La tolleranza ai guasti

di Anna Pugliese

Inizialmente sviluppate per quelle applicazioni da cui dipendono direttamente delle vite umane, le tecniche di tolleranza ai guasti sono oggi utilizzate in contesti sempre più vasti.
Da un po' di tempo la sigla «Fault Tolerant» appare anche su alcuni prodotti per personal computer. In questo articolo sono presentate le basi di questa metodologia di progettazione

Vi prego, vi prego, non piangete! Su, asciugatevi le lacrime... non fate così, altrimenti scoppio a piangere anch'io. Prima o poi doveva capitare... chi lo avrebbe mai detto! Ebbene sì, l'articolo di Appunti di Informatica che state per leggere, dopo ben 28 numeri di onorato servizio, non è opera del sottoscritto, ma di una mia ex collega d'università (ora è felicemente laureata) particolarmente esperta di sistemi distribuiti, tolleranza ai guasti, reti locali e geografiche, sistemi operativi multiprogrammati e non, programmazione parallela, architetture parallele di qualsivoglia natura, macchine data-flow o più in generale non Von Neumann. Esattamente come dire che (quasi) vi lascio in buone mani. Il «quasi» tra parentesi non riguarda, ovviamente, la qualità delle mani di cui sopra, ma semplicemente che la mia «anima» continuerà imperterrita a svolazzare tra queste righe finché la rubrica stessa continuerà ad esistere (quanto ancora? difficile dirlo!). Del resto credo proprio che sia un mio e un vostro diritto (... a Napoli dicono: «ogni scarrafone è bell'a mamma soja»).

adp

Prologo

Immaginate per un attimo di trovarvi in uno Shuttle. È vero ne occorre parecchia di immaginazione. Ma voi, immaginate ugualmente. Mancano pochi secondi alla partenza, da bravi piloti aerospaziali incrociate le dita e aspettate che sia tutto finito, che la navicella abbandoni razzi serbatoio e che terminata l'ascesa inizi l'orbita attorno alla terra. Tutto è avvenuto secondi i piani, o meglio, secondo programma. I calcolatori di bordo (ben più di uno, naturalmente) hanno guidato la navetta esattamente sulla traiettoria prestabilita tenendo costantemente nonché scrupolosamente sotto controllo l'intero velivolo. Tutto è andato bene, compreso il fatto magari che sul più bello uno dei computer ha dato forfait per un guasto alla CPU. Tutto previsto: i rimanenti computer ben consci dell'accaduto hanno dominato la situazione senza battere ciglio né facendosi prendere dal panico...

La terminologia

Un GUASTO è la causa meccanica o algoritmica di un errore. Un ERRORE è un evento consistente in una transizione di stato erronea (vi rimandiamo al riquadro di pag. 192 per ulteriori chiarimenti) fra due stati del sistema, dove erronea sta ad indicare la non conformità alle specifiche del sistema.

Un INSUCCESSO è costituito dalla manifestazione esteriore di un errore.

Queste tre definizione sono quelle maggiormente riconosciute dagli addetti ai lavori. I tre termini inglesi che designano, guasto, errore ed insuccesso, sono rispettivamente «Fault», «Error» e «Failure».

Il termine AFFIDABILITÀ sta invece ad indicare la misura del grado di fiducia che può essere riposta nel corretto funzionamento del sistema; in altre parole, l'affidabilità è il contrario della predisposizione agli insuccessi: un sistema è affidabile in misura inversamente proporzionale al numero di insuccessi che in esso si verificano in un certo intervallo di tempo.

È bene dire subito che è praticamente impossibile avere un sistema nel quale non si verificano guasti, nonostante esistano sistemi più CORRETTI di altri, cioè sistemi nei quali la probabilità di guasti è notevolmente ridotta rispetto ad altri.

La progettazione di sistemi con un alto grado di correttezza è un'attività che coinvolge due aspetti: quello hardware e quello software; le tecniche utilizzate nei due casi sono distinte tra loro.

Per l'aspetto riguardante l'hardware, l'attività si esplica nella ricerca di materiali e tecnologie di costruzione, mediante il cui impiego si giunga alla produzione di componenti che, sottoposti a verifiche empiriche (avete presente il peso di 5 kg messo nel cestello delle lavatrici?), dimostrino una resilienza maggiore.

Per quanto riguarda invece, l'aspetto software, il problema è a prima vista più semplice, in quanto basterebbe produrre del software corretto ma, non so se lo sapevate, il programmatore perfetto non è ancora stato trovato (chi si offre volontario?), per cui la speranza è riposta nel cosiddetto debugging dei programmi, ovvero in quegli strumenti capaci di scoprire le «marachelle» dei programmatori. Purtroppo, sfortuna volle che un certo signore (i lettori che si sono interessati di calcolabilità dovrebbero ricordarne il nome), abbia già tirato fuori il suo bel teorema che dice più o meno così: «Rimboccatevi le maniche e cercateveli ad uno ad uno questi bug, perché se aspettate di inventare un metodo che sappia dirvi se ce ne sono o meno, invecchierete!».

Morale della favola: i sistemi corretti non esistono (nessuna macchina è migliore del suo inventore), per cui con i guasti dovremo sempre fare i conti.

Ma tutto ad un tratto che ti tirano fuori questi benedetti informatici?

La tolleranza ai guasti

Potremmo azzardarci un po' e chiamarlo «ciclo di vita del guasto». Nome a parte, il processo esiste davvero e può essere schematizzato nel modo seguente:

GUASTO → ERRORE → INSUCCESSO

La tolleranza ai guasti spezza la cate-

na nell'anello di congiunzione tra errore ed insuccesso. Ma tiriamone fuori la definizione esatta: la TOLLERANZA AI GUASTI è la capacità, che può avere un sistema, di continuare a funzionare correttamente, eventualmente sacrificando un po' d'efficienza, anche in presenza di guasti.

In altre parole: scoprire gli errori e trattarli in modo che essi non si manifestino come insuccessi.

Per farlo occorrono 4 fasi:

1 Rilevazione dell'errore
2 Valutazione del danno
3 Ripristino dell'errore
4 Trattamento del guasto.

È doveroso fare una considerazione molto importante sulla tolleranza ai guasti e cioè che: i meccanismi per la sua realizzazione sono fondamentalmente applicabili a qualsiasi livello del sistema. Questa caratteristica scaturisce dall'aver fondato la tolleranza ai guasti su un concetto, lo stato del sistema visibile ad un certo livello, che è esso stesso applicabile uniformemente a qualsivoglia livello del sistema.

Consideriamo più in dettaglio le fasi sopracitate. Dovrebbe essere chiaro che se vogliamo trattare gli errori in modo tale che essi non si manifestino come insuccessi, dobbiamo riuscire a scoprire quando gli errori si verificano. Secondo la definizione data precedentemente un errore si verifica nel momento in cui, l'esecuzione di una certa operazione su un certo oggetto, fa transire il sistema in uno stato diverso da quello che ci si attendeva in conformità alle specifiche del sistema stesso. Rilevare gli errori consiste allora nell'accorgersi che lo stato interno del sistema è errato. Ora, poiché non necessariamente esistono valori dello stato interno errati in assoluto, dobbiamo fare in modo che ciò accada; un modo per farlo è quello di inserire ridondanza negli stati interni, in modo che essi, oltre a contenere le informazioni necessarie, contengano anche delle informazioni di controllo. Senza entrare nei dettagli delle possibili informazioni di controllo ai vari livelli, diremo che esse sono verificabili a Runtime mediante opportuni test capaci di stabilire se tali informazioni sono corrette o meno. In altri termini, quello che si fa è di effettuare il controllo su una parte dello stato che serve solo a questo scopo. Per fare un esempio, potremmo tirare fuori il cosiddetto «bit di parità»: un bit aggiunto ad un blocco di altri bit, che deve valere 1 se la somma degli altri bit è dispari, 0 altrimenti. In tal modo il controllo è più semplice da effettuare, proprio perché è un controllo standard. Attenzione: è evidente che un controllo siffatto potrebbe fallire; non necessariamente un errore presente nel blocco di bit considerati può essere rilevato dal bit di parità.

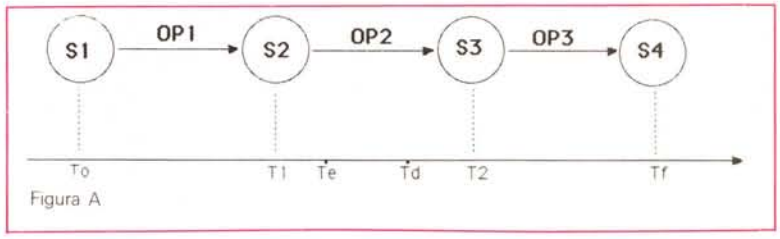

Figura A

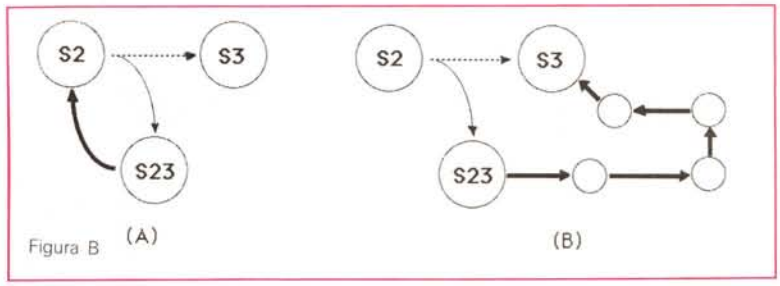

Figura B

Figura B - Le due fondamentali tecniche di ripristino dell'errore: Backward Error Recovery (A) riassegna al sistema lo stato precedentemente memorizzato; Forward Error Recovery (B) provoca ulteriori transizioni di stato del sistema fino a farlo giungere nello stato S3.

L'esempio fatto è quello di un diffuso meccanismo di rilevazione degli errori a livello delle comunicazioni tra moduli hardware; questo meccanismo implementa a livello hardware, una politica di rilevazione, quella delle informazioni di controllo, che è sicuramente applicabile a qualunque livello del sistema. Questo vale per tutte le politiche di tolleranza ai guasti.

Abbiamo già detto che l'intero stato del sistema è troppo vasto per poterci sviluppare sopra dei meccanismi di tolleranza ai guasti, e per semplificarlo abbiamo considerato lo stato del sistema come formato solo dagli stati significativi, ignorando gli altri, ad un certo livello. Per semplificare ancora di più, osserviamo che ogni stato significativo ad un certo livello, è composto dall'insieme degli stati assunti da ognuno degli oggetti virtualizzati da quel livello. Per cui, un sistema tollerante ai guasti viene ad essere un sistema che, ad un certo livello Li, non virtualizza semplici oggetti Oi, ma oggetti resilienti; in altre parole il livello Li non solo interpreta le operazioni sugli Oi, ma ne assicura un'esecuzione tollerante ai guasti. I meccanismi di rilevazione degli errori, a questo punto, possono limitarsi a porre sotto controllo, separatamente, ognuno degli oggetti resilienti. Queste semplificazioni tuttavia, comportano la necessità di sviluppare politiche di valutazione del danno (fase 2 della tolleranza ai guasti) per riuscire a scoprire se l'errore verificatosi in un certo oggetto ha, o meno, contaminato, altri oggetti. Di solito però, queste politiche, non vengono tradotte in espliciti meccanismi applicati a tempo di esecuzione, ma incorporate in quelle delle altre fasi.

Non intratteniamoci oltre sulla fase 2, né sulla fase 4 cioè quella del trattamento del guasto, limitandoci a dire, a proposito di quest'ultima, che essa consiste nella riconfigurazione del sistema, giungendo addirittura, nel caso di sistemi da utilizzare in applicazioni veramente critiche (si pensi alle centrali nucleari ed ai satelliti), all'automatica sostituzione dei componenti guasti.

Occupiamoci invece, della fase 3: la chiave di volta della tolleranza ai guasti.

L'Error Recovery

Abbiamo visto che un sistema tollerante ai guasti è un sistema «con gli occhi aperti». Abbiamo detto cosa, questi occhi aperti, devono tenere sotto controllo. Abbiamo capito che in tal modo esso è capace di scoprire gli errori ovunque essi siano annidati. Ma che vuol dire: trattarli in modo che essi non si manifestino come insuccessi?

Osserviamo la figura A. Questa volta S1, S2, S3 ed S4, non rappresentano l'intero stato del sistema ma quella parte di stato che è lo stato interno proprio dell'oggetto sul quale sono in esecuzione le operazioni op1, op2 ed op3; la

parte rimanente dello stato del sistema può essere trascurata supponendo che non sia coinvolta dalle tre operazioni considerate.

Le tre operazioni provocano, o dovrebbero provocare nel rispetto delle specifiche del sistema, una serie di transizioni di stato che portano dallo stato S1 allo stato S4.

In realtà gli stati intermedi assunti dall'oggetto non sono solo S2 ed S3, ma anche una serie di altri stati non significativi. Con riferimento all'asse temporale presente in figura A, supponiamo che un guasto presente nel sistema, provochi un errore nell'istante Te, e che tale errore venga rilevato nell'istante Td. Abbiamo detto che il guasto è presente nel sistema, in quanto non è né possibile né importante stabilire l'istante in cui un guasto si verifica, così come non si può stabilire quanto tempo intercorre tra il verificarsi di un guasto ed il suo manifestarsi come errore (potrebbe anche non provocare mai errori). In seguito all'istante Td il controllo del sistema viene affidato ai meccanismi di Error Recovery.

Un corretto funzionamento avrebbe consentito all'esecuzione dell'operazione op2 di far transire l'oggetto dallo stato S2 allo stato S3. In seguito all'errore invece, l'oggetto ha raggiunto uno stato interno, chiamiamolo S23, non solo non significativo, ma erroneo.

Esistono a questo punto due possibilità per ripristinare lo stato del sistema: riportare l'oggetto nello stato S2 oppure «trascinare» l'oggetto fino allo stato S3. A queste due possibilità corrispondono due politiche di Recovery:

Un po' di definizioni

La strutturazione a livelli

Un sistema d'elaborazione è rappresentabile mediante la definizione di tutti i livelli da cui esso è composto.

Con riferimento alla figura 1, osserviamo come i livelli del sistema siano sovrapposti l'uno all'altro; il loro numero è imprecisato in quanto ogni livello può essere implementato esso stesso mediante ulteriori livelli.

Ciò che caratterizza in linee generali il livello Li, è: Oi che è l'insieme degli oggetti (o risorse) fornite dal livello, le primitive in Pi, che non sono altro che le operazioni eseguibili sugli oggetti del livello mediante chiamate fatte dai livelli superiori, ed il linguaggio LANi attraverso il quale sono implementate le primitive di Pi.

Per evitare che le cose restino troppo campate in aria, osserviamo la figura 2 in cui si è riportato un esempio di quali potrebbero essere i livelli di un sistema d'elaborazione.

Consideriamo il livello L3 che corrisponde a quello del Nucleo del sistema operativo. Tipici oggetti appartenenti ad O3, cioè messi a disposizione dei livelli superiori, sono i processi. Una delle primitive di P3 potrebbe allora essere l'attivazione di un processo. Si pensi all'unità a dischi di un computer.

Il software necessario al suo funzionamento è proprio uno di tali processi: il gestore dell'unità. È evidentemente impensabile mantenere tale processo costrantemente in esecuzione. Ecco allora che in un programma scritto in LAN4, si incontra l'invocazione della primitiva d'attivazione di processo, la quale è implementata da un programma scritto in LAN3 e costituito, probabilmente, da tante invocazioni di primitive appartenenti a P2, cioè da istruzioni Assembler. Queste ultime, a loro volta, non sono altro che primitive di P1, ognuna di esse quindi consisterà in una sequenza di microistruzioni, appartenenti al linguaggio di microprogrammazione LAN1, che verranno infine implementate direttamente in hardware da un sistema di reti sequenziali.

Già da queste brevi considerazioni, si capisce come la progettazione dei sistemi di elaborazione sia un processo che abbraccia molti aspetti. Se si pensa poi, che i progettisti dei vari livelli sono quasi sempre diversi fra loro, anche se spesso appartengono alla stessa azienda, si comprende come la strutturazione a livelli sia un'astrazione necessaria per avere una visione uniforme di tutti gli aspetti riguardanti l'architettura dei sistemi di elaborazione.

La necessità di una visione uniforme, assume una grande importanza nel momento in cui si debbano prevedere meccanismi e politiche di funzionamento, a loro volta uniformi, nel senso di applicabili indistintamente ai vari livelli. È il caso in cui ci si trova quando si vogliono progettare dei sistemi affidabili.

Interpretazione e virtualizzazione

Per maggiore chiarezza è il caso di spiegare un attimo il significato di queste due parole: interpretare e virtualizzare; anche per poter accennare ad un'altra parola che resterà in secondo piano in questa rassegna: tradurre.

L'interpretazione è il processo consistente nell'esecuzione delle istruzioni scritte nel linguaggio di un certo livello del sistema, mediante l'esecuzione di altri programmi scritti in qualcuno dei linguaggi dei livelli sottostanti.

Così è possibile che un programma applicativo scritto in Pascal che supponiamo essere, con riferimento alla figura 2, il linguaggio del livello L5, venga implementato con una soluzione di tipo interpretativo, così come è possibile utilizzare una soluzione di tipo compilativo (o traduttivo). In tal caso il programma scritto in Pascal verrà tradotto, a tempo di compilazione e non a tempo d'esecuzione, in un programma scritto nel linguaggio di uno dei livelli inferiori.

Interpretare e compilare dunque, sono i due modi per scendere di livello nel sistema. Virtualizzare sta ad indicare esattamente la cosa opposta.

La virtualizzazione consiste nella stesura di un certo numero di primitive richiamabili dall'esterno, capaci di simulare o, per l'appunto, virtualizzare l'esistenza di un insieme di risorse o oggetti o tipi di dato che dir si voglia.

Per concludere, osserviamo la figura 3. In essa è mostrato il livello Li al quale è disponibile il linguaggio LANi; gli oggetti manipolabili in LANi sono quelli virtualizzati dal supporto a tempo d'esecuzione STE(Li), mentre tale supporto interpreta le operazioni invocate dai programmi residenti in Li.

La macchina a stati

Il passo successivo è ora quello di estrapolare da uno dei livelli del sistema di elaborazione, una caratteristica che vedremo essere comune a tutti gli altri: la rappresentabilità mediante stato interno.

Il livello del sistema del quale si parla è quello costituito dai microprogrammi codificati all'interno di una ROM che è parte di quel sistema di reti sequenziali universalmente conosciuto con il nome di PROCESSORE.

Questo livello è quello immediatamente

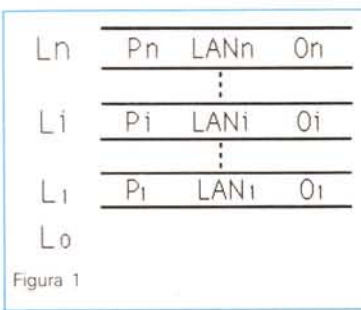

Figura 1

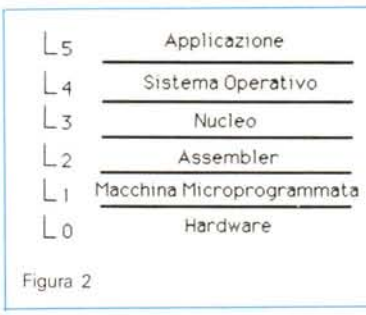

Figura 2

1 Backward Error Recovery
2 Forward Error Recovery.

La prima consiste nel provocare la transizione da S23 ad S2, mentre la seconda provocherebbe la transizione da S23 ad S3. Da qui deriva il nome delle due strategie e cioè: ripristino dell'errore «verso dietro» (backward) o «verso avanti» (forward). La figura B schematizza il funzionamento delle due politiche.

La politica forward è applicabile solo in casi particolari, mentre quella backward, essendo più generale, è la più diffusa fra le due.

Per poter implementare Backward Error Recovery è necessario memorizzare lo stato dell'oggetto in qualche apposita struttura dati, ogni volta che una nuova operazione dev'essere eseguita sull'oggetto. Solo nel caso in cui l'operazione termina con successo, il vecchio stato dell'oggetto, può essere scartato, memorizzando il nuovo stato al suo posto.

Prima di concludere, accenniamo ad una tecnica che realizza contemporaneamente sia la rilevazione dell'errore che il suo ripristino. Consiste nel replicare gli oggetti ed eseguire le operazioni su ognuna delle copie. Se ogni oggetto è triplicato (questa particolare tecnica è conosciuta con il nome di «ridondanza modulare tripla»), è possibile applicare un modulo votatore avente lo scopo di confrontare gli stati di ognuna delle tre copie dell'oggetto, in modo da rilevare un eventuale errore, in una delle copie, e correggerlo automaticamente assegnando ad ogni copia lo stato corretto ottenuto come il risultato di una votazione a maggioranza fra i tre.

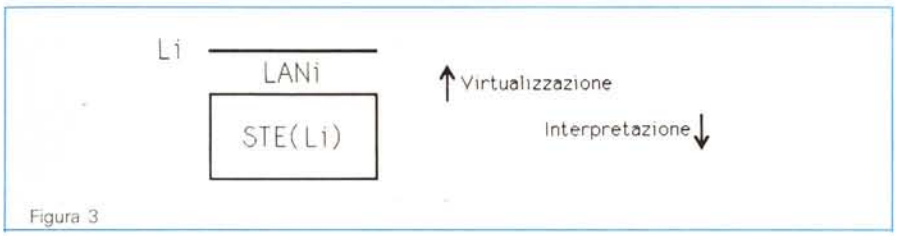

Figura 3

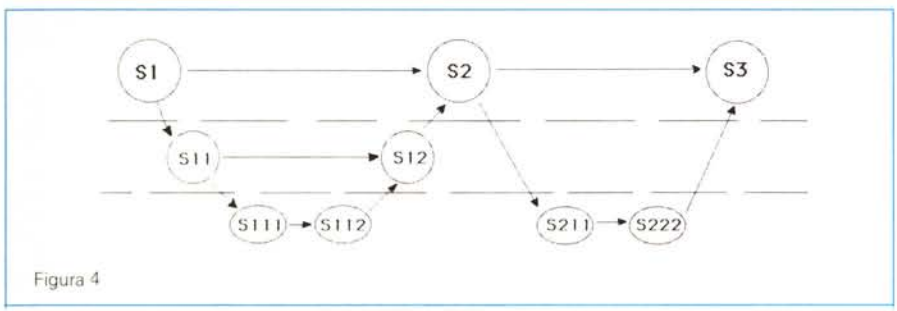

Figura 4

sottostante al livello Assembler o, per essere più precisi una volta per tutte, al livello del Linguaggio Macchina.

Un sistema di reti sequenziali, come è noto ai lettori che hanno avuto modo di interessarsi a questi aspetti, può essere interamente descritto mediante tabelle; una di esse descrive interamente in modo in cui il sistema evolve nel tempo in conseguenza degli INPUT ricevuti, in modo tale che, in qualsiasi istante, è possibile conoscere quale sia lo stato interno del sistema e quindi sapere tutto di esso.

In generale, l'idea di una macchina che istante per istante è interamente caratterizzata dal suo stato interno (da qui il nome di macchina a stati), è applicabile anche ai livelli superiori. Ad esempio, dato un processo appartenente al nucleo del sistema operativo, è possibile caratterizzarlo interamente tramite uno stato interno scelto ad hoc, come quello costituito dall'insieme dei valori presenti nella struttura dati «descrittore di processo» ed altri. Data una coppia di processi, allora, anche essa può essere considerata come avente uno stato interno, e precisamente quello costituito dalla coppia i cui elementi sono gli stati interni di ognuno dei due processi.

Un intero sistema di elaborazione, in questo modo, può essere visto come una macchina a stati.

Il vantaggio di avere un modello come quello della macchina a stati, cui affidarsi per rappresentare un sistema, sta nel fatto che in tal modo è possibile sempre conoscere lo stato d'avanzamento del sistema semplicemente conoscendo un insieme, per quanto grande esso sia, di valori: il suo stato interno.

Un'ultima complicazione

In effetti è vero: lo stato interno dell'intero sistema è troppo vasto e complesso per poterci sviluppare sopra i meccanismi di tolleranza ai guasti cui vogliamo andare a parare.

La strutturazione a livelli ci viene ancora una volta in aiuto, sia per spiegare il problema, sia per risolverlo.

Osserviamo la figura 4. Il sistema si trova, in un certo istante nello stato S1. A livello L del sistema, non interessa stabilire quale, sono invocate due operazioni; la prima fa transire il sistema da S1 ad S2 e la seconda da S2 ad S3. Le due operazioni tuttavia, richiedono del tempo per essere eseguite; inoltre esse sono interpretate da programmi dei livelli sottostanti in modo tale che l'esecuzione della prima operazione comporti l'esecuzione di tre operazioni del livello L-1 (spero che sia chiara la notazione relativa) che provocano le transizioni di stato: da S1 a S11 da S11 ad S12 e da S12 ad S2. A sua volta l'operazione che fa transire il sistema da S11 ad S12 è in realtà implementata mediante l'esecuzione di tre operazioni di livello L-2. Per quanto riguarda l'operazione che porta da S2 a S3, dalla figura si vede che essa è implementata direttamente in L-2, attraverso tre operazioni.

Non è il caso di nascondere che l'esempio presentato è altamente approssimativo. Esso potrebbe essere un esempio reale solo se il livello L considerato fosse un livello bassissimo del sistema, nel senso che è molto vicino a quello della macchina hardware. Del resto è impossibile presentare un esempio reale di transizione di stato a livelli alti, in quanto gli stati intermedi assunti dagli oggetti dei livelli inferiori hanno più o meno una crescita esponenziale, e sarebbe impossibile poterli seguire tutti.

Per farla breve: se noi volessimo guardare nel «profondo» di un computer, in un istante qualsiasi e volessimo rappresentarne lo stato interno, il più delle volte dovremmo arrenderci.

Ben diversi sarebbero i risultati se noi, aguzzando l'ingegno, ci mettessimo ad osservare, nel loro evolversi, gli oggetti di un preciso livello, e tenessimo traccia degli stati interni significativi per gli oggetti di quel livello, classificando come non significativo tutto ciò che, in corso di esecuzione nei livelli sottostanti, non ha ancora prodotto effetti visibili al livello considerato.

Sempre con riferimento alla figura 4, la nostra strategia ci permetterebbe così di sapere che il sistema si trova nello stato S1 fintantoché esso non transisce nello stato S2.

Detto in altre parole, quello che noi stiamo tentando di fare è di rendere «atomiche», nel senso di immediate, le operazioni del livello considerato, e le rendiamo atomiche in un modo molto semplice: «tenendo traccia».

Le pubblicazioni Technimedia

AUDIOREVIEW
La più qualificata rivista italiana di elettroacustica ed alta fedeltà

MCMICROCOMPUTER
La più diffusa e più autorevole rivista italiana di informatica

OROLOGILE MISURE DEL TEMPO
La prima rivista per chi conosce il valore del proprio tempo

Technimedia
Via Carlo Perrier, 9 - 00157 Roma - Tel. 06/4513931