteristiche dei prodotti presenti sul mercato variano in modo sensibile da caso a caso, sia per ciò che riguarda le funzionalità offerte che per altri aspetti quali il tipo di utenza alla quale lo strumento è indirizzato (progettisti piuttosto che manager o programmatori) o la classe di macchine per la quale esso è stato sviluppato (PC piuttosto che mainframe). Tuttavia esistono una serie di aspetti di carattere prettamente tecnologico che sono comuni alla maggior parte degli strumenti oggetto del nostro studio: in particolare, è possibile identificare *tre tecnologie* che giocano un ruolo essenziale per la costruzione di moderni CASE tools.

Interfacce grafiche avanzate.
Come si è già accennato in precedenza, nel corso degli ultimi anni vi è stato un enorme sviluppo delle tecnologie legate alle interfacce uomo-macchina: oggi la maggior parte degli strumenti di supporto allo sviluppo del software è costruita su macchine che dispongono di schermi grafici ad alta risoluzione e mouse. Inoltre, sia nel mondo dei personal computer che in quello delle workstation scientifiche, sono stati definiti e sono ormai universalmente adottati sistemi per la gestione di finestre (per esempio, Microsoft Windows e X-Window) che permettono di far coesistere su uno stesso schermo fisico più schermi virtuali, ciascuno dei quali viene dedicato ad una specifica applicazione. L'interazione tra uomo e macchina è di tipo grafico ed è basata sull'uso di metafore grafiche quali pop-up menu, icone e dialog-box.

Reti locali.
L'avvento dei personal computer e delle workstation grafiche ha mutato il modo secondo il quale un utente utilizza i servizi informatici: se nel corso degli anni sessanta e dei primi anni settanta lo sforzo dei produttori di computer era quello di fornire macchine *multi-user* sempre più potenti a cui potessero essere collegati centinaia di terminali alfanumerici, nel corso degli

(*) Il presente lavoro costituisce una sintesi e revisione di un precedente articolo scritto con Carlo Ghezzi e presentato alla Giornata AICA sul CASE, tenutasi a Milano il 30 novembre 1989.

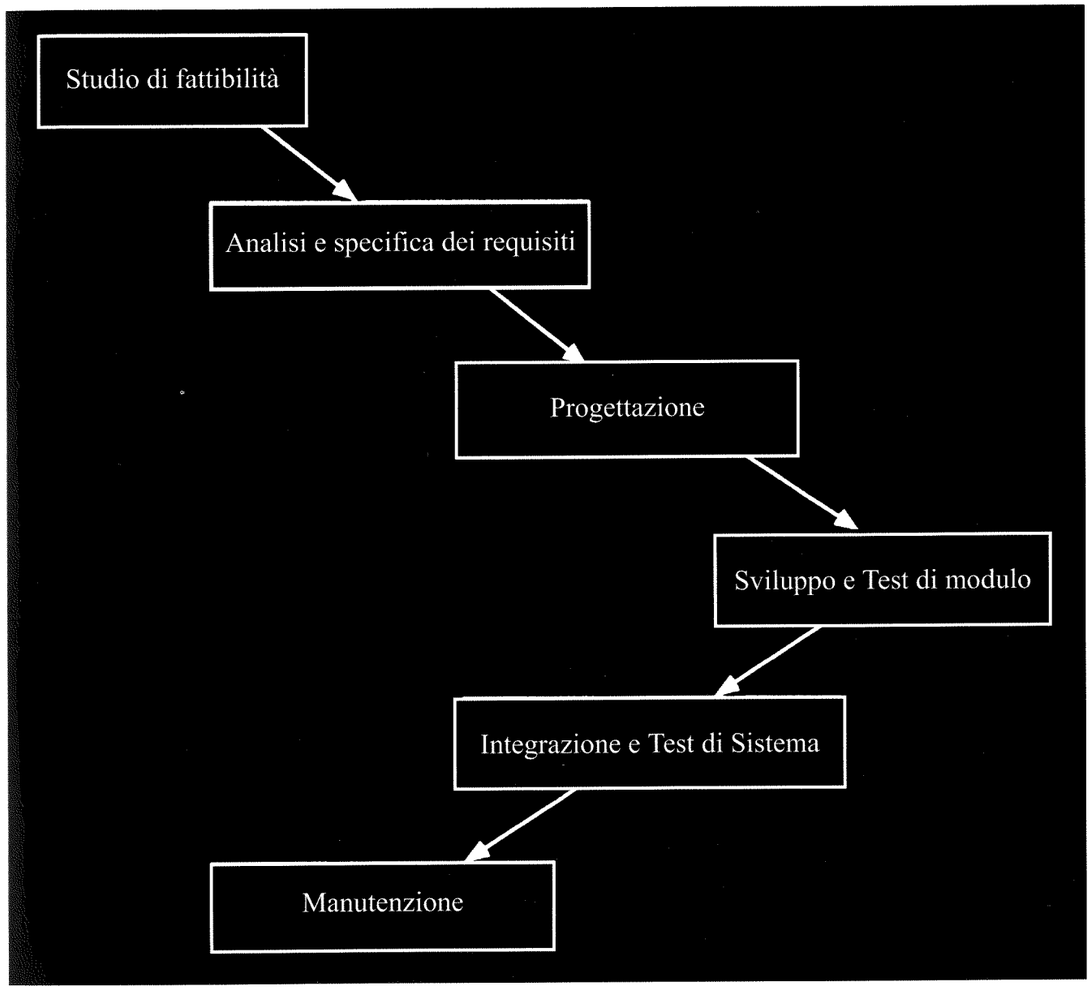


Fig. 1a) - Lo schema mostra il ciclo di vita tradizionale di un prodotto software.
L'introduzione delle tecniche CASE consente l'impiego di nuovi modelli di sviluppo del software, che sono presenti nelle figure seguenti (1b - 1e), a partire dalla analisi della applicazione.

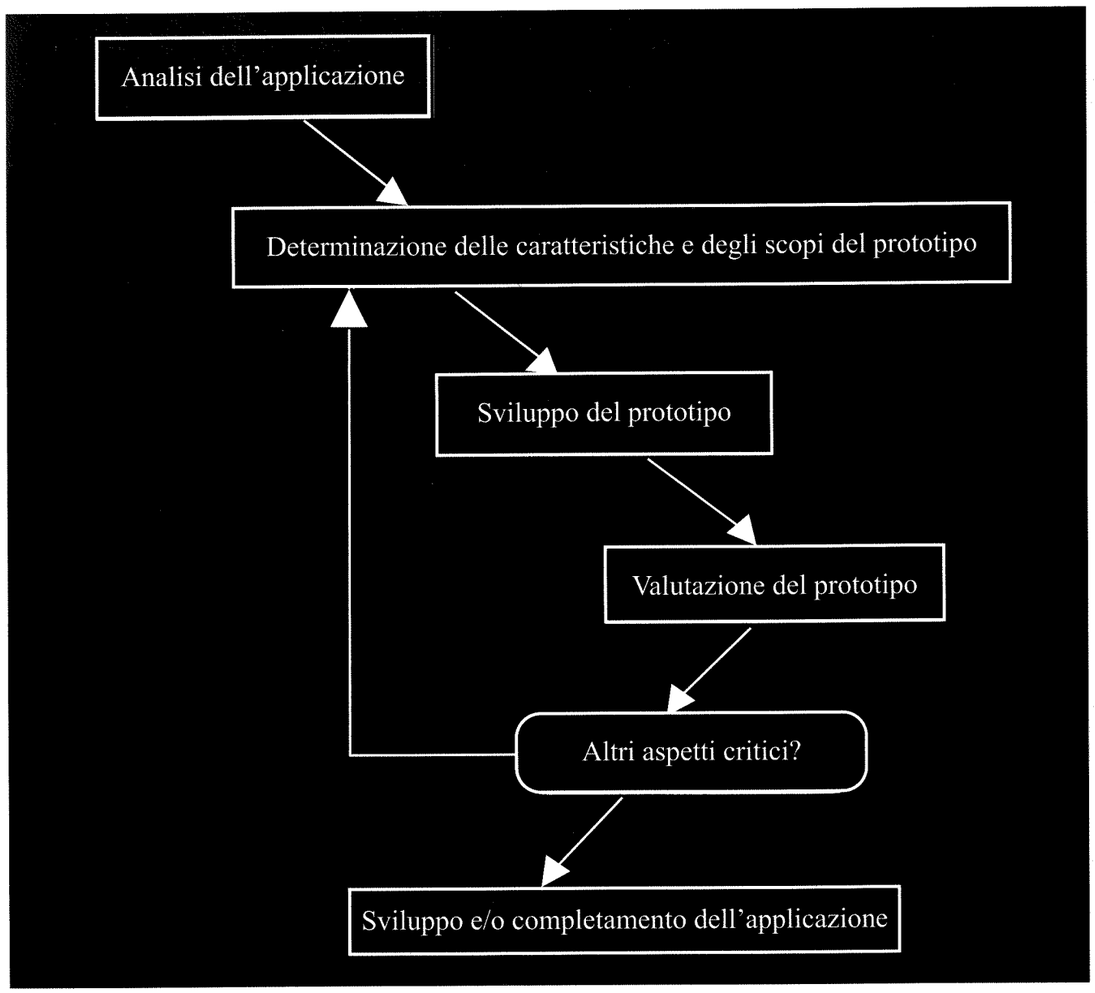


Fig. 1b) - Ciclo di vita basato sulla prototipazione.

anni ottanta vi è stato il grande sviluppo dell'informatica distribuita basata sull'uso delle reti locali di calcolatori (LAN). Tali reti permettono la creazione di nuove strutture informatiche basate sull'interconnessione di potenti macchine *multi-task* e *single- user* (tipicamente PC e workstation), che interagiscono scambiandosi messaggi o accedendo a risorse comuni integrate sulla rete (file server, mail server, printer server, . . .). Questo tipo di strutture si va sempre più affermando come la piattaforma ideale per la costruzione degli ambienti di supporto allo sviluppo di software.

Data Base Management Systems.
La gestione delle informazioni prodotte durante le varie fasi del ciclo di vita passa sempre più attraverso l'utilizzo di DBMS, che presentano il grosso vantaggio di controllare in modo efficace l'accesso concorrente ad uno stesso insieme di informazione da parte di più utenti e di mascherare all'esterno la struttura fisica dei dati. L'uso di un DBMS permette una più facile integrazione dei diversi strumenti presenti nell'ambiente: i risultati prodotti da uno specifico strumento vengono memorizzati in basi di dati specializzate (denominate spesso *Data Dictionary o Data Repository o Encyclopedia)* e possono essere riutilizzati in un secondo momento da altri componenti dell'ambiente di sviluppo.

2. Le tipologie di prodotti CASE

Nei successivi paragrafi viene proposta una classificazione dei prodotti CASE esistenti oggi sul mercato: essa ha lo scopo di fornire un quadro complessivo di riferimento delle diverse tecnologie al momento disponibili (si veda anche

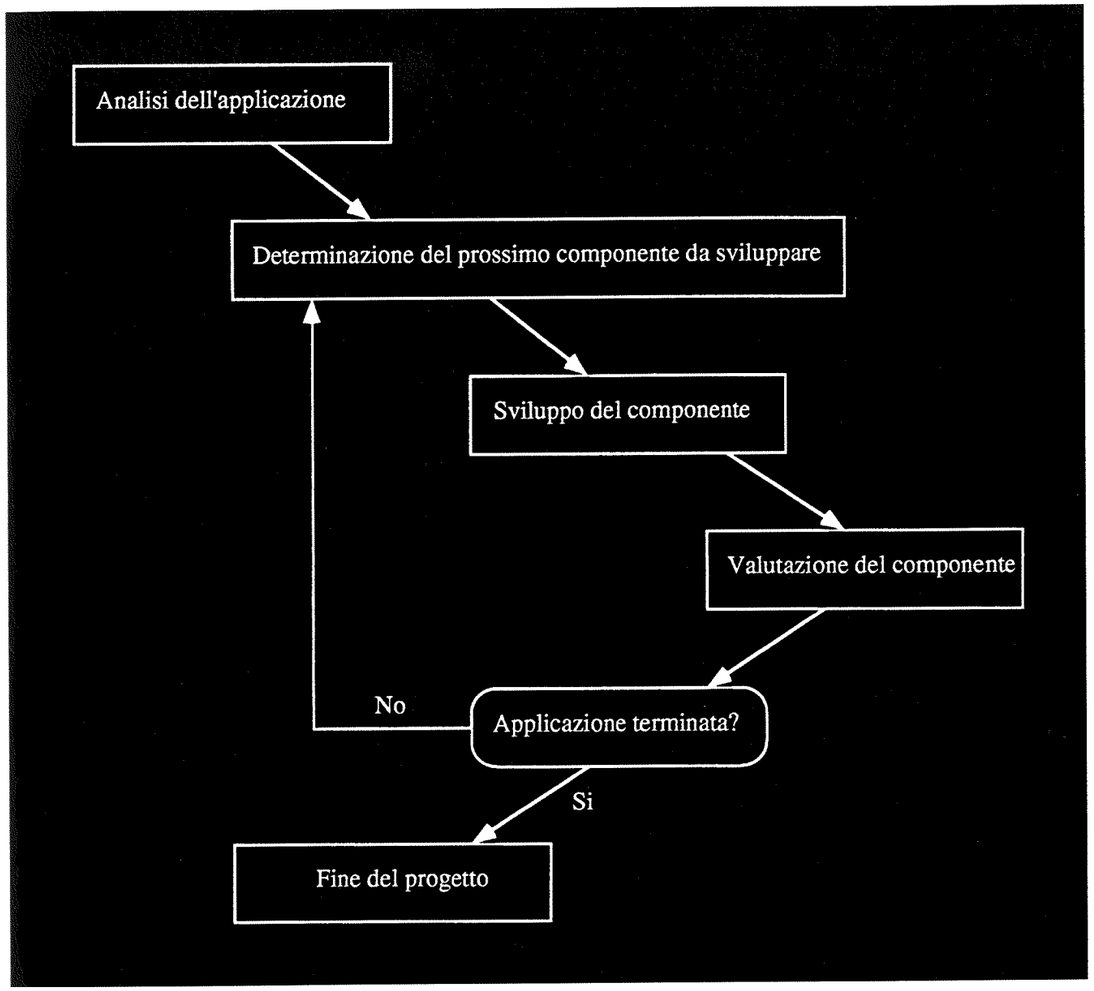


Fig. 1c) - Ciclo di vita evolutivo.

[FGM+90, Cap. 7]). Tale classificazione è articolata su due categorie di prodotti (strumenti ed ambienti) che si differenziano per il diverso grado di copertura delle varie fasi del ciclo di vita del software: in particolare verranno classificati sotto la voce *strumenti* quei prodotti che tendono ad automatizzare attività specifiche all'interno del ciclo di vita (per esempio lo sviluppo o la manutenzione), mentre sotto la voce *ambienti* verranno inseriti quei prodotti o quelle combinazioni di prodotti che offrono un supporto informatico integrato alle varie attività legate all'intero ciclo di produzione e gestione del software. E chiaro che classificazioni di questo tipo possono spesso risultare arbitrarie o parziali e sono funzionali ad uno scopo preciso: per esempio [DEFH87] affronta lo stesso problema di classificazione con l'intento di indicare le evoluzioni tecnologiche in corso nel settore degli ambienti di supporto allo sviluppo del software. Nel nostro caso l'intento è quello di fornire un sintetico quadro di riferimento che permetta di classificare gli strumenti e gli ambienti disponibili in base alle funzionalità offerte e al tipo di utilizzo per il quale sono stati progettati e realizzati.

3. Strumenti

3.1 Strumenti tradizionali per la programmazione in piccolo
Questa classe di strumenti racchiude al suo interno tutti quei prodotti che sono stati sviluppati per fornire un primo supporto all'attività di sviluppo del software. Ciascun prodotto di questa classe tende a risolvere un problema specifico del programmatore: un ambiente costruito utilizzando questo tipo di strumenti fornisce solo le funzionalità essenziali che sono necessarie per poter scrivere e mandare in esecuzione un programma scritto in un tradizionale linguaggio di

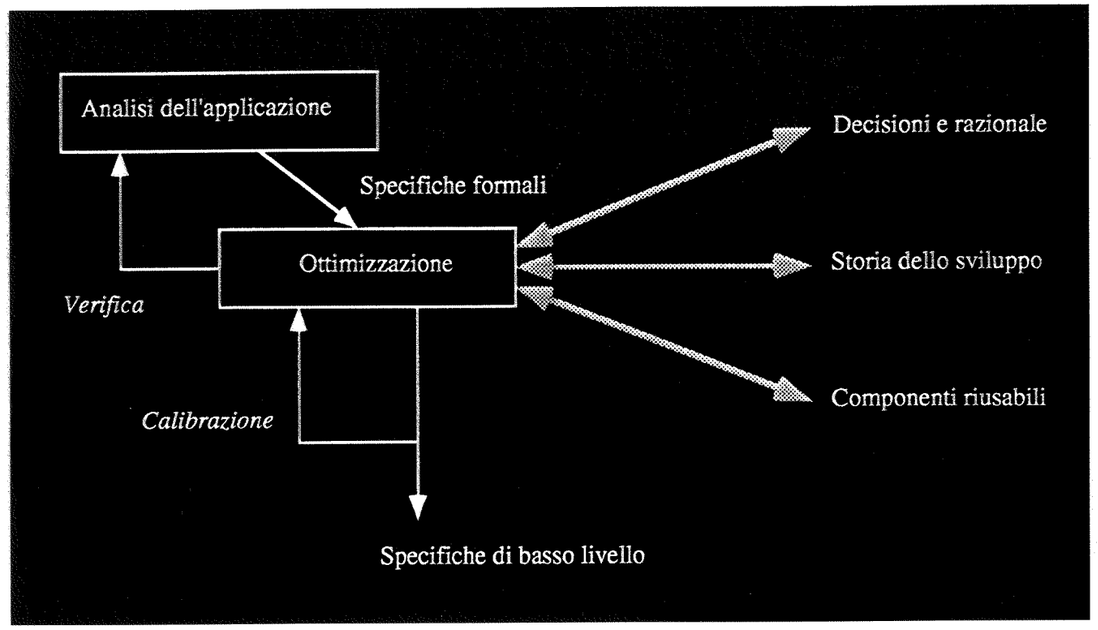


Fig. 1d) - Ciclo di vita trasformazionale.

programmazione. I prodotti che tipicamente fanno parte di questa classe sono text editors, compilatori, interpreti, linkers, debuggers,. . .

3.2 Strumenti integrati di sviluppo e debugging

Una evoluzione diretta degli strumenti visti nel precedente paragrafo sono i prodotti che integrano all'interno di un unico tool un editor, un compilatore e spesso anche un debugger. Tipici esempi di questo tipo di strumenti sono il Quick-C della Microsoft e la serie Turbo della Borland. Gli strumenti di questa classe permettono di passare molto facilmente attraverso le diverse fasi dell'attività di sviluppo: spesso essi forniscono all'utente una serie di opzioni preferenziali che lo indirizzano in maniera rapida attraverso i diversi componenti dell'ambiente.

3.3 Strumenti di supporto al testing e alla validazione

L'attività di testing è particolarmente complessa e critica e necessita di strumenti di supporto che aiutino lo sviluppatore a pianificare le varie prove e a tenere traccia dei risultati dei singoli test: tipicamente, alcuni dei problemi che maggiormente assillano i dipartimenti di Controllo Qualità (CQ) sono la determinazione dei test necessari per soddisfare uno specifico criterio di copertura o l'organizzazione e memorizzazione dei dati di test in opportune banche dati, allo scopo di rendere ripetibile il processo di test ogniqualvolta si voglia ricertificare la qualità del sistema (test di regressione).

Inoltre, la crescente dimensione e complessità dei programmi pone il problema di identificare criteri qualitativi che indichino al progettista quali parti del codice sviluppato presentano maggiori problemi dal punto di vista della affidabilità e della modificabilità del codice. A tal scopo, nel corso degli ultimi anni sono state definite varie metriche di qualità del software (si veda [FGM+90]).

Recentemente sono apparsi sul mercato alcuni prodotti che affrontano tali problematiche. Le funzionalità che, seppur in diversa misura, sono presenti in questi strumenti sono sostanzialmente di due tipi:

Misura della complessità.

Sono stati sviluppati alcuni prodotti che permettono di calcolare varie misure di qualità del software (secondo alcune delle metriche a cui si è fatto cenno in precedenza) e, quindi, di indicare quali sono i punti del programma sviluppato che, in base alle assunzioni fatte nel modello, possono creare i maggiori problemi di leggibilità e manutenibilità.

Gestione dell'attività di test.

Alcuni strumenti forniscono indicazioni su come condurre l'attività di test, identificando, per esempio, tutte le condizioni che determinano i diversi cammini all'interno del programma. Si noti che questo non costituisce di per sé una generazione automatica di casi di test, operazione spesso non eseguibile in modo automatico a meno che non si proceda ad una esecuzione simbolica del codice. In generale, gli strumenti oggi sul mercato si limitano a tenere traccia delle attività già svolte dall'utente e a produrre in modo semplice ed efficiente la documentazione relativa. Nel mondo della ricerca, al contrario, sono disponibili prototipi di strumenti

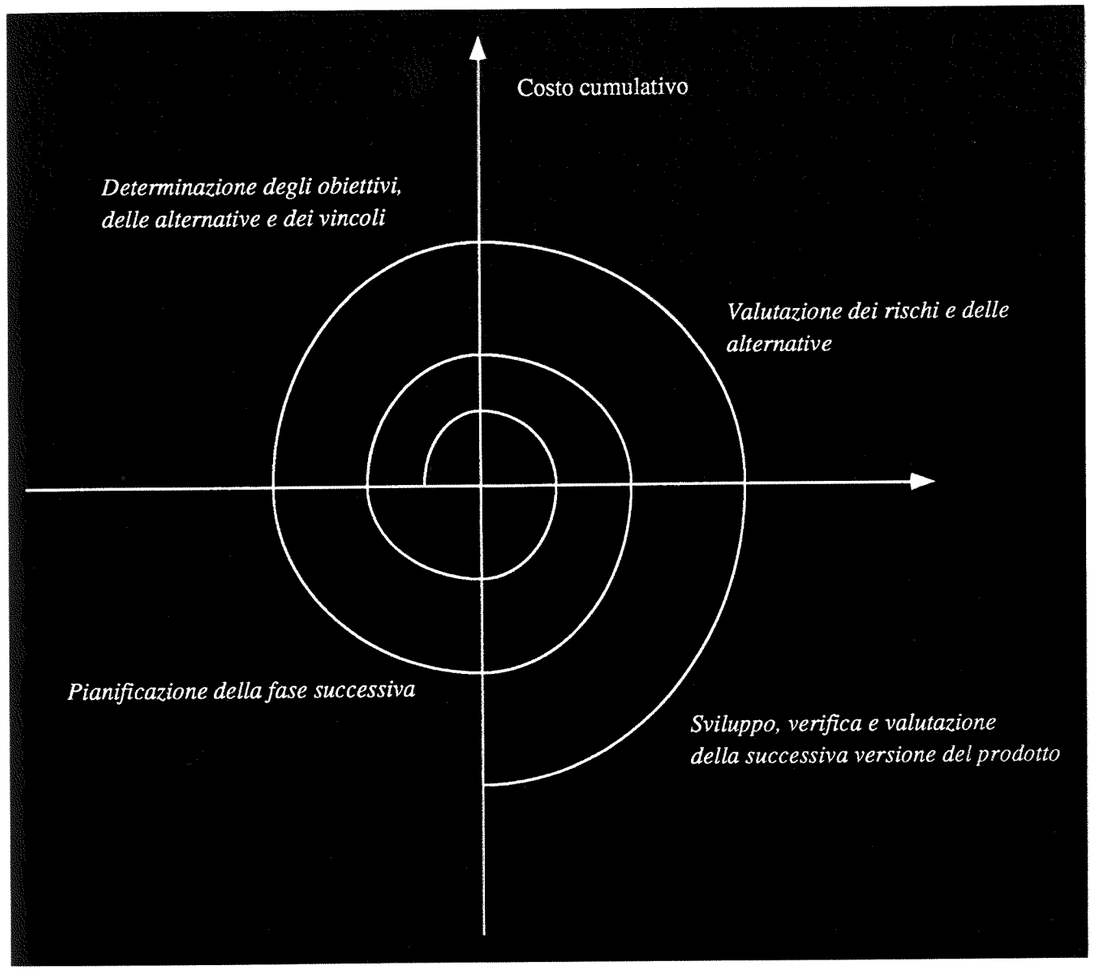


Fig. 1e) - Modello a spirale ("spiral model") dello sviluppo del software.

che forniscono funzionalità più avanzate quali, ad esempio, generazione automatica di casi di test e valutazione incrementale della copertura dei cammini già ottenuta attraverso i test eseguiti.

Strumenti presenti sul mercato che ricadono in questa classe sono il toolkit prodotto da McCabe & Associates, basato sulle omonime metriche, Logiscope e Tefax.

3.4 Strumenti per la gestione delle configurazioni

Le problematiche legate all'attività di gestione delle configurazioni e delle versioni sono particolarmente critiche e di complessa soluzione. Per questo motivo sono stati sviluppati vari strumenti che semplificano il compito di chi deve gestire il processo di sviluppo e manutenzione di un prodotto software. Le funzionalità tipiche di questi prodotti sono le seguenti:

Produzione automatica di oggetti.

Alcuni strumenti di questa classe permettono di ricostruire in modo automatico componenti "software" (nell'accezione più ampia che questo termine può avere: programmi, librerie, documentazione, ...), eseguendo il minimo numero di passi necessari. Se, per esempio, l'oggetto da produrre è un programma eseguibile, a fronte di una modifica in uno dei moduli sorgenti che costituiscono il programma, è possibile fare sì che vengano effettuati quei passi di rigenerazione che sono correlati al solo modulo modificato (tipicamente ricompilazione del modulo e linking dell'intero programma).

Gestione delle versioni.

Questa funzionalità fa riferimento alla possibilità che alcuni strumenti hanno di tenere traccia della evoluzione di un modulo sorgente. In

particolare è possibile gestire l'albero delle versioni attraverso due operazioni fondamentali:

1. Inserimento di una nuova versione di un programma.
2. Reperimento di una qualsiasi versione precedentemente memorizzata.

Per realizzare queste funzioni viene utilizzato il concetto di *delta,* cioè l'insieme di variazioni esistenti nel codice sorgente tra due successive versioni. Normalmente gli strumenti di questa classe memorizzano in modo tradizionale la prima versione del modulo sorgente, mentre, per tenere traccia della sua evoluzione, conservano la lista dei delta che permettono di ottenere, attraverso ricostruzioni successive, le varie versioni prodotte.

Gestione delle configurazioni.

Come abbiamo visto, nella gestione di un prodotto è necessario poter gestire diverse configurazioni, cioè insiemi di moduli che costituiscono le varie evoluzioni del prodotto stesso. Tali insiemi possono differenziarsi tra loro per vari motivi che sono sostanzialmente riconducibili a variazioni nelle versioni dei moduli utilizzati, o a inserimenti e cancellazioni di moduli. Alcuni prodotti offrono la possibilità di gestire le configurazioni di una applicazione, permettendo di memorizzare e reperire in modo efficiente la descrizione dei moduli utilizzati per costruire una specifica configurazione.

Gestione dell'accesso concorrente.

In un ambiente in cui più programmatori devono cooperare per lo sviluppo di un programma complesso, è necessario che vi sia un meccanismo di controllo per identificare eventuali conflitti nell'accesso a singoli componenti del programma. Alcuni strumenti forniscono meccanismi che permettono di inibire il verificarsi di situazioni critiche o di segnalarne la presenza nel caso tale criterio di controllo sia ritenuto troppo vincolante.

3.5 Strumenti di supporto alla specifica.

In questa classe possono essere collocati molti degli strumenti che oggi vengono commercializzati nel mercato degli strumenti CASE: in particolare vengono considerati in questo paragrafo i prodotti sviluppati per fornire un supporto automatico alle fasi di specifica dei requisiti e di progetto delle applicazioni.
Tuttavia, se è certamente vero che tutti questi strumenti si rivolgono ad una stessa fase del ciclo di vita, in realtà essi presentano funzionalità e caratteristiche molto diverse tra loro in quanto si basano su metodologie e approcci estremamente differenti. In particolare, si possono identificare diversi criteri di classificazione che possono essere utilizzati per meglio collocare ciascun strumento all'interno del panorama complessivo del mercato.

Strumenti semiformali o formali

Questo primo criterio fa riferimento al fatto che gli strumenti sul mercato si basano su metodologie di tipo semiformale o formale. Tra le prime possono essere ricordate notazioni quali i diagrammi di flusso dei dati (DFD), i diagrammi SADT, i diagrammi basati sulla notazione di Ward-Mellor (una estensione dei DFD). Tali notazioni permettono solo un controllo di correttezza puramente sintattico, in quanto la loro semantica non è definita in modo formale. Al contrario, strumenti basati su notazioni formali, quali, ad esempio, le reti di Petri (vedi fig. 2) e gli automi a stati finiti, permettono di ottenere significativi risultati in termini di automatizzazione di attività legate al processo produttivo.

Tipologia del linguaggio di specifica.

La maggior parte degli strumenti di questa classe presenti sul mercato si basa su notazioni grafiche e ha come componente principale un editor che consente all'utente di disegnare direttamente sullo schermo il diagramma che descrive l'applicazione. In altri casi, sicuramente meno frequenti rispetto alla maggioranza dei prodotti oggi sul mercato, l'utente utilizza linguaggi di altissimo livello, basati fondamentalmente o su notazioni algebriche o su linguaggi logici.

Tipi di applicazioni.

Le metodologie, le notazioni e i formalismi sviluppati per descrivere uno specifico problema possono essere, in realtà, suddivise in classi a seconda del settore applicativo a cui si rivolgono. È abbastanza usuale identificare una prima classe di notazioni che si rivolgono principalmente al mondo EDP (DFD, SADT, diagrammi Entità-Relazioni, ...) dove l'aspetto essenziale dell'applicazione è costituito dai meccanismi di trasformazione dei dati. Al contrario, esiste una seconda classe di modelli che sono maggiormente adatti alla descrizione di applicazioni di controllo di processo, di avionica o, in generale, di problematiche in cui il flusso di controllo ha una importanza prevalente rispetto alla elaborazione vera e propria delle informazioni (reti di Petri, automi a stati finiti, . . .). Tenendo conto di questo tipo di distinzione è possibile classificare gli strumenti considerati in questo paragrafo a seconda che includano un supporto per metodologie del primo tipo, del secondo o per entrambe.

Funzionalità di meta-editing.

Molti strumenti basati su funzionalità grafiche oggi sul mercato offrono all'utente la possibilità di modificare l'aspetto o, in generale, la sintassi dei simboli che costituiscono il linguaggio grafico di specifica. Questo tipo di funzionalità può avere una duplice giustificazione. Da un lato, all'interno di una organizzazione può accadere che siano già utilizzate notazioni con una sintassi differente da quella fornita dallo strumento e si vuole poter configurare l'ambiente di specifica in modo da favorirne l'adozione da parte dei progettisti e garantire l'uniformità di rappresentazione. Dall'altro, diviene sempre più frequente la necessità di disporre di un ambiente che consenta

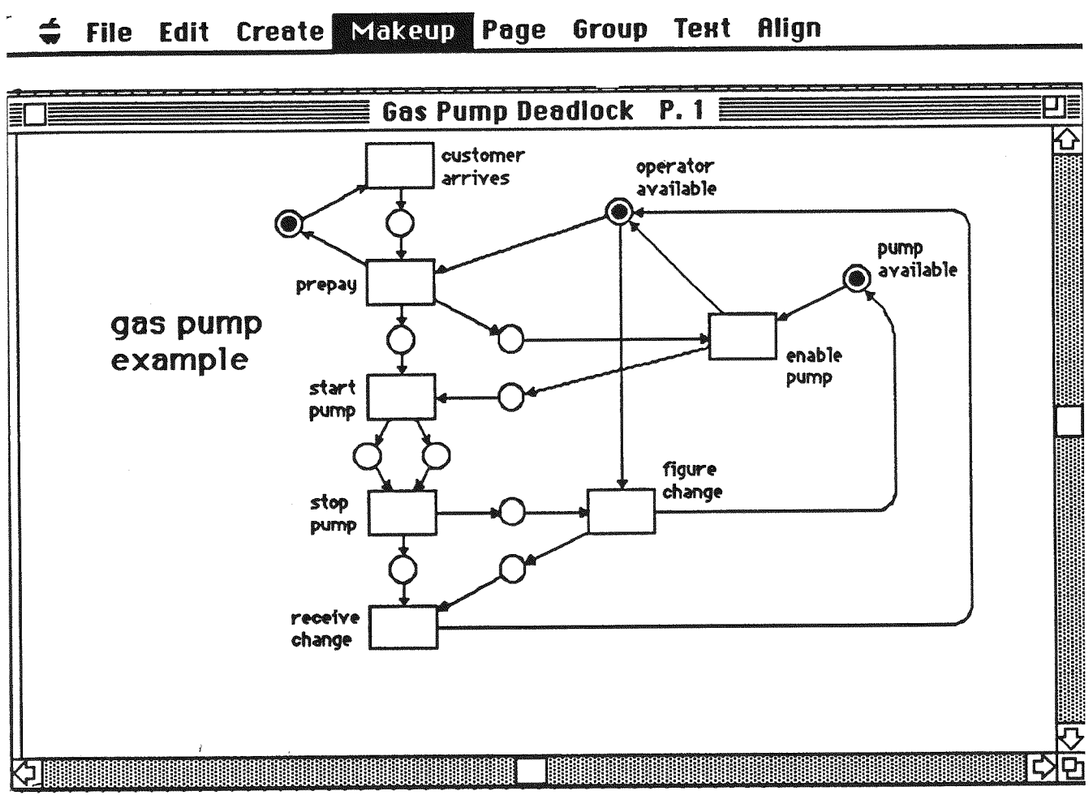


Fig. 2 - Rete di Petri prodotta con uno strumento di supporto alla specifica (Metadesign).

di creare specifiche notazioni grafiche che meglio risolvono particolari necessità applicative.

Tipici esempi di prodotti che ricadono nella classe degli strumenti di supporto alla specifica sono Excelerator, Information Engineering Workbench, Software Through Pictures, Teamwork (tutti basati su notazioni grafiche semiformali), Statemate, ASA (notazioni grafiche formali) e Refine (notazione di tipo linguistico/logico). Un esempio di strumento con funzionalità avanzate di metaediting è Virtual Software Factory (VSF).

3.6 Strumenti di supporto alla manutenzione

E ormai universalmente riconosciuto che - al di là dell'eliminazione di errori residui - la manutenzione del software è in realtà una attività continua di adattamento e estensione delle applicazioni, che si rende necessaria in quanto le esigenze del committente variano e si estendono a seconda delle condizioni ambientali all'interno del quale egli opera, e che l'applicazione in un qualche modo modella, e delle modificazioni che l'uso stesso dello strumento informatico apporta al suo tradizionale sistema di gestione e manipolazione delle informazioni. In quest'ottica risulta evidente che tutti gli strumenti utilizzati durante lo sviluppo e il test sono in varia misura utilizzati anche per l'attività di manutenzione.
Tuttavia, vi sono alcuni strumenti che sono stati specificatamente sviluppati per rendere più agevole il compito di chi manutiene una applicazione, specie nel caso in cui si abbia a che fare con software sviluppato senza utilizzare tecniche di programmazione strutturata e non sufficientemente commentato. In particolare, nel mondo scientifico e, soprattutto, in quello industriale vi è una crescente attenzione nei confronti di quelle metodologie e tecniche che vengono complessivamente identificate con il termine di *Reverse Engineering.* In realtà sotto questa espressione di recente introduzione si celano diversi tipi di strumenti che presentano funzionalità molto diverse tra di loro e non sempre particolarmente innovative rispetto a ciò che altri prodotti già sul mercato possono offrire.

Strumenti per la formattazione dei programmi (*Pretty- printing).*
Alcuni strumenti forniscono funzionalità per la formattazione dei programmi secondo convenzioni dipendenti dai linguaggi utilizzati,

che mirano a rendere visibili i costrutti linguistici utilizzati all'interno del programma. In alcuni casi viene prodotta anche una rappresentazione diagrammatica del programma (il grafo di flusso) che permette una più facile interpretazione del flusso di controllo. Si noti che funzionalità di questo tipo sono state già sperimentate e integrate all'interno di prodotti da tempo sul mercato: il grado di innovazione e di differenziazione da essi introdotto sembra così limitato da non meritare una particolare enfasi o caratterizzazione.

Strumenti per la ristrutturazione dei programmi.
Questi strumenti, benchè non molto numerosi e con diversi livelli di completezza nelle loro funzionalità, permettono di ristrutturare e semplificare la struttura di un programma, modificandone il codice sorgente. In generale, essi sono in grado di ricavare una rappresentazione astratta del codice sorgente, utilizzandola in seguito per il processo di semplificazione e rigenerazione del programma. Tale processo tende da un lato ad eliminare caratteristiche indesiderate (come ad esempio i *go-to*) e, dall'altro ad identificare anomalie del codice come, ad esempio, istruzioni non raggiungibili o variabili usate senza essere inizializzate. Inoltre, tali programmi effettuano anche la formattazione del programma sorgente e la generazione automatica di documentazione di corredo del codice esaminato. Uno strumento che può essere considerato rapresentativo di questa classe è Recoder, un ristrutturatore di programmi COBOL.

3.7 Strumenti per la gestione di basi di dati

Gli strumenti per la gestione di basi di dati sono ormai molto diffusi in molti settori applicativi e presentano un larga fascia di funzionalità. In questo contesto non si vogliono considerare quei prodotti che costituiscono di per sè un ambiente di sviluppo (quali ad esempio gli ambienti della IV Generazione che verranno esaminati in un prossimo paragrafo), quanto quei componenti che hanno la forma di librerie run- time di procedure richiamabili dall'interno di programmi scritti utilizzando linguaggi tradizionali, quali COBOL o C. Molti di questi prodotti sono stati sviluppati per completare le funzionalità offerte dai sistemi della IV Generazione, permettendo ai programmatori di sviluppare parti critiche dell'applicazione in linguaggi di più basso livello e, quasi sempre, più efficienti, continuando però ad accedere alla stessa base di dati.
Una categoria particolarmente interessante di prodotti è quella che permette di integrare all'interno dei linguaggi tradizionali istruzioni del linguaggio per la gestione di basi dati relazionali SQL. Questi strumenti, in generale, permettono di inserire direttamente nel codice del programma le istruzioni nel formato SQL nativo: il programma viene quindi analizzato da un precompilatore che sostituisce al codice SQL le opportune chiamate alle routines in grado di soddisfare la richiesta dell'utente (si fa spesso riferimento a questa modalità di utilizzo del linguaggio SQL con l'espressione *Embedded* SQL). Un prodotto particolarmente significativo che ricade in questa classe è Informix ESQL/C, che integra appunto SQL con il linguaggio C.
In generale, gli strumenti per la gestione di basi di dati possono essere útilizzati in due diversi contesti:

Base di dati applicativa.
Ovviamente, essi possono essere incorporati in applicazioni di tipo tradizionale in cui è necessario gestire grandi quantità di informazioni (applicazioni gestionali, bancarie, amministrative, ...). In quest'ambito, vi è una forte tendenza ad utilizzare sempre più DBMS basati sul modello relazionale, in quanto risulta più semplice garantire l'indipendenza dell'applicazione dalla struttura fisica dei dati.

Ambienti di supporto alle attività di ingegneria del software.
Se l'applicazione che deve essere sviluppata è in realtà essa stessa uno strumento CASE o un ambiente completo di supporto allo sviluppo di software (SEE), il discorso diviene più articolato, in quanto i requisiti che vengono posti all'ambiente di gestione delle basi di dati sono più complessi di quanto accade nel mondo delle applicazioni tradizionali. Infatti, se è vero che molti prodotti CASE di derivazione sia industriale che accademica oggi presenti sul mercato si basano su DBMS di tipo convenzionale (come il già citato ESQL/C), va sottolineato che esiste una vasta area di ricerca che attraversa trasversalmente le due comunità scientifiche (basi di dati e ingegneria del software) e che tende alla definizione e allo sviluppo di DBMS specializzati, che meglio supportino lo sviluppo di moderni SEE. Esempi di progetti di ricerca in questo settore sono quelli relativi allo sviluppo del DBMS per SEE Cactis ([HK88]), o alla definizione di piattaforme per lo sviluppo di SEE (descritte nella seconda parte del capitolo), che includono al loro interno un componente dedicato alla gestione delle informazioni presenti nell'ambiente.

Questi strumenti utilizzano modelli di dati più complessi di quello offerto dai DBMS relazionali, in quanto gli oggetti che devono essere manipolati all'interno di un SEE hanno caratteristiche differenti rispetto a quelle delle tradizionali basi di dati applicativa cui si è fatto cenno in precedenza (per una introduzione a tali problematiche si veda [Ber87]).

3.8 Strumenti per lo sviluppo di interfacce utente

La crescente richiesta di interfacce utente di immediato e facile uso ha fatto si che in molte applicazioni il componente di maggiore complessità risulta essere lo strato di software che gestisce il colloquio tra utente e programma. Tale strato deve essere in grado di fornire al programmatore applicativo meccanismi standard e predefiniti per la gestione della visualizzazione dei dati sullo schermo e lo scambio di messaggi (comandi - risposte) tra

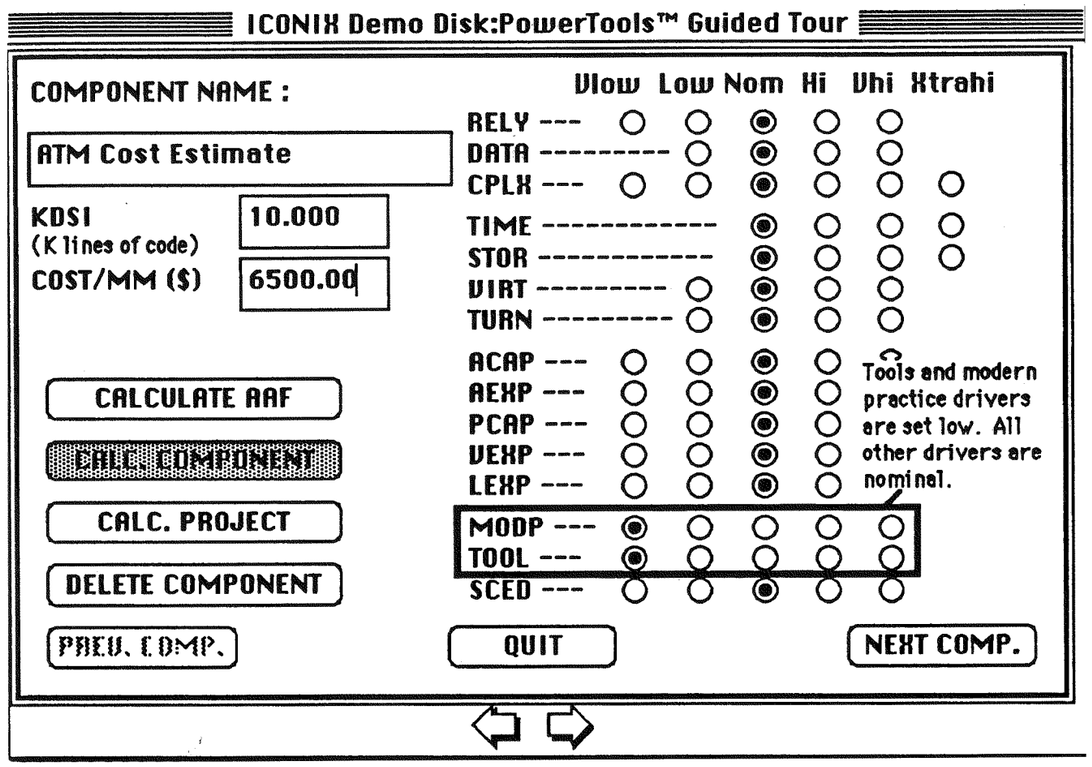


Fig. 3 - Definizione dei coeficienti del COCOMO in uno strumento per la stima dei costi (CoCoPro).

utente e applicazione; inoltre, deve realizzare *metafore* (cioè trasposizioni nel mondo informatico di oggetti del mondo reale) dell'ambiente di lavoro tradizionale che permettano di ottenere interfacce di facile apprendimento e in grado di assistere l'utente anche quando questi non conosce esattamente come una determinata funzionalità deve essere invocata o commette delle imprecisioni nella specifica del comando.
Spesso, per sottolineare le caratteristiche tipiche di una moderna interfaccia utente, vengono utilizzate due sintetiche espressioni inglesi: *DWIM - Do What I Mean* e *WYSIWYG - What You See Is What You Get.* Queste due espressioni stanno ad indicare la capacità dell'interfaccia utente di tollerare incompletezze o errori nelle richieste dell'utente o anche che l'oggetto elaborato dall'utente (documentazione, diagrammi, schemi, . . .) viene in ogni istante presentato sullo schermo nell'esatto formato che avrà una volta stampato.
Nel corso di questi ultimi anni sono stati sviluppati un certo numero di prodotti di supporto per la creazione di interfacce utente, quali ad esempio XWindow o la coppia Windows-Presentation Manager in ambiente MS-DOS. Tali prodotti hanno da un lato lo scopo di semplificare il compito dello sviluppatore delle applicazioni, che viene a disporre di un ricco insieme di funzioni predefinite per la gestione dello schermo e degli aspetti grafici, e dall'altro permettono di realizzare applicazioni con interfacce omogenee e gestibili in modo integrato in uno stesso ambiente di visualizzazione. Inoltre, spesso questi prodotti forniscono anche meccanismi per lo scambio di dati tra programmi, favorendo perciò, oltre che una integrazione delle applicazioni che potremmo definire operativa (appaiono sullo schermo in contemporanea e con interfacce utente omogenee), anche una integrazione di tipo funzionale ([Mye88]).

3.9 Strumenti per la pianificazione ed il controllo del progetto

Le principali problematiche legate alla gestione di un progetto sono riconducibili a due attività specifiche: l) stima del costo e delle risorse necessarie al completamento di un progetto e 2) pianificazione e controllo d'avanzamento. Gli strumenti oggi disponibili sono pertanto catalogabili in due classi:

Strumenti di stima basati su modelli algoritmici.
In questa classe sono catalogabili i molti strumenti che sono stati sviluppati basandosi su metodologie di stima dei costi quali il COCOMO (vedi [Boe81] e la fig. 3).

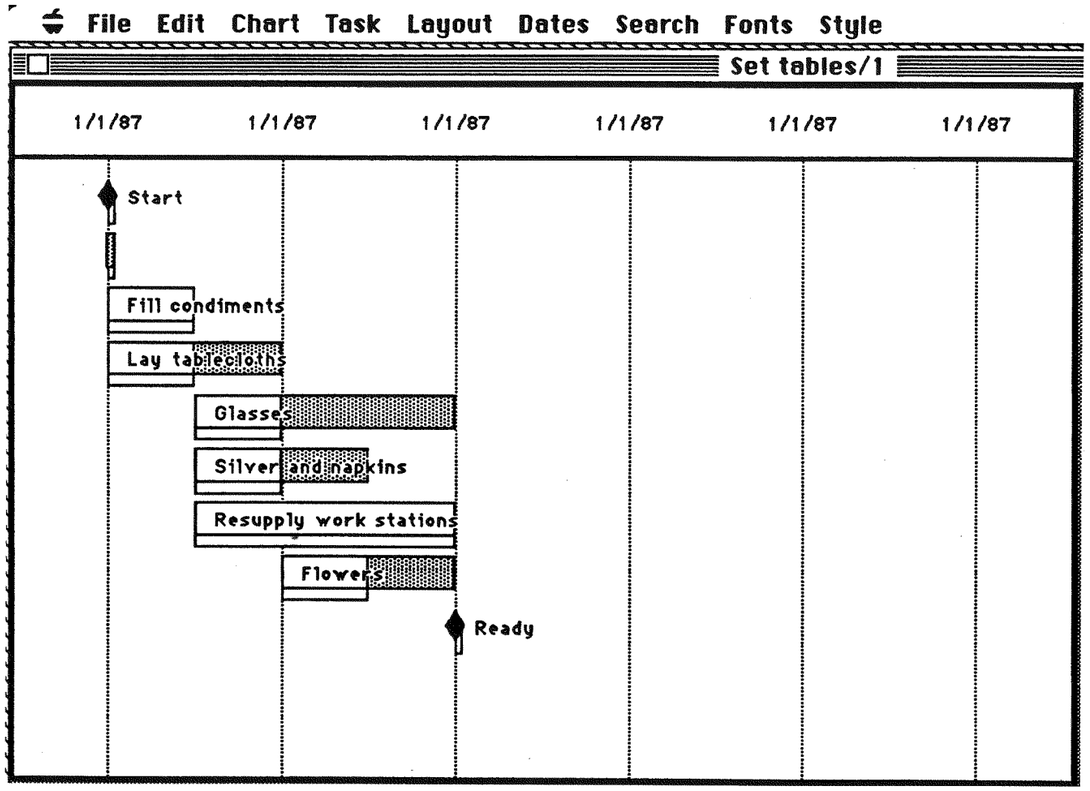


Fig. 4 - Carta di Gantt prodotta da uno strumento di pianificazione di progetto (MacProject).

Questi strumenti consentono all'utente di specificare i vari parametri del modello e di indicare la dimensione stimata dell'applicazione espressa in linee di codice o attraverso la tecnica dei Punti Funzione (vedi [AG83]): il programma calcola quindi direttamente i valori relativi al costo, alla durata del progetto e, in generale, tutte le informazioni che sono calcolabili in base al modello utilizzato.
In alcuni strumenti basati sul modello COCOMO, è possibile memorizzare in un archivio storico le informazioni relative alle stime già effettuate dall'utente: il sistema utilizza tali dati per raffinare i parametri che adattano la forma delle equazioni del modello alle caratteristiche dello specifico problema applicativo e dell'ambiente di sviluppo.

Strumenti di pianificazione e controllo.
Questi strumenti si basano su alcune notazioni molto diffuse, quali le carte di Gantt e i diagrammi PERT, e permettono di descrivere la struttura di un progetto, cioè l'insieme di attività in cui esso si articola, la loro distribuzione temporale e le risorse necessarie al loro compimento. Questi strumenti sono spesso basati su tecniche che consentono all'utente di operare direttamente sul diagramma (Gantt o PERT; ved. fig. 4) con un aggiornamento immediato del layout complessivo del progetto ogniqualvolta l'utente modifica le caratteristiche di una delle attività.

Alcuni esempi tipici di prodotti sono WICOMO, CoCoPro, Project Workbench e Superproject Expert (i primi due sono strumenti di stima dei costi basati sul COCOMO mentre gli altri sono strumenti di pianificazione e controllo).

3.10 Spreadsheet
I fogli elettronici (in inglese *spreadsheets o Worksheets)* sono tra i più diffusi strumenti di produttività individuale oggi presenti sul mercato: il prodotto più venduto degli ultimi anni nel mondo dei personal computers è proprio il famoso Lotus 1-2-3 che è stato distribuito in tutto il mondo in milioni di copie.
Il foglio elettronico si basa su un concetto molto semplice: l'utente ha a disposizione una matrice di celle ciascuna delle quali può contenere numeri, stringhe di caratteri o formule che utilizzano come dati i valori memorizzati in altre celle. Quando il contenuto di una cella viene modificato, lo strumento provvede a ricalcolare il valore di

tutte le formule che vi fanno riferimento diretto o indiretto (cioè attraverso altre formule): in questo modo è possibile realizzare modelli che permettono analisi del tipo "cosa succede se . . .", che, secondo una diffusa terminologia inglese, sono chiamate analisi *what-if.*
Ad esempio si supponga di avere realizzato un foglio elettronico che implementa le formule del COCOMO: modificando i valori di uno o più coefficienti correttivi è possibile avere l'immediata valutazione delle formule del modello e, quindi, valutare l'impatto che i diversi coefficienti hanno sulla stima finale. Tipicamente le formule di un foglio elettronico vengono espresse attraverso l'utilizzo di linguaggi che hanno una struttura sintattica molto semplice, ma dispongono di funzioni predefinite molto potenti (la maggior parte delle quali dedicate alla soluzioni di problemi finanziari e statistici) e permettono la creazioni di semplici programmi, chiamati spesso *macro*, che permettono di preconfigurare le operazioni compiute in modo sistematico dall'utente.
La possibilità di creare in modo molto semplice questi programmi ha fatto sì che il foglio elettronico si sia diffuso enormemente presso professionisti e imprese, che lo utilizzano per creare modelli di problemi finanziari, gestionali, amministrativi e anche di tipo tecnico: il foglio elettronico è in pratica divenuto uno degli strumenti di sviluppo di applicazioni più diffusi nel mondo degli utenti finali. In questo panorama vi sono due usi del foglio elettronico particolarmente significativi nell'ambito dell'analisi che stiamo compiendo:

1. strumento di supporto per l'attività di direzione e controllo di un progetto;

2. strumento utilizzato per sviluppare applicazioni (macro) da distribuire presso utenti finali.

Si noti che la semplicità e l'efficacia con le quali si possono utilizzare i fogli elettronici hanno portato in molti casi a degenerazioni nell'uso di questi strumenti: nati e strutturati per compiere analisi di tipo qualitativo, dove cioè è necessario analizzare in modo sofisticato pochi dati, oggi spesso il foglio elettronico viene utilizzato per la gestione di dettaglio del problema dell'utente (per esempio nel settore del controllo gestionale), fatto che mette inevitabilmente in luce i limiti di questa classe di strumenti, limiti dovuti alla loro struttura e concezione.

3.11 Strumenti per la gestione di ipertesti.
Gli ipertesti sono una delle più recenti innovazioni nel settore della gestione di informazioni: in realtà essi furono già studiati e sperimentati negli anni sessanta, ma la loro definitiva affermazione è avvenuta nella seconda metà degli anni ottanta grazie all'introduzione di strumenti di basso costo, utilizzabili su personal computers anche da utenti poco esperti ([Con87]).
Un ipertesto è sostanzialmente un documento strutturato in modo da permetterne un *attraversamento non lineare.* Normalmente, in un documento di tipo tradizionale, le informazioni in esso contenute possono essere esaminate attraverso una scansione di tipo sequenziale che riflette il modo in cui tali documenti sono costruiti e memorizzati, cioè come sequenze di caratteri. Negli ipertesti, un documento è costituito da oggetti di varia natura memorizzati in un data base: gli oggetti sono collegati tra loro attraverso legami che consentono a chi esamina il documento di accedere alle diverse informazioni seguendo uno qualsiasi dei collegamenti definiti. Ogni oggetto può essere un testo, un'immagine o un grafico: i collegamenti permettono di associare in modo libero ad un oggetto le informazioni ad esso correlate. Da un punto di vista esterno, l'associazione avviene inserendo nell'oggetto di partenza un simbolo grafico che funge da pulsante di attivazione e che permette di richiamare l'oggetto collegato: per esempio, a fianco di un testo che descrive l'andamento economico di una società si possono inserire i richiami ai grafici che descrivono visivamente le variabili in gioco, attivabili in seguito attraverso l'uso del mouse. Le recenti evoluzioni dei gestori di ipertesti permettono di utilizzare all'interno di un documento anche informazioni di tipo vocale o sequenze televisive: in questo caso i documenti così costruiti si dicono *ipermediali.* Gli utilizzi tipici degli ipertesti sono sostanzialmente due:

Creazione di documenti complessi di facile consultazione.
Tipici esempi possono essere i manuali in linea *(help-on-line)* per programmi applicativi, presentazioni (guide per il visitatore, presentazioni d'affari, ...) in cui si vuole lasciare all'utente del documento la possibilità di scegliere quali parti leggere e in quale ordine o, anche, la creazione di documentazione elettronica di programmi che viene così a sostituire gli ormai assai voluminosi e poco leggibili manuali.

Prodotti per utenti finali.
Gli ipertesti possono essere utilizzati per creare veri e propri prodotti per utenti finali quali programmi elettronici di formazione (per il cosiddetto CAE - *Computer Aided Education)* o applicazioni di automazione d'ufficio.

3.12 Strumenti per lo sviluppo di sistemi esperti.
Nel corso dell'ultimo decennio vi è stato un crescente interesse da parte sia della comunità scientifica che di quella industriale verso le tecnologie legate allo sviluppo dei *sistemi esperti* e alla creazione e manipolazione delle cosiddette *basi di conoscenza.* I programmi sviluppati in questo ambito sono stati costruiti utilizzando una tipologia di strumenti che presentano un certo numero di caratteristiche distintive rispetto agli altri strumenti esaminati in precedenza: da un lato essi offrono dei modelli di dati particolarmente sofisticati, in grado di gestire in modo assai raffinato elaborazioni su informazioni di tipo simbolico, e dall'altro presentano diversi paradigmi linguistici che consentono di affrontare secondo

la strategia più adatta specifici problemi applicativi.
Pur rimandando alla letteratura specializzata per un approfondimento di questo settore in rapida evoluzione (si veda per esempio [Wat86]), è possibile evidenziare due categorie di strumenti comunemente utilizzati per lo sviluppo di sistemi esperti e per la costruzione di basi di conoscenza. La distinzione tra queste due categorie, in pratica, non è sempre ben marcata.

Linguaggi di programmazione.
In questa categoria vengono di fatto considerati alcuni linguaggi di programmazione tradizionali che si sono dimostrati particolarmente adatti allo sviluppo di sistemi esperti. In particolare, LISP e Prolog si stanno ormai affermando come i linguaggi più diffusi ed utilizzati sia in ambito accademico che industriale, in quanto gestiscono in modo molto sofisticato aspetti quali l'elaborazione di informazioni simboliche e la definizione di strategie di risoluzione dei problemi. Oltre ad essi, si sono diffusi linguaggi orientati ad oggetti quale Smalltalk, ed estensioni linguistiche che hanno sovrapposto ai linguaggi funzionali (tipo LISP) o logici (tipo Prolog) dei meccanismi di strutturazione modulare basati sul paradigma ad oggetti. Nella prima classe ricadono FLAVOURS e LOOPS; nella seconda ricade Vulcan. Ricordiamo infine un ulteriore paradigma linguistico che trova largo impiego in questo settore: il paradigma a regole. Secondo tale paradigma la descrizione dell'algoritmo è fornita mediante regole del tipo *if-then,* che associano al verificarsi di una condizione una serie di azioni corrispondenti. Un esempio di linguaggio che ricade in questa categoria è OPS5.

Linguaggi per la rappresentazione della conoscenza.
Questo tipo di strumenti è solitamente costituito da un potente linguaggio specializzato per la costruzione di sistemi esperti e da un insieme di strumenti di supporto allo sviluppo del codice. E possibile scomporre ulteriormente questo gruppo di prodotti in due sottoclassi, considerando separatamente i cosiddetti gusci (o scheletri o "*shell*") di sistemi esperti e i sistemi general-purpose.

• Un sistema del primo tipo è sostanzialmente un sistema esperto al quale è stata sottratta la conoscenza del proprio specifico dominio applicativo, divenendo, così, un guscio utilizzabile per la costruzione di sistemi esperti diversi rispetto a quello di partenza (anche se, spesso, sono utilizzabili solo nello stesso settore applicativo).

• Nella seconda classe di strumenti possono essere considerati gli strumenti di tipo generale con i quali è possibile affrontare problemi di domini applicativi estremamente diversi tra loro: sostanzialmente, essi permettono di gestire con maggiore precisione e flessibilità le informazioni della base della conoscenza. Tipici esempi di strumenti di questa classe sono KEE, Knowledge Craft e ART.

4. Ambienti

Come si è già detto all'inizio, con l'espressione ambiente di sviluppo intendiamo considerare l'insieme degli strumenti che vengono utilizzati per progettare, costruire e gestire una specifica applicazione informatica. Prima di descrivere le classi di prodotti oggi disponibili, è però opportuno distinguere due grosse tipologie di ambienti di sviluppo, caratterizzate dalla relazione che essi hanno con l'ambiente operativo all'interno del quale l'applicazione finale verrà ad operare. Tale distinzione è ortogonale, in generale, rispetto alla classificazione che verrà presentata nel seguito: di fatto accade molto spesso che in una stessa classe di ambienti vengano utilizzati, in prodotti diversi o perfino in diverse versioni di uno stesso prodotto, entrambi gli approcci. I due diversi approcci possono essere così definiti:

Approccio integrato: l'ambiente di sviluppo è omogeneo con quello run-time (stesso HW, stesso software di sistema).

Approccio host-target: l'ambiente run-time è diverso da quello di sviluppo.

Un semplice esempio di approccio del primo tipo si ha considerando lo sviluppo di un programma per ambienti quali Unix o VMS: di solito per tali operazioni si utilizza la stessa macchina (o una equivalente) sulla quale a regime funzionerà l'applicazione. Il secondo approccio viene seguito quando l'architettura HW/SW run-time non è utilizzabile come ambiente di sviluppo o non è conveniente dal punto di vista economico. Vediamo due tipici casi:

1. I programmi di controllo del funzionamento di un aereo sono eseguiti da calcolatori specializzati che hanno un'architettura HW particolare (si tratta quasi sempre di sistemi ridondanti), controllata da un sistema operativo specializzato per la gestione e il controllo in tempo reale dei dati sul volo. Questi programmi di norma non possono essere sviluppati sul calcolatore specializzato in quanto sarebbe troppo costoso e spesso non fattibile sviluppare l'intero ambiente di sviluppo su questo tipo di HW. A questo scopo si utilizzano macchine general-purpose che dispongono di un ambiente di sviluppo completo e che include, di solito, un simulatore dell'ambiente operativo reale e un cross-compilatore in grado di generare il codice per il processore effettivamente utilizzato sull'aereo: in questo modo l'applicazione può essere completamente codificata e convalidata nell'ambiente host prima di essere collaudata definitivamente nell'ambiente operativo target.

2. Molte applicazioni EDP richiedono l'utilizzo di grossi mainframe in grado di sopportare i notevoli carichi operativi imposti dall'utenza (dimensioni delle banche dati, numero di transazioni al secondo, . . .). Spesso, sviluppare un'applicazione direttamente su questi ambienti risulta molto oneroso a causa dei notevoli costi di gestione di tali macchine e del relativo software di sviluppo. Per questi motivi, molti prodotti per mainframe dispongono oggi di versioni dell'ambiente di sviluppo operanti sotto MS-DOS: in tal modo è possibile sviluppare e testare una applicazione su PC e trasferirla su mainframe solo quando lo sviluppo è terminato.

4.1 Ambienti di sviluppo orientati ad un linguaggio

Gli ambienti di sviluppo basati su linguaggi sono caratterizzati dal fatto che il sistema operativo e gli strumenti che popolano l'ambiente sono specializzati per un particolare linguaggio di programmazione: in alcuni casi lo stesso hardware viene realizzato in funzione dell'ambiente di sviluppo (si veda il caso delle ormai numerose macchine LISP). Tipici esempi di tali ambienti sono il Cornell Synthesizer, Smalltalk, Interlisp e Cedar, sviluppati rispettivamente per i linguaggi Pascal, Smalltalk (l'ambiente ed il linguaggio hanno lo stesso nome), LISP e Mesa. In questi ambienti l'aspetto che viene maggiormente enfatizzato è la possibilità per l'utente di utilizzare uno stile di programmazione di tipo esplorativo o incrementale: infatti, l'ambiente permette di passare in modo pressochè immediato dalla scrittura del codice alla sua esecuzione e test, consentendo al programmatore di sviluppare e provare velocemente semplici prototipi di programmi che possono essere in seguito raffinati, arricchiti e integrati. Questo tipo di approccio è reso possibile dall'architettura di questi sistemi che sostanzialmente integra in un'unica unità sia l'ambiente di sviluppo che quello run-time, per sviluppare i quali viene spesso utilizzato lo stesso linguaggio al quale sono indirizzati (per esempio, l'ambiente di sviluppo Smalltalk è scritto in Smalltalk, mentre Interlisp in LISP). La conseguenza principale di questa scelta architetturale consiste nel fatto che il progettista può facilmente integrare nel programma sviluppato funzionalità presenti nell'ambiente di sviluppo o estendere l'ambiente stesso inserendo nuovi componenti sviluppati in proprio e che si ritiene possano essere utilmente riutilizzati in futuro.
Alcune caratteristiche comuni di questi ambienti possono essere così sintetizzate:

• I programmi non vengono memorizzati come semplici stringhe di caratteri, ma si tende ad utilizzare formati che permettono di registrare anche informazioni sulla semantica del linguaggio: per esempio nell'ambiente Rational sviluppato per il linguaggio Ada, i programmi vengono memorizzati in un formato interno, basato sul modello DIANA.
Per accedere ai programmi memorizzati in questi formati intermedi l'utente ha a disposizione strumenti che gli consentono di navigare attraverso le strutture da lui create *(browsers).*

• Molti degli ambienti basati su linguaggi includono al loro interno funzionalità per gestire la programmazione in grande. Cedar, ad esempio, consente di descrivere la struttura, intesa come gerarchia di moduli, che costituisce un programma, tenendo traccia in modo automatico della evoluzione delle versioni e aiutando l'utente nella selezione dei componenti necessari per la costruzione di una specifica configurazione. Rational include funzionalità per la gestione dell'accesso concorrente di più utenti allo stesso insieme di moduli.

Complessivamente, gli ambienti orientati al linguaggio permettono una grande flessibilità durante la fase di sviluppo: tuttavia, presentano alcune controindicazioni che possono essere riassunte in due punti:

1. L'estrema dinamicità nella creazione e modifica dell'ambiente, se da un lato consente di ottenere una estrema flessibilità, facilitando e ottimizzando notevolmente lo sviluppo di prototipi, alla fine può creare grossi problemi di gestione e comprensione della struttura dell'ambiente che, come abbiamo visto, può crescere inglobando parti di programmi sviluppati dagli utenti stessi.
2. Questi ambienti, essendo sostanzialmente ideati e costruiti attorno ad uno specifico linguaggio, mal si prestano o, addirittura, rendono inattuabili operazioni quali trasporto di applicazioni da/su altri ambienti, sviluppo in altri linguaggi, integrazioni con applicazioni esterne all'ambiente.

4.2 Ambienti della IV Generazione

Gli ambienti della IV generazione costituiscono uno dei maggiori sviluppi degli ultimi anni in quanto hanno notevolmente contribuito a mutare l'approccio utilizzato nello sviluppo di applicazioni di tipo EDP o commerciale in genere (vedi [FG87]). Essi sono nati come naturale evoluzione degli ambienti tradizionali di sviluppo, quali quelli centrati sui linguaggi COBOL e Pascal, con lo scopo di tenere conto al meglio di quelle che sono le peculiarità tipiche delle applicazioni di tipo EDP:
1. Relativa semplicità delle elaborazioni da compiere sui dati.
2. Notevoli complessità e dimensioni delle basi di dati utilizzate.
3. Necessità di costruire complesse interfacce uomo-macchina che siano allo stesso tempo di semplice comprensione ed utilizzo.
4. Necessità di sviluppare prototipi dell'applicazione in breve tempo in modo da fornire all'utente finale uno strumento con il quale verificare le proprie necessità applicative e la comprensione che di esse ha avuto il progettista.
5. Necessità di fornire strumenti di sviluppo che consentano di costruire, modificare e integrare il patrimonio di applicazioni di una azienda in modo incrementale, controllato ed efficiente.

Quest'ultima osservazione fa riferimento alla situazione in cui si vengono a trovare la maggior parte delle organizzazioni che dispongono di un sistema informativo: le crescenti richieste di programmi che coprano nuove problematiche applicative e di modifiche e cam-

biamenti ai programmi esistenti, che tengano conto delle mutate esigenze dell'azienda, pongono ai responsabili dei centri EDP grosse sfide organizzative, tecnologiche e finanziarie.
Le principali funzionalità oggi disponibili nei prodotti sul mercato sono le seguenti:

Gestione dell'interazione con l'utente.
I prodotti di questa classe dispongono di sofisticati strumenti per lo sviluppo di menu interattivi, di interfacce utente di tipo tradizionale/alfanumerico e per la generazione di rapporti. Di solito nell'ambiente sono presenti editors per la costruzione delle maschere video e del tracciato dei report, gestori runtime che permettono all'utente di mostrare a video le maschere e utilizzarle dall'interno di un programma applicativo, generatori di rapporti che, in base alla definizione del rapporto, sono in grado di estrarre le informazioni richieste dalla base di dati, elaborarle e stamparle sul dispositivo di uscita prescelto (video, stampante, file). Disponendo di questi strumenti risulta molto più rapido ed efficiente sviluppare l'interfaccia tra utente e sistema e verificarne la validità.

Gestione della base di dati.
Tutti gli ambienti di questa classe utilizzano, per la gestione delle informazioni proprie dell'applicazione, un sistema per la gestione di basi di dati (Data Base Management System - DBMS): in alcuni casi l'ambiente della IV Generazione dispone di un suo DBMS, ma accade spesso che il prodotto permetta di interfacciarsi direttamente con alcuni dei molti DBMS oggi disponibili sul mercato. Per esempio, un ambiente come Paradox, assai diffuso tra gli utenti di PC, dispone di propri meccanismi per la gestione delle basi di dati, mentre Focus, un linguaggio pesantemente utilizzato nei grossi centri EDP, può integrarsi assai facilmente con diversi tipi di basi di dati (DB2, DL/l, . . .).

Gestione degli aspetti procedurali.
Per poter descrivere gli aspetti procedurali di una applicazione, gli strumenti di questa classe utilizzano diversi approcci, spesso non mutuamente esclusivi, ma che si integrano o differenziano in modo sensibilmente diverso al variare del prodotto. In generale, ai due estremi, possono essere identificati due approcci:

- Ambiente guidato dal linguaggio (*language-driven*): in questo caso l'utente ha a disposizione un unico linguaggio (chiamato appunto Linguaggio della IV generazione-abbreviato 4GL) che include tutte le funzionalità per la gestione degli aspetti procedurali e delle altre funzionalità viste in precedenza (gestione dell'interfaccia utente e gestione delle basi di dati). Il linguaggio è di solito interpretato, anche se sempre piu spesso sono disponibili anche compilatori in grado di generare, al termine del processo di sviluppo, una applicazione chiusa. Inoltre, l'interprete è corredato di editors interattivi per la creazione delle definizioni di maschere e reports. Tipici esempi di strumenti di questo tipo sono IDEAL e Informix 4GL.

- Ambiente guidato dalle maschere (*form-driven*): in questo caso non esiste un vero e proprio linguaggio di programmazione in quanto la struttura dell'applicazione è determinata dai menù che la costituiscono (definiti spesso in modo interattivo dall'utente) e dalla sequenza di maschere e reports ad essi collegati (ad ogni voce del menù si assegna o un menù di livello inferiore o l'attivazione di una maschera o di un report). L'utente ha di norma la possibilità di associare a particolari punti della definizione di una maschera (per esempio un campo) o ad eventi relativi alla gestione della maschera stessa a run-time (arrivo del cursore in campo, pressione di un tasto funzione, . . .) una serie di istruzioni di controllo in un semplice linguaggio specializzato, o il richiamo di un programma scritto in linguaggio esterno (per esempio, SQL o C). Lo scopo di questi frammenti di codice è quello, per esempio, di attivare una nuova maschera, di effettuare controlli semantici sul singolo dato o di gestire le condizioni di errore. Un esempio tipico di strumento che ricade in questa classe è Oracle.

Ambienti di sviluppo.
Un ambiente di sviluppo della IV Generazione è di solito contraddistinto da una notevole facilità di utilizzo, resa possibile dall'uso di estese funzionalità di spiegazioni ed aiuti in linea e di tecniche di interazione incentrate sulla presentazione all'utente di quelle che sono, di volta in volta, le sole operazioni possibili nell'attuale contesto di lavoro: l'utente, cioè, deve sempre meno interagire attraverso comandi da presentare ad un interprete anonimo (così come avviene quando ci si siede davanti ad un terminale di un sistema tradizionale), in quanto l'ambiente di sviluppo fa uso di quelle stesse tecniche, basate sull'uso di menù interattivi, che saranno poi inserite nel prodotto sviluppato. Un altro aspetto interessante di questi ambienti di sviluppo consiste nel fatto che forniscono strumenti, seppur spesso molto semplici, di supporto al testing e alla manutenzione (in particolare, per il debugging dell'applicazione e per la gestione di insiemi di moduli). In generale, essi tendono a configurarsi come ambienti autonomi all'interno dei quali svolgere tutte le operazioni relative alla gestione di un applicazione.

Dopo aver presentato questa breve panoramica delle funzionalità presenti negli strumenti oggi sul mercato, è opportuno esaminare un'altro tipo di classificazione che distingue i prodotti in base all'uso tipico per cui sono stati sviluppati e, quindi, alla fascia di utenti a cui sono indirizzati.

Ambienti di produzione.
Gli strumenti di questa classe sono tipicamente utilizzati per lo sviluppo di applicazioni finali da grossi centri EDP, che devono provvedere allo sviluppo dei programmi applicativi per l'organizzazione in cui sono inseriti, o da case di software che sviluppano e distribuiscono applicazioni per utenti finali. Esempio tipici di questi prodotti sono i già citati IDEAL (principalmente utilizzato su grossi mainframe), Informix 4GL (disponibile su una grande fascia di macchine che va dai personal computer a macchine di grosse dimensioni) e alcuni prodotti su personal computer come Omnis, disponibile sia per gli ambienti MS-DOS che Apple Macintosh. In questa classe vanno considerati anche un insieme di prodotti che vengono di solito chiamati generatori di applicazioni COBOL, che pur condividendo le proprietà comuni agli strumenti già citati, si distinguono per il fatto che sono in grado di generare, a partire da una descrizione della applicazione data in termini di menù, maschere e controlli procedurali, il codice COBOL corrispondente. Questo permette una più semplice integrazione tra i programmi sviluppati in modo tradizionale e quelli prodotti all'interno dell'ambiente della IV Generazione. Telon, Pacbase e Transform sono tipici esempi di questi prodotti, che di solito operano su mainframe.

Ambienti per "infocenter".
Nei grossi centri EDP diviene sempre più spesso necessario affiancare all'ambiente di produzione vero e proprio un ambiente che supporti l'utenza nella generazione veloce di applicazioni di supporto alle attività dell'azienda: un esempio tipico è costituito dalla generazione di report per l'alta dirigenza che riassumono l'andamento di alcune variabili relative allo stato dell'azienda. Questo tipo di applicazioni si differenzia da quelle che sono tipicamente realizzate con l'ambiente di produzione in quanto l) nascono da esigenze non sempre prevedibili a priori, 2) non hanno requisiti operativi stringenti (tempi di risposta, . . .), 3) devono poter accedere a diverse basi di dati a seconda della struttura organizzativa dell'azienda e delle tecnologie in essa utilizzate, 4) spesso devono effettuare complesse elaborazioni statistiche e produrre risultati in forma grafica di alta qualità (queste esigenze vengono spesso catalogate sotto la voce "infocenter"). Per questi motivi sono stati sviluppati una serie di ambienti, tra i quali i famosi Ramis, Focus e Nomad, che vanno incontro alle esigenze prospettate, integrando e complementando le funzionalità offerte dall'ambiente di produzione. Anche questi prodotti vengono per lo più utilizzati su grossi mainframe.

Ambienti per uso personale.
Con l'enorme diffusione dei PC sono stati sviluppati un notevole numero di strumenti per l'uso personale, che si pongono cioè l'obiettivo di supportare l'utente finale, spesso non esperto, nella gestione ed elaborazione diretta delle informazioni relative alla propria attività professionale o, più in generale, di suo interesse personale.

Gli ambienti di questa classe offrono interfacce utente estremamente sofisticate e la possibilità di integrare facilmente le informazioni della propria base di dati con quelle prodotte da altri strumenti quali fogli elettronici ed elaboratori di testi. Chiaramente questi strumenti mal si prestano, almeno nella generalità dei casi, ad attività di sviluppo di applicazioni complesse o con forti requisiti in termini di tempi di risposta e di dimensione della base di dati applicativa.

In merito agli ambienti per uso personale va fatta un'ulteriore osservazione: in alcuni casi, i prodotti su personal computer presentano funzionalità molto ricche che ne permettono un uso sia di tipo personale che per la produzione di applicazioni. Paradox, prodotto dalla Ansa Software/Borland, dispone di una interfaccia utente molto sofisticata che consente all'utente non esperto di gestire le proprie basi di dati in modo molto semplice ed efficace. Allo stesso tempo, tuttavia, dispone di un ambiente di sviluppo (con un proprio linguaggio della IV Generazione) che consente la realizzazione di applicazioni di significativa complessità. Alcuni prodotti, come lo stesso Paradox, dispongono di generatori di applicazioni in grado di produrre il programma 4GL corrispondente alla descrizione che ne viene data in termini di menu, interazione con l'utente e operazioni di gestione dei dati.

4.3 Toolkits
Con toolkits si intendono gli ambienti costruiti per aggregazione di strumenti lascamente integrati tra loro: di norma essi includono editors, compilatori, interpreti, linkers, debuggers, strumenti di gestione delle configurazioni e delle versioni e altri componenti che coprono uno specifico compito all'interno del ciclo di vita (recentemente anche *Upper CASE* e strumenti di *Reverse Engineering)* .
I toolkits nascono come naturale evoluzione dei sistemi operativi standard, sono per loro natura indipendenti da un particolare linguaggio di programmazione e non impongono in alcun modo un controllo o indicazione su come le diverse attività del ciclo di vita devono essere condotte. Gli strumenti che costituiscono un toolkit interagiscono di norma attraverso la creazione e la modifica di files il cui formato è determinato da convenzioni molto semplici. Spesso si tratta di files di caratteri o comunque byte-stream (per esempio, tutti i files prodotti da editors, compilatori e linkers); in altri casi, esistono alcuni formati di interscambio e filtri di ingresso/uscita che consentono così di importare o esportare le informazioni attraverso i diversi tools dell'ambiente. Nella maggior parte dei casi questi files non contengono alcuna informazione semantica che permetta di evidenziare la struttura concettuale delle informazioni in essi contenute. Si consideri, per esempio, un ciclo *for* di un programma sorgente C: esso è di norma memorizzato attraverso la stringa di caratteri che lo costituisce. Ciascun tool che accede al file, per poter riconoscere l'informazione sulla semantica del ciclo *for,* deve necessariamente ripetere il processo di analisi sintattica del file sorgente. Queste considerazioni hanno come importante implicazione il fatto che l'integrazione tra i diversi tools risulta realizzabile solo ad un livello molto basso, in quanto le informazioni risultano scarsamente strutturate e con un contenuto semantico non direttamente intelligibile. Di fatto, è responsabilità dell'utente garanti-

re che le informazioni fluiscano correttamente attraverso i diversi strumenti dell'ambiente. I vantaggi derivanti dall'uso dei toolkits possono sostanzialmente essere riassunti nei seguenti punti:

1. E possibile estendere l'ambiente in modo molto semplice, aggiungendo nuovi strumenti che rispettino le semplici regole che determinano i formati dei files dell'ambiente.
2. Risulta abbastanza facile ritagliare uno specifico ambiente di lavoro, considerando un sottoinsieme degli strumenti forniti nel toolkit.

L'esempio classico di toolkit è rappresentato dal Programming WorkBench di Unix (PWB-si veda [DHM78]): in generale, è possibile considerare in questa classe tutti gli ambienti costruiti per aggregazioni successive di diversi prodotti.

4.4 Piattaforme per SEE

A partire dall'inizio degli anni ottanta vi è stata una crescente attenzione, specie nel mondo industriale, nei confronti delle problematiche legate al trasporto degli strumentí CASE su diverse architetture HW/SW e allo loro integrazione. La situazione tipica che si è venuta a creare in questi ultimi anni vede da un lato lo sviluppo di un grande numero di strumenti CASE e dall'altro l'affermarsi di diversi sistemi operativi (o addirittura, nel caso di Unix, di diverse versioni dello stesso sistema operativo). Di fatto, ciò porta ad una proliferazione di strumenti scarsamente integrati o integrabili tra loro e ad una crescente difficoltà nella gestione del processo di distribuzione e manutenzione di tali prodotti sulle molteplici architetture software presenti sul mercato. Si noti che il problema dell'integrazione presenta due diversi aspetti complementari:

1. E opportuno che gli strumenti di un ambiente abbiano la stessa interfaccia utente e, soprattutto, lo stesso modello logico di interazione.
2. Le informazioni che vengono scambiate dai diversi strumenti devono essere gestite in modo integrato ed omogeneo all'interno dell'intero ambiente. In particolare, è necessario disporre di concetti applicativi comuni a diversi tools: per esempio, un editor e un compilatore devono avere una "idea comune" di ciò che essi intendono per "programma sorgente". Inoltre, devono esistere componenti software che permettono di accedere e gestire in modo controllato tutte le informazioni prodotte da uno qualsiasi dei tools che costituiscono l'ambiente.

Per soddisfare le esigenze a cui si è fatto cenno, nel corso degli ultimi anni sono stati studiati, sviluppati e anche proposti come standard, alcuni ambienti che estendono le funzionalità del sistema operativo, fornendo servizi e librerie di procedure che consentono di progettare e costruire strumenti realmente integrati e portabili. Essi pertanto dovrebbero costituire uno strato di funzionalità sulle quali definire il vero e proprio SEE. In realtà, va sottolineato che a) i prodotti di questo tipo commercializzati sono a tutt'oggi molto pochi, e b) nello sviluppo di tali strumenti l'enfasi maggiore è stata spesso posta sul discorso portabilità, mentre il problema dell'integrazione non ha trovato ancora soluzioni o proposte di lavoro pienamente mature e praticabili da un punto di vista industriale. Lo sviluppo di questi ambienti, chiamati spesso piattaforme per lo sviluppo di SEE (*SEE platforms*), è stato portato avanti principalmente all'interno di progetti di ricerca internazionali o da grossi organismi interessati alla standardizzazione dei prodotti e degli ambienti di sviluppo (tipicamente il DOD, Department Of Defense del Governo degli Stati Uniti, la CEE, la NATO). Gli esempi più significativi di piattaforme sono il Portable Common Tool Environment - PCTE ([BGMT88]), sviluppato da un consorzio di società europee nell'ambito delprogramma di ricerca comunitario ESPRIT, e la proposta del DOD denominata CAIS - Common APSE Interface Set (dove APSE sta per Ada Programming Support Environment-[Obe88]).

Tra i prodotti commerciali è possibile ricordare la piattaforma denominata Atherton Software BackPlane che presenta funzionalità particolarmente interessanti ed evolute.

4.5 Ambienti integrati di progetto e sviluppo

Gli ambienti che ricadono in questa classe costituiscono di fatto i primi tentativi di costruzione di SEE centrati intorno al concetto di Dizionario Dati *(Data Dictionary)* integrato, la base di dati che contiene tutte le informazioni relative all'ambiente di sviluppo. In particolare, i produttori di strumenti ed ambienti di supporto allo sviluppo di applicazioni EDP stanno sempre più tentando di integrare strettamente il mondo degli Upper CASE con quello degli ambienti della IV generazione: da un lato si hanno infatti funzionalità di supporto alla definizione dei requisiti e alla specifica di progetto dell'applicazione e, dall'altro, un ambiente completo per lo sviluppo e la manutenzione dei programmi.
Nel settore delle applicazioni EDP, il mercato presenta strumenti CASE basati sostanzialmente su notazioni di tipo semiformale (DFD, ER Diagrams, . . .), che includono anche componenti per la simulazione dell'interfaccia utente (in genere editor di maschere video e simulatori di menu): tali prodotti sono basati su un Dizionario Dati che contiene tutte le informazioni che vengono create dal progettista durante l'interazione con lo strumento. Molti strumenti della IV generazione sono in grado di accedere a questo Dizionario Dati, estraendone le informazioni in esso memorizzate e riutilizzandole per lo sviluppo vero e proprio della applicazione: tuttavia, poichè a livello di specifica vengono utilizzate tecniche di tipo semiformale, la possibilità di riutilizzo automatico dei dati di progetto risulta nella sostanza abbastanza limitato. In molti casi, vengono generate in modo automatico le maschere di interazione video e gli scheletri dei programmi

(cioè gli alberi di chiamata dei sottoprogrammi) ricavati dagli diagrammi di struttura, che descrivono la scomposizione di un sistema in moduli. In altri casi è possibile avere un supporto automatico per la generazione dei modelli fisici delle basi di dati a partire dalle informazioni contenute nei diagrammi Entità-Relazione.

Uno degli ambienti di sviluppo di questo tipo che si sta oggi maggiormente affermando in molti centri EDP è costituito da uno strumento Upper CASE per applicazioni EDP integrato con un generatore di applicazioni COBOL: un esempio tipico è fornito dall'accoppiata Excelerator /Telon. Excelerator è uno degli strumenti Upper CASE più diffusi oggi sul mercato mentre Telon è, come abbiamo visto in precedenza, un generatore di applicazioni COBOL.

Nel mondo della ricerca, vi sono alcuni progetti particolarmente significativi che hanno come obiettivo la realizzazione di SEE integrati. Esempi tipici sono Arcadia e ESF (Eureka Software Factory).

5. Le ricerche nel settore

Da quanto visto in precedenza si evince chiaramente che esistono ancora molti aspetti del ciclo di vita del software non ancora supportati in modo adeguato: benchè vi sia stato un forte sviluppo del settore sia dal punto di vista teorico che da quello delle realizzazioni industriali, rimangono ancora molti i punti da affrontare e da risolvere in modo pienamente soddisfacente.

- In particolare, un primo problema estremamente critico riguarda l'integrazione degli strumenti che vengono a cooperare all'interno dell'ambiente di sviluppo: i problemi in quest'area riguardano da un lato l'interfaccia utente e dall'altro la condivisione delle informazioni prodotte ed elaborate nell'ambiente.
- Una seconda area di criticità è caratterizzata dal fatto che, se si vogliono rendere sempre più automatizzate o automatizzabili le diverse attività del ciclo di vita del software, risulta necessario definire ed adottare notazioni grafiche e linguaggi che abbiano una semantica definita formalmente: è solo in questo modo, infatti, che le informazioni fornite in un certo momento dall'utente al SEE possono essere in seguito rielaborate e riutilizzate da altri strumenti presenti nell'ambiente. Tipico è il caso delle specifiche, la maggior parte delle quali sono oggi prodotte utilizzando notazioni informali o semiformali: come è stato ampiamente discusso, ciò rende di fatto impossibile la realizzazione di strumenti (semi)automatici quali esecutori delle specifiche (di supporto alla prototipazione rapida) o generatori di codice.

Un'analisi di quelle che sono le attuali linee di tendenza della ricerca a livello mondiale mostra come questo problema sia di fatto all'attenzione di molti studiosi del settore. In particolare si possono evidenziare le seguenti linee di ricerca:

1. Linguaggi formali per la specifica del software.
2. Linguaggi formali per la definizione del processo di sviluppo e manutenzione del software.

Quest'ultimo argomento riscuote oggi una notevole attenzione da parte degli studiosi in quanto si ritiene che sostanziali miglioramenti nella qualità ed efficacia degli ambienti di sviluppo (SEE) siano ottenibili solo descrivendo in modo il più possibile preciso (o addirittura algoritmico) quella che è la sequenza di attività compiuta normalmente da chi sviluppa e, in generale, gestisce un'applicazione informatica. A questo proposito è stata creata l'espressione *Programmazione del processo* (Process Programming) ([Ost87]), che vuole appunto sottolineare la stretta analogia che esiste, almeno in linea di principio, tra gli approcci che devono essere seguiti nella descrizione dell'applicazione e del suo stesso processo di sviluppo. Esempi di progetti in cui questo argomento è al centro delle attività dei ricercatori sono il già citato progetto Arcadia ([TSY+88]), a cui partecipano alcune università ed enti di ricerca americani, e le attività svolte nell'ambito del progetto di ricerca finanziato in ambito europeo denominato ESF (Eureka Software Factory). Tali progetti si pongono come obiettivo la realizzazione di un ambiente completo di supporto allo sviluppo di SEE, che vede nel process language uno degli argomenti di ricerca di maggior interesse.

Riferimenti bibliografici

[AG83] A.J. Albrecht e J.E. Gaffney. Software function, source lines of codes and development effort prediction: a software science validation. *IEEE Transactions on Software Engineering,* 9(11):639-648, Novembre 1983.

[Ber87] P.A. Bernstein. Database system support for software engineering - an extended abstract. In *Proceedings of the Ninth International Con.ference on Software Engineering,* pagine 166-178. IEEE, 1987.

[BGMT88] G. Boudier, F. Gallo, R. Minot, e I. Thomas. An overview of PCTE and PCTE+. In *Proceedings of the Third ACM SIGSOFT/SIGPLAN Symposium on Software Development Environments*, pagine 248-257. ACM, 1988.

[BOE81] B.W. Boehm. *Software Engineering Economics.* Prentice Hall, 1981.

[CHI88] a cura di E.J. Chikofsky. *Software Development Environments. Computer-Aided Sofware Engineering (CASE).* IEEE Computer Society Press, 1988.

[CON87] J. Conklin. Hypertext: an introduction and survey. *IEEE Computer,* 20(9):17-41, Settembre 1987.

[DEFH87] S.A. Dart, R.J. Ellison, P.H. Feiler, e A.N. Habermann. Software development environments. *IEEE Computer*, 20(11):18-28, Novembre 1987.

[DHM78] T.A. Dolotta, R.C. Haight, e J.R. Mashey. UNIX time-sharing system: the programmer's workbench. *Bell System Technical Journal,* 57(2), Luglio 1978.

[FG87] A. Fuggetta e C. Ghezzi. Strumenti della IV generazione. In *AICA Checkpoint.* AICA, 1987.

[FGM+90] A. Fuggetta, C. Ghezzi, S. Morasca, A. Morzenti, e M. Pezzè. *Ingegneria del Software.* Mondadori Informatica, 1990.

[HK88] S.E. Hudson e R. King. The Cactis project: database support for software environments. *IEEE Transactions on Software Engineering,* 14(6):709-719, Giugno 1988.

[MYE88] B.A. Myers. A taxonomy of window manager user interface. *IEEE Computer Graphics and Applications,* pagine 65-84, Settembre 1988.

[OBE88] P.A. Oberndorf. The Common Ada Programming Support Environment (APSE) Interface Set (CAIS). *IEEE Transactions on Software Engineering,* 14(6):742-748, Giugno 1988.

[OST87] L. Osterweil. Software processes are software too. *In Proceedings of the Ninth International Conference on Software Engineering,* pagine 2-13. IEEE, 1987.

[TSY+88] R.N. Taylor, R.W. Selby, M. Young, F.C. Belz, L.A. Clark, J.C. Wileden, L. Osterweil, e A.L. Wolf. Foundations of the Arcadia environment architecture. In *Proceedings of the Third ACM SIGSOFT/SIGPLAN Symposium on Software Development Environments,* pagine 1-13. ACM, 1988.

[WAT86] D.A. Waterman. A *guide to expert systems.* Addison-Wesley, 1986.

La protezione del software: a che punto siamo?

CARLO FALCETTI
Bull Italia
Direzione Brevetti e Licenze
Pregnana Milanese

1. Introduzione

Per le teorie economiche una risorsa di qualunque tipo, in grado di soddisfare dei bisogni, acquista rilevanza economica e diventa un bene in senso economico, con un valore di mercato, solo se è disponibile in quantità limitata rispetto alle necessità. Un prodotto software, una volta creato, è riproducibile per sua natura in quantità illimitata, a un costo irrisorio. Si verifica quindi una situazione paradossale per cui, in assenza di strumenti che rendano possibile controllare la quantità del prodotto sul mercato, il software non potrebbe assumere rilevanza economica. Infatti la domanda potrebbe essere soddisfatta a costo zero e qualunque investimento sostenuto per sviluppare nuovi prodotti di questo tipo sarebbe scoraggiato, se non esistesse a priori la certezza di un recupero dei costi o se non vi fossero strumenti che consentono di controllare la quantità del prodotto che viene immessa sul mercato e quindi di praticare politiche di commercializzazione del bene economico. È in questa ottica che viene ripresentato, con i dovuti aggiornamenti, il problema della protezione del software già affrontato su questa rivista, per gli aspetti giuridici, in un lontano 1974 e 1975 (Quaderni di Informatica, N. 2 e N. 3).

La protezione, di qualunque tipo essa sia, mira essenzialmente a controllare la quantità del bene disponibile sul mercato, in modo che questo sia suscettibile di apprezzamento e acquisti un valore economico. Per quanto non esclusivo del software, il problema della sua valorizzazione acquista una peculiarità e una rilevanza di gran lunga superiore a quella riscontrabile per altri tipi di beni. Valorizzare un bene di questo tipo significa trarne il massimo profitto utilizzando strumenti di commercializzazione idonei o inventandone di nuovi. Significa anche trovare degli strumenti appropriati di protezione del bene e se non esistono, cercare di modificare gli strumenti esistenti in modo da renderli idonei allo scopo o crearne dei nuovi. In questo senso l'esperienza del SW è davvero esemplare per la ricchezza di dibattito che si è sviluppata in merito alle forme di tutela di cui poteva beneficiare e per le conseguenze che si sono avute in termini di innovazioni legislative e corrispondentemente di politiche di sfruttamento commerciale.

2. La protezione

Assicurare la protezione di un bene è sempre un concetto relativo e presuppone l'identificazione degli eventi che si intendono evitare e la messa in atto di accorgimenti preventivi o repressivi. Chi è in possesso di un bene può semplicemente preoccuparsi che questo non gli venga sottratto (e il modo più semplice è tenerlo nascosto e non fare sapere agli altri che lo si possiede), ma può anche cercare di impedire che altri vengano in possesso dello stesso tipo di bene, in modo da assicurarsi una condizione di privilegio o di monopolio. Come posizione intermedia può anche preoccuparsi che lo stesso tipo di bene non diventi disponibile ad altri a condizioni economiche più vantaggiose, ossia a un minor costo. La società civile ha da tempo riconosciuto che certe forme di monopolio, non solo sono ammissibili ma sono utili alla comunità. Le legislazioni brevettuali e quelle relative al diritto di autore, e le relative convenzioni internazionali garantiscono un monopolio, entro certi limiti, rispettivamente alle invenzioni industriali e all'espressione formale di creazioni intellettuali che soddisfano a determinati requisiti. Naturalmente questo diritto di monopolio è riconosciuto a tutti e non a pochi privilegiati e comporta, secondo da che lato si guardi questa pattuizione o statuizione sociale, anche dei doveri.

Con la creazione di diritti personali di qualsiasi natura si instaura anche l'obbligo di rispettare i diritti personali degli altri. Da un lato abbiamo un bene per la cui acquisizione si è sostenuto un costo, e quindi un interesse a proteggerlo. Dall'altro abbiamo tutta una serie di eventi possibili messi in atto da chi non possiede il bene per acquisirlo a basso costo, atti lesivi dell'interesse del possessore del bene. Nel mezzo, come un muro di cinta, abbiamo delle possibili forme di protezione basate su norme di legge e su comportamenti del possessore del bene. L'insieme delle norme di legge definisce una

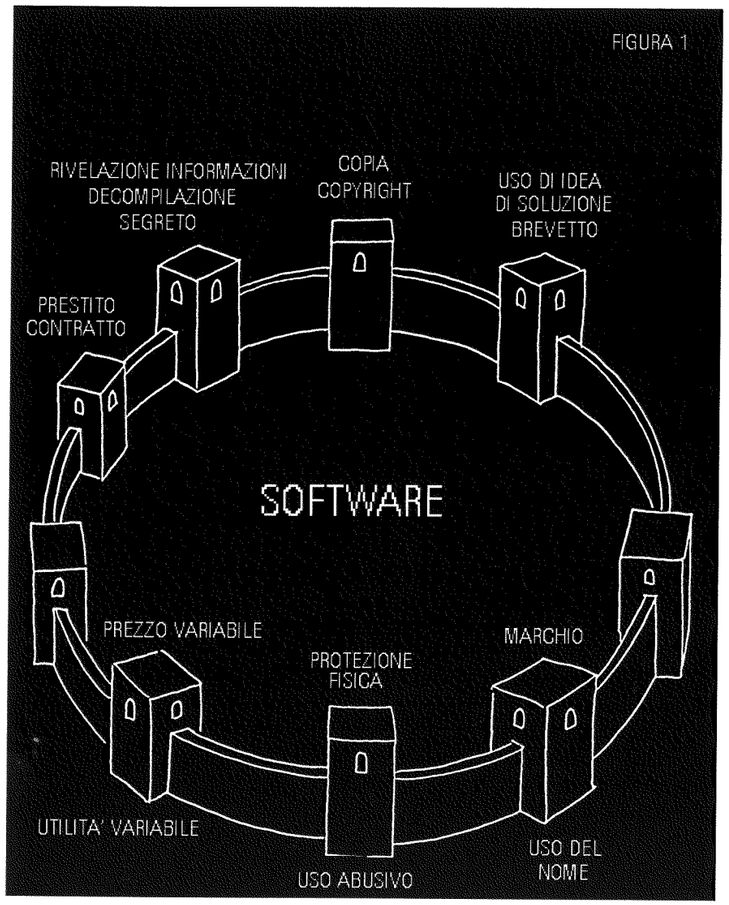


Fig. 1 - Una metafora grafica del problema della protezione del software.

sfera di protezione che identifica gli atti lesivi considerati abusi (in senso giuridico). Se gli abusi costituiscono un sottoinsieme degli atti lesivi, allora la sfera di protezione risulta piena di buchi che occorre tamponare, prevedendo forme nuove di protezione giuridica o comportamenti appropriati. (Figura 1).
Il problema può sembrare banalizzato ma queste considerazioni elementari, se tenute presenti, rendono abbastanza chiaro il quadro in cui il SW, al di là delle disquisizioni puramente intellettuali, ha trovato la sua identità come bene da proteggere e con quali forme, e in definitiva la sua miglior valorizzazione. Non si deve tuttavia pensare che il caso SW sia chiuso o quasi chiuso: con l'espressione SW ci si riferisce a un bene che non solo è estremamente diversificato, ma anche estremamente dinamico, soggetto a continua evoluzione ed estremamente pervasivo. La situazione attuale deve quindi essere considerata come un traguardo raggiunto ma non definitivo.

3. Una cosa di nome Software

Vale la pena di richiamare brevemente l'oggetto del discutere, ossia quella strana cosa che si chiama software.
Software può essere definito come l'insieme di informazioni astratte che concorrono alla gestione di un calcolatore elettronico e al suo impiego per la risoluzione di singoli problemi, oppure come insieme di procedure logiche e informazioni formalizzate in modo da poter essere comprese ed eseguite da un calcolatore elettronico. Anche se le classificazioni sono sempre convenzionali e discutibili, in relazione alle funzioni specifiche cui il software è destinato si usa fare la distinzione tra software applicativo e software di base. Il software applicativo è costituito da programmi di lavoro o programmi utenti, cioè dagli effettivi programmi che l'utente di un sistema di elaborazione dati ha interesse che siano eseguiti, quali il controllo di un processo, l'elaborazione degli stipendi del personale, l'aggiornamento dei registri delle scorte di magazzino, fatturazioni ecc. Il software di base è costituito da tutto quell'insieme di programmi, compilatori, assemblatori, supervisori che corredano uno specifico sistema di elaborazione dati e gli consentono di svolgere i programmi utenti. In particolare i programmi supervisori, o sistemi operativi provvedono ad organizzare il sistema di elaborazione, gli danno una fisionomia, lo gestiscono, intervengono in caso di errori e di malfunzionamenti. Mentre i programmi applicativi possono essere scritti ignorando, completamente o quasi, la reale struttura in cui dovranno essere svolti, con linguaggi di programmazione di tipo universale, quali FORTRAN,COBOL,PASCAL,BASIC, il SW di base e in particolare i programmi supervisori sono intimamente connessi all'hardware. Ogni tipo di sistema di elaborazione richiederà pertanto il suo proprio programma supervisore e più in generale ogni tipo di calcolatore avrà il suo software di base.
Una ulteriore classificazione che si può operare riguarda il linguaggio di descrizione del programma. Un programma può essere scritto in un linguaggio ad alto livello, non necessariamente un linguaggio standard, che essenzialmente descrive la pluralità di flussi logici eseguiti dal programma e la loro interazione. Il prodotto che ne risulta è definito come "sorgente". Ma per operare, questo programma deve essere convertito o tradotto, in blocco o istruzione per istruzione (compilatori e interpreti), in un altro programma, che risulta intelligibile alla macchina e usa il suo linguaggio, detto appunto linguaggio macchina.
Una ulteriore classificazione può essere fatta in funzione dei risultati finali conseguiti dal prodotto softuare. Si può quindi distinguere tra programmi industriali, per la gestione di processi industriali, programmi scientifici e statistici, per

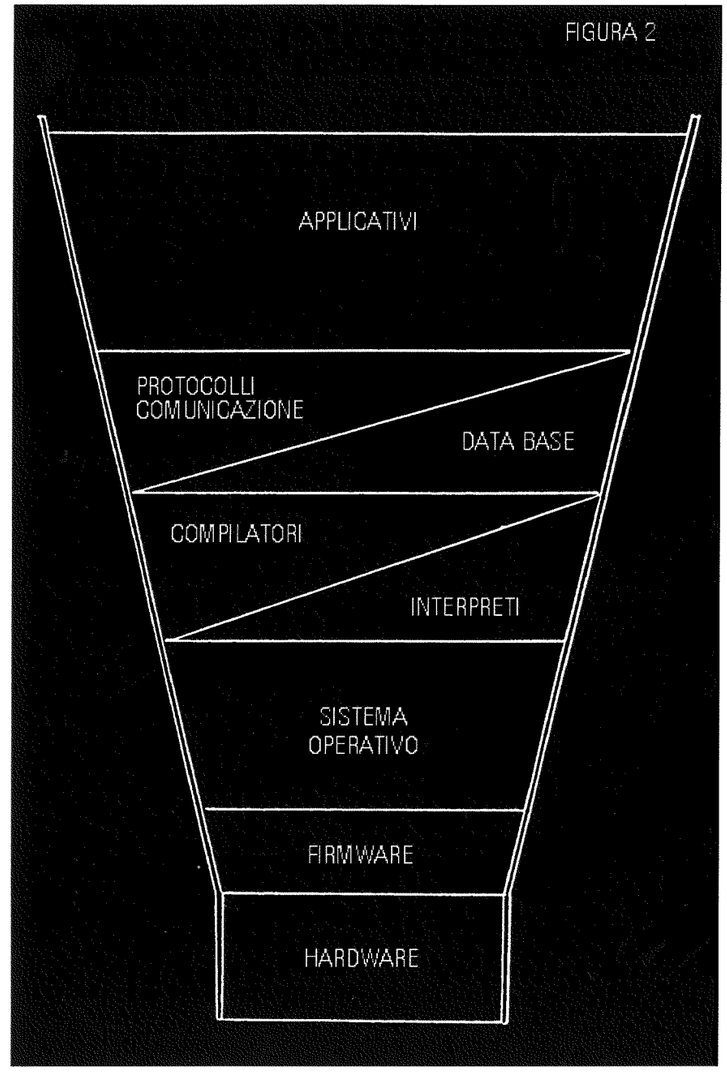


Fig. 2 - Il software come un imbuto che finisce nell'hardware.

l'esecuzione di calcoli, programmi gestionali e amministrativi, programmi per il trattamento di testi e di immagini.
Dalla semplice elaborazione di dati numerici, nel caso rappresentativi di variabili fisiche e dallo stretto ambito dei centri scientifici, degli impianti industriali e degli enti amministrativi, tipico dell'infanzia del SW, si è passati a funzionalità più ampie e diffuse capillarmente, nell'ambiente domestico, scolastico, di lavoro, nel campo del turismo, dei trasporti, dell'assistenza sanitaria, dell'agricoltura, dell'editoria, della diffusione e raccolta di informazioni.
Se in alcuni casi la presenza del SW è tangibile, nel senso che è facile identificare un supporto del programma in forma di disco o nastro magnetico, in altri casi il SW è presente in forma non immediatamente visibile, celato in una macchina, dispositivo o semplice oggetto, come una automobile, una lavatrice o una macchina da cucire o anche solo una carta di credito o scheda intelligente. Infatti dove c'è un elaboratore, non importa quanto piccolo, vi è sempre un minimo di funzioni di programma per controllarlo, e poco importa chiamarle Software o Firmware.
In questa varietà di beni, omnicompresa dalla dizione "Software" è evidente come il possessore, creatore, inventore del bene possa avere prevalenti interessi a una protezione di un certo tipo piuttosto che di altro.
A conclusione, la Fig. 2 riassume graficamente l'idea di software, come un imbuto che finisce nell'hardware.

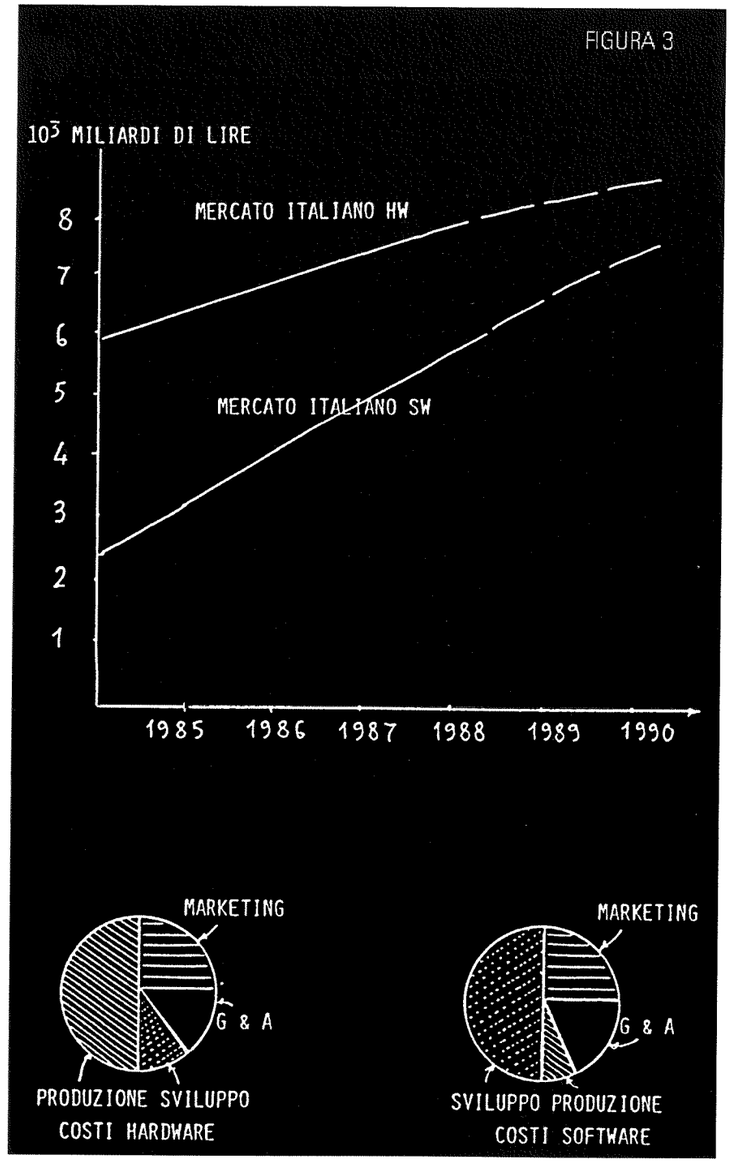
4. Necessità della protezione

Fin qui abbiamo considerato la protezione del SW come un naturale interesse del creatore di questo tipo di bene, strumentale alla sua valorizzazione e diversificato in funzione della sua specifica destinazione di impiego. Vi è tuttavia un aspetto generale da considerare. Il SW è un bene il cui costo di sviluppo e manutenzione è estremamente alto in rapporto alla semplicità e relativa non onerosità con cui può essere replicato. Mentre per un bene fisico prodotto in serie, ad esempio per l'HW, il costo di ogni esemplare del bene è costituito essenzialmente dai costi di produzione, con una incidenza dei costi di progetto sul costo di ogni esemplare che non supera il 10%, per il SW è vero il contrario, ossia il costo di replicazione è una frazione marginale del costo di sviluppo, anche se questo viene ripartito su una pluralità di copie. Inoltre l'operazione di replicazione non richiede in generale competenze, attrezzature o investimenti di particolare rilievo.
Per chi crea dei prodotti software, la sua protezione non è pertanto una questione di interesse ma una necessità.
In assenza di protezione non vi sarebbe nessun incentivo a investire nello sviluppo del bene SW, per disporne, se poi questo, una volta creato, risultasse disponibile agli altri a un costo irrisorio.
Questa necessità non riguarda un fenomeno sporadico e un numero limitato di persone, ma un mercato mondiale che ha raggiunto annualmente dimensioni dell'ordine del centinaio di miliardi di dollari.
Solo in Italia il mercato del SW è attualmente stimabile a più di 5.000 Miliardi di lire (v. Fig. 3), con più di 50.000 addetti, senza comprendere il SW "bundled" ossia quello che viene fornito come parte integrante di un prodotto HW, sia questo o meno un prodotto specificatamente informatico. Il fenomeno ha quindi rilevanza sociale e giustifica considerazione a livello politico e provvedimenti di natura legislativa se opportuni.
Un altro aspetto del SW è la sua universalità e trasferibilità. La maggior parte dei prodotti SW è diffusa e utilizzata a livello internazionale e facilmente trasferibile da un paese all'altro vuoi su un supporto fisico, vuoi attraverso sistemi di telecomunicazione, via cavo o etere.
L'esigenza di protezione non è quindi limitata territorialmente ma sentita a livello internazionale e richiede forme di protezione armoniche e coerenti a livello internazionale.

5. La brevettabilità del software

La più estesa forma di protezione accordata dalle diverse legislazioni nazionali alle opere dell'ingegno è costituita dal brevetto per invenzione. Il brevetto per invenzione è un diritto di monopolio limitato nel tempo, che la legge riconosce ad alcune categorie di opere creative e precisamente alle invenzioni industriali. Diritto di monopolio, anzitutto nel senso di "ius escludendi alios" ossia diritto di impedire agli altri di mettere in atto in qualunque forma l'oggetto del brevetto. In secondo luogo, per metterne in evidenza la forza e la portata, diritto di sostanza e non di forma, nel senso che tutto quanto ricade nell'ampia idea di soluzione rivendicata nel brevetto, indipendentemente dai diversi modi in cui può essere realizzata, rientra in questo diritto di monopolio. Cosa vi è di meglio per chi abbia realizzato un programma, che consente di controllare una macchina, un processo industriale o di gestire la contabilità di un'azienda, che riuscire a impedire agli altri di effettuare le stesse cose e beneficiare di questa situazione di monopolio?
Tentativi in questo-senso si sono avuti fin dagli anni '60 e hanno aperto la lungo dibattuta questione se il SW sia brevettabile o meno. Non vale qui la pena di esaminare i diversi casi giudiziari discussi dalle diverse corti competenti in diversi paesi.
Francamente non potrei dare una lista completa nè discuterla perchè da parecchio tempo la questione ha perso di ogni importanza per me, come credo per molti.
Essenzialmente la questione da risolvere è se un programma sia una successione logica di operazioni eseguite su informazioni che dà come risultato altre informazioni, quindi un algoritmo, una serie di passi logici che possono essere eseguiti anche mentalmente o qualcosa di diverso. Se si conclude che un programma è un puro processo logico se ne esclude la brevettua-

Fig. 3 - Il mercato italiano del software (in alto) e la diversa ripartizione dei costi dei prodotti software rispetto ai prodotti hardware (in basso).



lità, perchè in modo implicito o esplicito, la legislazione brevettuale dei diversi paesi esclude dalla brevettabilità le formule matematiche, i principi e le scoperte scientifiche, i metodi di conduzione aziendale, le regole di gioco. Di fatto un programma può essere qualcosa di più: può essere il modello logico del comportamento di una macchina o dell'evoluzione di un processo industriale. Diventa per questo brevettabile? La risposta da dare mi sembra logicamente un no.
Brevettabile non è il modello logico, ma la macchina o il processo descritto dal modello logico. Se il programma è il modello logico di comportamento di un calcolatore e si ottiene un brevetto per un calcolatore che si comporta nel modo descritto dal suo modello logico non si finisce col riconoscere la brevettabilità del programma?
Inizialmente (mi riferisco agli anni '60) la risposta data a questo interrogativo è stata discordante da paese a paese: in Francia e nella Repubblica Federale Tedesca è prevalsa la tesi che il SW non è in alcun caso brevettabile e nel 1968 la Francia ha modificato la sua legge brevettuale per sancirne espressamente la non brevettabilità. In Gran Bretagna è stata riconosciuta

la brevettabilità del SW come modo nuovo e originale di operare di un calcolatore. Negli Stati Uniti il Patent Office ha reso ufficiale nel 1968 delle direttive per l'esame di domande di brevetto relative a programmi che in sostanza affermano la non brevettabilità dei programmi in quanto tali e la loro brevettabilità in quanto processi che producono effetti di natura materiale o che presuppongono nuovi apparati per la loro esecuzione.

Nel corso del tempo l'interpretazione dei programmi come qualcosa che produce effetti di natura materiale perchè comunque, indipendentemente dalla loro finalità ultima, agiscono in un calcolatore e possono determinarne una migliore utilizzazione, si è progressivamente estesa. Anche in Giappone l'Ufficio Brevetti ha pubblicato nel 1976 degli standard di esame per invenzioni relative a programmi di calcolatore, abbastanza singolari nella loro impostazione. In sintesi si riconosce la brevettabilità di un programma se il suo nucleo inventivo può essere ricondotto all'uso di una legge naturale. Cosi' se un programma è sviluppato per risolvere un problema di scacchi, o per definire quale è la più redditiva o meno rischiosa forma di investimento di un capitale, il programma non può essere ricondotto allo sfruttamento di una legge naturale ma di regole matematiche di gioco, o di previsioni economiche e di "leggi" che in qualche modo descrivono i fenomeni economici. Viceversa se il programma è usato per calcolare la profondità e consistenza di un giacimento di petrolio attraverso l'analisi di segnali geofisici, che obbediscono a leggi naturali di propagazione, il programma può essere brevettabile.

Il quadro internazionale si è infine orientato, verso la fine degli anni '70 e i primi anni '80, con l'entrata in vigore della Convenzione sul Brevetto Europeo, verso il diniego della brevettabilità del SW in quanto tale, ma il riconoscimento come brevettabili di processi e apparecchiature, anche se tra i loro elementi caratterizzanti figura un programma.

6. La convenzione per il brevetto europeo

La Convenzione europea dei brevetti, entrata in vigore il 7 Ottobre 1977 e a cui hanno ormai aderito quasi tutti i paesi europei, prevede esplicitamente (Art. 52.1C) che i programmi di calcolatori siano esclusi dalla sfera delle invenzioni brevettuali: L'esclusione riguarda però (Art. 52(3)) i programmi in quanto tali.
E nella formulazione della convenzione e nella sua ratifica è quindi prevalsa la tesi che i programmi di calcolatore in sè sono una lista di operazioni logiche registrate su un supporto e quindi non brevettabili. Si è però riconosciuto fin dall'inizio, che i programmi quando eseguiti su un calcolatore potevano dare dei risultati industriali nello svolgimento di processi e di metodi. Si trattava quindi di definire in che modo e in quale misura i programmi potevano rientrare tra le invenzioni brevettabili come componenti essenziali o strumenti per lo svolgimento di processi, o metodi brevettabili.

Le direttive originali per l'esame delle domande di brevetto europee non fornivano alcun aiuto sostanziale per identificare il confine tra invenzione brevettabile e SW non brevettabile, limitandosi a indicare che una invenzione rivendicata in termini di calcolatore provvisto di un certo programma o di un procedimento per far funzionare un calcolatore non acquisiva per questo il requisito di contributo tecnico allo stato dell'arte. Per questo, dopo aver dibattuto a lungo il problema, sono state pubblicate nel 1985 nuove e più puntuali direttive:

1. L'invenzione rivendicata deve portare a un contributo di natura tecnica, indipendentemente dalla forma usata nelle rivendicazioni,

2. La natura tecnica del contributo deve essere valutata indipendentemente da altri requisiti, in particolare quelli dell'industrialità (suscettibilità di applicazione industriale) della novità ed originalità.

3. Con riferimento specifico ai programmi si dà per scontato che macchine controllate da programmi e processi di produzione e regolazione controllati da programmi sono brevettabili e che il modo di operare interno di un calcolatore, anche se il calcolatore è noto in se, può essere brevettabile se produce un effetto tecnico (ossia migliora le prestazioni o aumenta le funzionalità del calcolatore).

Come queste direttive di esame siano utillizzate e fino a che punto l'Ufficio Europeo dei Brevetti si sia spinto nel riconoscere la brevettabilità del SW è illustrato da alcune decisioni delle camere di ricorso EPO pubblicate recentissimamente nel Giornale Ufficiale EPO. Una domanda di brevetto, della IBM, relativa al coordinamento e al controllo delle comunicazioni interne tra programmi in una pluralità di processori interconnessi mediante una rete di telecomunicazione, quindi relativa tipicamente a funzionalità di sistema operativo o supervisore, è stata riconosciuta come relativa a una invenzione che risolve un problema tecnico e quindi brevettabile, anche se non richiede alcun particolare calcolatore per operare.
Anche per un'altra domanda di brevetto relativa a un metodo per visualizzare uno tra più messaggi comprendenti una frase costituita da più parole e indicativa di eventi occorsi in un dispositivo di ingresso/uscita in un sistema di elaborazione di testi, il metodo comprendendo una pluralità di operazioni logiche (ricezione di un segnale da dispositivo, richiamo di un programma di generazione di messaggi, che attraverso opportune tavole costruisce per passi il messaggio, lo carica in un buffer e una volta completato lo visualizza), la camera di ricorso EPO ha concluso che fornire una indicazione visiva di eventi occorsi in un apparato è essenzialmente un problema tecnico, che scopo dell'invenzione non era un modo particolare di presentazione di informazioni, ma la generazione di queste informazioni, che la generazione mediante un programma e tabelle di memoria era incidentale alla soluzione del problema e che quindi la brevettabilità era ammissibile se erano verificati gli altri requisiti di brevettabilità.
In questa decisione, l'unico aggancio dell'invenzione con un problema tecnico risolto si trova nel fatto che le informazioni visualizzate riguardano uno stato di una macchina e sono generate da una macchina. Viceversa nel caso di una domanda di brevetto per un metodo che permette di riassumere automaticamente un documento, di memorizzare il riassunto e di ritrovarlo interrogando un calcolatore, si è ritenuto che il metodo (eseguito da programma) non avesse alcuna rilevanza tecnica e si basasse su criteri puramente intellettuali per produrre o ritrovare informazioni puramente intellettuali.
I limiti entro cui la brevettazione del SW è ammissibile sono quindi sufficientemente se non rigorosamente delineati, tenendo presente che le decisioni dell'Ufficio europeo dei brevetti fanno in buona misura "testo". Vorrei solo aggiungere che probabilmente, nel caso precedentemente citato di metodo per la generazione automatica di riassunti e di ricerca di documenti, se si fosse portata evidenza di un più efficiente impiego delle risorse HW, l'Ufficio Europeo dei Brevetti avrebbe preso una decisione diversa, favorevole alla brevettazione.

7. I limiti del brevetto

Si può quindi concludere che virtualmente tutto il software è brevettabile, non in sè, ma in quanto metodo o procedimento secondo cui un calcolatore è fatto funzionare.
Tutto questo è interessante per capire fino a che punto lo strumento brevettuale può essere utilizzato come tutela, ma risulta abbastanza accademico per l'interesse prevalente dei produttori di SW.
Risolto il problema se il SW rientri nella categoria delle opere dell'ingegno che sono brevettali, occorre verificare, caso per caso, se il particolare prodotto software possiede gli altri requisiti per la brevettabilità ossia la novità e l'originalità.
La verifica di sussistenza di questi requisiti è possibile anche se difficile e onerosa (e questa difficoltà è stata invocata come pretesto per sostenere la tesi di non brevettabilità) e porterebbe a concludere, come nel caso dei prodotti HW, che solo un numero relativamente modesto di prodotti SW, per aspetti limitati e facilmente aggirabili, possiede questi requisiti. Viceversa il prodotto SW richiede una qualche forma di tutela indipendentemente dal suo contenuto inventivo.
Da questo punto di vista, l'istituto del brevetto industriale, se può essere convenientemente utilizzato in alcuni casi, non si presenta come la più opportuna forma di protezione perchè non porta in maniera generale alla tutela desiderata.
Se confrontiamo il numero di domande di brevetto che riguardano programmi di calcolatore e il numero di domande di brevetto che riguardano circuiti elettronici, architetture di sistemi, tecnologie dei circuiti, dobbiamo concludere, tenuto conto della distribuzione degli investimenti di ricerca e sviluppo, nelle due aree HW, SW (nell'area del SW eguagliano, se non superano quelli dell'area HW) che, di fatto, poco peso è dato alla protezione del SW con lo strumento del brevetto.
Le ragioni principali di ciò sono, secondo me, le seguenti.
Anzitutto la verifica di sussistenza dei requisiti di novità e originalità rende incerto il successo nell'ottenimento della tutela brevettuale.
Inoltre le procedure di brevettazione sono procedure piuttosto onerose e comportano la descrizione dell'invenzione in modo tale che un esperto del ramo possa mettere in atto l'invenzione senza difficoltà, quindi la sua pubblicazione.
Equivalgono a rivelare, senza un onere sostanziale per il pubblico, seppure con un ritardo di 18 mesi dal deposito, informazioni che altrimenti si potrebbero ottenere solo con un lungo e costoso lavoro di reverse engineering.
Poichè la protezione accordata dal brevetto riguarda il metodo o processo eseguito dal programma o una apparecchiatura fatta funzionare in un certo modo, la violazione del brevetto avverrebbe solo nel caso di uso del programma.
In tutti quei paesi in cui non vige il principio del "Contributory infringement" si potrebbe avere una commercializzazione di copie del programma che non è perseguibile essendo perseguibile solo l'utilizzatore del programma.
Le procedure di brevettazione oltre che onerose sono lunghe e incerte e i pieni diritti conferiti col brevetto sono esercitabili solo con la concessione del brevetto, quando ormai può essere troppo tardi per recuperare delle posizioni commerciali danneggiate.
Chiunque produce SW è consapevole che se può ottenere col brevetto una posizione di esclusiva per i suoi prodotti, corre anche il rischio di vedersi preclusa la possibilità di operare liberamente sul mercato per effetto dei brevetti dei concorrenti. All'onere di brevettazione dei propri prodotti si aggiunge, e non è affatto irrilevante, l'onere di verificare che i propri prodotti non violino diritti brevettuali di terzi.
Meglio quindi non spingere nell'uso dello strumento brevettuale, favorendo fenomeni di emulazione che potrebbero rendere difficile e pericoloso operare nel campo del software.
Meglio invece cercare di utilizzare forme di tutela meno monopolistiche ma più efficaci nella repressione di comportamenti lesivi. L'interesse prevalente e generale di un produttore di SW non è quello di ottenere una esclusiva e di impedire agli altri di realizzare lo stesso tipo di prodotto, con gli stessi costi e magari con un ritardo di qualche anno, ma quello di evitare che la concorrenza o i potenziali utenti possano copiare a costo zero o quasi, in tempo zero o quasi, il prodotto. Come si può constatare, si tratta di un tipo di protezione che si scosta alquanto da quella derivante da un titolo di brevetto e molto più vicina a quella che è riconosciuta dalla legge sul diritto di autore.
Il che porta a considerare il diritto d'autore o altre forme di tutela come più opportune.

8. Software e diritto di autore

Per chiarire cosa sia il diritto di autore, o "copyright" secondo la terminologia anglosassone, non v'è nulla di meglio che rifarsi ai primi articoli, relativi a tale diritto, del nostro Codice Civile.
"Art. 2575 - Oggetto del diritto - Formano oggetto del diritto di autore le opere dell'ingegno di carattere creativo che appartengono alle scienze, alla letteratura, alla musica, alle arti figurative, alla architettura, al teatro e alla cinematografia, qualunque ne sia il modo e la forma di espressione.
Art. 2576 - Acquisto del diritto - Il titolo originario dell'acquisto del diritto di autore è costituito dalla creazione dell'opera, quale particolare espressione del lavoro intellettuale.
Art. 2577 - Contenuto del diritto - L'autore ha il diritto esclusivo di pubblicare l'opera e di utilizzarla economicamente in ogni forma e modo, nei limiti e per gli effetti fissati dalla legge.
L'autore, anche dopo la cessione dei diritti previsti dal comma precedente, può rivendicare la paternità dell'opera e può opporsi a qualsiasi deformazione, mutilazione o altra modificazione dell'opera stessa, che possa essere di pregiudizio al suo onore e alla sua reputazione".
La citazione che precede, limitata ai soli articoli che interessano al nostro scopo, per quanto si riferisca al Codice Civile italiano, può considerarsi rappresentativa delle fondamentali disposizioni di legge che, in quasi tutti i paesi del mondo, tutelano i diritti di autore e mette in evidenza che, anche se storicamente il diritto di autore è nato come privilegio o come diritto di monopolio di natura patrimoniale connesso agli editori e stampatori, esso, nelle legislazioni moderne, si è esteso a coprire le piu' svariate categorie di opere intellettuali e a tutelare non solo i diritti, diciamo così patrimoniali o di sfruttamento economico dell'opera (in ogni forma e modo) riconosciuti all'autore (o al suo avente causa), ma anche diritti morali o della personalità dell'autore (rivendicazioni di paternità, opposizione a deformazioni pregiudiziali).
Vista così l'ampia portata del diritto di autore, che si estende ben al di là del campo della stampa e della letteratura, è stato immediato pensare che in qualche modo il "software", o almeno parte di questo, potesse godere di tutela nell'ambito di tale diritto.
Si è quindi esaminato se esso possiede i requisiti richiesti. È principio stabilito della giurisprudenza che il requisito essenziale consiste nel carattere creativo intellettuale individualizzante dell'opera, per cui l'opera si distingue da quelle che l'hanno preceduta per un "quid novi" apportato in modo inconfondibile dall'autore (o dagli autori).
In un'opera dell'ingegno si suole identificare tre elementi costitutivi: l) il soggetto o argomento 2) il contenuto psichico o ideologico o forma interna, ossia la particolare concezione che del soggetto ha l'autore, 3) la forma di espressione o forma esterna.
Il "quid novi" apportato dall'autore è sempre, necessariamente ma non esclusivamente di carattere formale: non c'è originalità se non c'è forma espressiva o forma esterna nuova, in quanto la forma esterna è il veicolo con cui si manifesta il soggetto ed anche il contenuto psichico.
Un secondo requisito, generalmente invocato per definire l'appartenenza di un'opera a quelle dell'ingegno protette dal diritto di autore, è la sua destinazione a una comunicazione con il pubblico in funzione di rappresentazione intellettuale.
In altre parole, mentre una macchina, ancorchè brevettata è risultato di una attività inventiva, ha per destinazione non di essere conosciuta, ma di essere utilizzata praticamente, un'opera dell'ingegno, protetta dal diritto di autore è destinata ad essere conosciuta e utilizzata dal pubblico a livello intellettivo.
Così un libro è destinato ad essere letto e convoglia con questo un certo patrimonio informativo; un dipinto o una scultura sono destinati a essere visti e convogliano con questo un certo appagamento estetico, così come una composizione musicale, che è destinata ad essere ascoltata.
Perchè ciò sia possibile, occorre che la forma espressiva sia in qualche modo intelligibile. Per quanto riguarda il primo punto (carattere creativo individualizzante) non si è mai posto in discussione che il software possa godere di questo requisito essenziale. Anche se un programma usa un linguaggio convenzionale, regole strutturali e sintattiche di quel linguaggio ed è condizionato nel suo intreccio alla funzione che deve svolgere, senza concessioni a finalità o esigenze estetiche o culturali, l'aspetto di creazione intellettuale, in forma di impegno espressivo di fenomeni, di problemi, e procedure di risoluzione, può certamente sussistere.
Per quanto riguarda il secondo punto (destinazione a comunicazione con il pubblico) si sono sollevati in passato due tipi di critiche. In primo luogo che un programma, nelle forme in cui è normalmente distribuito e utilizzato non è affatto intelliggibile, perchè usa codici e linguaggi convenzionali e perchè il mezzo fisico di supporto non si presta in generale a una comunicazione intelligibile. In secondo luogo, si sostiene, un programma non è destinato ad essere inteso o goduto intellettualmente, ma solo ad essere utilizzato. Sono ambedue argomentazioni abbastanza oziose in quanto l'uso di linguaggi convenzionali nella espressione di un programma non preclude la comprensione del programma a chi conosce tali linguaggi convenzionali, l'uso di supporti fisici non direttamente leggibili non preclude la manifestazione della forma espressiva del programma attraverso strumenti e la destinazione utilitaristica non preclude una funzione diversa quale la comunicazione di contenuti informativi a chiunque sia interessato a ricevere questi contenuti.
Diverse sentenze in Italia e all'estero hanno infatti sancito l'applicabilità del diritto d'autore al SW precisando nel tempo che tale diritto è riconosciuto non solo ai video giochi, nei quali oltre che una destinazione alla comunicazione con il pubblico si può anche riconoscere una qualche finalità estetica, non solo ai programmi che interagiscono con l'utente per mezzo di maschere, ma a qualunque programma, indipendentemente dal fatto che nel corso del suo uso si realizzi una qualsiasi comunicazione intellettuale con l'utente.
Il principio è ormai così scontato che ci si può chiedere come mai in passato si siano ritenute opportune azioni legislative nei diversi paesi per inquadrare espressamente il SW tra le opere tutelate dal diritto di autore, e come mai iniziative legislative in questo senso siano ancora in corso.

9. Portata e limiti della protezione offerta dal diritto di autore

Il problema di fondo non è tanto se il software rientra tra le opere dell'ingegno protette dal diritto di autore e se è necessaria una menzione esplicita di questo fatto, ma se la protezione che ne deriva è adeguata. A differenza del monopolio brevettuale, che impone un divieto assoluto di utilizzazione dell'invenzione protetta, il diritto di autore impone un divieto di fare delle copie dell'opera protetta o delle traduzioni o delle elaborazioni poste in commercio. Non ne impedisce il prestito, a titolo gratuito od oneroso, la cessione a terzi, non ne limita il godimento e soprattutto non preclude l'impiego del contenuto informativo. Le conseguenze di questo fatto sono molteplici. Si può ipotizzare la creazione autonoma di due programmi identici, o funzionalmente equivalenti, senza violazione del diritto di autore ma con evidente conflitto di interessi sul mercato.
Si può ipotizzare lo studio e l'analisi di un programma o con espressione corrente il "reverse engineering" per la ricreazione di un programma funzionalmente equivalente.
Il contenuto informativo non è infatti tutelato ma diventa, una volta che il prodotto SW sia reso disponibile al pubblico, di patrimonio pubblico.
Inoltre, una volta che il supporto contenente il pogramma è stato venduto non si può, sulla base del solo diritto di autore, imporre alcuna limitazione a un suo uso che non implichi l'esecuzione di copie.
Oltre a tutto l'effettuazione di copie per uso personale e con mezzi non idonei alla distribuzione costituisce un lecito previsto. È sufficiente questa protezione oppure occorre qualcosa di più? La risposta è scontata.
Il diritto di autore offre una ragionevole tutela a costo nullo. Le convenzioni internazionali garantiscono questa tutela estensivamente a livello mondiale, senza necessità di adempiere ad alcuna formalità.
Anche la pur minima formalità richiesta in USA per beneficiare del "Copyright" ossia l'apposizione obbligatoria della annotazione di riserva su ogni copia dell'opera tutelata è diventata accessoria e non mandatoria con la ratifica della convenzione di Berna da parte degli Stati Uniti. Tuttavia la tutela offerta dal diritto di autore non è perfetta.
Scopo essenziale delle riforme legislative avvenute in molti paesi e delle iniziative legislative in corso non è solo quello di sciogliere ogni dubbio sulla applicabilità del diritto di autore al SW, ammesso che ciò sia ancora necessario, ma piuttosto quello di perfezionare l'estensione della tutela.
Questa strada è apparsa come la più facilmente percorribile e suscettibile di risultati immediati in alternativa alle molteplici proposte, che sono state avanzate e poi abbandonate, per l'istituzione di strumenti di protezione specifici.

10. La riforma del diritto di autore

Ritengo che la prima legge sul diritto di autore che è stata modificata per tener conto delle specifiche esigenze del SW sia quella degli Stati Uniti (12 Dicembre 1980).
È stato definito cosa si intendeva per programma di calcolatore ossia "un insieme di dichiarazioni o istruzioni da usare direttamente o indirettamente, in un calcolatore per ottenere un certo risultato".
I programmi di calcolatori non sono stati direttamente enunciati come categoria di opere protette, ma indirettamente con l'introduzione di una sezione (SS117) ad essi dedicata si è riconosciuto in maniera esplicita che ad essi si applica il "Copyright".
Per quanto riguarda il diritto riconosciuto, questo è stato oggetto di una precisazione che, seppure espressa in termini di limitazione, è di fatto una espansione della tutela.

- Si è riconosciuto che il legittimo possessore di una copia di un programma per calcolatore può effettuare o far effettuare un'altra copia del programma
- Se questa copia o adattamento è un atto essenziale nell'uso del programma in un calcolatore e non è usata in altro modo oppure
- Se la copia o adattamento è per soli scopi di archivio e se viene distrutta nel caso che il possesso del programma cessi di essere legittimo.

Vi sono molte implicazioni in questo linguaggio.
Anzitutto viene riconosciuto che una registrazione fatta nella memoria di un calcolatore è una "copia" nel senso del diritto di autore ossia è fissata su un supporto tangibile (celle di memoria) in maniera sufficientemente "permanente" per poter essere percepita (con opportuni mezzi) riprodotta o comunicata per un periodo superiore a una durata transitoria.
Tale copia è permessa solo nei limiti necessari per l'uso del programma e non per altri.
Questa può considerarsi una conquista fondamentale, per la tutela del SW, anche se espressioni come "uso" e "durata" transitoria" possono risultare equivoche.
Se l'uso del programma presuppone una operazione di replicazione o adattamento e questa è ammessa solo in caso di possesso legittimo di una copia, qualunque altro uso della copia in possesso legittimo o qualunque uso di una copia illegittima costituisce una violazione del diritto di autore.
Il concetto di possesso legittimo, distinto dal semplice possesso è

abbastanza chiaro in giurisprudenza e distingue quella fattispecie in cui il bene di cui si acquisisce il possesso, ossia la disponibilità, è messo a disposizione a condizioni definite entro limiti determinati, da chi ha la proprietà del bene.

Conclusione: se il possesso legittimo è ottenuto con un atto di compra-vendita del supporto di programma e senza limitazioni espresse l'unico uso consentito è quello con un calcolatore e non in altro modo.

La legge sul copyright non pone restrizioni sulla "intensità" dell'uso, sul "luogo" dell'uso, sul "calcolatore" in cui si realizza l'uso, sul "trasferimento" del possesso, sulle "modalità" dell'uso, sul "beneficiario" dell'uso, ma configurando un atto (caricamento in memoria di calcolatore) come copia, crea la premessa per esercitare dei diritti assoluti che rendono possibili e giuridicamente tutelabili e controllabili unilateralmente nell'ambito del copyright limitazioni di questo tipo.

Il rapporto contrattuale che legittima il possesso può definire dei limiti e delle regole nell'uso del bene anche più restrittive di quelle previste dal copyright.

Ancora più restrittiva è la revisione del diritto d'autore, entrata in vigore in Francia nel 1985.

Oltre a definire che il "logiciel" rientra nelle categorie delle opere protette, stabilisce (Art. 47 - Titolo V della legge 85-660 del 3 luglio 1985) che "qualsiasi riproduzione diversa dalla preparazione di una copia di salvaguardia da parte dell'utilizzatore, così come qualsiasi uso non espressamente autorizzato dall'autore o dal suo avente diritto è passibile di sanzione cioè è illegale).

Per radicalizzare il concetto si può dire che per la legge francese il possesso di una copia di programma, legittimato da un atto di compravendita non è sufficiente a legittimare l'uso ma il contratto di compravendita deve essere accompagnato da una autorizzazione unilaterale all'uso o da un contratto di licenza.

11. La direttiva CEE per la protezione del software

Anche se la maggior parte dei paesi europei, e dei paesi extraeuropei più industrializzati riconosce in modo esplicito, per effetto di aggiornamenti legislativi, che il SW rientra tra le opere tutelate dal diritto di autore e gli assicura una tutela specifica, non in tutti i paesi è stata predisposta una normativa specifica, nè dove questa esiste essa risulta uniforme.

Viceversa il SW, come bene facilmente distribuibile a livello internazionale, richiede per quanto possibile delle forme di tutela uniformi, che rendano possibile una commercializzazione effettuata con modalità uniche per i diversi paesi, e che non offrano la possibilità di vanificare certi aspetti della tutela perchè quello che è proibito o illecito in un paese risulta invece legittimo in un altro.

E con questo obiettivo di armonizzazione, che la Comunità Economica Europea ha preparato e intende promulgare quanto prima, una direttiva comunitaria i cui principi possono essere così sintetizzati:

- Il software rientra nella categoria delle opere tutelate dal diritto di autore, come opera letteraria.
- La tutela non si estende a idee, principi, logica su cui è basato il pogramma.
- Il titolare del diritto può controllare in modo esclusivo la replicazione, in tutto o in parte, l'adattamento e la distribuzione del programma.
- L'uso del programma, la sua visualizzazione, esecuzione, trasmissione o memorizzazione sono atti soggetti a limitazioni, in quanto implicano una riproduzione.

Le uniche deroghe a questi diritti di controllo si hanno per la vendita o messa a disposizione del pubblico senza accordi contrattuali: è conferita con ciò una licenza implicita ad effettuare tutti gli atti necessari all'uso del programma, compreso il suo adattamento, nonchè la libertà di alienazione e circolazione della copia.

Viene inoltre introdotto il concetto di violazione indiretta dei diritti esclusivi, che si verifica con l'importazione, il possesso ed il commercio consapevole di copie illecite (anche se non se ne fa copia) oppure con la fabbricazione, importazione, possesso, commercio dei così detti "sprotettori".

È anche confermata la cumulabilità della tutela offerta dal diritto di autore con altre eventuali forme di protezione giuridica, come brevetto, marchio, concorrenza sleale, segreto industriale, e normative contrattuali.

Sono secondari, per gli scopi di questa relazione, gli altri aspetti considerati dalla direttiva quali la paternità dei programmi, i beneficiari della tutela, la durata della tutela.

La direttiva è ancora in discussione, in quanto esistono, negli ambienti interessati, divergenze di natura formale e sostanziale su alcuni punti, in particolare sulla libertà di operare il così detto reverse engineering e sui limiti che devono essere posti a questa attività, ma è prevedibile che possa divenire efficace già nel 1991, con modifiche che non ne cambieranno in maniera rilevante la portata.

12. La protezione offerta dal diritto di autore.

È ormai ragionevolmente consolidato, e lo sarà ancor di più con l'armonizzazione richiesta dalla direttiva CEE, che nel caso dei programmi di elaboratore, non solo la riproduzione intesa come effettuazione di una copia, ma anche l'uso del programma, in quanto presuppone una riproduzione, sono attività proibite senza l'autorizzazione del titolare del diritto, autorizzazione che si esprime attraverso la stipulazione di un contratto di licenza, nei limiti e alle condizioni previste dal contratto, o, nel caso di vendita di supporti del programma, come licenza implicita o derogazione prevista dalla legge.

Ci sono due conseguenze fondamentali di questo approccio:

- La prima è già stata accennata ed è che il titolare del diritto può limitare contrattualmente l'uso del programma.
- La seconda conseguenza è che, in assenza di una autorizzazione contrattuale o di una licenza implicita, qualsiasi uso di un programma è proibito dalla legge.

Mentre un accordo contrattuale è efficace tra le parti, il divieto imposto dal diritto di autore è efficace "erga omnes" e il diritto di autore costituisce un "diritto assoluto".

Gli orientamenti attuali in materia di diritto di autore eliminano quindi una grave limitatezza della protezione assicurata dal diritto comune in materia contrattuale: una eventuale violazione contrattuale rimane confinata in sè stessa e non ha implicazioni sul diritto assoluto.

Facciamo un esempio: in assenza di un diritto assoluto, un programma potrebbe essere riprodotto e trasferito a terzi in violazione di impegni contrattuali assunti.

Una rivalsa per violazione di contratto e risarcimento del danno subito potrebbe essere esercitata solo nei confronti della parte inadempiente, con nulle o scarse possibilità di azione nei confronti di terzi.

L'esistenza di un diritto assoluto, e di una normativa che disciplina i limiti d'uso di un programma in assenza di obbligazioni contrattuali, può rendere superflua la messa in atto di pratiche commerciali di distribuzione del SW per mezzo di contratti di licenza, semplificando e rendendo più economico il processo di distribuzione. È questa per esempio la politica seguita dalla distribuzione di videogiochi e programmi per home computer, che sono oggetto di vendita del relativo supporto senza sottoscrizione di obbligazioni contrattuali e sono reperibili in edicole, librerie, rivendite di giornali e negozi di prodotti elettronici.

Lo stesso vale nel caso di vendita di macchine e apparecchiature, quali stampanti, governi unità periferiche, terminali di visualizzazione, ricezione, trasmissione messaggi, che intrinsecamente contengono del "FIRMWARE" ossia dei programmi essenziali al loro funzionamento.

Nella maggior parte dei casi, tuttavia, e per i prodotti di maggior valore, si preferisce fare ricorso agli impegni contrattuali come strumento complementare alla tutela offerta dal diritto d'autore (e dal brevetto quando applicabile) per definire limiti d'uso e garantire nel caso la segretezza del prodotto, come consentito nell'ambito del diritto comune.

13. Le clausole contrattuali tipo

Le clausole contrattuali che possono essere adottate differiscono secondo i casi. Come già detto il mercato del SW è molto eterogeneo e può riguardare sistemi operativi, utilities, applicativi.

Gli operatori interessati possono essere i produttori originali, intermediari di distribuzione, utenti finali, fornitori di servizi, produttori di HW che intendono completare la loro offerta con prodotti SW sviluppati da terzi ma adattati a esigenze di un HW specifico.

Esiste quindi una grande varietà di contratti nei quali si può identificare tutta una serie di clausole tipo, correlate tra loro per assicurare una tutela e una valorizzazione del prodotto.

Poichè il costo di sviluppo del SW è la componente principale di costo e il costo di produzione (o riproduzione) è irrilevante, il massimo profitto dal bene, la sua valorizzazione, si ottiene con una politica di distribuzione che ne consenta la massima diffusione in modo che i costi di sviluppo si ripartiscano su una pluralità di copie e di utenti.

Così facendo si può arrivare a un prezzo di mercato ragionevole e tale da poter soddisfare la domanda in volumi sufficienti a consentire il recupero dei costi e il profitto dell'operazione.

Lo studio delle leggi economiche (Figura 4) consente di identificare per un dato bene (leggi programma) una curva di domanda che definisce le quantità di un bene che il mercato è disposto ad acquistare (ascissa) in funzione del suo prezzo (ordinata).

Questa curva mette in evidenza che l'utilità di un bene per i diversi soggetti che compongono il mercato è variabile: per alcuni è molto alta, per altri è bassa.

In condizioni di monopolio, il produttore del bene può usare tale curva per prevedere i potenziali ricavi in funzione del prezzo di mercato che stabilisce per il bene (curva dei ricavi a prezzo fisso).

Confrontando la curva dei ricavi con la curva dei costi totali di produzione, funzione dei volumi, si può anzitutto verificare se esiste una condizione di mercato per cui i ricavi superano i costi.

Si può anche determinare quale è la condizione di mercato prezzo-volume che consente di realizzare il massimo profitto (con il confronto tra curva dei costi marginali e curva dei ricavi marginali, che sono rispettivamente la derivata della curva dei costi totali e della curva dei ricavi totali). Se la curva dei costi è sempre superiore a quella dei ricavi non vi è apparentemente alcun interesse a intraprendere l'iniziativa commerciale di produzione e distribuzione del bene.

È questo il caso di molti prodotti SW.

In realtà la curva dei ricavi a prezzo fisso descrive una situazione in cui il prezzo è invariabile.

Ma se l'utilità di un bene è variabile da soggetto a soggetto perchè non imporre al mercato un prezzo variabile in funzione dell'utilità variabile per ogni soggetto? Se ciò è possibile la curva dei ricavi si trasforma in una curva che anzichè essere una campana è sempre crescente (integrale della curva di domanda) e può mettere in evidenza che l'iniziativa commerciale può essere profittevole.

Come realizzare questa condizione di prezzi differenziati?

Per i beni convenzionali si usa generalmente la politica dei prezzi variabili e decrescenti nel tempo ma sempre superiori al costo marginale) per attrarre al prodotto nuove fasce di mercato. Per il SW, oltre a questa politica, se ne possono adottare altre che non sono discriminatorie da cliente a cliente ma selettive in funzione dell'uso e del vantaggio ricavabile dal prodotto.

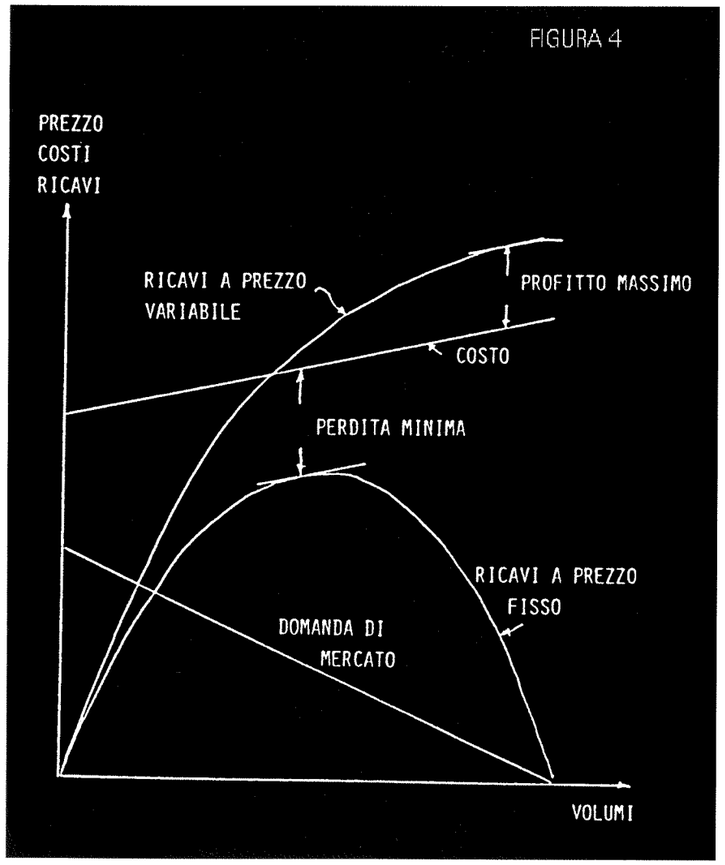


Fig. 4 - Le leggi economiche generali valgono anche per i prodotti software.

L'uso del bene è distribuito nel tempo: si può quindi chiedere un corrispettivo (canone) in funzione della durata di impiego del bene.
L'utilizzazione avviene attraverso macchine più o meno potenti con produttività del bene più o meno alta. Si può quindi praticare una politica di prezzi in funzione dello strumento HW con cui il SW deve operare.
L'utilizzazione può avvenire con modalità diverse da utente a utente, e quindi con utilità diverse. Si può quindi differenziare il prodotto in modo da consentirgli di operare solo con certe modalità e non con altre e praticare una politica di prezzi differenziati.
La maggior parte delle clausole contrattuali, che qualche volta possono apparire all'utente come vessatorie, perchè impongono dei precisi limiti all'utilizzazione del bene, hanno questo scopo: praticare un prezzo del bene proporzionato al vantaggio che l'utente ne ricava.
Altre clausole hanno motivazione diversa, servono essenzialmente ad evitare che attraverso un trattamento negligente del bene, esso possa cadere in mano a persone di pochi scrupoli che le possano duplicare, ritoccare e immettere sul mercato o usare in concorrenza al prodotto originale.
Le clausole che impegnano alla segretezza sulle informazioni contenute nel programma sono solo alcune delle clausole di questo tipo, ma vi si possono comprendere anche quelle relative ai divieti di alienazione e di prestito, nonchè quelle che prevedono diritti di auditing per verificare che il prodotto sia usato nei limiti e con le cure richieste.
Vi sono anche clausole che il fornitore propone perchè sono imposte da esigenze di legge o di regolamentazione imposte da accordi internazionali.
Molti prodotti SW sono considerati di interesse strategico perchè potenzialmente suscettibili di impiego, oltre che nel campo industriale e commerciale, anche a fini militari, e comunque tali da portare a progressi tecnologici pregiudizievoli all'interesse dello stato o di alcuni stati.
Non è quindi infrequente trovare clausole che limitano territorialmente la circolazione del bene o ne vietano l'esportazione, salvo autorizzazione delle autorità a ciò preposte, che in alcuni casi possono anche essere autorità estere (Dipartimento del Commercio e Dipartimento di Stato degli Stati Uniti).
Vi sono infine clausole che si possono definire standard nel senso che si trovano in tutti i tipi di contratti, quali clausole di garanzia o limitazione di garanzia, limitazioni di responsabilità per violazione di diritti di terzi, legge applicabile e foro competente, sulle quali non è qui il caso di soffermarsi.
Vale invece la pena di considerare un ultimo aspetto della protezione del software, ossia la funzione preventiva o repressiva della tutela.

14. Prevenzione e repressione

Il fatto che il SW possa godere di una tutela brevettuale, di una tutela nell'ambito del diritto di autore, di una tutela contrattuale, non significa che violazioni dei diritti esistenti non possano verificarsi.
Significa solo che in caso di violazione, l'abuso può essere punito e represso.
Da questo punto di vista le norme giuridiche poste a tutela del SW costituiscono solo un deterrente.
Non è poi detto che la punizione e repressione dell'abuso possano risultare così semplici da mettere in atto, così economiche e così indennizzanti da giustificare la loro applicazione sistematica.
Si possono identificare al riguardo due tipi di abusi principali.

- Abusi concorrenziali: un programma viene copiato e messo sul mercato in concorrenza al prodotto originale.
- Abusi di utente: un programma, abusivamente duplicato o meno, è usato senza autorizzazione o al di fuori dei suoi limiti.

Nel secondo caso il danno subito è generalmente limitato e la repressione dell'illecito può risultare più onerosa dell'indennizzo. In alcuni casi può anche significare la perdita di un cliente e di un "business" molto più grosso di quello costituto dall'oggetto dell'illecito.
Basti pensare a una azienda informatica che vende calcolatori, e che al tempo stesso fornisce SW, e servizi di assistenza.
Prima di reprimere con mezzi legali un abuso connesso al SW ci penserebbe due volte.
Allora perchè non dotare i programmi di opportuni dispositivi antifurto?
È noto che non esiste dispositivo di protezione che non possa essere violato.
Ma ciò comporta una competenza specifica che non tutti possiedono, delle risorse che non tutti possiedono, dei costi che vanno rapportati ai vantaggi che si ottengono e dei rischi che non tutti osano correre.
Così, se è vero che nel caso di abusi concorrenziali è possibile che si verifichino tutte queste circostanze, competenze, risorse, entità del lucro, disponibilità al rischio, nel caso dell'utente comune ciò è abbastanza improbabile.
Certamente esistono persone dotate di una competenza specifica per le quali l'elusione di strumenti fisici e logici di protezione a difesa del SW, è quasi un gioco, o diventa persino una sfida gratificante, indipendentemente dal ritorno economico, ma per la maggior parte degli utenti è vero il contrario.
Così le stesse leggi dell'interesse economico possono essere strumentalizzate per ottenere l'effetto contrario, scoraggiando qualsiasi azione abusiva.
Non è qui il caso di dilungarsi sulle diverse tecniche di protezione fisica che possono essere usate nel caso del SW.
Per chi può essere interessato, questi Quaderni hanno trattato recentemente il problema in maniera semplice e al tempo stesso esaustiva con un articolo di Eugenio Orlandi (Pirateria del software: tecniche di protezione fisica; Quaderni di informatica, Anno XV N.3, 1989).
Devo solo aggiungere, in tema di protezione fisica, che la difficoltà di assicurare una correttezza esaustiva dei programmi, si traduce di fatto in una protezione.
Chi usa delle copie abusive di un programma non può in generale beneficiare di tutte quelle forme di assistenza che i fornitori mettono a disposizione degli utenti, vuoi gratuitamente, vuoi a pagamento e di fronte a un problema nell'uso di un prodotto è completamente abbandonato a se stesso.
Se si vuole beneficiare di questa assistenza è quindi preferibile fare ricorso alla qualità globale dei servizi offerti dai produttori del SW e dai loro distributori autorizzati.
Il fenomento della pirateria del SW è certamente un fenomento diffuso che a dispetto di tutte le forme di tutela predisposte, esiste e continuerà ad esistere. È però prevedibile che l'evoluzione della tecnologia e l'espansione del mercato ne consentiranno un sostanziale ridimensionamento.
Per esempio l'immissione sul mercato di prodotti software registrati su CD-ROM renderà certamente più onerose le operazioni di replicazione meccanica, che sarà possibile solo in officine specializzate.
Le operazioni di replicazione logica su supporti di tipo diverso sarà ancora possibile a basso costo, ma potranno e possono già essere ostacolate con meccanismi di accesso a chiave logica. Anche il prezzo dei prodotti avrà il suo peso.
Con l'aumento delle dimensioni del mercato sarà possibile distribuire i costi di sviluppo su maggiori volumi, abbassando i prezzi unitari.
Almeno a livello di abusi commessi da utenti il vantaggio economico derivante dall'abuso potrebbe diventare marginale.
Per quanto riguarda gli abusi concorrenziali, la protezione più efficace consiste ancora nel riconoscimento giuridico del SW come beneficiario del diritto di autore con la possibilità di esercitare tutte le azioni legali che la legge prevede.
Un inasprimento delle sanzioni penali previste per le violazioni del diritto di autore potrebbe ridimensionare il fenomeno.
Per quanto riguarda l'Italia è stato recentemente presentato alle camere un disegno di legge (N. 4367) che prevede delle pesanti sanzioni penali per la abusiva duplicazione e commercializzazione del SW.
Vi sono molti punti discutibili in tale proposta e, in assenza di una chiara identificazione dei reati, vengono suscitate incertezze con possibili conseguenze di così rilevante portata che un giudizio negativo non può essere taciuto.
Pur tuttavia, l'iniziativa è in sè lodevole ed è auspicabile che il legislatore ne tenga conto con i dovuti correttivi perchè, una volta convertita in legge svolga effettivamente il ruolo per cui è nata.

Progetto di cinematismi mediante computer

IVANO TANTURLI

SICAD
Società Italiana Computer Aided Design
Milano

1. Introduzione

La realizzazione di un modello tridimensionale ad elementi solidi con l'ausilio del calcolatore è oggi cosa ormai comune.

Ma la possibilità di simulare i suoi movimenti, evidenziandone in tempo reale le caratteristiche cinematiche e dinamiche, è ottenibile soltanto con un software altamente sofisticato.

Per non trattare l'argomento in modo generico, faremo riferimento qui ad uno specifico sofware, lo IMP (Integrated Mechanism Program), che costituisce un esempio paradigmatico delle eccezionali possibilità oggi offerte ai progettisti meccanici dalle tecniche CAD (Computer Aided Design).

Un programma come IMP consente la simulazione di sistemi meccanici bi e tridimensionali con uno o più gradi di libertà nei nodi di collegamento.

La facilità con la quale si possono determinare le caratteristiche cinematiche, statiche e dinamiche di un sistema meccanico dipendono essenzialmente dal numero e dal tipo (traslazione o rotazione) dei gradi di libertà in gioco e dall'orientamento dei collegamenti che formano il sistema fisico.

Con il programma IMP le geometrie tridimensionali dei diversi elementi sono facilmente realizzabili mediante l'utilizzo del modellatore solido GMS, un sistema software che funge da preprocessor.

Con GMS tutte le informazioni inerenti le geometrie, le posizioni dei baricentri, le caratteristiche di massa e d'inerzia di ogni singolo elemento costituente il sistema sono calcolate e registrate nel data base di IMP.

Dalla simulazione del sistema meccanico si possono avere tutte le informazioni in terminì di velocità, accelerazioni e forze (statiche e dinamiche) necessarie alla definizione delle condizioni peggiori di esercizio del meccanismo studiato.

Infine, tutte queste informazioni possono confluire ed essere utilizzate mediante la realizzazione di un modello ad elementi finiti (*).

A tale scopo si può usare il software COSMOS/M, con il quale si possono verificare strutturalmente tutti gli elementi che compongono il sistema studiato.

Nella figura 1 si mostrano le varie fasi di progettazione ed i programmi che si possono usare per risolvere completamente lo studio dei meccanismi.

2. Il modellatore di solidi

Il preprocessor GMS di cui si avvale IMP è caratterizzato da un'ampia versatilità e facilità d'uso.

Il programma dispone di un gran numero di funzioni geometriche bi e tridimensionali (per esempio cerchio, quadrato, cubo, prisma, parallelepipedo, etc.), predefiniti nella libreria del sistema, che possono essere generate semplicemente scegliendone il nome e definendone le caratteristiche specifiche (per un cerchio o una sfera il raggio, per un cubo le dimensioni di un lato, per un parallelepipedo le lunghezze dei tre lati, ecc.).

In pratica, GMS si comporta come un modellatore parametrico.

È evidente però che l'utente deve avere la possibilità di realizzare qualsiasi forma geometrica, ed è per questo che GMS possiede la funzione di *sketch*, con la quale si possono realizzare forme bidimensionali qualsiasi.

Nell'esempio di figura 2 si è generata una sezione geometrica ricorrendo all'uso di punti, di archi e di raccordi (*spline*).

È a questo punto che GMS diventa molto potente. Infatti con i comandi di generazione per estrusione, per rotazione, per "sweeping" e per copia si possono realizzare modelli solidi di qualsiasi forma.

Ma dove GMS diventa veramente eccezionale è nelle applicazioni dell'algebra Booleana, cioè unione e sottrazione di più oggetti e generazione di solidi formati dall'interferenza di altri elementi.

La figura 3 mostra il risultato ottenuto unendo una sfera con un parallelepipedo, nella figura 4 si evidenzia il risultato dovuto ad una sot-

(*) Sul metodo degli elementi finiti, si veda l'articolo dell'Autore apparso sul n. 39 (1990) di questa rivista.

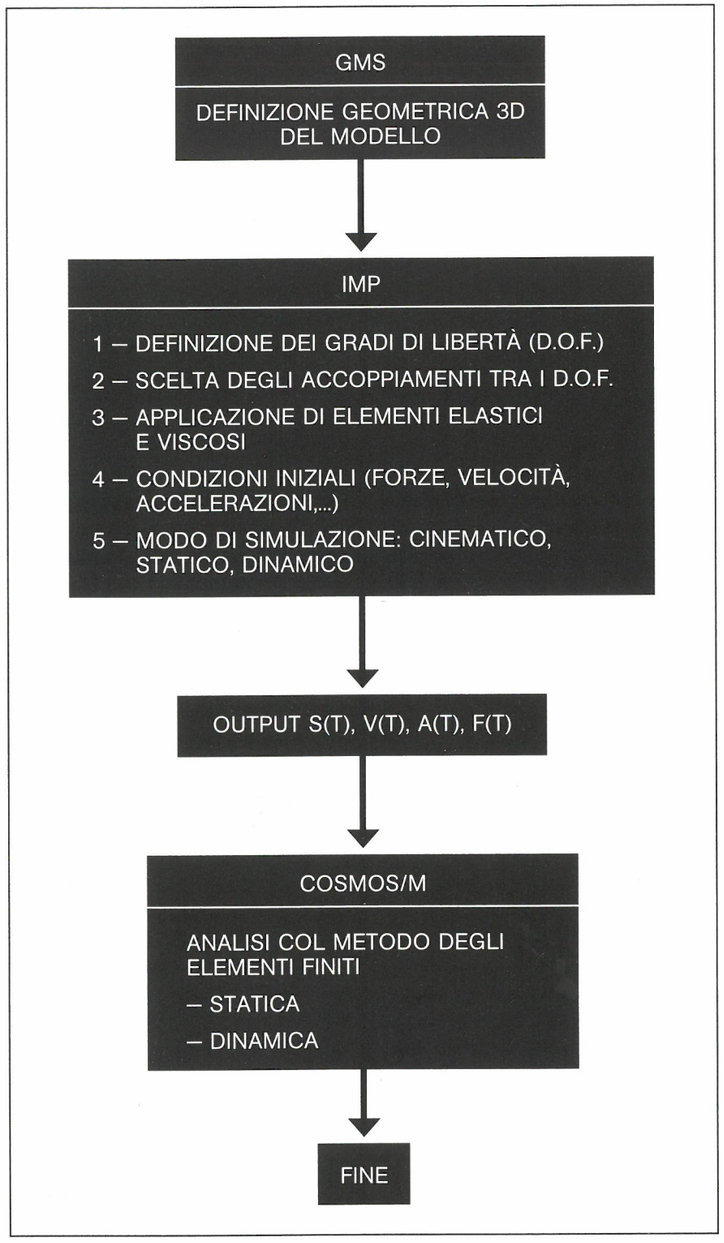


Fig. 1 - Iter progettuale con la sequenza dei vari moduli CAD.

trazione, mentre nella figura 5 si è rappresentata la funzione d'interferenza compiuta per l'esempio di figura 4.

Completano le capacità di GMS le varie funzioni di traslazione e rotazione che permettono di posizionare i diversi oggetti solidi nella loro corretta collocazione spaziale.

Nella figura 6 si mostra una fase intermedia in cui si sta realizzando un modello complesso.

È soprattutto in questa fase che si può ricorrere alle funzioni di *view dinamico* (rotazione, traslazione e zoom in tempo reale) con e senza effetto *shade.*

Infine è da ricordare che qualsiasi oggetto realizzato in GMS mantiene sempre la sua identità ed è in qualsiasi momento "editabile" con tutte le informazioni relative al numero di punti, linee, superfici, alle dimensioni d'ingombro massime, al baricentro, all'area superficiale, al volume nonchè ai momenti d'inerzia.

3. Il programma di simulazione

IMP è un programma di analisi e di progettazione che può essere usato facilmente, a costi molto contenuti,

Fig. 2 - Esempio di generazione di geometrie mediante la funzione di "sketch".

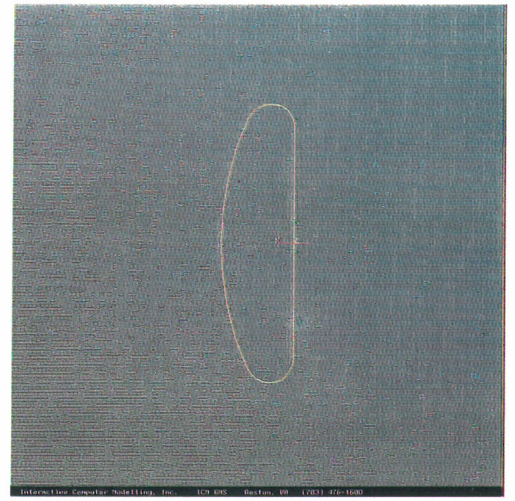


Fig. 3 - Operazione Booleana di unione. La figura mostra il risultato ottenuto unendo una sfera e un parallelepipedo.

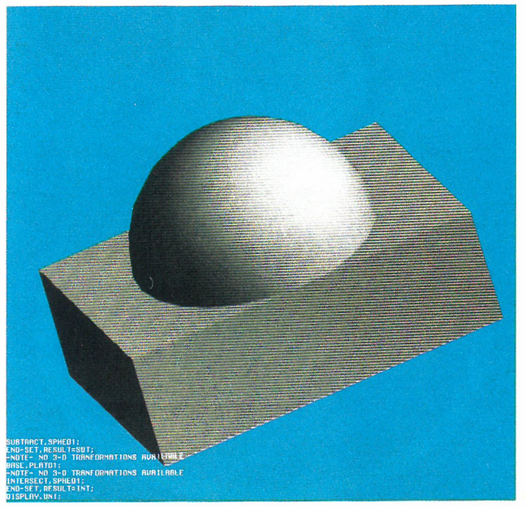


Fig. 4 - Esempio di operazione Booleana di sottrazione.

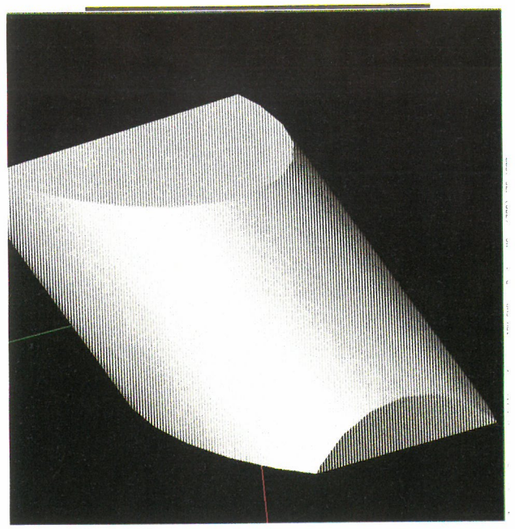


Fig. 5 - Operazione di interferenza compiuta sul modello di fig. 4.

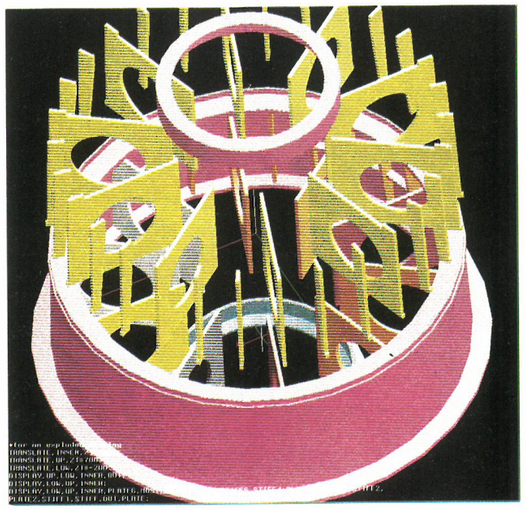


Fig. 6 - Fase intermedia di realizzazione di un modello complesso.

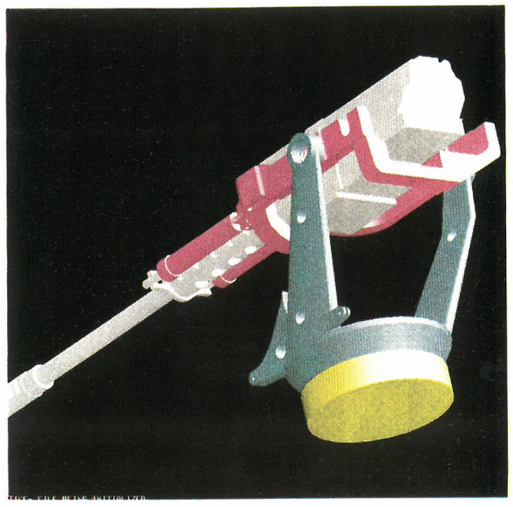


Fig. 7 - Sistema meccanico con più gradi di libertà.

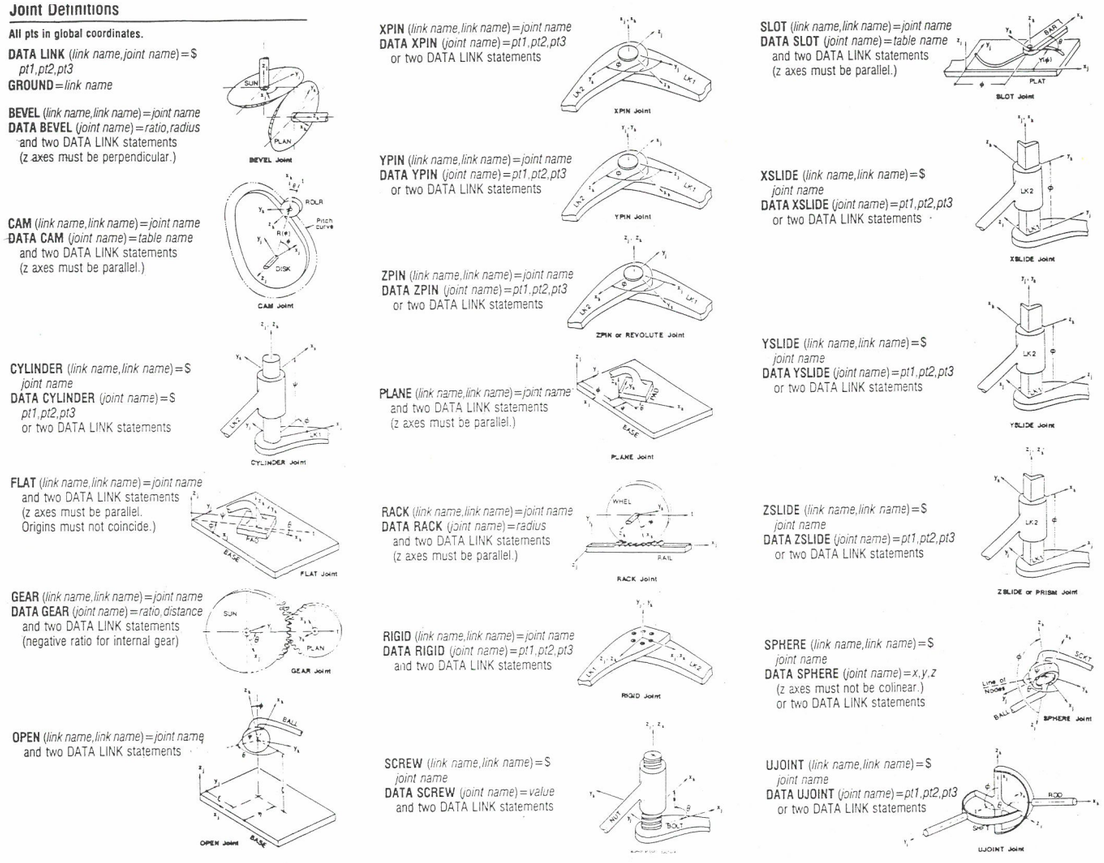


Fig. 8 - La libreria di elementi meccanici di IMP.

nel campo della simulazione di sistemi meccanici formati da collegamenti rigidi bi e tridimensionali comunque orientati e con uno o più gradi di libertà (vedi fig. 7).

La libreria dei collegamenti è formata da una vasta gamma di elementi tra i quali: spine, guide, viti, ingranaggi cilindrici, ingranaggi conici, cremagliere e pignoni, eccentrici rotanti, intagli e giunti cilindrici, universali, sferici ed a bussola in qualunque combinazione aperta o a loop chiuso (vedi fig. 8).

Inoltre, elementi tipo molle e smorzatori viscosi lineari e non lineari possono essere introdotti nei giunti o agire fra punti specificati nel movimento dei diversi collegamenti.

È possibile nell'analisi considerare anche gli effetti delle masse e dell'accelerazione di gravità.

Facciamo un esempio, in modo da chiarire maggiormente quello che si è detto in precedenza. Supponiamo di avere realizzato con GMS il modello di figura 9, identificato nei suoi componenti da 3 elementi, cioè la guida orizzontale e verticale che formano un unico oggetto chiamato "base", la barra a forma di Y nominata "leva 1" con i perni che s'inseriscono nelle fenditoie della guida ("base") ed infine la barra centrale collocata nella parte superiore della guida verticale della base (identificata con il nome "leva 2").

In questo caso, la barra a forma di Y (leva 1) ha due vincoli cinematici di traslazione rispetto alla "base" che rimane fissa; in più, lungo il suo asse medio c'è la traslazione del perno della manovella sottostante ("leva 2") la quale ruota di moto circolare uniforme vincolata alla "base".

In questo esempio abbiamo in totale quattro gradi di libertà (tre di traslazione e uno di rotazione).

Con IMP, realizzare questo manovellismo è molto semplice. Basta scegliere il tipo di vincolo che si desidera realizzare ed attribuirlo agli elementi che lo compongono; tutto questo semplicemente selezionando con il cursore dai menù che di volta in volta appaiono sulla destra del video.

Fatto ciò, è sufficiente collegare le caratteristiche di massa e d'inerzia, selezionandole con il cursore, ed at-

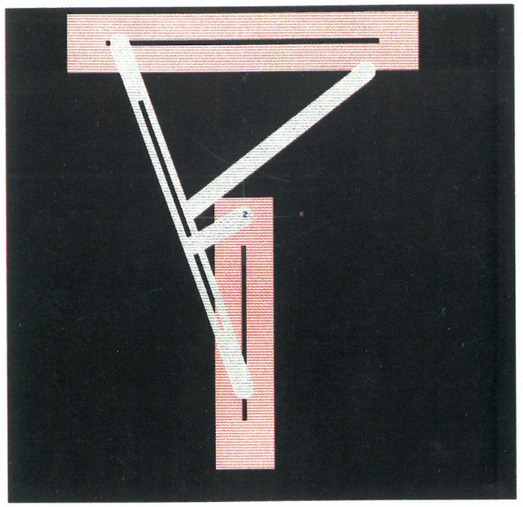


Fig. 9 - Studio di un manovellismo con IMP.

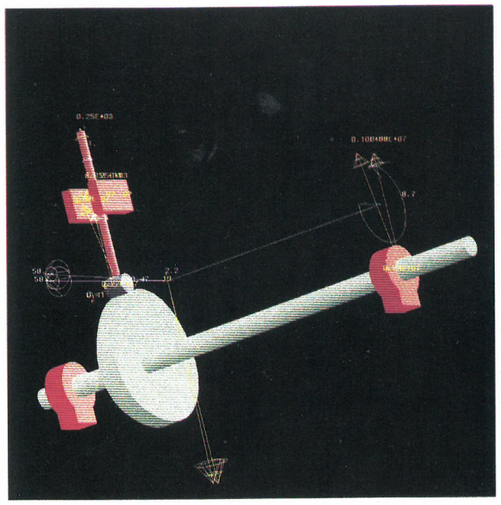


Fig. 10 - Studio di un albero con eccentrico a camma.

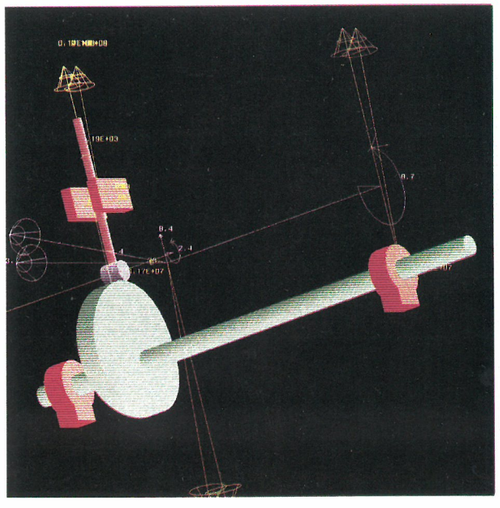


Fig. 11 - Lo stesso modello di fig. 10 in un istante diverso.
Le caratteristiche cinematiche e dinamiche sono cambiate sia in direzione che in modulo.

tribuire la legge oraria (se si vuole simulare in modo cinematico) all'elemento che dovrà pilotare il sistema (in questo caso si fornirà la variazione nel tempo dell'angolo della manovella chiamata "leva 2").

A questo punto si potrà eseguire la simulazione e successivamente si potrà usare il modulo di postelaborazione per vedere i risultati dell'analisi appena fatta.

In quest'ultima fase, veramente spettacolare, si potrà vedere il meccanismo muoversi (come in un filmato) con le leggi del moto imposte dall'utente. Tutto questo mentre il progettista potrà sfruttare le possibilità di *view dinamico* per effettuare rotazioni, traslazioni e zoomate in tempo reale con e senza effetto *shade.*

Inoltre egli potrà mostrare, richiamando a video e vedendoli variare sia in direzione che in modulo (anche con il valore numerico), i vettori che rappresentano le forze, le accelerazioni, le velocità e tutte le caratteristiche cinematiche e dinamiche che possono ritenersi utili.

Il sistema meccanico, una volta formato, può essere pilotato da forze applicate o da moti imposti i quali possono essere funzioni specificate del tempo e della geometria del sistema.

Il software IMP è in grado di simulare il sistema meccanico in tre differenti modi:

- cinematico (geometria)
- statico (equilibrio)
- dinamico (risposta nel tempo)

In tutti e tre i modi, il programma è in grado di calcolare le posizioni, le velocità, le accelerazioni, le reazioni vincolari, sia statiche che dinamiche, le frequenze proprie, i rapporti di smorzamento e le funzioni di trasferimento delle piccole oscillazioni (modi principali) del sistema simulato.

4. Iter di progettazione

L'ingegnere progettista ha a disposizione due approcci per realizzare un nuovo meccanismo. Il primo, detto "sintetico", implica l'utilizzo di soluzioni metodologiche conosciute, le quali richiedono la conoscenza delle dimensioni geometriche e la definizione di tutte le proprietà del modello. Il secondo approccio si basa invece sul processo di iterazione e si avvale di tutte le tecniche scientifiche conosciute in proposito.

Generalmente il primo metodo è il più diretto, ma sfortunatamente è applicabile ad una ristretta classe di problemi. Il secondo, anche se più lungo, permette di risolvere qualsiasi problema.

IMP è un programma che si può utilizzare quando si è costretti a ricorrere al secondo metodo in quanto mette a disposizione dell'utente una grande capacità di analisi e di calcolo.

Nel processo di progettazione ha una grande importanza l'interazione tra l'uomo e il computer ed IMP

rappresenta un utile mezzo, flessibile e di facile uso, in grado di realizzare le idee del progettista senza restringere la libertà di scelta.

Topologia

Nella prima prima fase di studio di un sistema meccanico si deve includere l'analisi topologica, cioè il riconoscimento dei vincoli, del numero e del tipo di giunzioni, l'ordine con il quale i vincoli e le giunzioni sono introdotte, il numero e la forma dei cicli (loop) cinematici. La determinazione di questi ultimi è un aspetto essenziale.

In questa fase dell'analisi IMP utilizza algoritmi basati sulla teoria dei grafi (rif. [3]) ed è in grado di riconoscere se un ciclo non è propriamente chiuso o se il vincolo fisso non è definito. In tal caso esso conclude che il problema è mal posto e termina presentando all'utente un messaggio diagnostico.

Quando si inizia la simulazione del moto di un sistema, i dati numerici più importanti sono rappresentati dalla posizione relativa e dell'orientamento degli elementi che formano i vari giunti.

Inoltre, la posizione del sistema è dettata dal valore di certe variabili, le quali sono dette "coordinate generalizzate" del sistema e sono in numero eguale a quello dei gradi di libertà del sistema stesso.

In generale, le coordinate generalizzate (DF) sono controllate da agenti esterni, chiamati "coordinate generalizzate specificate" (SGC).

I restanti gradi di libertà assumono valori che sono valutati o dall'equilibrio statico o dalle condizioni dinamiche dipendenti dal modo della simulazione.

Ciò posto, le "coordinate generalizzate libere" (FGC) sono definite dalla relazione:

$$FGC = DF - SGC$$

FGC può assumere un valore compreso tra 0 e DF.

Quando FGC = 0 la posizione del sistema è definita da considerazioni solo cinematiche, mentre quando FGC = DF le coordinate generalizzate assumono valori che sono determinati dal modo della simulazione.

Modo cinematico

Nel modo cinematico l'utente deve specificare l'intervallo del moto ed i valori per ciascuna coordinata generalizzata (per variabili generalizzate si intendono quelle grandezze che permettono di determinare la posizione di qualunque punto del sistema in qualunque istante; il loro numero deve essere uguale al numero di gradi di libertà del sistema), fornendo velocità, accelerazioni e forze applicate. IMP risolverà il sistema calcolando le grandezze d'interesse nell'intervallo specificato.

Quando IMP lavora in modo cinematico è in grado di riconoscere qualsiasi situazione in cui si abbiano le coordinate generalizzate libere (FGC) non volute ed in tal caso IMP mantiene costante il loro valore.

Modo statico

Nei sistemi vincolati caricati da forze stazionarie, si può desiderare di specificare solo le coordinate generalizzate (SGC) e seguire le posizioni delle FGC assunte in condizioni di equilibrio statico.

Oppure si può voler vedere cosa succede al sistema in termini di po-

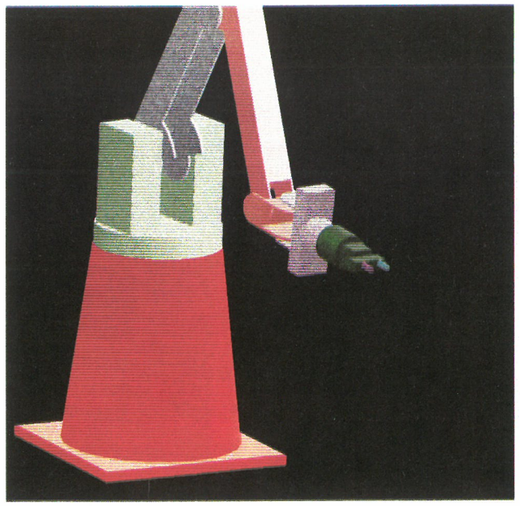


Fig. 12 - Simulazione dinamica di un robot.

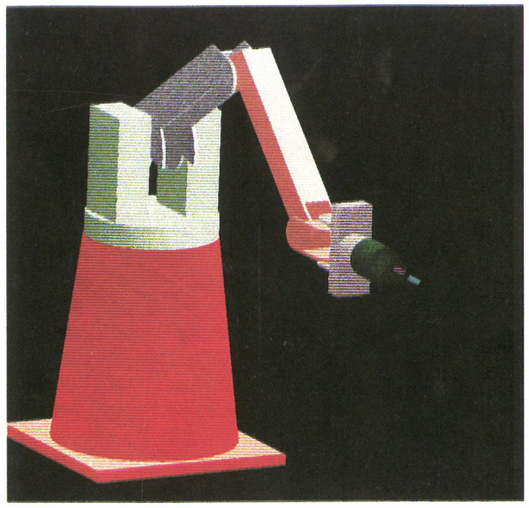


Fig. 13 - È lo stesso robot di fig. 12 ma in posizione diversa.
Le "dita" della mano hanno subito una rotazione ed una traslazione che simulano una operazione di "presa".

sizioni di equilibrio assunto o quando si incrementa l'intensità di forze agenti.

Le posizioni delle FGC sono continuamente valutate dalla soluzione della equazione statica:

$$[K] * \{dX\} = \{dF\}$$

in cui la matrice di rigidezza [K] è di ordine (FGC * FGC), {dX} è il vettore delle posizioni di FGC e {dF} è il vettore delle forze nette sbilancianti nei FGC.

Modo dinamico

Si possono identificare due distinti tipi di problemi dinamici: il primo tipo è caratterizzato dall'ampiezza delle piccole oscillazioni di un sistema rispetto alla sua posizione di equilibrio statico (o quasi statico); il secondo tipo è caratterizzato dal moto delle grandi ampiezze e richiede l'utilizzazione delle non linearità geometriche.

L'esempio di fig. 10 e 11 mostra lo studio di un'albero con eccentrico a camma in due diversi istanti (frame 25 e 50). Dalle figure si può vedere come le caratteristiche cinematiche e dinamiche siano cambiate sia in direzione che in modulo.

Nell'esempio di fig. 12 e 13 è rappresentata invece la simulazione dinamica di un robot in due posizioni diverse. Dalle figure si può osservare come il braccio sia esteso in modo diverso e come le "dita" della mano abbiano subito una rotazione ed una traslazione tali da simulare un'operazione di "presa".

5. Conclusione

Grazie a programmi CAD altamente sofisticati come quello brevemente presentato in questo articolo, il progettista meccanico dispone oggi di strumenti di lavoro di eccezionale potenza, flessibilità e facilità d'uso.

È opportuno sottolineare che disporre di questi mezzi appare ormai non tanto una opzione quanto una necessità.

Infatti, poter disegnare, verificare, modificare e ottimizzare un dispositivo meccanico senza dover costruire una serie di prototipi significa un risparmio non solo di costo ma anche di tempo. E quest'ultimo è sempre più importante. Anticipare la concorrenza e scegliere il più opportuno *"time - to - market"* è infatti, accanto ai tradizionali fattori di costo e prestazioni, la carta vincente nell'offerta di un prodotto.

Riferimenti

[1] P.N. SHETH, *A digital computer based simulation procedure for multiple degree of freedom mechanical system with geometric constraints,* Ph. D. Dissertation, University of Wisconsin, 1982.

[2] P.N. SHETH and J.J. UICKER, *IMP, (Integrated Mechanisms Program): A computer aided design analysis system for mechanisms and linkages,* Journal of Engineering for Industry, ASME Transactions, vol. 93, no. 4, 1981.

[3] T.A. PHELPS, *"Algorithms for determining the kinematic loops in mechanisms,"* MS Thesis, University of Wisconsin, 1983.

Carlo Peretti
Dialogo sull'informatica
Editore Le Monnier, 1990 (pag. 150)

Il volume intitolato "Dialogo sull'informatica", di cui è autore Carlo Peretti, Presidente Bull Italia, fa parte della collana "Tecnologia 2000" diretta da Bruno Visentini e Piero Barucci che è pubblicata dall'editore Le Monnier di Firenze.
La collana si rivolge principalmente ai giovani e ha l'obiettivo di fornire una panoramica dei più importanti settori dell'economia come strumento di orientamento delle loro scelte professionali.
Sono previsti 24 titoli che vanno dall'informatica all'agroindustria, dalla finanza al giornalismo, dal turismo alla pubblica amministrazione.
Caratteristica della collana è che gli autori dei vari volumi sono personaggi fra i più rappresentativi del rispettivo settore.
Il volume di Peretti è impostato in forma di intervista ed è articolato in sette parti.
Nella prima parte si fa una breve storia della elaborazione dei dati, dalla macchina di Pascal a quella di von Neumann.
Segue quindi una disamina dei concetti fondamentali dell'informatica: hardware, software, architettura, tecnologia elettronica, anche per fornire una terminologia di base del settore.
Allo stesso scopo, alla fine del volume è anche inserito un glossario dei termini più frequenti.
Nella terza parte si affronta l'informatica come fatto economico, fornendo indicazioni sulla sua struttura, dimensioni e tendenze. Quindi si passa a un esame dell'informatica italiana e, in questo contesto, viene in particolare approfondito il caso Bull Italia.
Il tema affrontato successivamente riguarda lo stato e le prospettive della cultura informatica in Italia.
Si parla del piano nazionale per la scuola secondaria, del ruolo dell'editoria e dei mass media e si accenna anche al contributo di Bull Italia.
Nella sesta parte vengono trattate le professionalità specifiche dell'informatica e le prospettive di occupazione del settore anche qui riferendosi, a titolo esplicativo, alla Bull Italia.
Infine, il volume si chiude con una discussione sul futuro dell'informatica e sui suoi impatti nell'economia e nella società.
L'intervista è preceduta da una presentazione dell'ing. Ottorino Beltrami, presidente dell'Assolombarda.