Honeywell

Honeywell Information Systems Italia

Via Laboratori Olivetti - Pregnana Milanese

UNIVERSITA'
DEGLI STUDI DI MILANO
ISTITUTO DI CIBERNETICA
Via Viotti, 5 - Milano

10/11

NOTE DI SOFTWARE

UNIVERSITA'
DEGLI STUDI DI MILANO
ISTITUTO DI CIBERNETICA

Honeywell
Honeywell Information Systems Italia

Note di software.
è un bollettino trimestrale, edito a cura del Centro di Ricerca e Progettazione della Honeywell Information Systems Italia e dell'Istituto di Cibernetica dell'Università di Milano.

Note di software.
intende costituire uno strumento per la rapida e informale circolazione di informazioni, idee ed esperienze nell'area della tecnologia del software. In particolare, intende stimolare il dibattito sulle metodologie orientate alla diminuzione del rapporto costo/qualità dei programmi, e sugli strumenti atti ad automatizzare l'attività di produzione del software in ogni suo aspetto.

La collaborazione a **Note di software** è libera.
In particolare, si sollecitano contributi nella forma di brevi monografie o bibliografie essenziali commentate sui seguenti temi: tecniche e strumenti per la programmazione strutturata e modulare, metodologie e strumenti per il testing e il debugging dei programmi, documentazione del software, aspetti organizzativi e gestionali nella costruzione di sistemi software di grandi dimensioni.

Chi desiderasse pubblicare un contributo è pregato di prendere contatto con il Comitato di Redazione (HISI - Via ai Laboratori Olivetti - Pregnana Milanese), o di inviare direttamente il dattiloscritto in triplice copia.
I contributi non dovrebbero superare le quattromila parole.

I lavori accettati per la pubblicazione non vengono revisionati da un comitato tecnico. Pertanto, ogni contributo pubblicato riflette le opinioni personali dei suoi autori, e non necessariamente quelle del Comitato di Redazione.
Eventuali revisioni saranno effettuate di comune accordo fra il Comitato di Redazione e i proponenti.

Note di software.
viene distribuito a chi ne faccia richiesta alla redazione.

Direttore responsabile: Paolo Ghelfi
Comitato di Redazione: Roberto Polillo - Wanda Anselmetti
Redazione: Franco Soresini

10 / 11

Aprile - Settembre 1979

Indice:

Autorizzazione del Tribunale di Milano n. 87 del 23 febbraio 1977

NOTE DI SOFTWARE

QUESTO NUMERO...

Questo numero doppio di NOTE DI SOFTWARE è dedicato alle applicazioni delle reti di Petri, uno strumento semplice e generale per la descrizione dei sistemi, già presentato, ai lettori della rivista, nello scorso numero 9.

Nel primo lavoro, G. Castelli, G. Haus e M. Maiocchi presentano una metodologia, basata sulle reti di Petri, per la stesura di specifiche funzionali di procedure EDP.

C. Simone e P. Rossi, in due articoli coordinati, mostrano come le reti di Petri possano essere convenientemente utilizzate per la descrizione di sistemi di processi concorrenti: nel primo lavoro, vengono rappresentate, mediante reti opportunamente estese, le strutture di sincronizzazione fra processi più frequenti; nel secondo, viene mostrato un esempio completo, riguardante un problema tipico di gestione di un protocollo di comunicazione.

Infine, A. Beltramini propone, con un esempio, l'uso delle reti di Petri per la progettazione di un sistema (hardware e software) di controllo di processo, realizzato con microprocessori.

Abbiamo chiesto al prof. G. Degli Antoni, direttore dell'Istituto di Cibernetica dell'Università di Milano, una breve introduzione sull'interesse che riveste, oggi, lo strumento proposto da Petri.

La Redazione

Ogni epoca ha problemi. Ogni problema sollecita le strutture culturali a sviluppare concettualità e strumenti capaci di affrontare e risolvere problemi.

Così, lo sviluppo dell'industrializzazione ha portato all'organizzazione scientifica del lavoro, nel cui ambito Taylor ha suggerito strumenti concettuali quali l'organigramma ed il diagramma a blocchi. Successivamente, la possibilità di impiego di tecnologie elettroniche ha portato allo sviluppo di elaboratori elettronici, per la cui programmazione Von Neuman ha introdotto il flow-chart.

Oggi i problemi sono molti e complessi e certo è arduo caratterizzarli nel loro insieme. Ma è difficile non riconoscere che la complessità e l'interazione fra le parti delle strutture sociali e tecnologiche, assieme alla loro autonomia parziale, introduce la necessità di saper distinguere ciò che è essenziale nel controllo da ciò che non lo è.

Ne consegue la necessità di saper esprimere, analizzare ed eventualmente risolvere i problemi che da tale ordine di cose derivano.
A questo immane compito Carl Adam Petri ha dedicato la sua vita. Fra i prodotti minimi della sua fatica sono le ben note reti di Petri, accanto ad una incessante attività tendente far capire ciò che di profond e difficile c'è nella descrizione dei sistemi. Complessivamente ne è nata, quindi, una teoria generale delle reti che pretende di fornire strumenti matematici (anche semplicissimi), nel cui ambito trova posto una più vasta sistemazione delle nozioni di indipendenza parziale o concorrenza, e nel cui ambito i problemi semplici e complessi, superficiali o profondi del parallelismo possono trovare una collocazione.

Perché, ad esempio, preferire le reti di Petri a linguaggi di programmazione che descrivano la concorrenza? Perché impiegarle per descrivere sistemi hardware? Perché impiegarle nella presentazione di sistemi? Come impiegarle nella loro soluzione.?
Tutte queste domande hanno una molteplicità di risposte, per cui risulta riduttivo licenziarle con semplici slogan. Ma una merita di essere fornita. Sia che si pensi che le Reti di Petri nel loro attuale sviluppo sono uno strumento definitivo o no, sia che si pensi che sono un linguaggio, o viceversa si pensi che sono oggetti, esse sono comunque uno strumento concettuale in fase di sviluppo, facilmente acquisibile nelle sue più dirette applicazioni, e che porterà dietro di sè futuri sviluppi che sembrano rompere la tradizionale difficoltà di comunicazione fra discipline umanistiche, scientifiche, tecniche.

G. Degli Antoni

CONTRIBUTI TECNICI

UNA METODOLOGIA DI STESURA E VERIFICA DI SPECIFICHE FUNZIONALI DI PROCEDURE E PROGRAMMI SOFTWARE

G. Castelli°, G. Haus °, M. Maiocchi°[a]

Riassunto

Il presente lavoro illustra una metodologia per l'analisi di procedure software e per la costruzione di documenti di specifiche funzionali che, partendo dalla descrizione informale dei requirements, consente una descrizione non ambigua che può essere verificata sulla base delle reali necessità e che mostra le eventuali deficienze nella definizione dei requirements, nel corso dell'attività di costruzione. La metodologia, basata sull'uso delle reti di Petri per la descrizione delle procedure, è organizzata in passi successivi di due classi distinte, in cui diversi tipi di descrizioni e di tests sono interfogliati.

Abstract

The paper illustrates a methodology for the analysis of software procedures and the construction of functional specifications documents which, starting from the informal expression of the requirements, obtains a non ambiguous description, which can be checked for consistency with the stated needs and which shows possible lacks in the requirements definition, during the construction activity. The methodology, based on the use of Petri nets for the description of the procedure, is organized in subsequent steps in which different kinds of descriptions and of checks are interleaved.

° Istituto di Cibernetica dell'Università di Milano
[a] Honeywell Information Systems Italia, Pregnana Milanese

Questo lavoro è stato sviluppato nell'ambito del Communication and Programming Project, progetto di cooperazione fra l'Istituto di Cibernetica dell'Università di Milano e la Honeywell Information Systems Italia. Esso è stato precedentemente presentato alla III Giornata di Studio su "La Programmazione Strutturata: metodologia e strumenti per l'analisi e il disegno" (Milano, 23-25 maggio 1979), e pubblicato negli atti di tale convegno.

Gli autori ringraziano G. Degli Antoni per i contributi determinanti allo sviluppo del lavoro ed inoltre S. Ruzzini, L. Zanella e B. Zonta per le fattive discussioni. Un particolare ringraziamento alla Società TEMA di Bologna ed a O. Sticchi soprattutto, per l'applicazione sperimentale della metodologia in ambiente produttivo.

1. INTRODUZIONE

L'attuale diffondersi di metodologie di programmazione, come Warnier [1], Jackson [2] e PHOS [3], fornisce una guida per la costruzione di un singolo programma, a partire dalle sue specifiche funzionali. Introdurremo una metodologia utile per la costruzione di specifiche funzionali per una procedura software completa fino al livello di singolo programma, partendo dai requirements fissati per la soluzione del problema.

La metodologia si basa sull'uso delle reti di Petri per la descrizione delle diverse attività della procedura e delle loro interazioni, su uno standard di documentazione, basato sulla tecnica PSPN, e su un preciso insieme di regole che guidano la costruzione del documento.

Le regole prevedono, secondo uno sviluppo top-down:

— i passi operativi da seguire;
— le verifiche sulle descrizioni ottenute;
— l'integrazione del documento.

2. RETI DI PETRI

Una rete di Petri è un grafo bipartito in cui si possono riconoscere due tipi di nodi: i *posti* e le *transizioni*. Archi orientati congiungono i due tipi di nodi in modo che non possano essere connessi elementi dello stesso tipo. La figura 1 rappresenta una rete di Petri formata da due transizioni, una con due posti di ingresso ed un posto di uscita, l'altra con un posto di ingresso, condiviso con la transizione precedente, e con due posti di uscita.

Le reti di Petri possono essere interpretate come la

RETRI DI PETRI

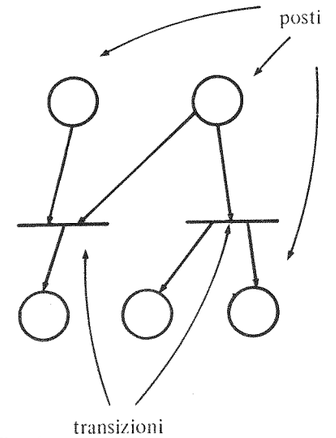


Fig. 1

descrizione di un processo in cui gli eventi possono verificarsi in dipendenza da un insieme di condizioni che determinano altri eventi: in particolare, ogni posto può essere interpretato come una *risorsa* ed ogni transizione come una *attività*: ogni attività può aver luogo solo quando sono presenti le sue risorse di ingresso ed il suo compimento produce le risorse di uscita, consumando quelle di ingresso (fig. 2).
In tale modo, le reti di Petri possono essere utilizzate per rappresentare processi produttivi ed, in particolare, programmi connessi in una procedura.

La presenza di una risorsa in una rete è rappresentata per mezzo di una *marca* in un posto (indicata con un punto) (fig. 3) e l'evoluzione temporale di una rete può essere rappresentata tramite il flusso delle marche nella rete stessa: quando una attività "scatta", viene eliminata una marca da ogni posto di ingresso e viene posta una marca in ogni posto di uscita. Per esempio, la rete della figura 3 rappresenta un processo in un laboratorio di analisi mediche: quando lo *sportello* è libero ed un *richiedente* è presente, l'operazione di *accettazione* può aver luogo producendo un *modulo di richiesta* che, insieme alla presenza di un *dottore* e di un *paziente*, può far scattare il *prelievo*, producendo un *campione* ed aggiornando il *modulo di richiesta*; l'attività di *refertazione* deve attendere la generazione di un risultato attraverso l'attività di *analisi*, producendo il *referto* finale.

INTERPRETAZIONE

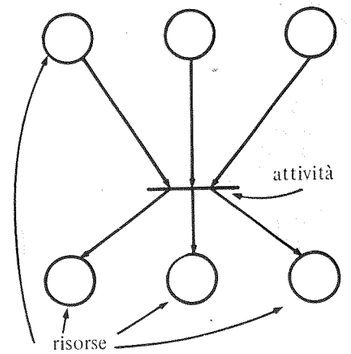


Fig. 2

3. MODIFICHE GRAFICHE DELLE RETI DI PETRI

Sono stati introdotti due tipi di modifiche grafiche delle reti di Petri [4] al fine di renderle più adatte ai nostri propositi: una consiste nell'utilizzare una rappresentazione altamente evocativa, l'altra in un insieme di reti in forma sintetica per rappresentare alcune situazioni comuni.

Nella figura 4 è rappresentata una rete evocativa, in cui ogni risorsa è stata rimpiazzata con un simbolo grafico che rappresenta l'entità del problema in forma mnemonica, e ciascuna transizione è stata rappresentata con una "scatola" che contiene la descrizione della trasformazione realizzata: la rete rappresenta in modo più conciso il lavoro del laboratorio suddetto, in cui la fase di accettazione e di prelievo sono state sintetizzate in una sola.

Il secondo tipo di modifica fa riferimento alla capacità di rappresentare l'alimentazione parallela o alternativa delle transizioni o la generazione delle risorse: per esempio, la prima rete della figura 5 è adatta a rappresentare le operazioni su un file A che può produrre o un file B o un messaggio di errore C; la seconda rete può rappresentare due programmi X e Y che accedono simultaneamente al file A; e così via.

Le estensioni della forma grafica illustrate non sono estensioni delle possibilità delle reti di Petri: ogni nuova notazione, infatti, può essere espansa

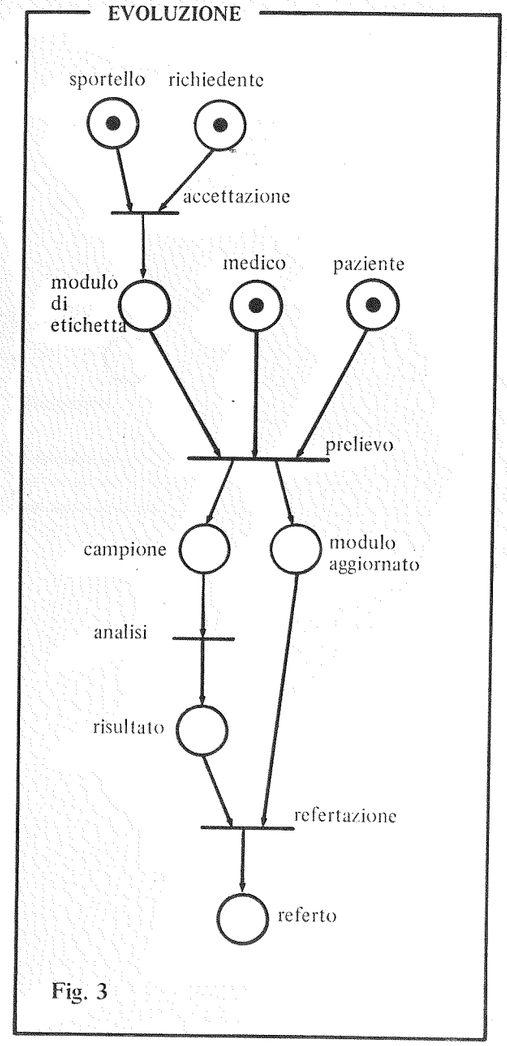


Fig. 3

in reti di Petri tradizionali, in cui tuttavia è difficile associare un significato, in termini di entità del problema, a ciascun elemento della rete; per esempio, la figura 6 mostra la costruzione della prima rete della figura 5 per mezzo delle reti di Petri usuali: nessun significato può essere associato alla risorsa α ed alle transizioni X e ρ.

La figura 7 mostra un'ulteriore estensione in cui è possibile riconoscere intuitivamente che la risorsa A produce, per mezzo dell'attività X, alternativamente la risorsa D o la coppia di risorse B e C.

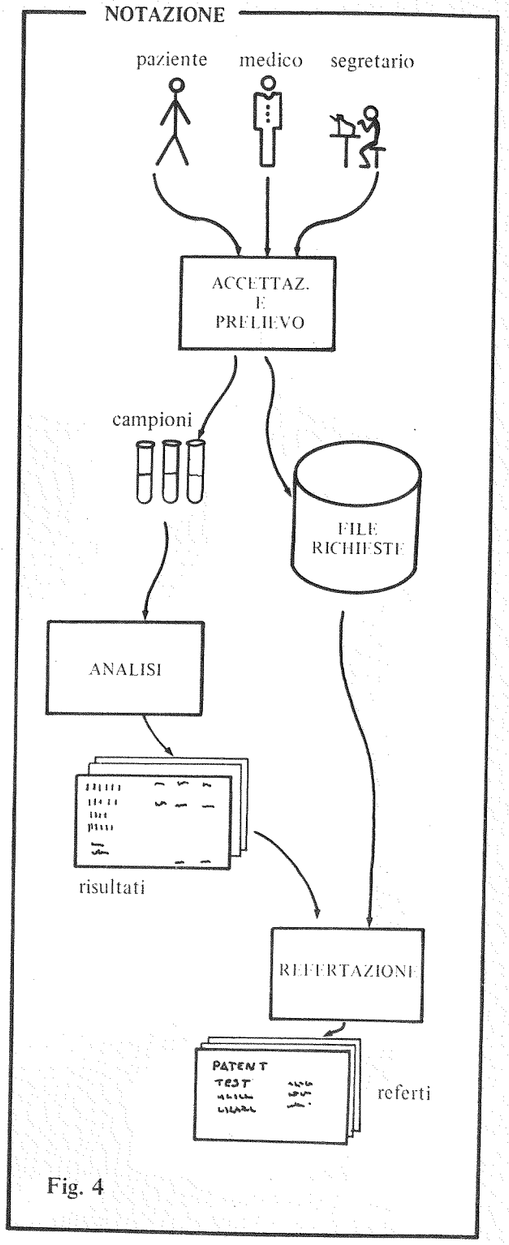


Fig. 4

4. LA METODOLOGIA

La metodologia proposta prevede la presenza di due tipi distinti di moduli PSPN: moduli in cui sono descritte reti composte da una sola attività e

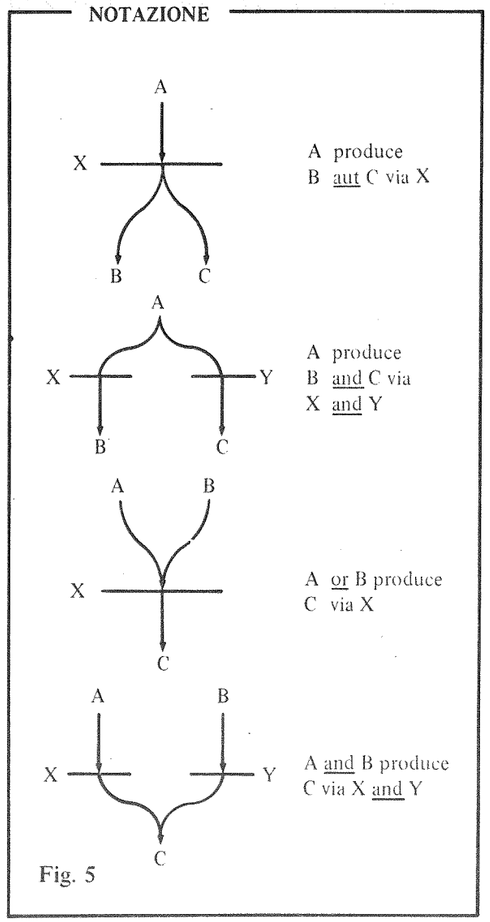


Fig. 5

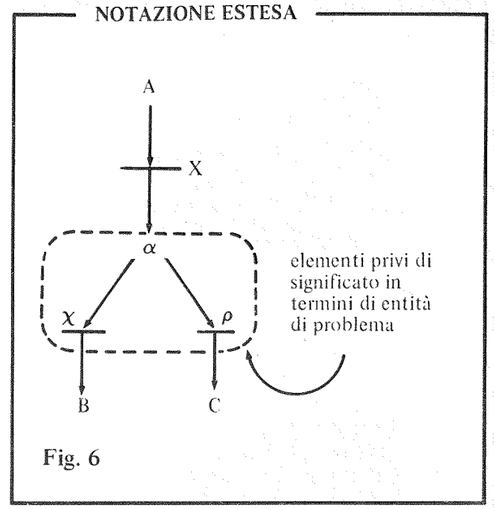


Fig. 6

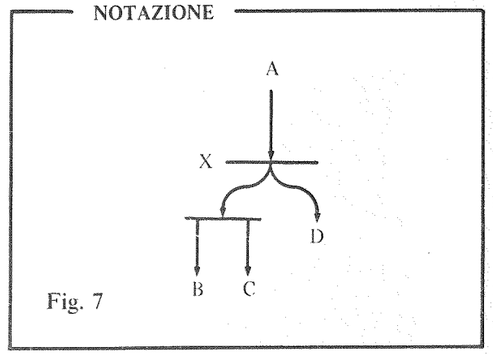


Fig. 7

moduli in cui sono descritte reti composte da più attività; i primi conterranno informazioni sulle caratteristiche delle risorse di ingresso e di uscita e sulla trasformazione operata per mezzo della transizione; i secondi conterranno informazioni sulle possibili interazioni tra le attività.

La metodologia è costituita dei seguenti passi:

1. descrizione della procedura come una unica transizione di una rete di Petri;
2. descrizione linguistica della rete di Petri di cui al punto 1., con una definizione concisa delle caratteristiche di ogni risorsa di ingresso o di uscita e con una descrizione il più possibile non procedurale della trasformazione realizzata;
3. creazione del dizionario dei termini particolari della procedura;
4. raffinamento della rete di Petri del livello precedente in un insieme ridotto di transizioni che condividono le risorse (eventualmente raffinate);
5. verifica della correttezza dei fini, attraverso un'ispezione della rete sia dal punto di vista locale della grafica che dal punto di vista strutturale; controlli delle semaforizzazioni (eventuali);
6. descrizione linguistica della rete con definizione delle relazioni tra le diverse transizioni della rete;

7. controlli linguistici sulla descrizione del punto precedente, sia da un punto di vista lessicale che sintattico;
8. aggiornamento del dizionario dei termini particolari della procedura;
9. isolamento di ogni transizione contenuta nella rete di cui al punto 4 eliminando ogni risorsa od arco non in relazione con essa;
10. per ogni transizione isolata:
    - 10.1. descrizione linguistica della rete ottenuta per isolamento, con una definizione concisa delle caratteristiche di ogni risorsa di ingresso o di uscita e con una descrizione il più possibile non procedurale della trasformazione realizzata;
    - 10.2. controlli linguistici sulla descrizione del punto 10.1., sia da un punto di vista lessicale che sintattico;
    - 10.3. aggiornamento del dizionario dei termini particolari della procedura;
    - 10.4. esecuzione dei passi da 4. a 10. se la transizione in oggetto è ulteriormente raffinabile;
11. descrizione della struttura logica dei files in Pascal.

Il processo di costruzione e verifica delle specifiche è definito in forma ricorsiva; esso ha termine quando non esistono più transizioni raffinabili.

4.1. Descrizione della procedura top-level

In questa fase l'intera procedura è rappresentata come un'unica transizione e le risorse coinvolte sono indicate in maniera concisa, considerando come una sola entità le risorse simili per uso o per natura (per esempio, tutti i files su disco sono indicati come un'unica risorsa). La figura 8 mostra un esempio di un sistema di terminali "cash & carry" che possono operare concorrentemente per realizzare operazioni di fatturazione: l'intera procedura è vista come un'unica transizione che ha come risorse di ingresso:

— il sistema inattivo che sarà attivato tramite comandi di inizializzazione;
— comandi di esecuzione, che permettono le operazioni di fatturazione da ciascun terminale, e che accedono a un insieme di files contenenti le informazioni relative ai clienti, ai prodotti, ecc.

Le risorse sono:

— le fatture stampate;

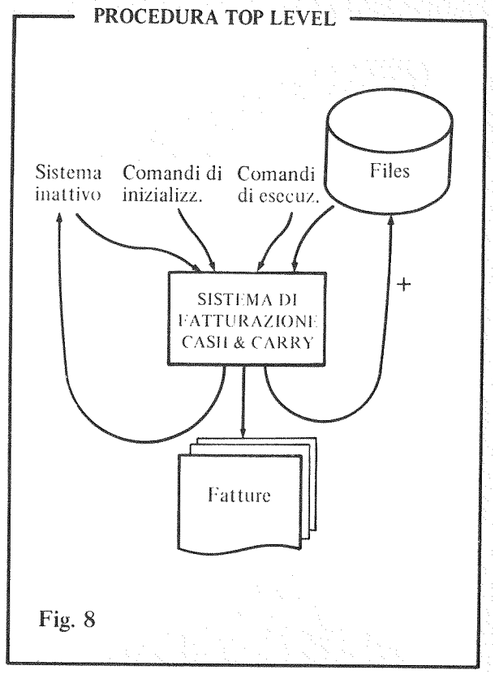


Fig. 8

— il sistema di terminali che ritornerà in uno stato inattivo;
— i files, che dopo le operazioni si troveranno in condizione aggiornata (il simbolo "+" associato all'arco indica un'operazione di aggiornamento).

4.2. Descrizione linguistica top-level

La fase 4.2. richiede la costruzione di una pagina descrittiva per il completamento di un modulo PSPN(°), in cui:

— ciascuna risorsa è descritta per mezzo delle sue componenti e del loro significato in relazione alle entità dei problemi;
— la trasformazione della transizione è specificata in una forma riassunta.

(°) La tecnica di documentazione PSPN [5], [7] richiede la costruzione di documenti costituiti da un insieme di moduli PSPN, ciascuno dei quali è costituito da una coppia di pagine, la sinistra delle quali contiene schemi riassuntivi, disegni o slogan, mentre la destra contiene il dettaglio verbale indentato della sinistra.

4.3. Creazione del dizionario

Il linguaggio utilizzato per le specifiche di sistemi software contiene spesso espressioni settoriali o dialettali: le espressioni dialettali possono essere usate in generale nella letteratura software o possono essere peculiari di un ambiente particolare, eventualmente di una sola azienda. In tal caso, è molto importante raccoglierli in un dizionario che specifica le parole non comuni ed il loro significato e che comparirà come un'appendice al documento di specifiche funzionali.

4.4. Raffinamento della rete

La fase di raffinamento consiste nell'esplosione dell'unica transizione che si considera in una rete in cui appaiono più transizioni, in corrispondenza con le diverse fasi di attività: l'esempio della figura 8 viene sviluppato come illustrato dalla figura 9, in cui vengono individuate tre diverse attività:

— la prima è l'inizializzazione del sistema che permette di avere terminali attivi in grado di accettare comandi;
— la seconda è la riconfigurazione del sistema che permette di attivare nuovi terminali o disattivare quelli attivi;
— la terza è la fase operativa con i terminali attivi a scopo di fatturazione.

In questa fase, le risorse della descrizione precedente sono correttamente distribuite tra le diverse transizioni, e la rete indica le relazioni di dipendenza temporale delle diverse attività.

È ora necessario analizzare la rete ottenuta per verificare la correttezza degli scopi.

4.5. Verifica della correttezza degli scopi

La rete descrive completamente la sincronizzazione temporale tra le diverse attività della procedura; è possibile quindi verificare la correttezza della descrizione in relazione ai requirements del problema. In particolare, la rete impone una precisa descrizione degli aspetti di sincronizzazione, cosicchè eventuali deficienze nei requirements saranno riconosciute nella rete come scelte di comportamento che devono essere analizzate per la approvazione.

L'attività di controllo è realizzata tramite due tabelle di verifiche che si riferiscono all'aspetto lessicale e sintattico della rete.

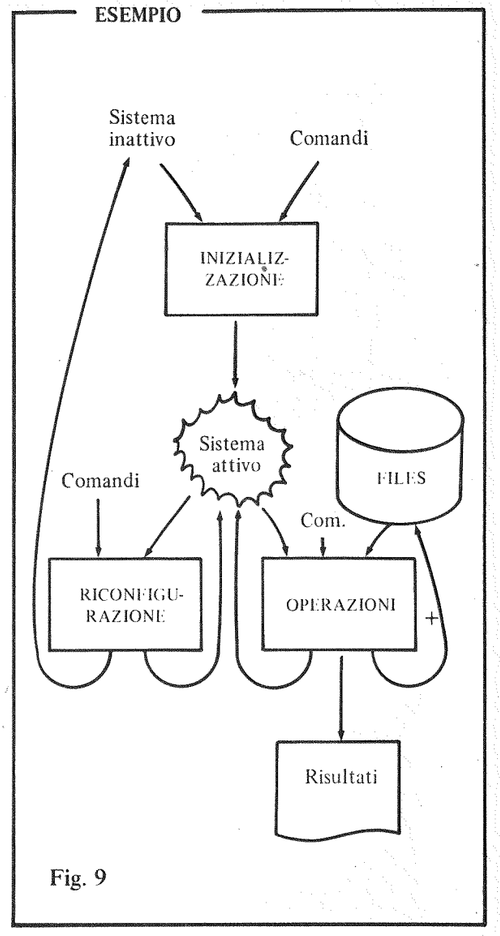
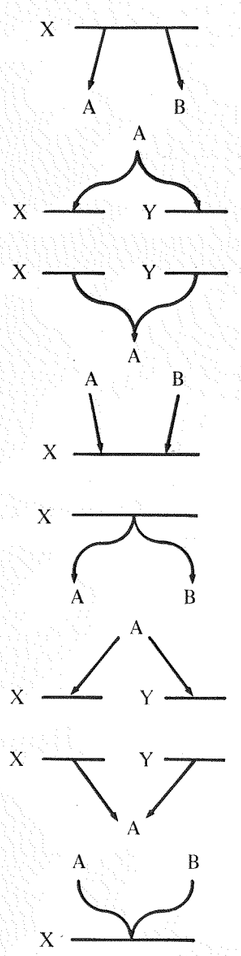


Fig. 9

La tabella illustrata nella figura 10 riassume ogni possibile situazione di una singola transizione e delle relative risorse di ingresso o di uscita; con questa tabella possiamo esaminare la correttezza della rete della figura 9:

— la *inizializzazione* richiede il sistema inattivo *e* qualche comando di inizializzazione (corretto), e genera il sistema attivo (corretto);
— la *riconfigurazione* richiede il sistema attivo *e* qualche comando di riconfigurazione introdotto da terminale (corretto), ma non può produrre simultaneamente il sistema attivo *ed* il sistema inattivo come descritto nella rete; si deve quindi introdurre una generazione alternativa; l'errore riscontrato è evidentemente dovuto ad un errore del progettista e non si possono, a

TAVOLE DI CONTROLLO



1. Generazione parallela
2. Alimentazione parallela
3. Generazione congiunta
4. Alimentazione congiunta
5. Generazione alternativa (aut)
6. Alimentazione alternat. (aut)
7. Generazione alternativa (vel)
8. Alimentazione alternativa (vel)

Fig. 10

questo punto, rilevare deficienze nei requirements;
- le *operazioni* richiedono il sistema attivo *e* i files *e* qualche comando operativo introdotto da terminale per avere le fatture (corretto), e generano i risultati (le fatture stampate), aggiornando i files *e* rilasciando il sistema attivo (corretto);
- il *sistema attivo* può alimentare alternativamente l'attività di riconfigurazione o l'attività delle operazioni, come mostrato in figura; non è possibile stabilire se la descrizione è corretta o meno in questo punto, ma è stata fatta una scelta nel disegno della rete: questo è stato reso possibile da una deficienza dei requirements; infatti, non è accettabile che l'attivazione di un terminale richieda la sospensione delle operazioni agli altri terminali: la rete deve quindi essere cambiata con una situazione di alimentazione parallela.

Il secondo tipo di verifiche da effettuare è illustrato dalla figura 11 in cui sono elencate situazioni topologicamente più complesse.

Riferendosi al medesimo esempio, aggiornato come in figura 12 secondo le indicazioni emerse dai controlli fin qui effettuati, si devono verificare e dichiarare:

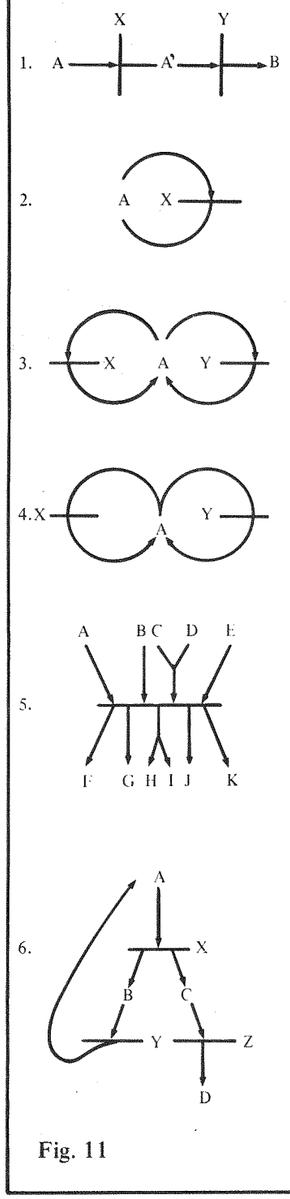
- i limiti nell'iterazione dell'attività delle operazioni (nel caso in oggetto, nessuno);
- i limiti nell'attivita di riconfigurazione (nessuno);
- la possibilità di accedere concorrentemente al sistema attivo (non ci sono problemi di sincronizzazione);
- opzionalità delle risorse: tutte sono obbligatorie.

Un'altra verifica, non ancora descritta, che deve essere realizzata, è legata alla complessità della rete: è molto difficile tenere sotto controllo una rete composta da molte risorse e transizioni, perciò è molto importante che ad ogni fase l'esplosione della rete generi reti piccole; ciò può in generale essere ottenuto per mezzo di una ridotta esplosione delle risorse (nell'esempio, l'unica risorsa espansa è il sistema, come *attivo* od *inattivo*): per esempio, negli sviluppi successivi sarà necessario distinguere i diversi files coinvolti; probabilmente sarà utile dividerli prima come files anagrafici a sola lettura e come files di lavoro aggiornati, e poi, dopo ulteriori raffinamenti, individuare le specifiche risorse come il file anagrafico dei clienti, il file di lavoro delle fatture ecc.

4.6. Descrizione della rete

La descrizione linguistica della rete, come descritta nel paragrafo 4.2., con un modulo PSPN, deve essere ripetuta per ogni livello di raffinamento: analizzeremo ora quali informazioni sono richieste in questa fase.

TABELLA DI CONTROLLO



1. Controlli devono inibire l'esecuzione di Y prima di X ?
2. Quanti cicli sono ammessi?
3. X e Y non possono essere concorrenti? Possono interfogliarsi?
4. X e Y possono esser concorrenti? Devono essere sincronizzati da semafori?
5. Tutte le risorse sono obbligatorie?
6. E' necessaria una coda per memorizzare le diverse generazioni di C?

Fig. 11

Per ogni risorsa è necessario specificare se occorrono sincronizzazioni (generalmente per le risorse residenti su disco) ed il relativo livello di inibizione (volume, record fisico, o livelli logici intermedi).

CORREZIONI (es)

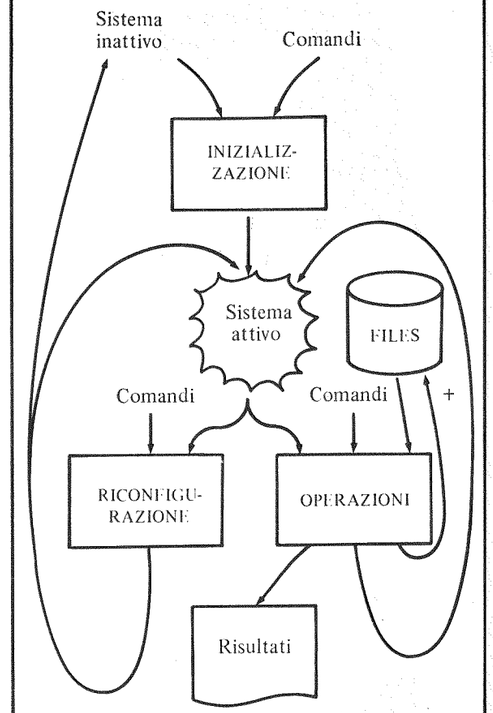


Fig. 12

4.7. Controllo linguistici

La parte destra di un modulo PSPN, cioè la descrizione linguistica della rete, deve essera anch'essa verificata: questi controlli potranno essere considerati molto noiosi, ma si riveleranno molto utili per l'individuazione delle ambiguità delle specifiche. I controlli si riferiscono ai diversi elementi che costituiscono le frasi:

- *articoli*: si deve verificare ed essere in grado di stabilire il motivo della loro assenza/presenza; se presenti, del fatto che siano definiti/indefiniti;
- *sostantivi*: ogni plurale o collettivo deve essere specificato da attributi per potere individuare l'insieme a cui il sostantivo appartiene;
- *aggettivi* ed avverbi: devono essere essenziali;

— *verbi*: il soggetto e, per i verbi transitivi, il complemento oggetto devono essere sempre espressi o in ogni caso non ambigui;
— *pronomi*: devono essere non ambigui;
— *congiunzioni*: devono essere usate propriamente; in particolare, per la congiunzione "o" è necessario specificare se significa "aut" o se significa "vel".

4.8. Aggiornamento del dizionario

In questa fase si aggiorna il dizionario creato nella fase 4.3. con i nuovi termini particolari della procedura emersi nel corso della stesura delle specifiche.

4.9. Isolamento delle transizioni

La specificazione della procedura prosegue con l'isolamento di ogni transizione della rete ottenuta dal precedente raffinamento eliminando ogni risorsa od arco non in relazione con la singola transizione.

4.10. Descrizione delle transizioni isolate

Per ogni transizione isolata sono richiesti i seguenti passi:

4.10.1. Descrizione della rete

L'obiettivo principale della descrizione è di definire concisamente ma non ambiguamente le risorse di ingresso e di uscita e la trasformazione della rete; per questo è richiesto che ogni risorsa sia descritta rispetto a:

— tipo di supporto fisico (disco, nastro, tastiera, schede, ecc.);
— caratteristiche fisiche e di organizzazione della risorsa (ad esempio, se si trattasse di un disco, se indexed, sequenziale, il formato della chiave, la dimensione, ecc.);
— la forma sintattica (caratteri permessi e sequenze di informazioni, se da tastiera; il tracciato del record se file su disco);
— le caratteristiche dei sostantivi plurali, cioè il loro numero (definito/indefinito; se definito: massimo e minimo valore) e gli attributi che definiscono l'insieme a cui appartengono.

La descrizione della trasformazione è realizzata per mezzo di un linguaggio che mescola il linguaggio naturale ai costrutti mostrati in figura 13, in modo il più possibile non procedurale; i costrutti si riferiscono a:

— operazioni da eseguire, senza specificare la loro sequenza;
— sequenze delle operazioni (talvolta è il modo migliore per specificare le trasformazioni);
— operazioni condizionali (ove connesse con il primo costrutto sono molto simili ai "guarded commands" di Dijkstra);
— operazioni su plurali.

SPECIFICHE NON-PROCEDURALI

1. NON-SEQUENZA

transf. α
transf. β
transf. γ

2. SEQUENZA

begin
transf. α
transf. β
transf. γ
end

3. CONDIZIONALE

if cond **then**
transf. α
endif

4. PLURALE

$\forall x \mid p(x)$ **do**
transf. α
end

Fig. 13

4.10.2. Controlli linguistici

Come al punto 4.7.

4.10.3. Aggiornamento del dizionario

Come al punto 4.8.

4.10.4. Ricorsione

Se la transizione in oggetto è ulteriormente raffinabile, effettuazione delle fasi 4,5,6,7,8,9,10 della metodologia. Alla fine del procedimento, la procedura sarà descritta da un "albero" di moduli PSPN i cui livelli dispari saranno di dettaglio delle attività ed i livelli pari saranno di raffinamento delle attività di livello di astrazione superiore.

4.11. Descrizione dei dati in Pascal

Per una completa descrizione dei programmi, viene descritta in linguaggio Pascal la struttura logica dei files esattamente come segue dai programmi disegnati con la metodologia. La figura 19 esemplifica questa fase.

5. IL PROCEDIMENTO PER RAFFINAMENTI SUCCESSIVI. UN ESEMPIO DI APPLICAZIONE

Il procedimento descritto nel paragrafo 4. prosegue fino a che una transizione corrisponde al livello di programma.

Non esiste un modo preciso per definire cosa si intende per programma, perchè, in realtà, non si possono imporre limiti allo sviluppo top-down;pur tuttavia possiamo tentare di dare la nostra definizione di "livello di programma" e di "procedura": una procedura è composta da un insieme di programmi per i quali la sequenza di controllo (o l'insieme delle sequenze di controllo) non può essere completamente fissato: possiamo pensare ad esecutori diversi (macchine o uomini) che lavorano concorrentemente o con qualche sincronizzazione; un programma è un insieme di operazioni per le quali le sequenze di esecuzione possono essere completamente fissate.

La definizione muove dal bisogno di diverse metodologie di disegno, quali ad esempio quella di Jackson, quella di Warnier, PHOS, che nel senso sopra esposto possono essere applicate ai programmi, di coprire i livelli più alti delle descrizioni delle procedure [6]. In tal modo, la costruzione delle specifiche funzionali si arresta al livello in cui le metodologie di programmazione vengono generalmente usate.

Diamo ora un esempio tratto da un'applicazione reale della metodologia descritta nei precedenti paragrafi. La procedura considerata è volta ad automatizzare con modalità interattiva la gestione degli acquisti di un'azienda.

L'intera descrizione, abbiamo detto, può essere vista come un albero di moduli PSPN; per ogni livello dell'albero vediamo un modulo, così da scendere dalla descrizione top-level fino alla descrizione di un programma. La figura 14 rappresenta la descrizione top-level della procedura e dà informazioni di massima sulle entità coinvolte; il raffinamento viene dato nel modulo PSPN successivo. La figura 16 descrive l'attività concernente gli ordini d'acquisto, cioè la prima transizione isolata dalla rete ottenuta con il raffinamento precedente. Questa viene a sua volta raffinata nel modulo successivo.

Ogni elemento del raffinamento corrisponde ad un modulo di programma. Nella figura successiva ne viene descritto uno.

Oltre alla descrizione del file ORDINI (nella figura 19) sono necessari altri allegati per completare la stesura delle specifiche; nell'esempio, gli allegati 1 e 5, cioè il formato delle maschere per il dialogo TP ed i tracciati di stampa rispettivamente.

6. ESPERIENZE

La metodologia illustrata è stata applicata con successo a svariate procedure non banali, come:

— una procedura per la gestione integrata di un laboratorio di analisi mediche;
— una procedura di fatturazione con terminali cash & carry in un ambiente multitasking;
— una procedura per la gestione automatizzata di una biblioteca;
— una procedura per la gestione automatizzata del bilancio.

I principali vantaggi del metodo emersi nel corso delle esperienze sono:

— la possibilità di individuare e sopperire alle deficienze dei requirements fornendo una completa descrizione funzionale;
— le indicazioni sull'architettura del sistema ricavate dal processo di sviluppo top-down, guidato dalle risorse necessarie e dalle eventuali esigenze di semaforizzazione.

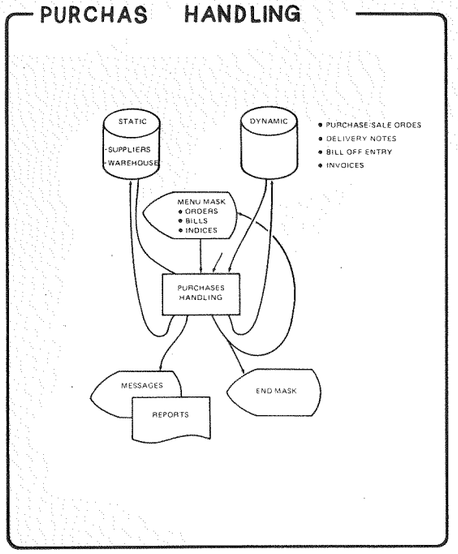


Fig. 14

PURCHASE HANDLING

The described procedure allows interactive purchases handling.

The procedure can start after an INTTIALIZATION phase which issued the proper MENU MASK on the video display; INPUT informations are required for the selection of the desired function among PURCHASE ORDERS, BILLS OF ENTRY and PAYABLE INVOICES; the menu offers also the STOP function, which end the execution, issuing an END MASK.

After the execution of a function (different from STOP), the initial MENU MASK will be issued, allowing new selections.

The procedure updates all the used files, which have been classified in two groupe:

. STATIC, for which the updating is performed on the content of the records, but not on their number;
. DYNAMIC, for which the updating is performed adding or deleting records, but not changing their content.

RESOURCES

Static files: SUPPLIERs , indexed
WAREHOUSe , indexed
Dynamic files: PURchase ORDERs , indexed
BILLS of entry, indexed
INVOICES , indexed
SALe ORDERs, indexed
DeLiVery NOTES , indexed
MENU MASK: see mask n.2, annex 1
END MASK : see mask n.3, annex 1
INPUT: "O"/"B"/"I"/"S" for the selection of the proper function (purchase orders,bills of entry,payable invoices, stop) and the informations required by each function.
MESSAGES: error and final massages issued by the functions.
REPORTS: different reports referring to the performed functions

ACTIVITY

. updating of static files and different reports on their content
. handling of BILLS of entry and related reports
. handling of INVOICES with different reports (including check computations).

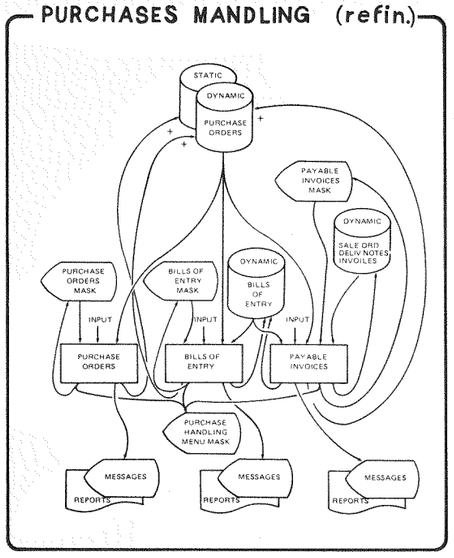


Fig. 15

PURCHASES MANDLING (refinement)

Different terminals can simultaneously operate with different functions,accessing concurrently the static files and the PUR-ORDER file.

The selection of a function causes the issuing of a proper mask proposing a lower level menu.

The BILLS file can be accessed concurrently by the functions "bills of entry" and "payacle invoices"; the other files are accessed only by the function "payable invoices".

Each function updates the files which uses.

Each function proposes, at the end, the purchase handling MENU KASK.

RESOURCES

STATIC FILES: .SUPPLIERs , indexed; logical unit: supplier semaphorised at logical unit level
.WAREHOUSE, indexed; logical unit: product semaphorised at logical unit level
Dynamic files:.PURchase ORDERs, indexed; logical unit: order, composed of a heading record and more detail records, semaphorised at logical unit level
.BILLS of entry, indexed; logical unit: bill; semaphorised at logical unit level
.other files as defined above

For other resources, see lower levels.

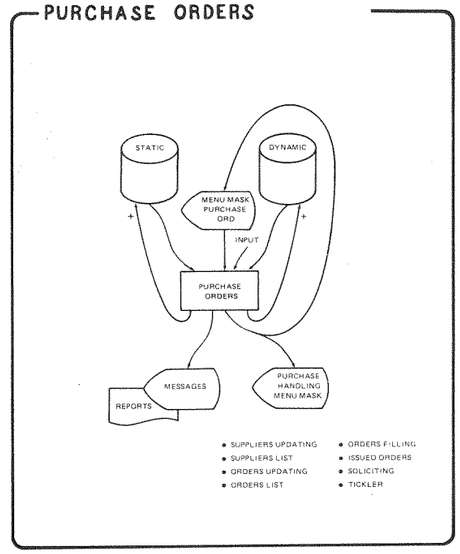


Fig. 16

PURCHASE ORDERS

This procedure part allows the selection and execution of one of the following functions: suppliers updating, suppliers list,orders updating, orders list,orders filling,issued orders, soliciting, tickler.

Different terminals can operate on different functions simultaneously.

The menu provides a stop function for exiting the "purchase orders" level, returning to the purchase handling MENU MASK.

At the end of each function execution the purchase orders menu mask is anow proposed.

RESOURCES

Static files:see previous level.
Dynamic files: PURchase ORDERs: see previous level and description in annex 4.
MENU MASK : see Mask n.4, annex 1.
Purchase handling MENU MASK: see Mask n.2, annex 1.
INPUT: "SU"/"SL"/"OU"/"OL"/"OF"/"IO"/"SO"/"TC" respectively for the functions "suppliers updating", "suppliers list","orders updating", "orders list", "orders filling", "issued orders","soliciting", "tickler". Informations required by the different functions.
MESSAGES: error messages or final messages issued by the different functions.
REPORTS: different reports referring to the different functions.

ACTIVITY

. updating and printout of the SUPPLIERs file
. updating and printout of PUR-ORDER
. printout of the orders to be filled
. printout of the issued orders, sorted on product/supplier or on supplier
. automatic printout of soliciting notes
. tickler printout on a specified time interval, sorted on supplier/product or on product.

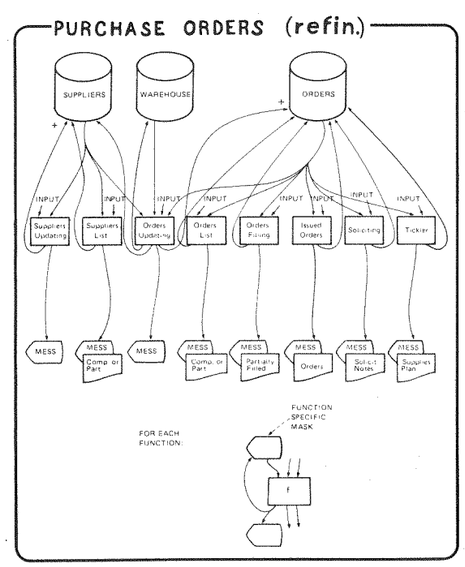


Fig. 17

PURCHASE ORDERS-(refinement)

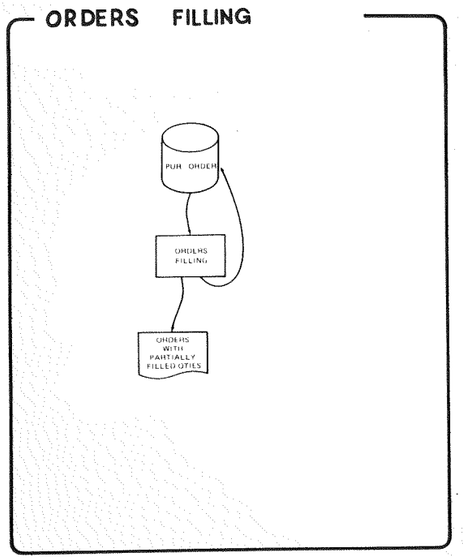
This procedure part allows one of the eigth indicated functions:

"suppliers updating", "suppliers list" and"orders updating" access concurrently the file SUPPLIERs (which is updated by the first one only); the file PUR-ORDER is accesses concurrently by "orders updating", "orders list","orders filling", "issued orders", "soliciting" and "tickler" (updated only by the firts one); the file WAREHOUSe is accessed as read only exclusively by "orders updating".

Each function returns the purchase orders menu mask.

RESOURCES

see precedent or next levels.



ORDERS FILLING

RESOURCES

files: PUR-ORDER, indexed, key=record type/order nurber/code; logical unit: order, composed of a heading record followed by more detail records, each of them referring to a single article; semaphorised on the key order number. See annex 4 for details.

REPORT: printout, for each record to be filled, of the informations present on the file PUR-ORDER, among which, the partially satisfied quantities.(see printout form 3,annex 5)

ACTIVITY

heading recprd with status = "filled":

. prints anagraphical data of the order, present on the heading
. prints informations contained on the detail records, for obtaining the printout form 3, annex 5.

Fig. 18

```
type orders = file order

type order = record of

        herd: supplier-info
        movens: file of movement

        end

type supplier-info = record of

        rec - type      : (0)
        order - no      : (1 .. 9999)
        code            : array [1 .. 5] of char
        address         : array [1 .. 28] of char
        issue - date    : date
        expire - date   : date
        amount          : (100 .. 9999999)
        status          : (issued, solicited, filled)

        end
```

```
type movement = record of

        rec - type      : (1)
        order - no      : (1 .. 9999)
        article - code  : array [1 .. 5] of char
        descript        : array [1 .. 14] of char
        price           : (1 .. 999999)
        quantity        : (1 .. 9999999)
        rec - date      : (date, '$')
        status          : (completed, partial, writ)
        future - use    : array [1 .. 14] of char

        end

type date = record of

        dry     : (1 .. 31)
        month   : (1 .. 12)
        year    : (0 .. 99)

        end
```

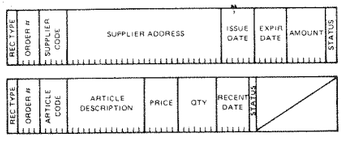


Fig. 19

7. BIBLIOGRAFIA

[1] J. Warnier, B.M. Flanagan, ENTRAINEMENT À LA PROGRAMMATION, Les Editions d'Organisation, Paris, 1972.

[2] M. Jackson, PRINCIPLES OF PROGRAM DESIGN, Academic Press Ltd, 1975.

[3] T. Bettinazzi, M. Maiocchi, A. Maserati, F. Monzani, E. Spoletini, E. Toscani, PHOS: UNA METODOLOGIA PER LA COSTRUZIONE VELOCE ED AFFIDABILE DEI PROGRAMMI, Note di Software n. 4 - 1977.

[4] J.L.Peterson, PETRI NETS, ACM Computing Surveys, vol. 9, n. 3, 1977.

[5] A. Cicu, G. Degli Antoni, M. Maiocchi, R. Polillo, G. Torriani, AN INTEGRATED MULTILEVEL DOCUMENTATION APPROACH, HIS Successful Software Management Techniques Conference, Bloomington (MN) USA, marzo 1978.

[6] G. Degli Antoni, B. Zonta, DOPO LA PROGRAMMAZIONE STRUTTURATA, Congresso annuale AICA, Pisa, 1977.

[7] G. Degli Antoni, M. Maiocchi, R. Polillo, PSPN: UN APPROCCIO AI PROBLEMI DI COMUNICAZIONE DELL'INFORMAZIONE TECNICA, Note di Software n. 6/7, 1978

SCHEMI DI INTERAZIONE FRA PROCESSI CONCORRENTI DESCRITTI MEDIANTE LE RETI DI PETRI

G.P. Rossi, C. Simone
Ist. Cibernetica - Univ. di Milano

Riassunto

In questo lavoro analizziamo alcune delle principali caratteristiche dei processi concorrenti che possono essere descritte da configurazioni "standard" di reti di Petri (eventualmente estese). Questo suggerisce un'attitudine nell'analisi di un problema e nella costruzione di un sistema di processi concorrenti che lo risolve.

Abstract

In this paper we analyze some of the characteristics of concurrent processes which can be described using "standard" Petri nets configurations. This approach reveals useful in analysing problems which can be solved by systems of concurrent processes, and in the construction of software implementing them.

1. INTRODUZIONE

L'implementazione di un sistema di processi concorrenti è in generale fortemente influenzato dalle caratteristiche del linguaggio di programmazione utilizzato, che impone le sue restrizioni alla logica del problema. È quindi utile poter disporre di un formalismo descrittivo in grado di rappresentare processi concorrenti indipendentemente dagli strumenti di implementazione, allo scopo di studiare il problema unicamente rispetto alla sua logica interna, consentendo così valutazioni sul grado di parallelismo, di efficienza, ecc., che saranno utili nella fase di implementazione.

Le reti di Petri sono uno strumento descrittivo adeguato allo scopo, anche se sono necessarie alcune estensioni per renderle più flessibili rispetto alle caratteristiche di ambienti concorrenti: infatti, esse non impongono vincoli alla descrizione del sistema e consentono una visione dinamica della sua evoluzione.
Un tale strumento, trasparente ai problemi implementativi, consente anche eventuali verifiche di consistenza e correttezza (tramite simulazione) del sistema descritto, in modo da fornire all'implementatore una struttura funzionale già collaudata.

Ricerca svolta con il contributo del CNR (contratto n. 79.00701.02.115.9672)

In questo lavoro, dopo una breve presentazione dell'interpretazione (da noi utilizzata) associata alle reti di Petri, analizzeremo alcune caratteristiche fondamentali dei processi concorrenti che possono essere descritte da "configurazioni standard" di reti di Petri (eventualmente estese). Questo fatto suggerisce un'attitudine nell'analisi di un problema e nella costruzione di un sistema di processi concorrenti che lo descrive e lo risolve.

Per la descrizione dei concetti fondamentali necessari per definire le proprietà sintattiche di una rete di Petri (definita in modo classico), si veda, ad esempio, il lavoro di Hack [1]; per una ricca bibliografia il lavoro di Peterson [2] o la più recente raccolta di Pless e Plünnecke [3].

2. INTERPRETAZIONE ASSOCIATA AD UNA RETE DI PETRI

Fissata la sintassi e la legge di scatto delle transizioni, ciò che definisce le capacità descrittive delle reti di Petri è l'interpretazione associata a *posti*, *marche* e *transizioni*.

Interpretando in modo classico [4], [5] i posti come *condizioni* e le transizioni come *eventi* (o *azioni*) che modificano lo stato della rete (cioè la marcatura), si ottiene uno strumento in grado di descrivere, di ogni sistema, sia proprietà statiche (il grafo bipartito), sia proprietà dinamiche legate alla sua evoluzione, rappresentata dagli spostamenti delle marche all'interno della rete secondo le regole di scatto delle transizioni.

Il grafo bipartito indica quali sono le precondizioni necessarie alla occorrenza di un evento (pensato come azione unitaria che avviene in un tempo nullo), e quali sono le postcondizioni che l'eventuale occorrenza renderebbe vere.

L'interpretazione di una marca dipende strettamente dal tipo di condizione associata al posto che la contiene. Due sono i possibili tipi di condizione che si possono associare come interpretazione ad un posto:

— un *predicato* sulla struttura dati: ad esempio, la presenza di una marca nel posto "messaggio arrivato", viene interpretata come il valore "vero" del predicato stesso;

— il *valore di un contatore* >0: ad esempio, il numero di marche contenute nel posto "numero di messaggi arrivati", è il valore di un contatore, il cui nome (associato al posto) indica la quantità da misurare.

Nel primo caso, affinchè l'interpretazione non perda significato, è necessario che il posto non possa contenere più di una marca.

La necessità di distinguere fra predicati e contatori risiede nel fatto che i primi non possono essere utilizzati per indicare quante volte un certo evento è occorso, e quindi per quante volte l'evento successivo (dipendente da esso) deve scattare.

Le reti di Petri, nell'interapretazione proposta, descrivono quindi soltanto *condizioni* sui dati manipolati dai processi del sistema in esame e le *azioni* che esse abilitano, e mai direttamente i dati stessi.

La struttura causale su cui si basa la descrizione di un sistema mediante le reti di Petri prevede che, se tutte le precondizioni di un evento sono vere, allora quest'ultimo può scattare; altrimenti, il processo che lo contiene entra in uno stato di attesa (stato di "wait" su una condizione). Questa struttura è sufficiente per descrivere situazioni di pura sincronizzazione, mentre è carente nei casi in cui si devono descrivere situazioni di controllo condizionale (if-then-else), in cui anche il *non* verificarsi di una condizione può abilitare l'occorrenza di un evento.

Da questa esigenza, è nata la prima generalizzazione delle reti di Petri, che introduce gli *inibitori* [6], cioè un nuovo tipo di archi del grafo bipartito che congiungono un posto ad una transizione e sono rappresentati da $\dashrightarrow$.

L'uso degli inibitori modifica le regole di abilitazione allo scatto come mostrato dalla fig. 1.

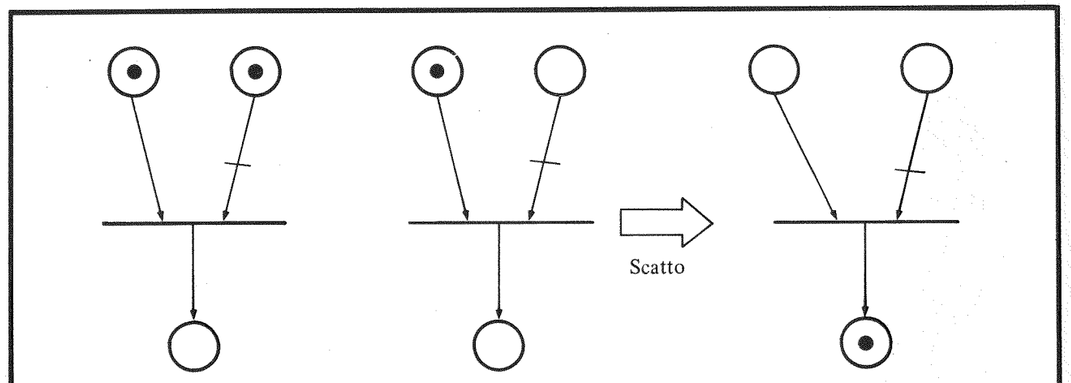
L'introduzione degli inibitori è compatibile con l'interpretazione fin qui data: se il posto rappresenta un predicato, la precondizione è legata al fatto che esso sia falso; se rappresenta un contatore, la precondizione equivale a richiedere che esso assuma il valore zero.

L'uso degli inibitori amplia la capacità descrittiva delle reti di Petri in quanto permette loro di descrivere qualunque funzionalità [6], ma impedisce di usare algoritmi di decidibilità di proprietà delle reti (per ulteriori dettagli su questo argomento si veda, per esempio, [1]).

3. LA COSTRUZIONE TOP—DOWN DELLA DESCRIZIONE DI PROCESSI CONCORRENTI

Perchè le reti di Petri siano uno strumento utile per la descrizione di processi concorrenti non è sufficiente definire la loro sintassi e la relativa interpretazione, ma è necessario dare alcuni criteri che ne guidino la costruzione.

È universalmente accettato che l'approccio top-down, in tutte le fasi di sviluppo di un prodotto



a. Transizione non abilitata

b. Transizione abilitata

c. Marcatura dopo lo scatto della transizione

Fig. 1

software (dall'analisi all'implementazione), è un criterio fondamentale per garantire alcune caratteristiche (leggibilità, modificabilità, ecc.) del prodotto finale. Il tentativo di definire che cosa significhi costruzione top-down di una rete di Petri ha portato ad un atteggiamento analogo a quello tenuto anni fa, per i flow-chart.

Alcuni studi recenti di Kotov [7], [8], hanno infatti individuato una serie di operazioni definite su reti più semplici, in modo da comporle in una più complessa che risulti, in qualche modo, "strutturata". Ma, mentre per i flow-chart le operazioni corrispondenti ai tre noti schemi di composizione di base (sequenza, selezione, iterazione) hanno avuto un immediato riscontro applicativo per il loro naturale significato, questo non è ancora avvenuto per le reti di Petri.

L'approccio di Kotov però limita i raffinamenti alle sole transizioni, mentre dall'esperienza emerge la necessità di espandere sia azioni che condizioni, cioè sia le transizioni che i posti.
Per questo motivo, proponiamo di seguire nella definizione delle modalità di raffinamento l'approccio che emerge nel lavoro di C.A. Petri [5]: in esso viene definito il concetto di *morfismo di reti di Petri*, come uno strumento utile per stratificare le reti da un livello più dettagliato ad uno meno specificato. Questo atteggiamento, tipicamente bottom-up, può essere utilizzato in modo top-down: senza entrare nei dettagli formali delle definizioni (ampiamente descritti nel lavoro citato), mostriamo intuitivamente il senso di questo approccio.

Ogni raffinamento può essere visto come il prodotto di una trasformazione della rete meno specificata in una più dettagliata, che si realizza sostituendo ad un posto o ad una transizione una struttura di rete (quindi una combinazione di posti e transizioni) con alcuni vincoli:

— devono essere mantenuti i collegamenti (archi) con gli elementi (posti o transizioni) che precedono e seguono l'elemento raffinato (*prossimità*);

— tali collegamenti devono mantenere gli orientamenti che avevano nella rete prima del raffinamento (*orientamento*).

Ad esempio, data la rete di fig. 2, il raffinamento T

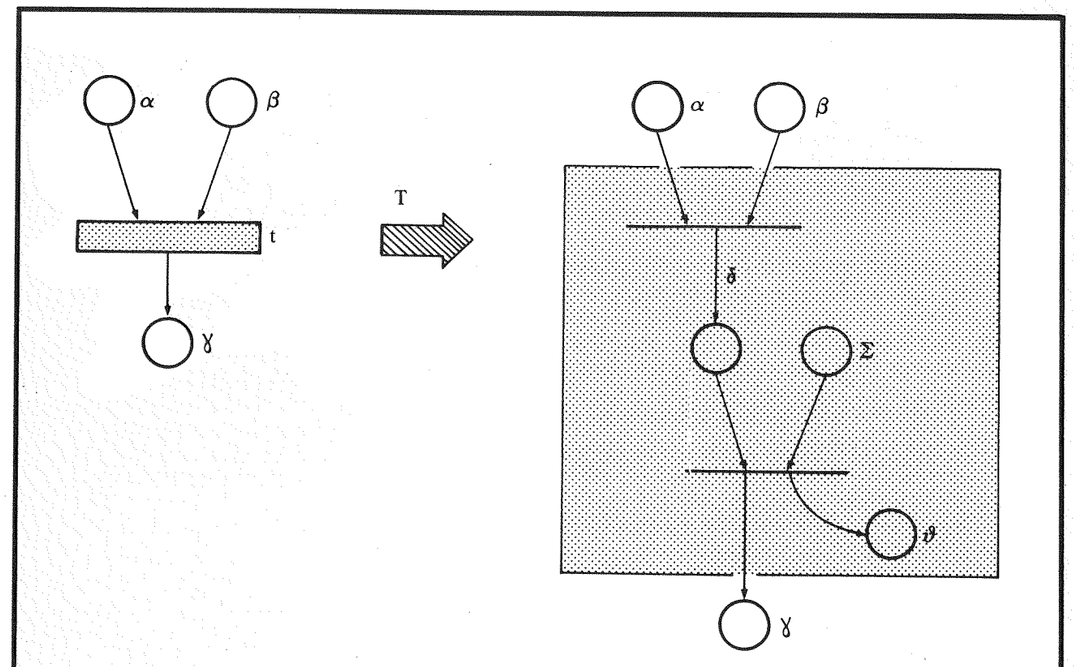


Fig. 2

trasforma α, β e γ in se stessi, e t nella rete inquadrata.

La prossimità è conservata, in quanto sono mantenuti gli archi uscenti da α e β; l'arco entrante in γ è pure conservato e tutti mantengono l'orientamento.

Analogamente, in fig. 3, T definisce il raffinamento di un posto che rispetta le regole di prossimità e orientamento, mentre in fig. 4 T non conserva la prossimità di α rispetto a t e in fig. 5 T non conserva nè la prossimità (rispetto a S) nè l'orientamento dell'arco α rispetto a t.

Intuitivamente, le due regole impongono che il raffinamento mantenga le preesistenti relazioni di causalità tra eventi e condizioni, mentre altre relazioni (nuovi posti e transizioni) possono sempre venire aggiunte, purchè non alterino quelle preesistenti .
Questo approccio, pur lasciando ampia libertà nei raffinamenti, impone vincoli concettuali che sono di ausilio nel controllare un logico sviluppo della descrizione del problema in esame, attraverso semplici e naturali vincoli strutturali.
In [12] è mostrato l'utilizzo di questo approccio in una rete interpretata.

4. GLI SCHEMI DI INTERAZIONE

Ci proponiamo in questo paragrafo di individuare alcune situazioni tipiche della comunicazione fra processi, e di dare per ciascuna di esse la corrispondente rappresentazione in termini di reti di Petri, individuando così delle *strutture di base*, che potremo modularmente comporre per descrivere le interazioni fra processi concorrenti di un qualsiasi sistema.

a) Il primo caso che analizziamo è quello di processi *disgiunti*, la cui esecuzione avviene concorrentemente ma senza reciproche interferenze (essi utilizzano insiemi disgiunti di risorse).
La situazione corrispondente, in termini di reti di Petri, è molto semplice (fig. 6).
(Le strutture dati modificate da un processo Pi devono essere necessariamente private a quel processo, ma processi disgiunti possono far riferimento a

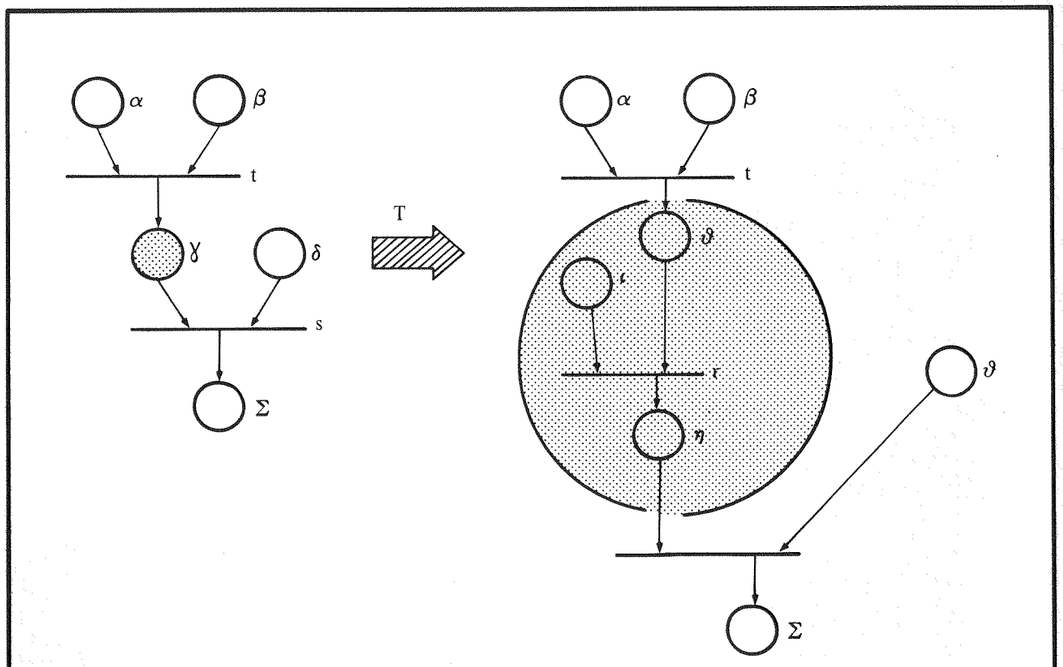


Fig. 3

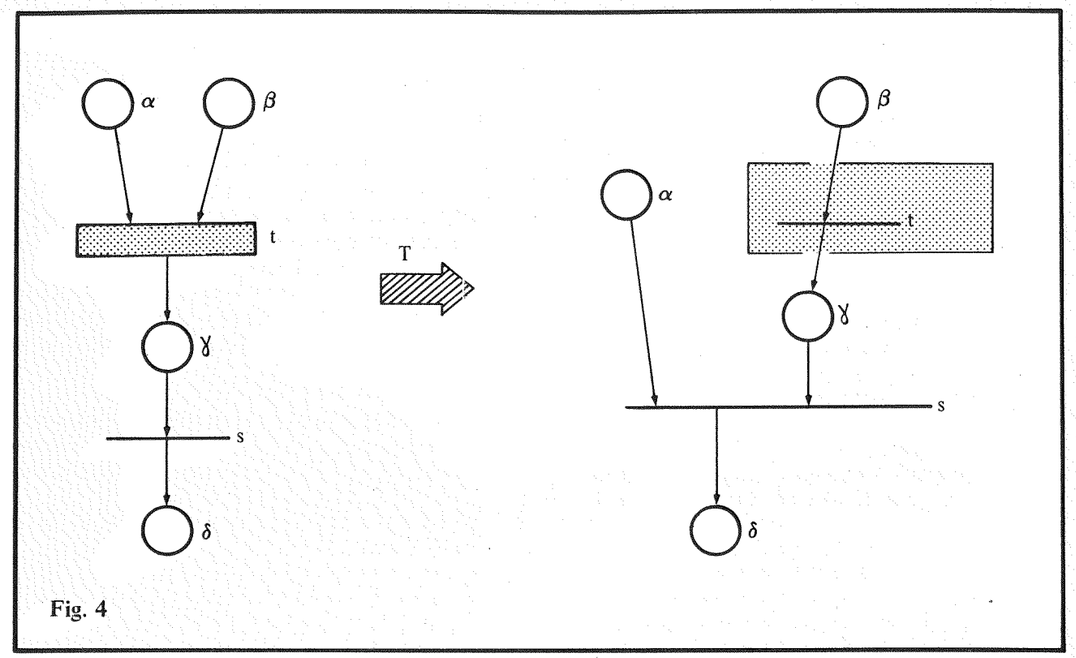


Fig. 4

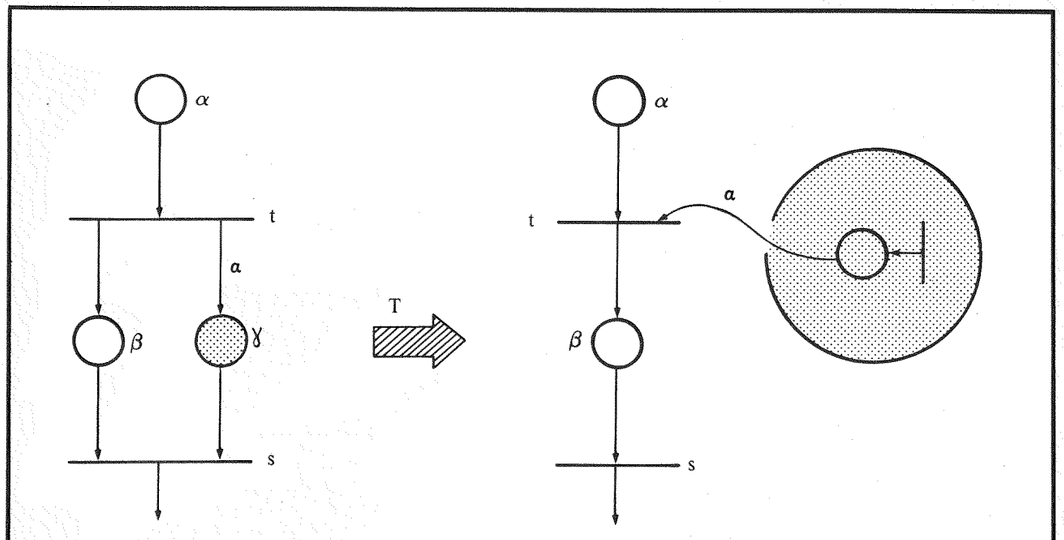


Fig. 5

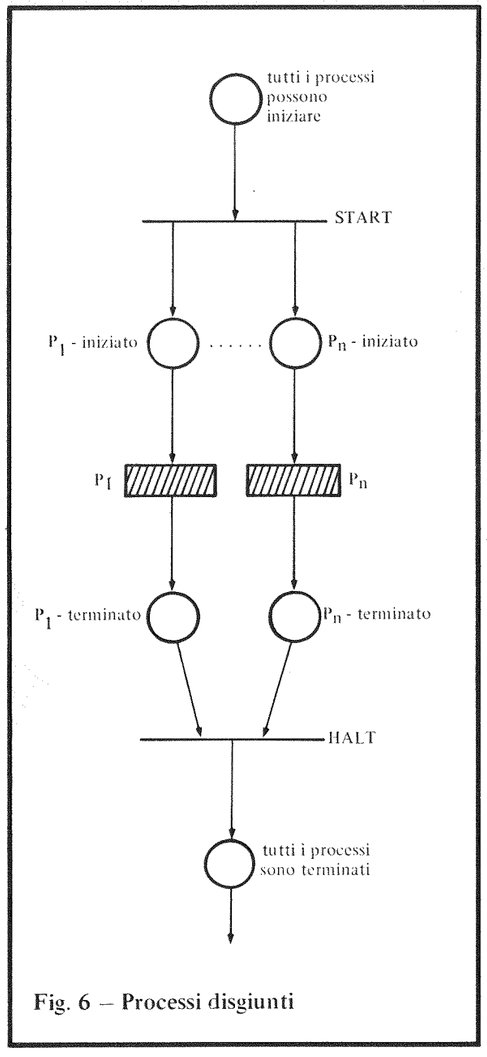


Fig. 6 – Processi disgiunti

risorse comuni non modificate da nessuno di essi).

b) Consideriamo ora il caso di processi *interagenti* (che fanno uso di risorse comuni); possiamo individuare tre situazioni tipiche:

— *i processi interagiscono ma non cooperano.* È il caso in cui per esempio, più processi accedono concorrentemente ad una risorsa comune (periferica, variabile, struttura dati, ecc.). La costruzione del sistema deve garantire unicamente che tale risorsa non sia utilizzata simultaneamente da più processi (mutua-esclusione), mentre non ha nessuna rilevanza chi e in che ordine vi accede. Lo schema è allora quello di fig. 7: quando un processo effettua una richiesta di accesso alla risorsa e questa è disponibile, esso la utilizza, per poi rilasciarla rendendo vero il predicato "risorsa disponibile".

— *Due processi interagiscono e si scambiano segnali.* È il caso in cui due processi devono sincronizzarsi. Lo schema che descrive questa situazione (fig. 8) usa una condizione, che rappresenta una variabile (cond A e cond B) col compito di registrare le segnalazioni che i processi si scambiano (fig.4).

Ad esempio, solo se il predicato cond B è vero, indicando che il processo Y ha spedito una segnalazione, il processo X può continuare la sua esecuzione.

Si noti che sincronizzazione è un termine generale per indicare ogni vincolo sull'ordine delle operazioni nel tempo. Infatti le regole di sincronizzazione specificano una precedenza o una priorità di operazioni, con limitazioni del tipo "l'operazione A deve essere eseguita prima dell'operazione B", oppure "un'operazione di priorità x deve essere eseguita solo dopo che tutte le operazioni di priorità maggiore sono state eseguite".

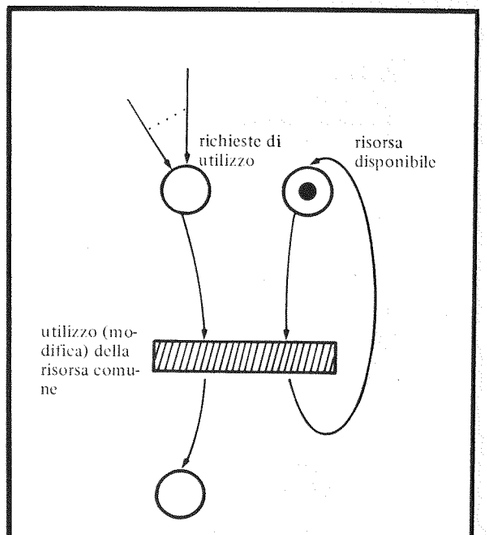


Fig. 7 – Mutua esclusione

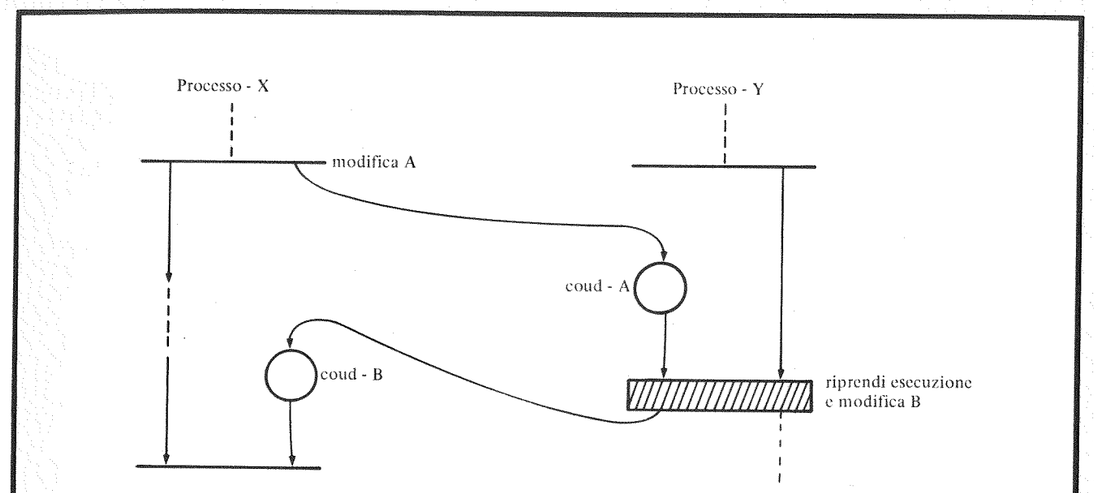


Fig. 8 – Sincronizzazione fra due processi X e Y

— *Due processi interagiscono scambiandosi segnali e cooperano scambiandosi messaggi.* Un esempio di questa situazione è quella in cui un processo A (che indichiamo come *sender*) "spedisce", tramite una struttura dati destinata alla comunicazione (*buffer*), un messaggio ad un processo B (*receiver*). La sincronizzazione fra i due processi è necessaria in quanto le due operazioni di *scrivi nel buffer* e *leggi dal buffer* devono essere eseguite in sequenza.
La rete di fig. 9 descrive questa situazione.

L'operazione di lettura dal buffer ha luogo solo se è vero il predicato *buffer pieno*, segnalato dal processo *sender*, mentre l'operazione di scrittura ha luogo solo se è vero il predicato *buffer vuoto*, segnalato dal *receiver*.

Il sistema che descrive il problema *sender* e *receiver* di fig. 9 è coerente con le esigenze del problema in quanto esiste un solo sender e un solo receiver. In questo caso, infatti, i valori dei due predicati *buffer pieno* e *buffer vuoto* sono sufficienti a garantire la sincronizzazione fra i processi, consentendo di accedere al buffer nella sequenza voluta e in modo alternato.

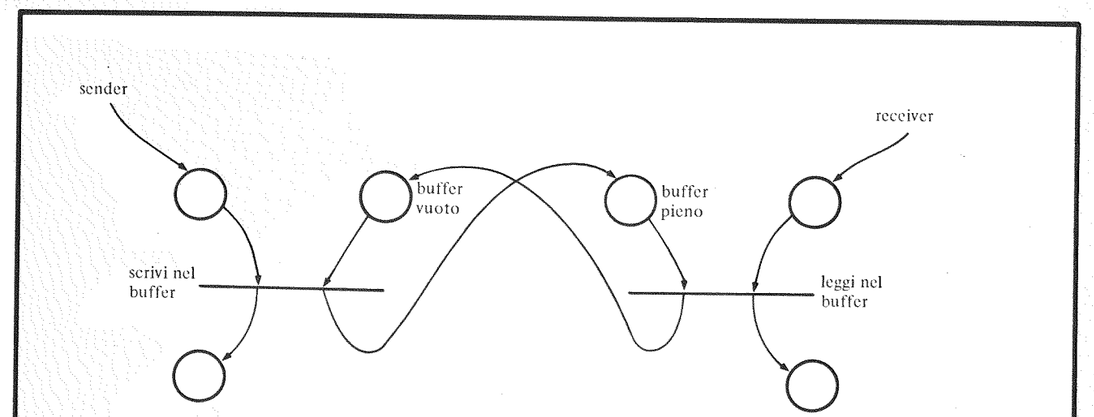


Fig. 9 – Sender e receiver su buffer ad un solo posto

La fig. 10 rappresenta una semplice estensione del caso precedente in cui si usa un buffer multiplo, in grado di contenere n messaggi. In questo caso, le due condizioni, *posti liberi* e *posti occupati*, sono contatori.

Si può notare che, in fig. 10, i due contatori sono sufficienti a gestire la sincronizzazione tra lettura e scrittura sul buffer, mentre la mutua esclusione sui singoli elementi del buffer multiplo può essere risolta al livello della implementazione della sua gestione (utilizzando due puntatori al "primo posto occupato" e al "primo posto libero").

Nel caso più generale di *multi-sender* e *multi-receiver*, le azioni *scrivi nel buffer* e *leggi dal buffer* può essere utile, per motivi ovvii di efficienza e leggibilità, inglobare tutte le operazioni che risolvono complesse situazioni di interazione fra processi, in un unico modulo (che qui indicheremo come *procedura*) che mascheri ai processi i meccanismi di sincronizzazione e di mutua esclusione e al quale i processi fanno riferimento con una chiamata a procedura (*call* nome-procedura) (fig. 12).

Questa attitudine rende indubbiamente più modulare la descrizione del sistema, ma crea il problema di rappresentare con le reti di Petri più istanze di una procedura.

La notazione delle reti di Petri fin qui vista permette di descrivere questa situazione mediante pesanti e poco chiare ripetizioni di sottoreti, (si veda la fig. 11). Per evitare ciò introduciamo una soluzione che permette di distinguere le varie istanze di una procedura utilizzando una sola copia della rete che la descrive.

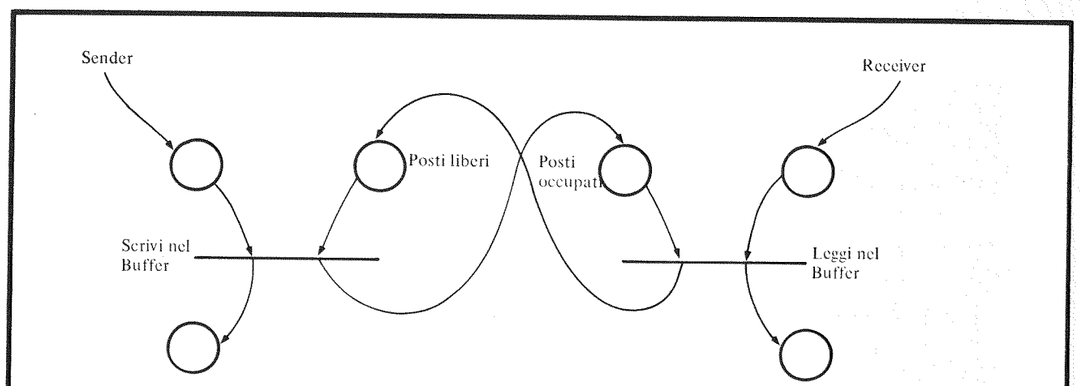


Fig. 10 – Sender e receiver su buffer multiplo

e le condizioni *buffer pieno, buffer vuoto* dovranno essere inserite in due regioni critiche, opportunamente protette, in modo che comunque sia garantito che un solo processo alla volta dello stesso tipo possa operare sul buffer.
A questo scopo in fig. 11 sono state introdotte le due condizioni *mutex-S* e *mutex-R*.

In casi come il precedente, in cui ci sono più livelli di interazione fra processi e in cui la struttura dati coinvolta è un elemento passivo rispetto all'evoluzione del processo (nel nostro caso il buffer è solamente uno strumento di comunicazione come potrebbe essere una linea fisica di comunicazione),

La soluzione sta nella *colorazione* delle marche che percorrono la rete, in modo che ogni istanza sia caratterizzata da un diverso colore.
Questa soluzione comporta una revisione sia della definizione della rete di Petri che delle regole di scatto.

In una rete di Petri si possono allora avere due tipi di posti:

— *posti condizione* (predicati o contatori), che mantengono le caratteristiche già viste;
— *posti controllo,* che contengono *marche colorate* e vengono qui indicati con ◎.

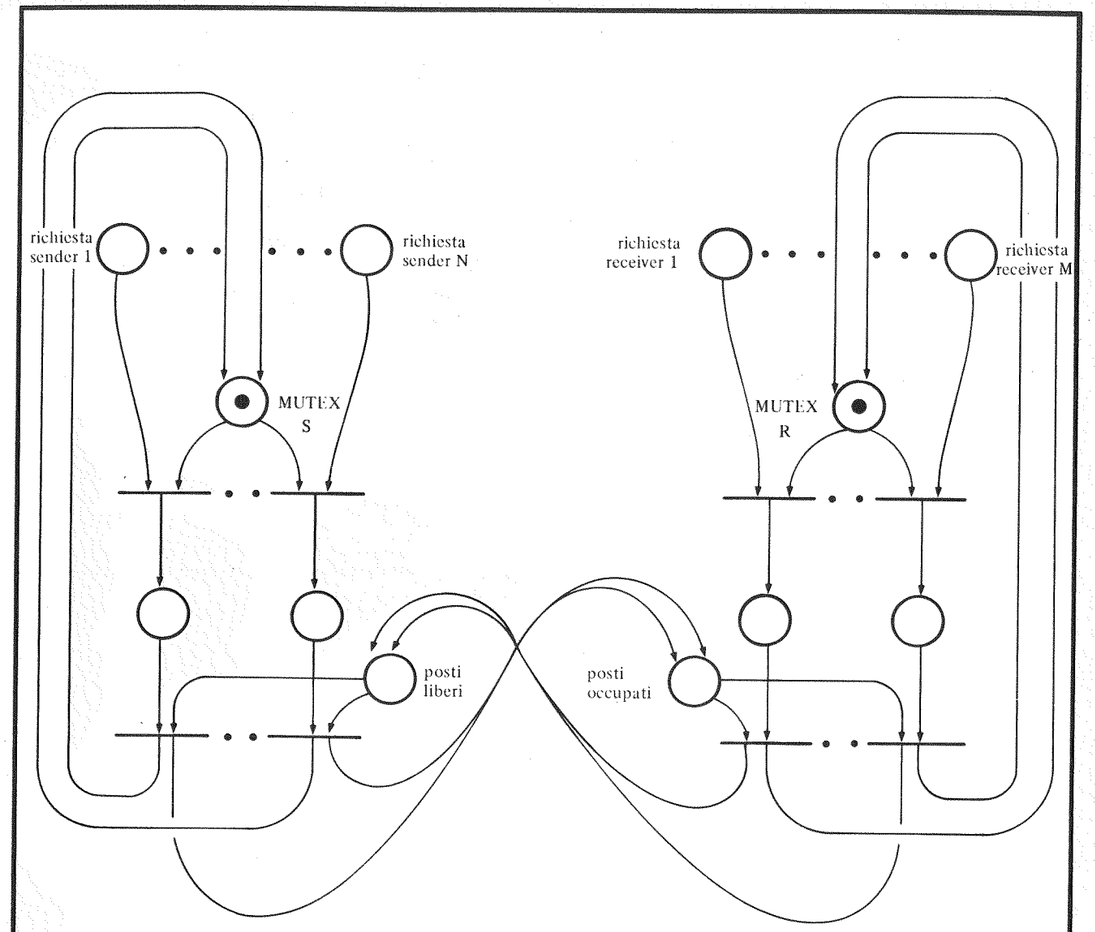


Fig. 11 – Multi sender e multi receivers

Rifacendoci alla fig. 13, vediamo che una transizione è abilitata allo scatto se i posti condizione hanno almeno una marca e i posti controllo hanno almeno una marca dello stesso colore ((a) e (b) in figura). La regola di scatto di una transizione resta dunque invariata per i posti condizione, mentre per i posti controllo si procede togliendo una marca di colore uguale da ogni posto di ingresso, e depositandola nel posto controllo di uscita.

Ritornando allora alla fig. 12, osserviamo che ogni processo chiamante la procedura deve essere caratterizzato da un colore diverso, e che i posti Attesa hanno il compito di garantire il ritorno del controllo al processo che ha appena percorso il corpo della procedura.

La qualificazione delle marche presenti in un posto di una rete di Petri è stata introdotta da K. Lautenbach, e ulteriormente precisata in un suo recente lavoro in cooperazione con H.J. Genrich [9]. La colorazione da noi proposta ne è un caso particolare, in cui la struttura algebrica associata alle marche (l'universo) è un insieme con il predicato di uguaglianza.
La qualificazione delle marche è stata, ad esempio, utilizzata in [10] con lo scopo di tener traccia della evoluzione della rete che ha portato a quella marcatura. Nel nostro caso, invece, ciascuna marca colorata può essere vista come un *puntatore*, che indica in quale punto l'esecuzione di un'istanza è arrivata ed associa ad essa informazioni (punto di rientro, indirizzo dello stack privato, priorità, ecc.) relative al processo corrente, ricevute all'atto della chiamata tramite la marca colorata che rende vera la condizione *inizio*.
Il corpo della procedura può allora essere costruito in modo indipendente, con il vincolo che la sua evoluzione porti marche colorate nel posto *fine*.

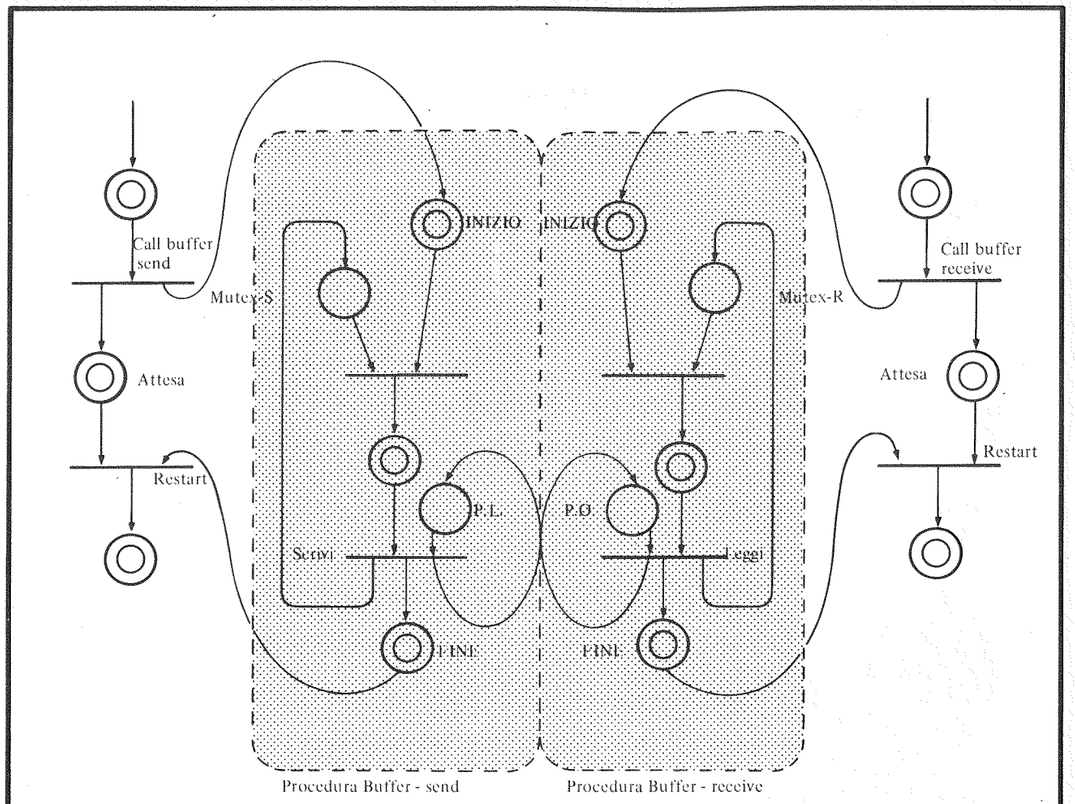


Fig. 12 – Uso di procedure nel caso multi - senders e multi - receivers

5. IL PROBLEMA DELLA INIZIALIZZAZIONE

Nell'interpretazione qui proposta, una volta determinate quali condizioni sono verificate all'inizio, l'evoluzione delle reti procede soltanto in dipendenza della marcatura iniziale.

Ciò non è sempre vero: può infatti verificarsi che alcune caratteristiche, intrinsiche alla struttura dei dati, determinino condizioni che influenzano lo sviluppo successivo della rete.

Supponiamo, infatti, che un processo P riceva un dato dall'esterno (da un altro processo, da un supporto fisico attraverso un'operazione di lettura, ecc.) e che, sulla base delle caratteristiche del dato stesso, operi una scelta su più evoluzioni possibili. Accade, quindi, che non solo l'acquisizione del dato, ma anche le sue caratteristiche sono precondizioni di successivi eventi differenti.
Come è possibile porre la marcatura corretta in queste precondizioni, dato che non lo può fare l'evoluzione precedente della rete?
È necessario ricorrere ad una fase di *inizializzazione intermedia* che si occupi di mettere una marcatura opportuna in un certo numero di posti, sulla base di *valutazioni esterne* all'evoluzione della rete

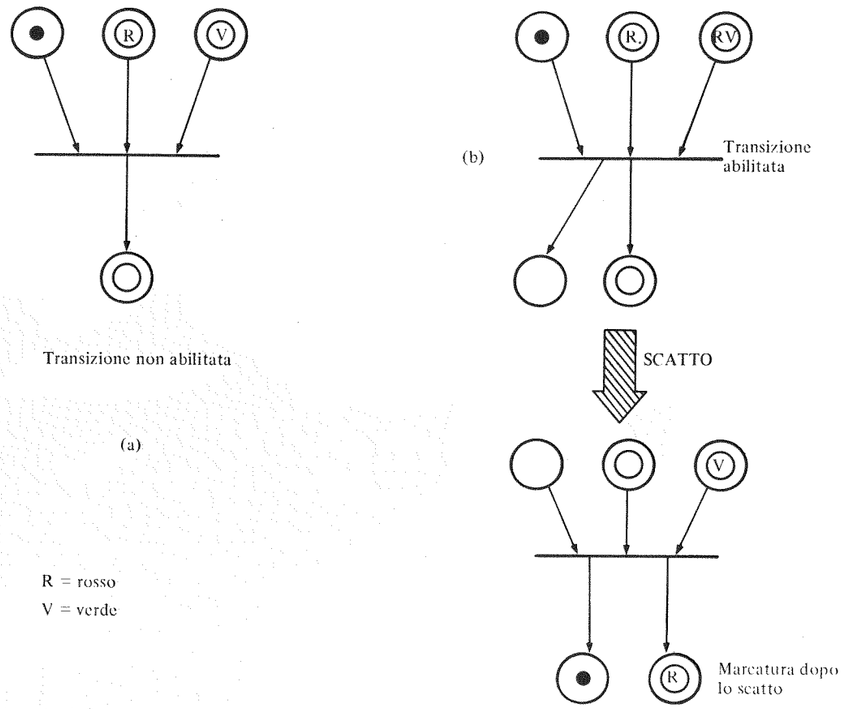


Fig. 13

stessa.

Introduciamo, quindi, una nuova transizione (che indicheremo con I ——○^{c}, in cui I sta ad indicare che si tratta di una inizializzazione intermedia e c è la condizione che determina la marcatura successiva). Essa si comporta come qualunque altra transizione per quanto riguarda le regole di abilitazione e scatto per i posti di ingresso, ma pone una marca nei posti di uscita, sulla base della valutazione del predicato che in esso è indicato, la cui verità *non* dipende dalla precedente evoluzione della rete.

In questo modo, le inizializzazioni intermedie sono viste come un procedimento deterministico (la valutazione di un predicato) simulato da un evento di tipo particolare.

6. CONCLUSIONI

L'aver individuato alcuni standard di comunicazione tra processi, permette di dare un strutturazione logica al problema che si deve risolvere mediante un sistema di processi concorrenti.

In una stessa applicazione, può essere presente più di uno degli schemi introdotti (si veda l'esempio in [12]).

Il cercare di separare le attività che rientrano in ciascuno schema permette un approccio top-down alla descrizione del problema, indipendentemente dagli strumenti linguistici che si utilizzano nella descrizione.
Tra questi ultimi, le reti di Petri hanno il vantaggio di fornire una descrizione dinamica, che si presta alla simulazione del sistema descritto e quindi allo studio di proprietà del sistema basato sull'analisi

della sua evoluzione. Le estensioni proposte non alterano questa caratteristica, al più rendono meno elementari i procedimenti di verifica di abilitazione e di scatto di ciascuna transizione [13].

Il formalismo qui presentato non vuole essere definitivo, ma semplicemente un approccio alla descrizione di processi concorrenti, indipendentemente da vincoli linguistici legati ad un particolare linguaggio di programmazione.

Due sono ancora i problemi aperti, concettualmente interessanti: la definizione di costruzione top-down di una rete e il problema delle inizializzazioni intermedie. Quest'ultimo è tipico dei linguaggi basati sullo schema condizione-azione: si ritrova infatti anche nel linguaggio proposto da Dijkstra [14], in cui per esprimere le inizializzazioni è necessario introdurre un costrutto (il *begin-end*) non basato sul tale schema.

Per quanto riguarda il primo, l'approccio di Petri, qui utilizzato, ci sembra ancora troppo strutturale e poco costruttivo nel senso che non fornisce sufficienti regole di verifica della validità del raffinamento rispetto alla funzionalità complessiva descritta dalla rete. Per questo motivo, un interessante sviluppo della ricerca sarà il cercare di correlare la definizione della interpretazione funzionale (quindi in un senso diverso da quello utilizzato in questo lavoro) secondo le linee indicate in [15] con il concetto di morfismo di reti di Petri.

7. BIBLIOGRAFIA

[1] M. Hack, DECIDABILITY QUESTIONS FOR PETRI NETS, Technical Report 161, Lab. Computer Science, M.I.T., giugno 1976

[2] J.L. Peterson, PETRI NETS, ACM Computing Surveys, settembre 1977

[3] E. Pless, H. Plünnecke, A BIBLIOGRAPHY OF NET THEORY, Gesellshaft für Mathematic und Datenverarbeitung, ISF-78-08, 1978

[4] C.A. Petri, CONCEPTS OF NET THEORY, Proceedings of the Symposium and Summer School on Mathematical Foundations of Computer Science, High Tatras, settembre 1977

[5] C.A. Petri, NON-SEQUENTIAL PROCESSES, Gesellshaft für Mathematic und Datenverarbeitung, ISF-77-01, 1977

[6] T. Agerwala, A COMPLETE MODEL FOR REPRESENTING THE COORDINATION OF ASYNCRONOUS PROCESSES, Hopkins Computer Research Report n. 32, John Hopkins University, Baltimore, luglio 1974

[7] V.E. Kotov, CONCURRENT PROGRAMMING WITH CONTROL TYPES, in "Quality Software", North Holland Pub. Co., 1978

[8] V.E. Kotov, AN ALGEBRA FOR PARALLELISM BASED ON PETRI NETS, Lecture Notes on Computer Science, n. 64, Springer-Verlag, 1978

[9] H.J. Genrich, K. Lautenbach, THE ANALISYS OF DISTRIBUTED SYSTEMS BY MEANS OF PREDICATE/TRANSITION-NETS, Lecture Notes in Computer Science n. 70, Springer-Verlag, 1979

[10] V. De Antonellis, G. Degli Antoni, G. Mauri, B. Zonta, FROM EDP PROCEDURES SPECIFICATION TO THE RELATED DATA BASE: A CONCISE VIEW, Working Conference on "Data Base Architecture" Venezia, giugno 1979

[11] P. Brinch Hansen, OPERATING SYSTEMS PRINCIPLES, Prentice Hall, 1976

[12] G.P. Rossi, C. Simone, PROCESSI CONCORRENTI IN UN SISTEMA MULTIPROCESSOR: DESCRIZIONE CON RETI DI PETRI, in questo numero di NOTE DI SOFTWARE

[13] C.A. Del Puppo, ASPETTI APPLICATIVI DELLE RETI DI PETRI, Tesi di Laurea in Fisica, Ist. di Cibernetica dell'Università di Milano, Anno Acc. 1978-79

[14] E.W. Dijkstra, A DISCIPLINE OF PROGRAMMING, Prentice Hall, 1976

[15] G. Castelli, F. De Cindio, G. De Michelis, G. Haus, M. Maiocchi, C. Simone, VERSO UNA METODOLOGIA PER LA COSTRUZIONE DI SPECIFICHE STRUTTURATE E CORRETTE DI PROCEDURE E PROGRAMMI, Convegno AICA, Bari, 1979

PROCESSI CONCORRENTI IN UN SISTEMA MULTICOMPUTERS: DESCRIZIONE CON RETI DI PETRI

G.P. Rossi, C. Simone
Istituto di Cibernetica - Univ. di Milano

Riassunto

Questo lavoro mostra un esempio di uso di configurazioni "standard" di reti di Petri (come suggerito in [1]), allo scopo di descrivere le modalità di comunicazione fra più processi distribuiti su una rete multiprocessor.

Abstract

This paper shows an example of use of "standard" Petri nets configurations (as suggested in [1]) to describe a communication protocol among processes distributed on a multiprocessor network.

1. INTRODUZIONE

In un altro lavoro in questo numero di NOTE DI SOFTWARE [1] si sono evidenziati alcuni schemi di interazione tra processi concorrenti e si è proposto un formalismo, basato sulle reti di Petri, che definisce per ciascuno di essi un modulo grafico che lo descrive. Tale formalismo permette la descrizione di più istanze di una stessa procedura, mediante l'utilizzo di marche colorate che percorrono la rete che la rappresenta, e la descrizione di processi e procedure per successivi livelli di raffinamento (cfr. § 3 del lavoro citato).

In questo lavoro, si vogliono applicare gli strumenti proposti alla descrizione di un sistema di processi che cooperano per realizzare un sistema di trasmissione dati.

Una delle principali difficoltà nella descrizione di questo tipo di sistemi risiede nell'intreccio di diversi livelli di astrazione: dai problemi puramente logici fino a quelli tipicamente realizzativi. Noi crediamo che il poter disporre di schemi standard e della loro rappresentazione mediante le reti di Petri sia un valido ausilio, in quanto essi aiutano ad identificare i diversi aspetti delle interazioni tra processi e, quindi, a fornire una descrizione in cui i diversi problemi (sincronizzazione, mutua esclusione, cooperazione) sono localizzati, livello per livello. Una descrizione così costruita consente, da un lato, di aumentare la comprensione della struttura logica delle interconnessioni delle varie componenti del sistema e, dall'altro, di mettere chi eventualmente dovrà occuparsi della fase implementativa in condizione di separare i vincoli logici del problema da quelli realizzativi, legati agli strumenti implementativi utilizzati.

Ricerca svolta con il contributo del CNR (contratto n. 79.00981.11.115.11252)

2. DEFINIZIONE DEL PROBLEMA

Il problema considerato è un tipico problema di comunicazione fra tasks distribuiti in un sistema multiprocessors.
Tale comunicazione è caratterizzata:

— dal *protocollo di comunicazione*, che definisce le modalità che governano lo scambio di messaggi tra i tasks;

— dal *supporto fisico*, cioè dalla struttura di interconnessione fra computers (linee fisiche con una determinata struttura topologica) su cui i messaggi viaggiano da un nodo trasmettitore ad un nodo destinatario.

Nell'esempio considerato, ciascun task può inviare due tipi di messaggi:

— un messaggio di *richiesta* di informazione: il task è detto *receiver*;

— un messaggio di *risposta*: il task è detto *sender*.

Ovviamente, ogni task può essere di volta in volta sender o receiver.

La procedura di dialogo, descritta in fig. 1, realizza un protocollo di comunicazione con le seguenti caratteristiche:

— un messaggio di risposta può essere inoltrato da un sender al rispettivo receiver se o solo se esiste una precedente richiesta di quest'ultimo (*messaggio di richiesta*);

— ciascuna coppia sender-receiver è univocamente determinata da una linea logica (*canale virtuale*) che idealmente li collega; ad ogni inizializzazione del sistema, è noto il numero di canali virtuali (indicato nel seguito con *cmax*) che esso contiene e, quindi, le coppie di tasks abilitate allo scambio dei messaggi (fig. 2);

— ad ogni istante, più tasks possono richiedere contemporaneamente di inviare un messaggio di richiesta o di risposta.

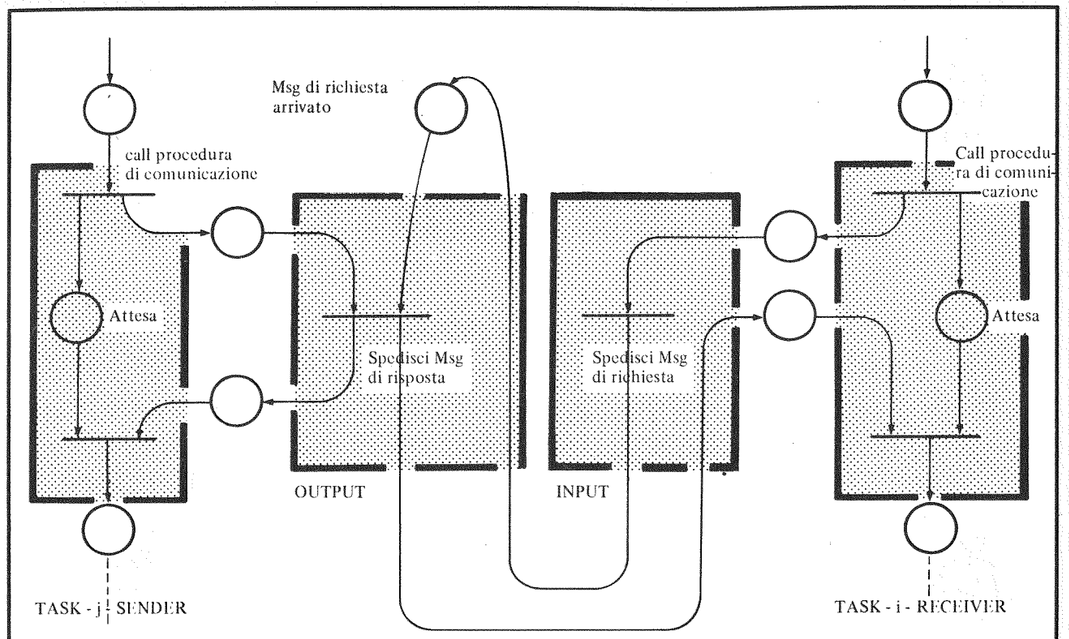


Fig. 1 – Procedura di dialogo

Tale procedura di dialogo, tratta da [2], è volutamente semplice e si rifà, in parte, ai protocolli di più basso livello (packet-level) delle reti di computers più complesse [3] [4].

3. LA DESCRIZIONE DEL SISTEMA

Il nostro obiettivo è quello di descrivere il software che realizza la procedura di dialogo appena definita.
La fig. 1 dà la descrizione al più alto livello: in essa sono del tutto trasparenti la collocazione dei tasks nella rete e le caratteristiche fisiche di quest'ultima. Siamo, quindi, nel caso di cooperazione tra processi mediante una struttura dati destinata alla comunicazione.

Il secondo livello di descrizione coinvolge gli aspetti implementativi dei canali virtuali. Bisogna, infatti, tener conto della struttura a nodi del sistema, sulla quale si ipotizza solo che tutti i nodi della rete abbiano la stessa potenzialità di calcolo rispetto alla gestione della comunicazione e costruire una struttura che realizzi i canali virtuali, alla quale i tasks di uno stesso nodo possono accedere simultaneamente per inviare messaggi di richiesta o di risposta. Due, quindi, sono le procedure di accesso alla struttura di dati di comunicazione: (fig. 3)

— *input-receive*, utilizzata dai receivers;

— *output-send*, utilizzata dai senders.

Inoltre, poichè più richieste di invio possono avvenire simultaneamente, è necessario disporre di un *buffer di nodo*, in cui i messaggi di entrambi i tipi vengono immagazzinati prima di essere effettivamente spediti. In questo modo, la frequenza delle richieste può essere asincrona rispetto alla frequenza di spedizione sulla linea (che nel seguito indiche-

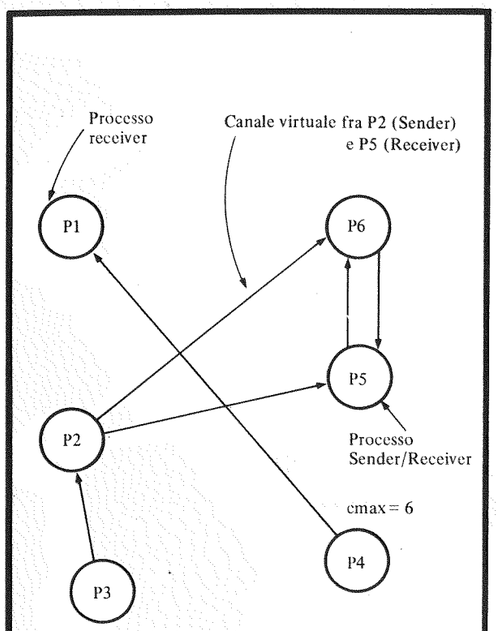


Fig. 2 – I canali virtuali - esempio

remo con *bus-link*), che dipende ovviamente dalle caratteristiche fisiche di quest'ultima.

Si incomincia, quindi, a delineare una struttura di software di nodo che contiene le suddette procedure ed il buffer (si veda fig. 8).

Descriviamo allora le prime.

3.1. Input-receive (fig. 4)

Ha il compito di:

— immagazzinare nel buffer il messaggio di richiesta;
— mettere in stato di attesa il task chiamante;
— risvegliarlo quando è arrivato il messaggio di risposta.

Le ultime due attività riguardano la realizzazione del protocollo di comunicazione; sono, dunque, attività di sincronizzazione su di una ben precisa condizione: l'arrivo del messaggio richiesto.

L'operazione di inserimento nel buffer è ancora realizzabile mediante una procedura: in fig. 5 l'evento non elementare relativo è chiamato *call buf-send*.

L'espansione delle condizioni non elementari impone delle scelte sulla struttura di dati, in modo da poter mettere in relazione l'arrivo di un messaggio di risposta con il risveglio del task che lo sta aspettando. A tale scopo, introduciamo una coppia di condizioni per ciascun canale virtuale ($1 \leq i \leq cmax$):

— Risp.(i): è il predicato che indica l'avvenuto arrivo del messaggio di risposta destinato ad un task receiver dell'i-esimo canale virtuale;
— TRec (i): è il posto di controllo che contiene una marca del colore associato al task receiver.

Le regole di scatto definite per le Reti di Petri con marche colorate garantiscono il corretto abbinamento: task-receiver, messaggio di risposta, sotto l'ipotesi che le TRec (i) siano inizializzate ciascuna con una marca del colore associato al task receiver dell'i-esimo canale virtuale.
Questo motiva anche l'arco rientrante nelle TRec (i), dopo lo scatto della transizione relativo, in modo da ripristinare le condizioni iniziali.

3.2. Output-send (fig. 6)

Ha il compito di:

— individuare il canale su cui si vuole spedire il messaggio di risposta;
— verificare se è già arrivata la relativa richiesta e, in caso affermativo, inserire il messaggio di risposta nel buffer;
— altrimenti, mettere il task chiamante in stato di attesa.

Come nel caso precedente, le condizioni non elementari si espandono in *cmax* predicati Rich (i), che indicano la presenza della richiesta da parte del receiver associato all'i-esimo canale, e in *cmax* predicati C (i), che indicano la richiesta di spedire un messaggio di risposta sull'i-esimo canale.

Il primo evento non elementare si espande mediante una richiesta di inizializzazione intermedia, che ha associata la condizione: *quale canale*?. Il passaggio di parametri ad una procedura (simulato dalla marca colorata) è, infatti, a tutti gli effetti, una operazione di acquisizione di informazione dall'esterno assimilabile ad una lettura (come de-

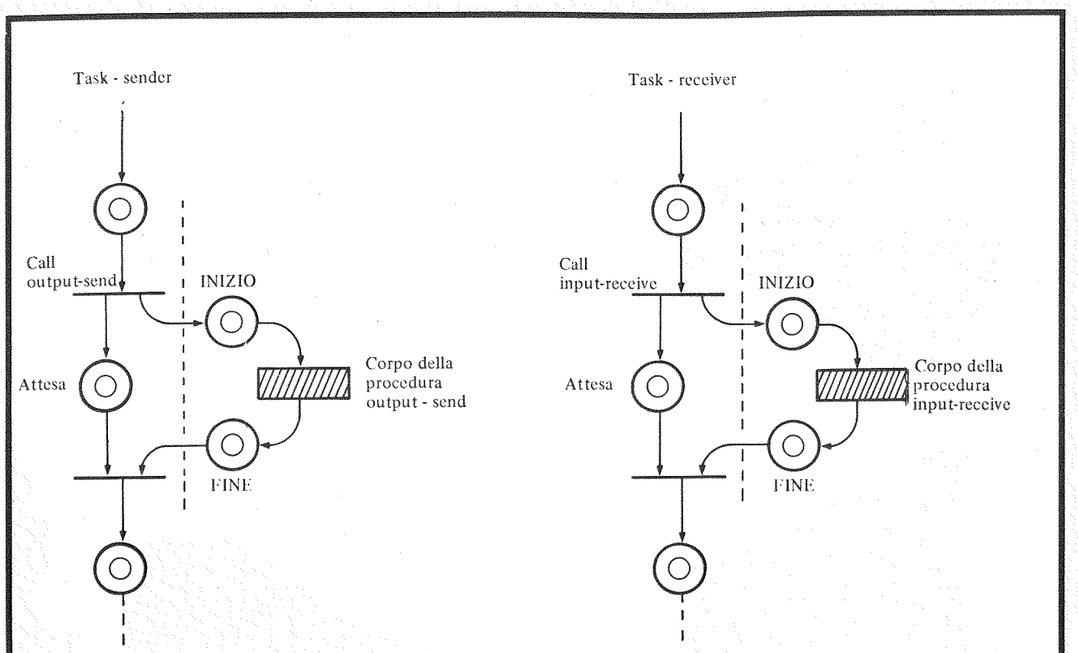


Fig. 3 – Tasks

scritto in [1]).

In fig. 7 è descritta la procedura *output-send*.

In fig. 8 è riassunto quanto si conosce della struttura di ciascun nodo: emerge con evidenza che il software di nodo è ancora incompleto. È compito del terzo livello di descrizione colmare questo vuoto: infatti, è necessario tenere conto di caratteristiche della linea fisica, del tutto inessenziali nei livelli precedenti.

Poichè, in generale, la struttura topologica della rete non prevede per ciascun canale virtuale un bus-link corrispondente, è possibile che esso sia realizzato mediante un cammino che parte dal nodo del task che invia il messaggio al nodo di quello che lo riceve, passando per nodi di transito (fig. 9). Questo implica che tre sono i possibili tipi di messaggi che possono giungere ad un nodo dalla linea in esso entrante:

— messaggi di richiesta di informazione destinati ad un task di quel nodo;
— messaggi di risposta destinati ad un task di quel nodo;
— messaggi di entrambi i tipi, ma destinati a tasks di altri nodi (*messaggi in transito*).

Il modulo A di fig. 8 contiene, dunque, un processo col compito di prelevare dal bus in entrata al nodo i tre tipi di messaggi: sia esso il *reader*. Esso può essere visto come un processo senza termine, che esce dallo stato di attesa quando giunge un messaggio al nodo dal bus.
In fig. 10 descriviamo questo processo: poichè a questo livello non interessa conoscere come avviene il prelevamento dei messaggi dal bus, indichiamo tale operazione con *input from bus-link*, senza specificarla ulteriormente.

La fase di analisi del messaggio arrivato deve stabilire il tipo del messaggio e il canale virtuale che identifica il destinatario: si utilizza quindi ancora lo schema di inizializzazione intermedia sulle condizioni: *quale tipo*? e *quale canale*?, rispettivamente.

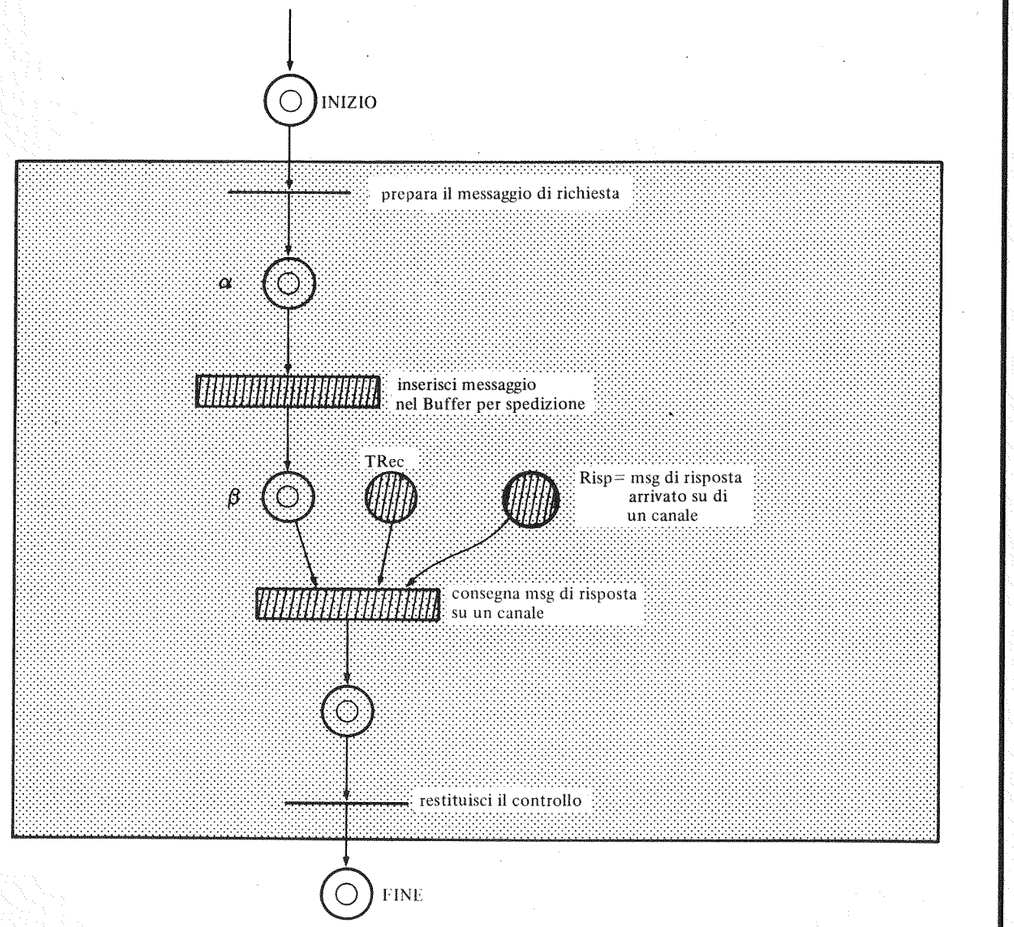


Fig. 4 – Procedura input - receive

Per i messaggi in transito, si può scegliere di inserirli nel buffer insieme a quelli in partenza dal nodo: un processo, detto *writer*, si preoccuperà di immetterli nel bus di uscita dal nodo insieme agli altri. Con questo, il software del nodo è completo. In fig. 11 diamo la descrizione completa del *reader*, ed in fig. 12 quella del *writer*, ricordando che esso utilizza una procedura devoluta al prelevamento di messaggi dal buffer (*buf-receive*) e una procedura per il loro inserimento nel bus di uscita (*output to bus-link*).

Restano ancora da specificare le procedure di accesso al buffer. Diamo la loro descrizione in fig. 13, rimandando a [1] per ogni commento a proposito. Notiamo soltanto che è necessaria una condizione di mutua esclusione per la *buf-send* (*mutex*), in quanto tutti i tasks del nodo più il *reader* la possono utilizzare, mentre tale condizione non è necessaria per la *buf-receive*, in quanto essa può esse-

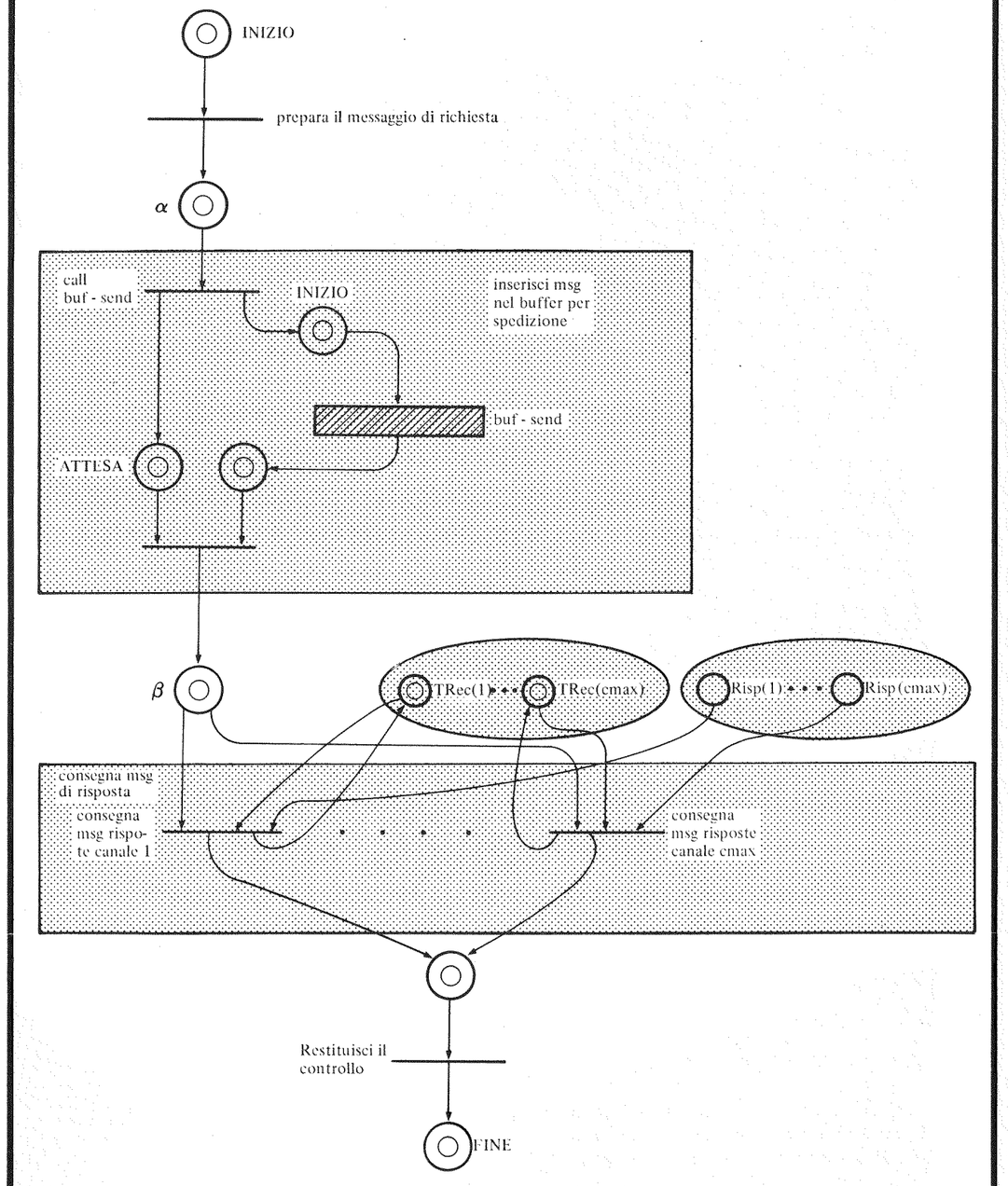


Fig. 5 – Procedura input - receive

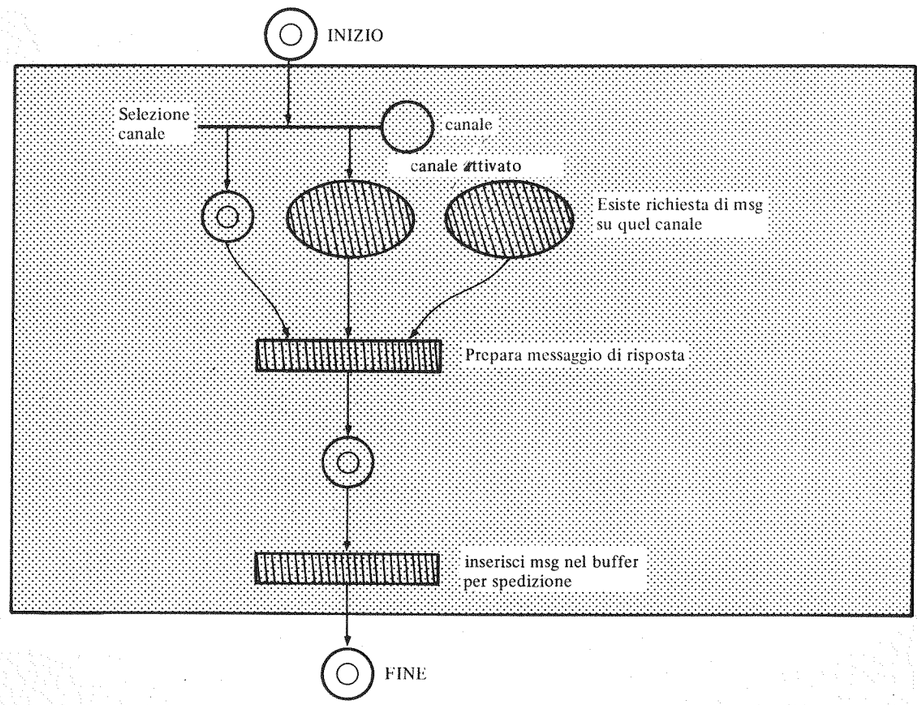


Fig. 6 – Procedura output - send

re utilizzata soltanto dal *writer*.

Nella fig. 14 è descritto il flusso del controllo e dei dati del software di ciascun nodo, mentre in fig. 15 si dà la descrizione dettagliata delle attività compiute da ciascuna delle sue componenti.
Da essa, emerge la complessa struttura del software di nodo, imperniata sulle condizioni di sincronizzazione Rich (i) e Risp (i), sulle procedure di accesso al buffer e sull'attività dei processi *reader* e *writer*.

4. STRUTTURA DI INTERCONNESSIONE

Per finire, diamo un cenno ad una possibile configurazione della struttura topologica del supporto fisico che realizza la trasmissione dei messaggi.

Una possibile struttura, come proposto in [2], è un semplice anello, in cui un numero fissato di nodi sono connessi ciclicamente da una linea unidirezionale (fig. 16). Tale struttura, ampiamente usata per semplici reti locali che connettono minicomputers, ha, come evidente contropartita alla sua alta modularità e semplicità di realizzazione, un limitato grado di riconfigurabilità ai guasti ed un alto tempo di consegna dei messaggi [5].

Quello che preme osservare, è che eventuali maggiori complessità della struttura fisica di comunicazione verrebbero scaricate unicamente sui processi *reader* e *writer* e sulle procedure di accesso al buslink, mentre i livelli di descrizione più alti non verrebbero modificati, se non in caso di un cambiamento della procedura di dialogo.

5. CONCLUSIONI

L'esempio esaminato in dettaglio nei paragrafi precedenti presenta molti e significativi aspetti di

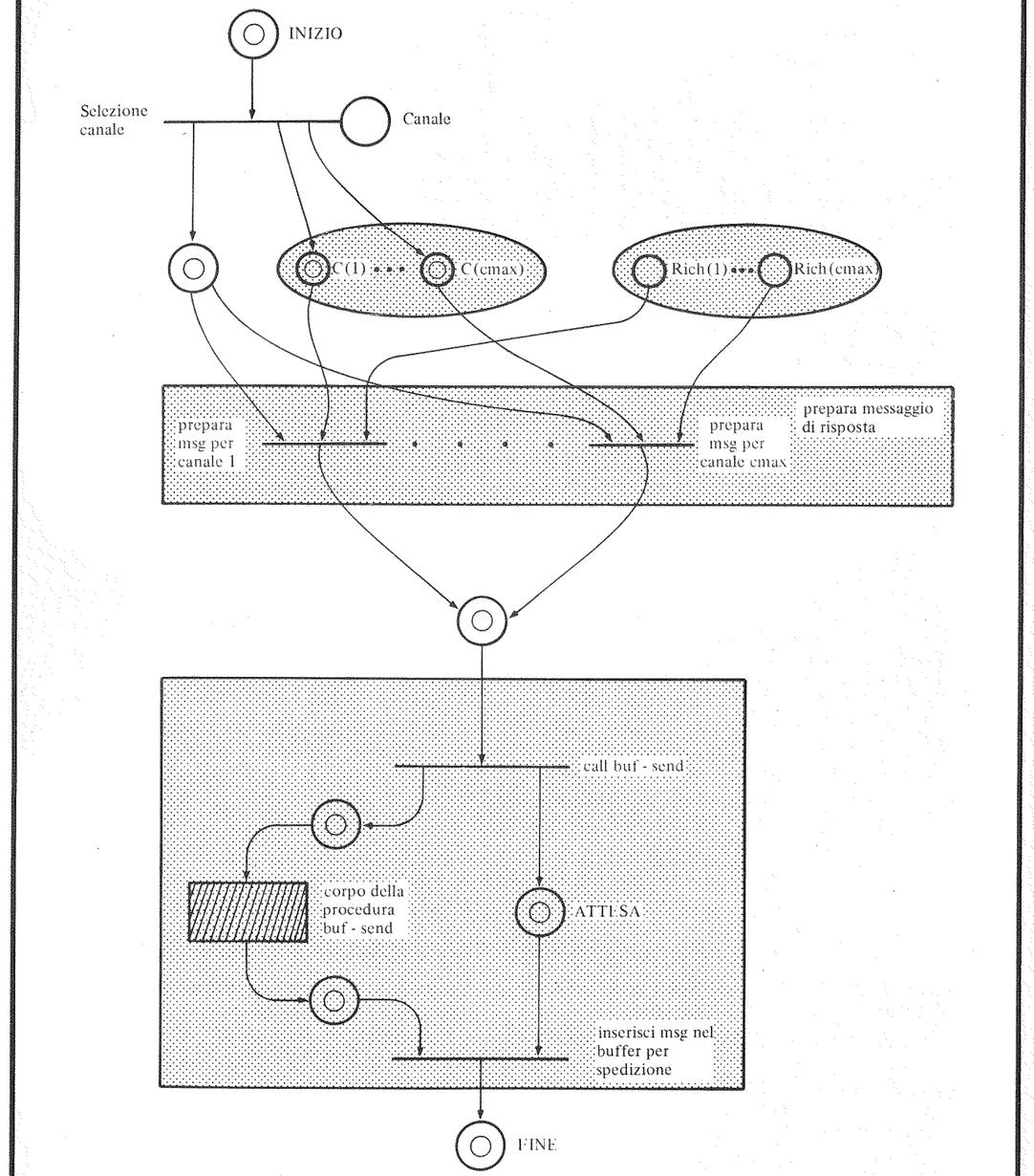


Fig. 7 – Procedura output - send

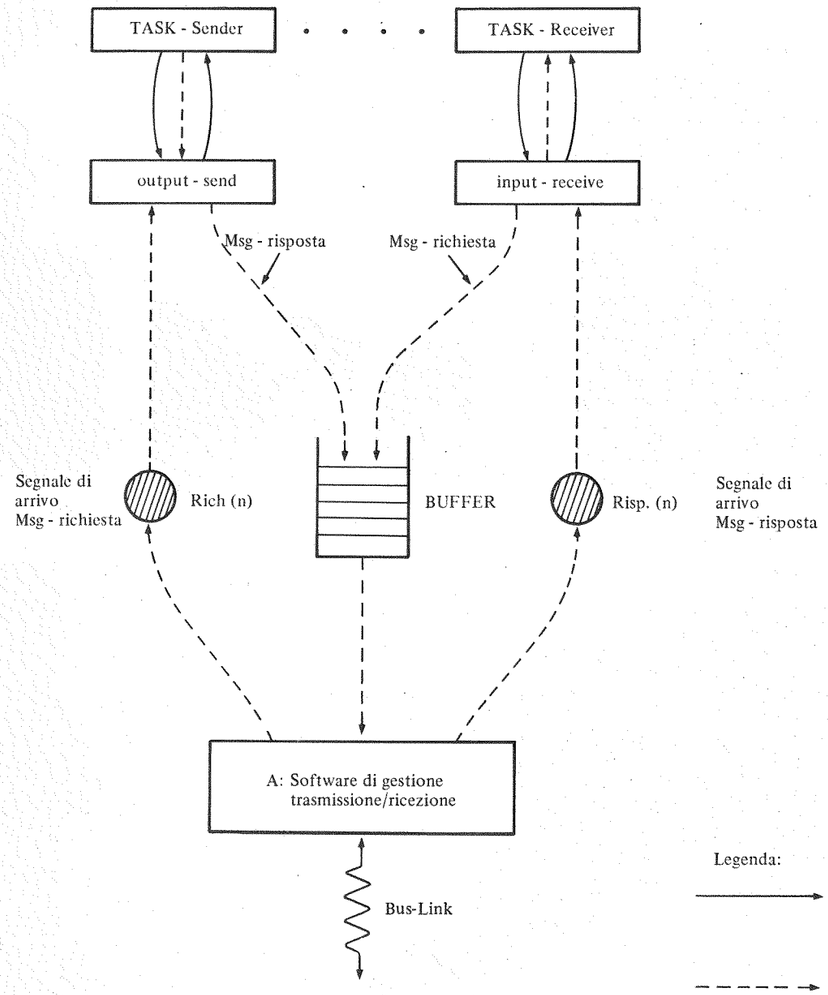


Fig. 8 – Software di ogni nodo della rete

concorrenza e interazione fra processi, complicati dal fatto che la comunicazione avviene, oltre che fra tasks di uno stesso nodo, anche fra tasks distribuiti su una rete.

Affrontare un tale problema sulla base di quanto proposto in [2] e costruire la descrizione logico-funzionale del sistema come fatto nel § 3:

- aiuta a strutturare il sistema in un numero di moduli (tasks) fra loro concorrenti;
- evidenzia tutti i punti di interazione fra processi, consentendo anche di simulare l'interfoliazione dei vari tasks;
- aumenta la competenza sul problema da affrontare;
- consente di visualizzare il più alto grado di parallelismo fra i tasks coinvolti, in quanto si procede in modo indipendente da vincoli implementativi (alcune restrizioni sull'accesso a risorse comuni, nella descrizione del sistema con i monitors di Concurrent Pascal [2], sembrano essere imposte dallo strumento descrittivo stesso, e non necessarie alla soluzione del problema);
- consente all'implementatore del sistema ottenuto di separare i vincoli logici da quelli implementativi.

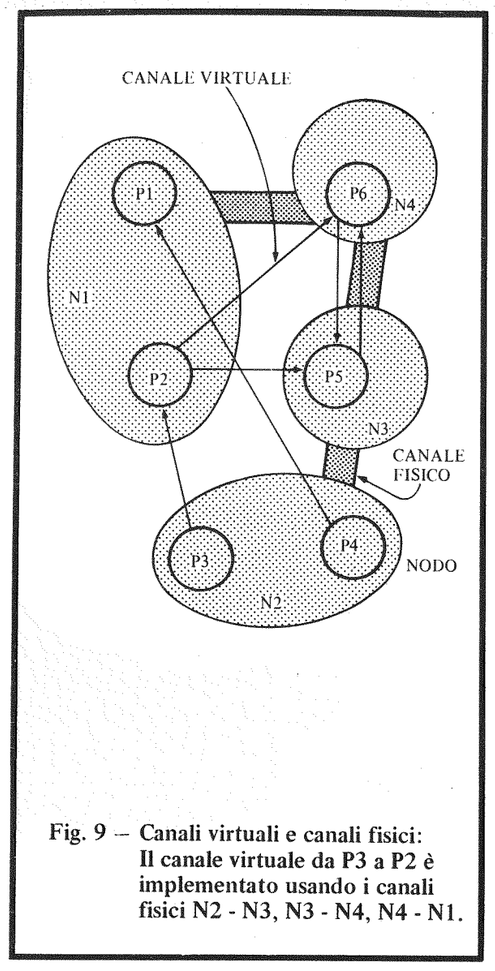


Fig. 9 – Canali virtuali e canali fisici: Il canale virtuale da P3 a P2 è implementato usando i canali fisici N2 - N3, N3 - N4, N4 - N1.

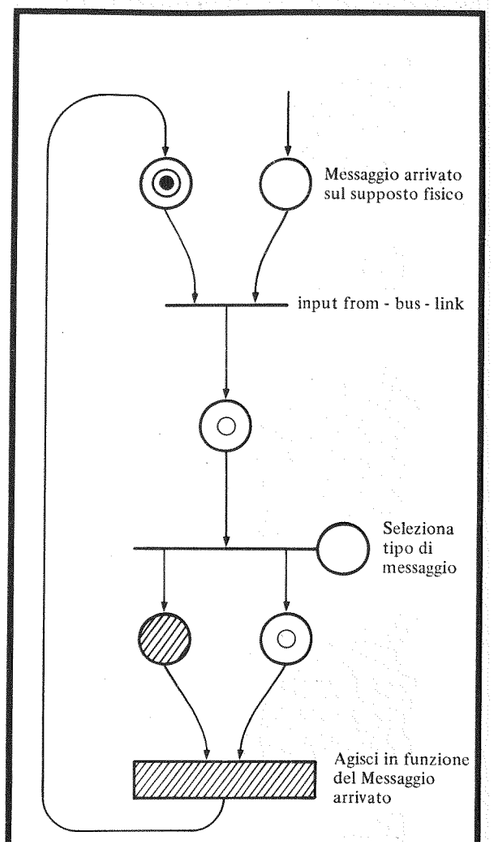


Fig. 10 – Processo Reader

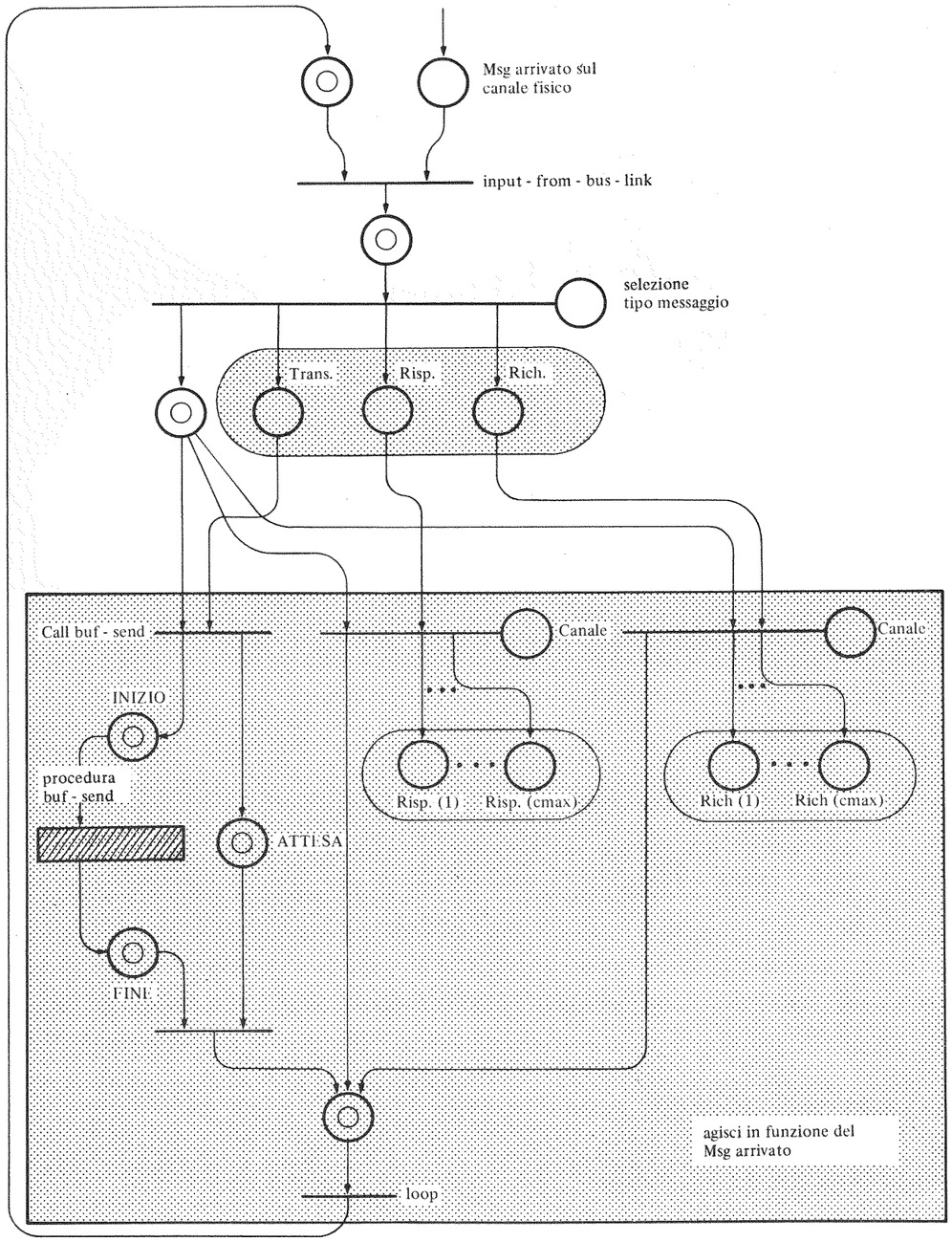


Fig. 11 – Processo reader

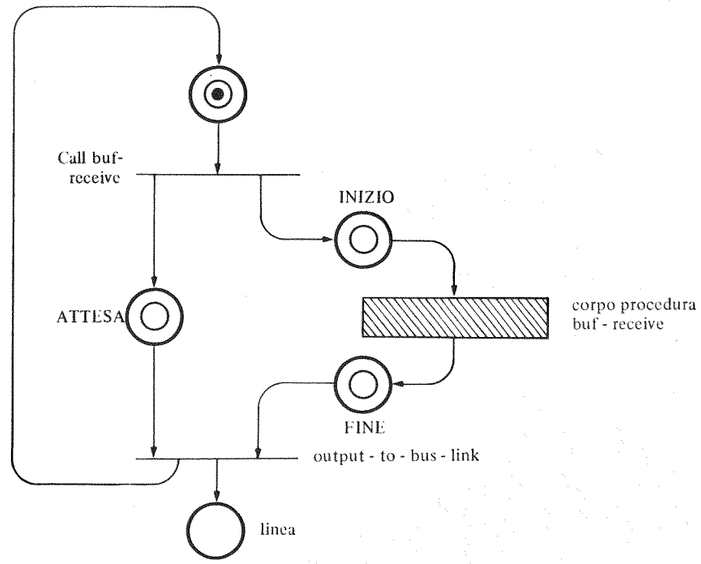


Fig. 12 – Processo writer

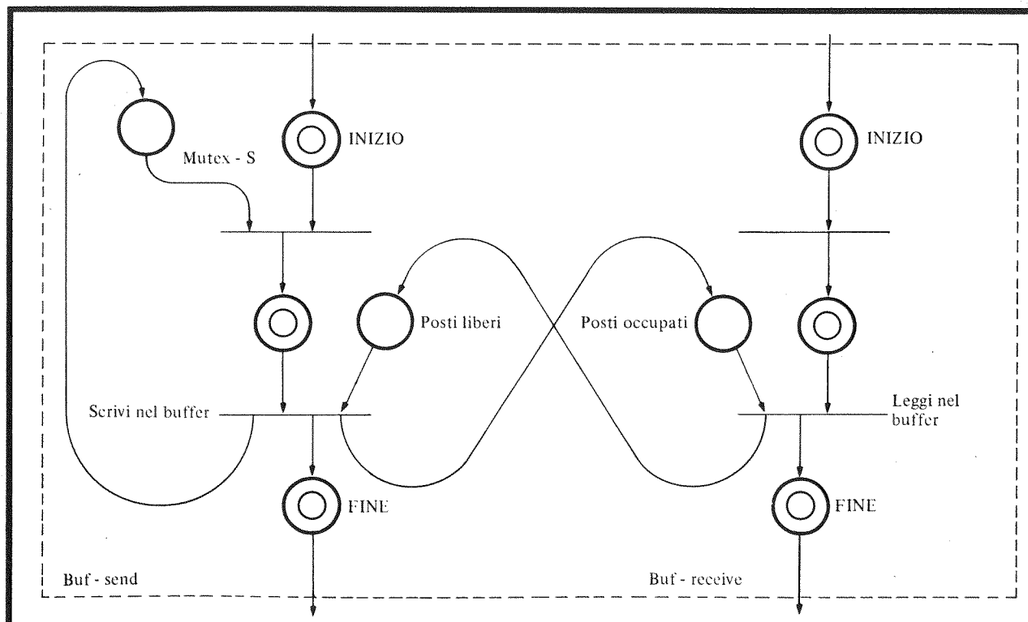


Fig. 13 – Procedure Buf - send e Buf - receive

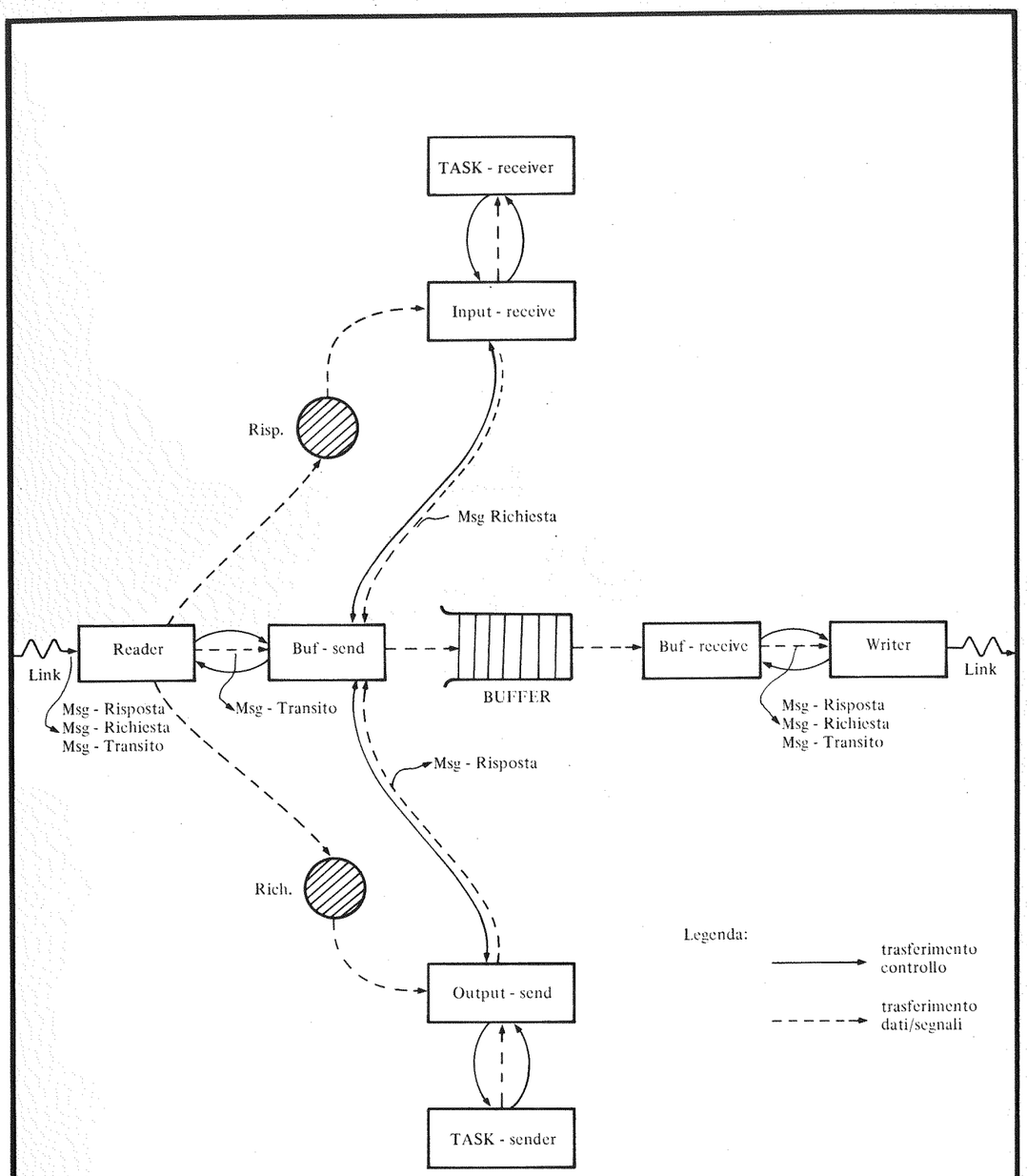


Fig. 14 – Schema generale del software dei nodi

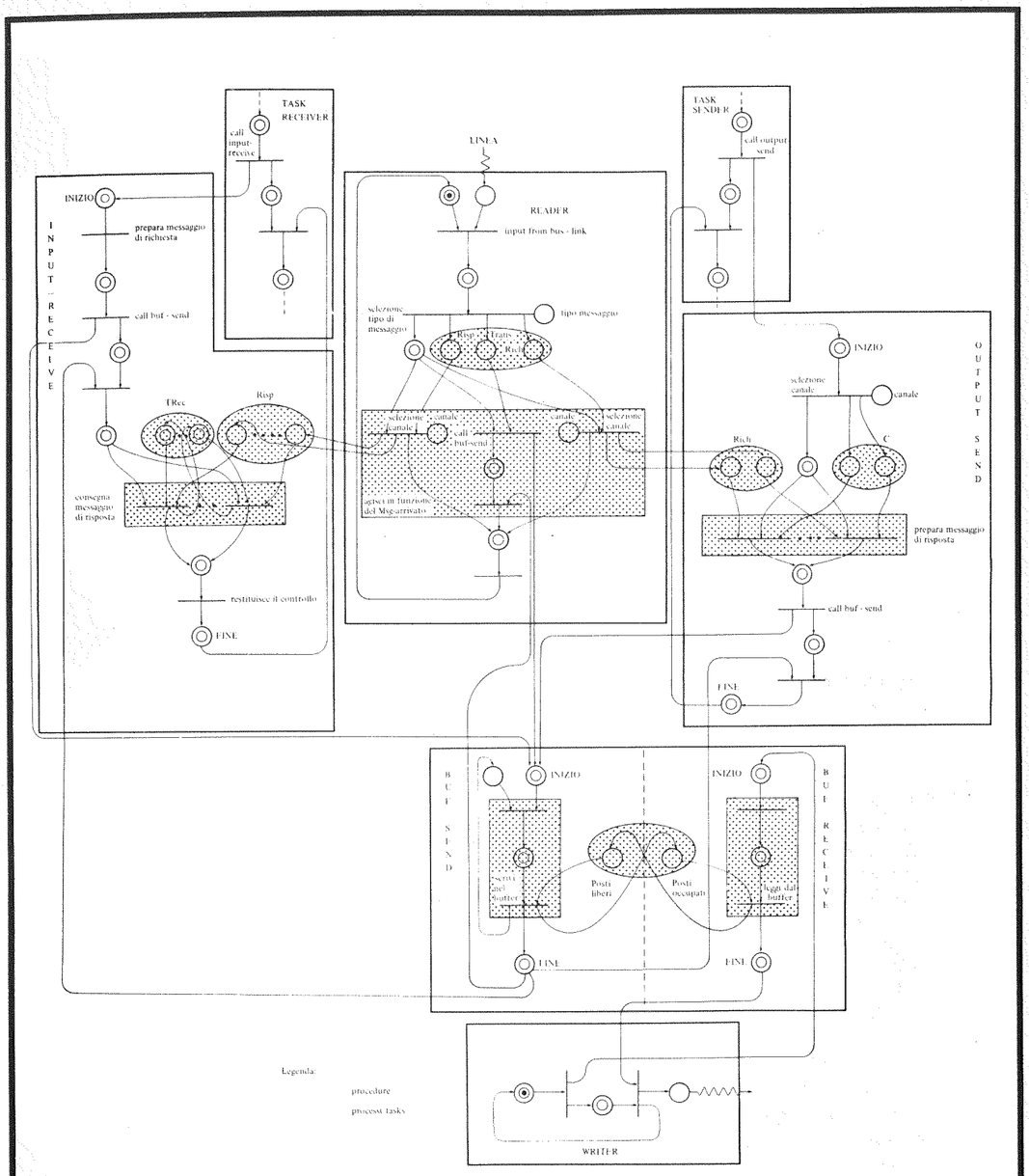


Fig. 15 – Schema complessivo del software di nodo

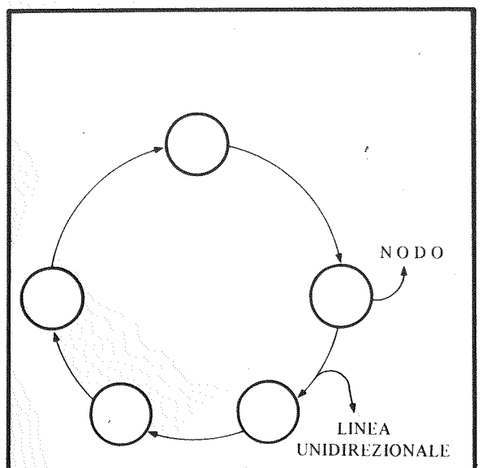


Fig. 16 – Struttura topologica ad anello

6. BIBLIOGRAFIA

[1] G.P. Rossi, C. Simone, SCHEMI DI INTERAZIONE TRA PROCESSI CONCORRENTI DESCRITTI MEDIANTE LE RETI DI PETRI, in questo numero di NOTE DI SOFTWARE

[2] P.B. Hansen, NETWORK: A MULTIPROCESSOR PROGRAM, IEEE Trans. on Software Eng., Vol. SE—4 N. 3 (1978)

[3] D.W. Davies, D.L.A. Barber, COMMUNICATION NETWORKS FOR COMPUTERS, Wiley & Sons (1976)

[4] Cost Project 11 EIN, SPECIFICATION OF THE INTERFACE BETWEEN A SUBSCRIBER COMPUTER AND THE NETWORK SWITCHING CENTRE, EIN/76/01/06

[5] G. Anderson, E. Jensen, COMPUTER INTERCONNECTION STRUCTURES: TAXONOMY, CHARACTERISTICS AND EXAMPLES, ACM Comp. Surveys, Vol. 7 N. 4 (1975)

UNA METODOLOGIA PER IL PROGETTO DI SISTEMI DI CONTROLLO DI PROCESSO BASATA SULLE RETI DI PETRI

A. Beltramini
Istituto di Cibernetica dell'Università di Milano

1. PREMESSA

Le maggiori difficoltà che si incontrano normalmente nella costruzione di apparecchiature per il controllo dei processi industriali sono connesse alla descrizione del comportamento del sistema, ovvero alla comprensione del suo funzionamento.

La difficoltà del rapporto tra l'utente ed il costruttore esige un approfondimento continuo, che normalmente avviene strada facendo e, talora, si protrae anche durante la costruzione delle apparecchiature riservando, non poche volte, amare sorprese che si evidenziano solo durante i collaudi del sistema.

Obiettivamente sono molti i fatti che influenzano tale rapporto e dipendono, oltre che dalla preparazione specifica degli interlocutori, dal fatto che non viene quasi mai applicato un "metodo" preciso di approfondimento del problema, metodo che implica una distinzione di ruoli tra l'utente che sa *che cosa bisogna fare* ed il costruttore che sa *come si fa*.

Accade non raramente che la descrizione del funzionamento sia riferita agli oggetti fisici che lo supportano o, ancora peggio, a precedenti e diverse realizzazioni che nulla hanno a che vedere con quanto si è in procinto di fare.
La sostituzione di una unità di controllo analogica con una a micorprocessore ha posto e pone anche difficoltà come quelle poco sopra evidenziate.

A supporto di questa affermazione adduciamo la cosiddetta difficoltà di costruzione del prototipo che ovviamente, al di là delle difficoltà obiettive connesse alla nuova progettazione, contiene anche quei margini di indeterminazione del comportamento del sistema e da qui l'introduzione di quelle ridondanze che qualche volta possono venire assimilate all'uso del cannone per l'uccisione della mosca, mentre, qualche altra volta, si scopre che l'apparecchiatura va rifatta per l'esistenza di qualche "buco" concettuale che stranamente non era mai emerso nei lunghi scambi di informazioni.

L'autore desidera ringraziare Gianni Degli Antoni per l'incoraggiamento e lo stimolo a tradurre un'esperienza professionale in un documento che può contribuire ad un approccio metodologico al problema del controllo dei processi industriali.

2. IL METODO

Diviene, pertanto, di grande importanza l'uso di un metodo che garantisca a utente e costruttore il pieno successo dell'intervento, con tempi e fatica ragionevoli. Metodo che renda scientifici l'approccio, la comprensione e la descrizione del problema mediante l'uso di tecniche che consentano omogeneità nella articolazione dei discorsi di vari livelli.

Così come si procede, nell'hardware, dallo schema a grandi blocchi agli schemi a blocchi ed infine agli schemi elettrici delle singole piastre corredati delle relative topografie, similmente si dovrà procedere nella descrizione del software, partendo da un livello generale e discendendo man mano alla descrizione particolareggiata dai singoli packages.

Questa tecnica di descrizione è motivata, tra l'altro, dal fatto che essa si deve rivolgere a diverse aree di utenti, che vanno dal livello manageriale a quelli di esercizio e di manutenzione.
Se la descrizione è corretta nel senso del progressivo sviluppo dei dettagli dall'alto al basso, sia per la parte hardware che per la parte software, oltre che costituire l'elemento fondamentale della progettazione, essa sarà la base per le documentazioni che dovranno accompagnare l'apparecchiatura durante l'esercizio.

L'utilità del metodo (top-down) è tale da consentire non solo la visione più corretta del sistema, ma anche la generazione di informazioni storicamente utili sia per il costruttore che per l'utente, articolate secondo i livelli di utenza che costituiranno per tutti gli operatori interessati un valido precedente.

3. LA MODULARITA'

Il costruttore accorto ha certamente notato come l'hardware, a qualsiasi genere di processo venga

dedicato, offre molti aspetti comuni a tutti i processi. Applicando a ciò una serie di considerazioni generalizzanti è possibile costruire la quasi totale struttura hardware con elementi modulari iterabili.

L'iterabilità consente agli stessi moduli di essere impiegati per coprire qualsiasi 'larghezza' sia richiesta alla funzione.

Per funzioni, invece, potrà essere sicuramente descritta la quasi totalità dei processi.

Se tutto ciò appare più semplicemente conseguibile per l'hardware, occorre perseguire la stessa impostazione anche per il software, ed ancora potremo proseguire considerando non più le strutture che consentono di svolgere certe funzioni ma le funzioni stesse. Ragionando per funzioni è possibile, tra i processi pensabili, unificarne alcune e da qui passare agli aspetti di modularità dei packages di software.

Se il lavoro viene fatto molto accuratamente, è possibile ottenere supporti modulari tali da rendere la progettazione più semplice e sicura, divenendo, essa, un insieme di scelte di oggetti hardware e software ormai noti e dal funzionamento garantito.

L'impiego di questi moduli introduce il concetto di architettura, che nel caso dell'hardware utilizza tecniche grafiche conosciute, anche se molte volte usate in maniera imprecisa specialmente ai livelli più alti, ovvero schemi a grandi blocchi e schemi a blocchi. Nel caso del software, invece, il problema della "comprensione-descrizione" è quasi sempre lasciato alla soggettività del progettista.

Pur non volendo approfondire l'interazione hardware-software, in questa trattazione riteniamo importante mostrare come sia possibile, utilizzando particolari strumenti, descrivere il funzionamento di un sistema in modo che da tale descrizione sia esplicitamente ricavabile l'architettura del software.

Le reti di Petri sono lo strumento che abbiamo utilizzato per la descrizione della architettura del funzionamento di un sistema.

4. APPLICAZIONI

La nostra attenzione viene ora rivolta alle apparecchiature di controllo di processi facenti uso di microprocessori, in quanto riteniamo con ciò di sodisfare alla più ampia casistica.

Tra le possibili apparecchiature di controllo, scegliamo quelle che effettuano il controllo digitale diretto "DDC", supponendo che su queste verranno concentrate le maggiori attenzioni, in quanto, a nostro avviso, esibiscono quelle caratteristiche di affidabilità e costi da renderle proponibili nello stragrande numero di applicazioni.

Come controllo digitale diretto intendiamo quel tipo di controllo che usa il microprocessore e le sue interfacce come unico elemento di gestione, senza ricorrere a back-up analogici su cui effettuare la supervisione, come avviene in una molteplicità di impieghi del calcolatore di processo.

Nelle macchine DDC l'arresto del microprocessore causa l'arresto ed il blocco del processo, mentre nelle apparecchiature con back-up analogico l'arresto del calcolatore conduce alla possibilità di gestione semiautomatica, con l'uso manuale delle regolazioni analogiche che precedentemente erano gestite dal calcolatore di processo.

5. ASPETTI STRUTTURALI

Consideriamo un sistema in cui distinguiamo una unità controllante ed un'unità controllata (Fig. 1): possiamo immediatamente suddividere i vettori di controllo in:

— IA Ingressi analogici;
— ISC Ingressi di segnali di stato (ON-OFF) provenienti dal campo;

ed altrettanto faremo per i vettori di comando, suddividendoli in:

— UA Uscite analogiche;
— USC Uscite di segnali di stato (ON-OFF) rivolti al campo;

L'unità controllante, infine, è assistita da una consolle operatore atta ad evidenziare mediante il vettore

— USO Uscite di segnali di stato rivolti all'operatore

quegli stati del sistema che richiederanno una particolare attenzione, e tra questi i segnali di emergenza e di allarme provenienti dal circuito di blocco.

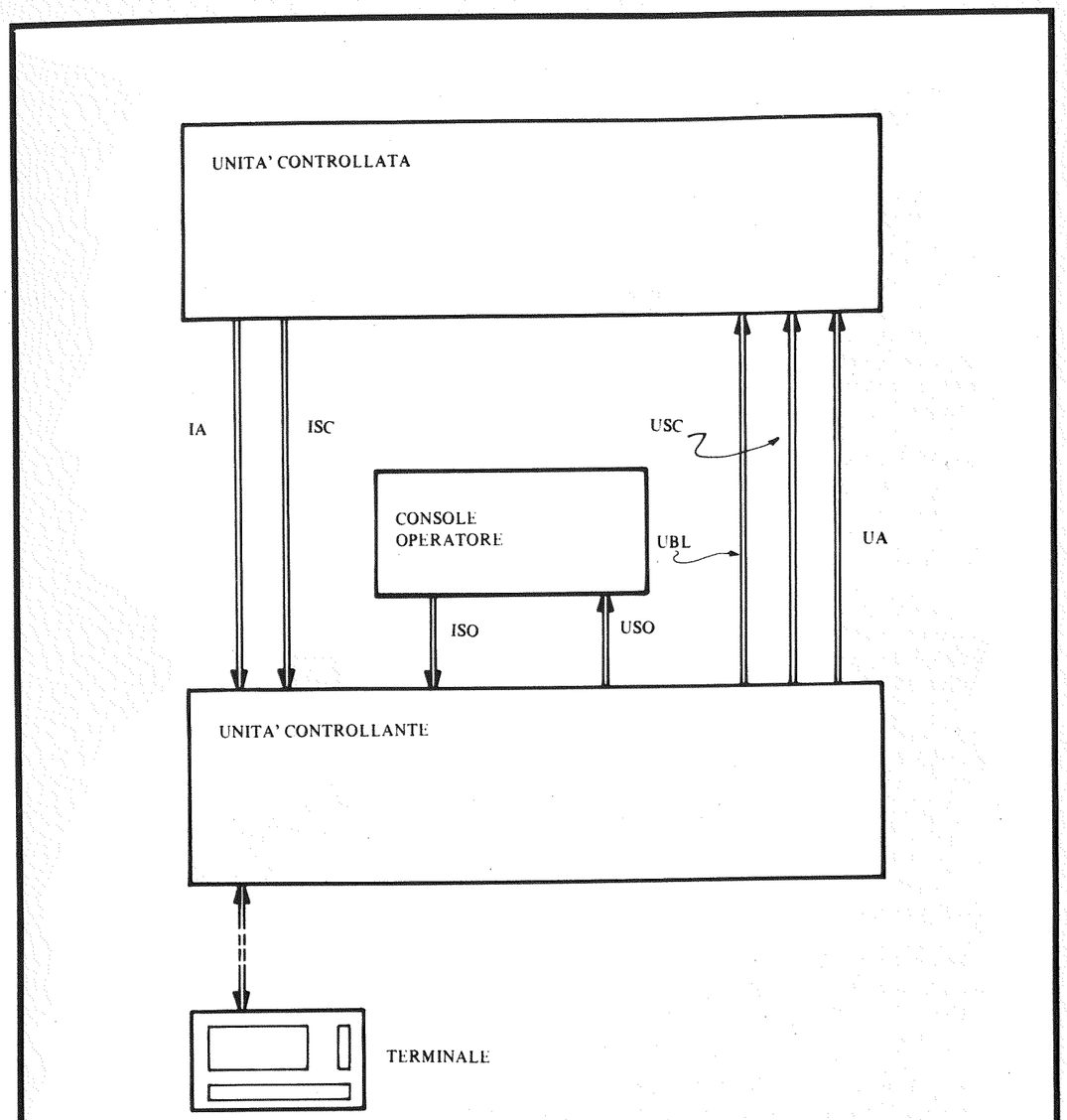


Fig. 1

Inoltre, dalla consolle operatore si muoverà il vettore

— ISO Ingressi di segnali di stato provenienti dall'operatore

che rappresenterà le informazioni relative all'operatore in campo, e tra queste quelle di emergenza o di allarme, tali da porre il sistema nelle condizioni di sicurezza.

Per scelta, le informazioni che si muoveranno da e per l'operatore saranno discrete, mentre le informazioni analogiche (set point, soglie di allarme etc.) utilizzeranno un terminale di I/0 dotato di video e tastiera, e potranno essere gestite anche in maniera remota.

Tra le informazioni provenienti dall'operatore, alcune avranno caratteristiche importanti, relative alla gestione in campo del sistema: ad esempio, i by-pass degli organi di attuazione, utilizzabili per controllarne il corretto funzionamento.

Tra queste, particolare importanza avranno le informazioni di emergenza che metteranno l'impianto in stato di blocco, a cui conseguiranno le condizioni di sicurezza degli attuatori.

Lo stato di blocco sarà portato al campo dalla azione del vettore

— UBL Uscita segnali di stato rivolta al o ai dispositivi di blocco;

e dalle concomitanti azioni fatte sui circuiti attuatori mediante il vettore USC.

Questi ultimi elementi descrittivi introducono un ulteriore e più approfondito livello della struttura delle interfacce che stanno tra il microcalcolatore ed il campo (Fig. 2).

Trascurando solamente gli aspetti elettrici, quali le separazioni galvaniche che esigono una trattazione particolare che non rientra negli scopi del presente documento, passiamo a considerare lo schema a grandi blocchi della figura 2.

Gli ingressi analogici IA provenienti dal campo sono direttamente portati ad un multiplexer analogico, adatto alla loro scansione e conversione.
La scansione degli stessi è esercitata dal microcalcolatore mediante il vettore OUa, mentre l'acquisizione della variabile convertita verrà effettuata dal vettore INa.

Tra gli ingressi analogici, vanno anche annoverate, "con esclusive finalità diagnostiche", le uscite analogiche che verranno portate al campo dalle uscite UAi dopo la conversione D/A del vettore OUd.
Gli ingressi di segnali di stato, provenienti dal campo ISC, sono inviati ad un circuito rivelatore delle avvenute variazioni, tale da fare in modo che venga consentita l'acquisizione solamente quando è avvenuta una variazione.
Chiaramente, occorrerà provvedere, tramite una preliminare acquisizione di un vettore di stato INg, alla individuazione di quel campo del vettore di ingresso (pedici b,c,d) in cui si è manifestata la variazione.

Gli ingressi di segnali di stato non sono solo provenienti dal campo, ma proveranno anche all'operatore, quali quelli di imposizione del blocco, di by-pass di comando degli attuatori etc., che avranno significati da comunicare al microprocessore.

Costituirà ancora un vettore in ingresso lo stato assunto dal circuito di blocco ISBL. Tutti questi vettori, per la loro caratteristica di asincronicità, mostreranno la loro avvenuta variazione tramite la preliminare acquisizione del vettore proveniente dallo status register. Infine si acquisiranno come variabili discrete le informazioni INe dello stato degli attuatori di potenza USC, senza che da essi venga generata la richiesta di lettura tramite lo status register, essendo sincroni con la attuazione dei comandi rivolti al campo.

Tra gli ingressi di segnali di stato, siano essi provenienti dal campo ISC o provenienti dall'operatore ISO, alcuni saranno doppiamente utilizzati: oltre ad essere acquisiti per l'informazione discreta che portano, il loro significato verrà inviato al circuito di blocco, che provvederà ad assumere gli stati conseguenti alla applicazione di quelle variabili. Come abbiamo già osservato, lo stato che si viene a determinare nel circuito di blocco condurrà alla generazione dei vettori UBL e ISBL, quest'ultimo verrà acquisito sempre tramite l'intervento dello status register.

La congruenza tra le informazioni di stato e di comando di blocco potrà essere così utilizzata per uso diagnostico degli stessi circuiti di blocco.

I circuiti di blocco saranno tali da garantire una sicurezza intrinseca del funzionamento, e la loro azione di porre in sicurezza l'impianto sarà comunque assicurata.

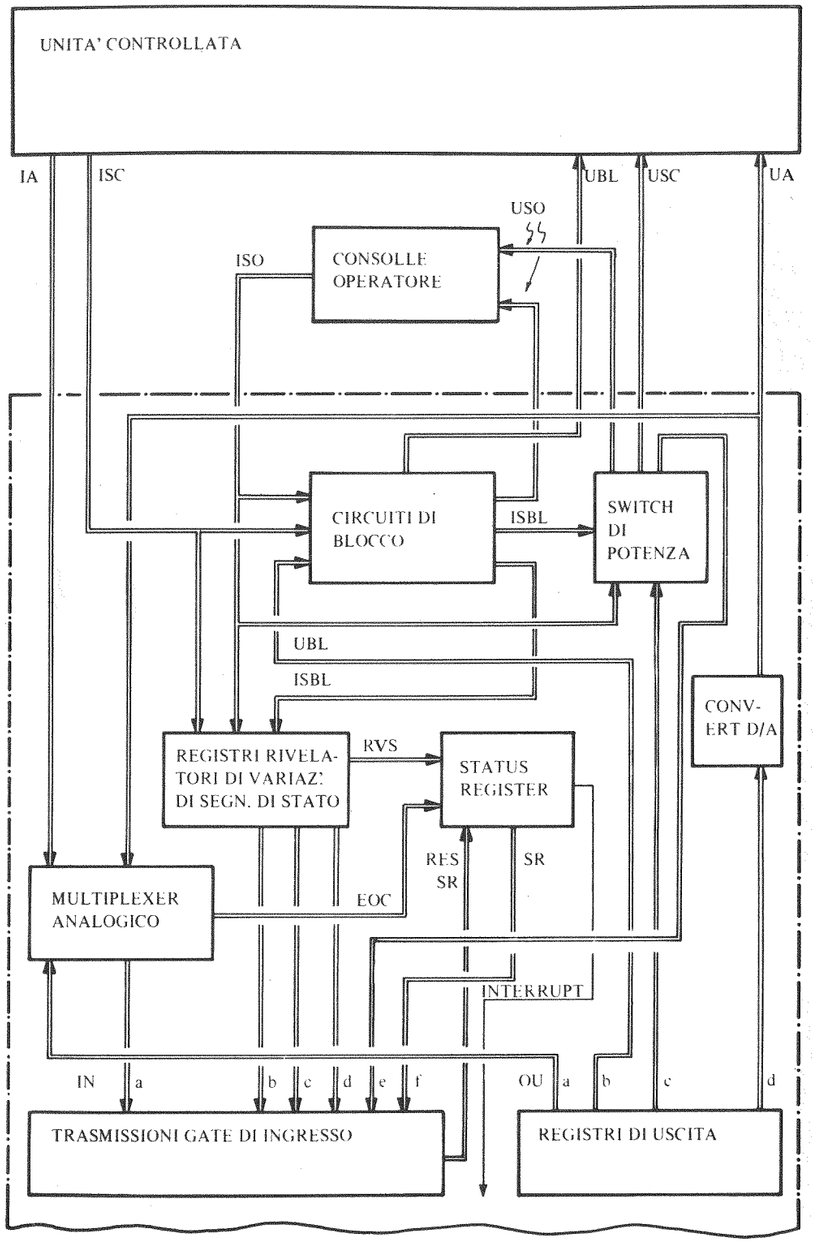


Fig. 2

I circuiti "timer di programma", non evidenziati nella figura 2, costituiranno una parte dei circuiti di blocco unitamente agli eventuali circuiti di restart.

Infine potranno essere inviate al campo le variabili analogiche UA provenienti dal vettore OUd.

In talune applicazioni, macchine a controllo numerico, potrebbero esistere altri vettori di ingresso, ad esempio quelli provenienti da encoder che comunque vengono assimilati agli ingressi di segnali di stato il cui significato potrebbe essere acquisito con tecniche particolari che vanno dal DMA, alla acquisizione continua come feedback del comando impartito ed infine alla acquisizione dei valori assunti in concomitanza ad altri segnali di stato che fungono da enable indicando il momento opportuno per la lettura.

6. ASPETTI FUNZIONALI

Dopo avere esaminato gli aspetti minimi strutturali della unità di controllo, possediamo ora gli indispensabili elementi a cui riferirci per procedere alla descrizione del funzionamento.

In maniera generale, iniziamo la visione delle tre attività fondamentali della unità controllante: l'acquisizione, la elaborazione e la attuazione (Fig. 3a), regolate da un programma (Fig. 3b) al quale viene comunicato man mano lo stato in cui si trova il sistema (Fig. 3c).
A queste si aggiunge l'attività di inizializzazione del nuovo ciclo di regolazione.

Prima di passare a descrizioni più particolareggiate, un gradino più in basso nel livello top-down, è opportuno stabilire che, se il processo fosse interamente sincrono, le "attività" si succederebbero come descritto nella figura 3. Tuttavia, non esistendo processi sincroni, la sincronicità è elemento di semplificazione descrittiva e di approssimazione di comportamento della unità regolante: occorre allora prevedere che, in qualsiasi stato si trovi il sistema, l'unità regolante sia predisposta alla acquisizione di eventi asincroni affinchè essi possano concorrere alla regolazione.

7. ACQUISIZIONI

Possiamo allora affermare che tutte le acquisizioni che il sistema effettuerà saranno gestite in maniera asincrona, siano esse le variabili di stato che hanno sorgenza spontanea (quali le variabili discrete tipo ISC e ISO, che sono da considerarsi tipici eventi asincroni), siano esse le acquisizioni di variabili analogiche, che sono decise sincronamente dal programma mediante l'indirizzamento di un apposito multiplexer, ma rilevate dopo lassi di tempo variabili da caso a caso e da ciclo a ciclo (Fig. 4) (°).

Infatti, nel primo caso, riferito ad un generico evento asincrono l'avvenuta variazione di uno o più bit costituenti ISC o ISO produce una operazione di confronto con il pattern precedentemente memorizzato.
La disuguaglianza attiva un temporizzatore, programmato con un ritardo tale da discriminare eventuali transizioni spurie dalle reali variazioni del pattern.
Al termine del tempo di ritardo, una ulteriore operazione di confronto atta ad accertare la persistenza della disuguaglianza tra il pattern precedentemente memorizzato e quello presente all'ingresso conduce, se la disuguaglianza permane, alla generazione del segnale RVS (registro variazioni di stato). Se invece si verificasse l'uguaglianza, non si avrebbe la generazione del segnale RVS in quanto sarebbe evidente il carattere spurio della avvenuta transizione, pertanto non significativa per il processo (Fig. 5b).
Abbiamo così condotto un filtraggio hardware delle variazioni dei segnali di stato di ingresso.

Nel secondo caso, invece, (Fig. 5a) l'attività rivolta dal programma alla acquisizione delle variabili analogiche, e pertanto sincrona con lo svolgimento dello stesso, conduce alla abilitazione di un meccanismo esterno, "il multiplexer" che nel caso più completo potrà comportarsi nel seguente modo.

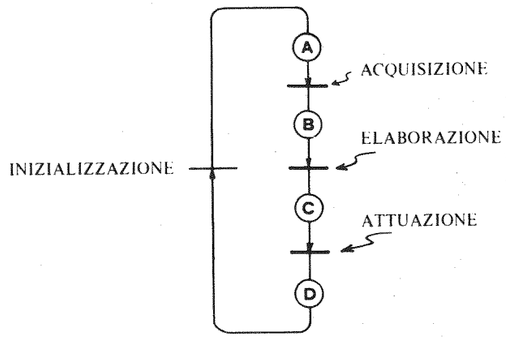
Il vettore di comando della misura, diviso in tre campi, abiliterà l'indirizzo della variabile, la preselezione del guadagno ed infine la eventuale preselezione del tempo di sample.

L'avvenuto indirizzamento del multiplexer decide la partenza di un tempo di ritardo atto a consentire l'esaurimento del transitorio relativo agli switch Hg wetted.

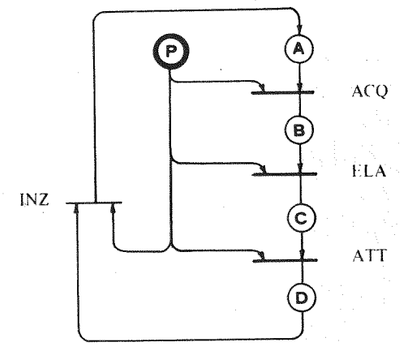
Al termine di questo transitorio, potrà iniziare il tempo di integrazione con il campionamento del

(°) Si noti che, nelle reti, useremo la notazione semplificativa:

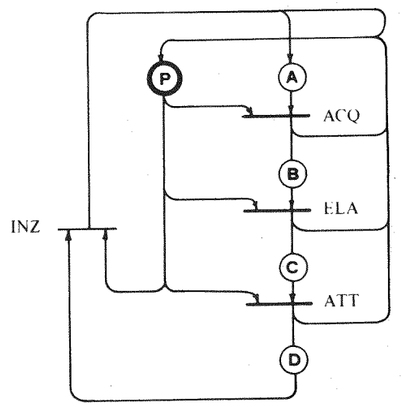
per indicare che la transizione b scatta quando non ci sono marche nel posto a.



3. a



3.b



3.c

Fig. 3

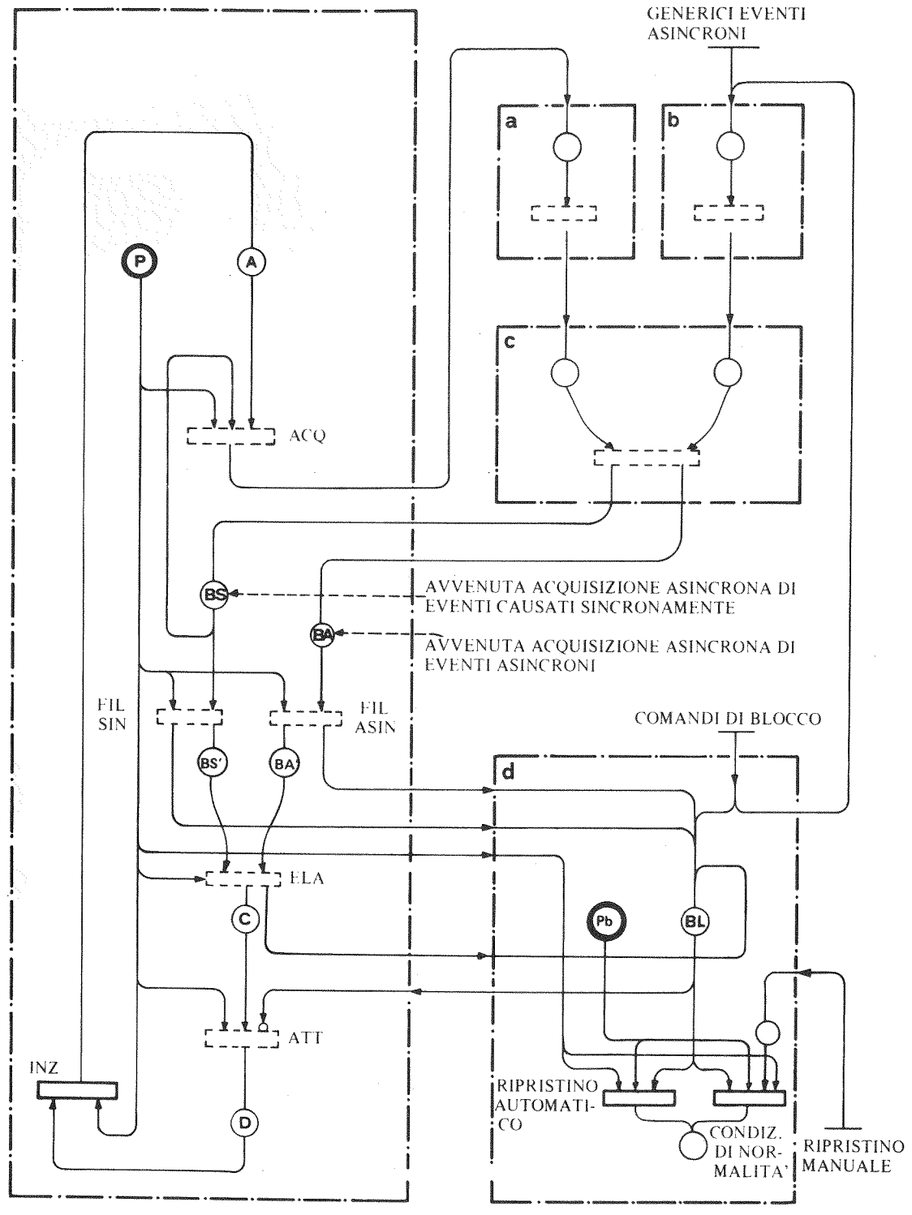

GENERICI EVENTI ASINCRONI
a
b
c
d
P
A
ACQ
BS
BA
AVVENUTA ACQUISIZIONE ASINCRONA DI EVENTI CAUSATI SINCRONAMENTE
AVVENUTA ACQUISIZIONE ASINCRONA DI EVENTI ASINCRONI
FIL SIN
FIL ASIN
BS'
BA'
COMANDI DI BLOCCO
ELA
C
Pb
BL
ATT
INZ
D
RIPRISTINO AUTOMATI-CO
CONDIZ. DI NOR-MALITA'
RIPRISTINO MANUALE

Fig. 4

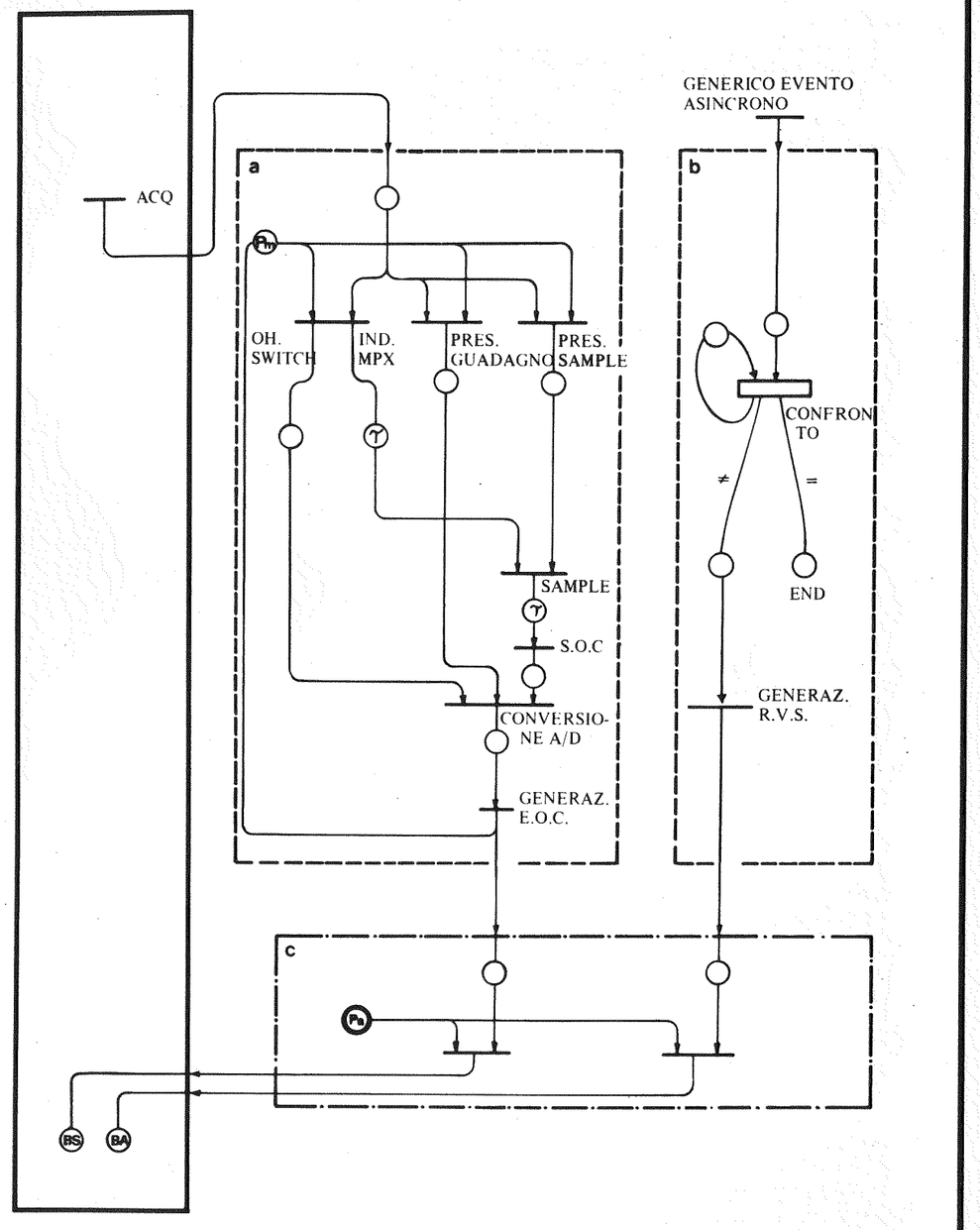

ACQ
a
b
OH. SWITCH
IND. MPX
PRES. GUADAGNO
PRES. SAMPLE
SAMPLE
S.O.C
CONVERSIO-NE A/D
GENERAZ. E.O.C.
GENERICO EVENTO ASINCRONO
CONFRON TO
≠
=
END
GENERAZ. R.V.S.
c
Pa
BS
BA

Fig. 5

segnale, campionamento che nel caso più raffinato potrà essere digitalmente preselezionato.

Con la transizione sample-hold avverrà la generazione del segnale S.O.C. (Start of conversion) a cui seguirà la conversione A/D.

Il segnale E.O.C. (End of conversion) annuncerà la avvenuta conversione e la presenza della nuova variabile da acquisire (Fig. 5a).

Il o i segnali RVS ed il segnale EOC dovranno provocare l'acquisizione asincrona del nuovo vettore, rispettivamente riferito alle variabili di stato o alle variabili analogiche (Fig. 6).

Per fare ciò, si utilizza un registro di stato [SR] in cui ci sarà un Flip - Flop [FF] per ogni parola o byte da acquisire, ovvero per ogni trasmission gate, che singola o multiple permetteranno l'acquisizione delle variabili.

Il set del o dei relativi FF dello SR genera il set del FF di interrupt.
La gestione di quell'interrupt da parte del programma conduce alla apertura della trasmission gate relativa allo status register SR che verrà acquisito ed al contemporaneo reset del FF di interrupt.

Il riconoscimento da parte del programma Pa della variabile da acquisire dovuto alla avvenuta comprensione del significato dello SR provoca l'apertura della relativa trasmission gate con il contemporaneo reset del FF relativo a quella variabile, a cui consegue la lettura della stessa.

Qualora una variabile fosse tale da occupare più trasmission gate, essendo la larghezza di queste strettamente vincolata alla dimensione del bus, il programma Pa le aprirà sequenzialmente azzerando, per ognuna di esse, il corrispondente FF dello SR.

Si verranno così a compilare due distinte tabelle, l'una per le variabili analogiche e l'altra per le variabili di stato (BS e BA).

È opportuno rilevare che il programma principale, una volta decisa l'attività di acquisizione sincrona potrà, dopo avere indirizzato il multiplexer, dedicarsi ad altre operazioni in attesa dell'evento asincrono con cui verrà acquisita la variabile relativa a quella misura.

Diviene allora interessante osservare come si svolge l'attività di acquisizione (Fig. 7).

Il programma abiliterà l'inizio di una delle sequenze di acquisizione delle variabili analogiche, in quanto potrebbero essere lette diverse tabelle in funzione di diverse attività operative o per particolari stati del sistema.

Questa operazione produce, per ogni suo passo, l'indirizzamento della variabile da acquisire.

Il meccanismo che soddisferà l'acquisizione e depositerà la variabile acquisita all'interno della costituenda tabella delle variabili analogiche informerà il meccanismo di indirizzamento di procedere ad una nuova misura, e così di seguito sino all'esaurimento delle stesse.

Solamente la fine di quelle misure previste produrrà l'"END" di quella sequenza, avvisando di ciò il programma principale P.

La transizione che produce l'indirizzamento del multiplexer avviserà, inoltre, dell'inizio del periodo di attesa della acquisizione una parte del programma di supervisione Pg, che potrà disporre dell'annunciato periodo di attesa per espletare altre attività.

8. ELABORAZIONE

L'elaborazione degli acquisiti segnali di stato ha normalmente maggiore priorità di quanto non abbiano le elaborazioni delle variabili analogiche, perché essi possono contenere informazioni di emergenza che devono tradursi in attuazioni immediate, anche se complementari ad attività di messa in sicurezza del sistema che sono ordinariamente condotte da oggetti hardware.

L'elaborazione preliminare di questi segnali di stato viene normalmente effettuata confrontando il valore del o dei vettori interessati con il contenuto di tabelle rigide, contenute in una parte dedicata del programma che chiameremo Ps che, in esito alle avvenute elaborazioni, conterrà se necessario, anche le azioni di regolazione (Fig. 8).

Se azioni attuative immediate non saranno richieste, si avviserà il programma generale delle nuove condizioni affinchè assuma il conseguente comportamento.
Esso sarà, comunque, avvisato anche nel caso di esiti attuativi e tra questi la messa in blocco dell'impianto.
L'elaborazione delle variabili analogiche viene ge-

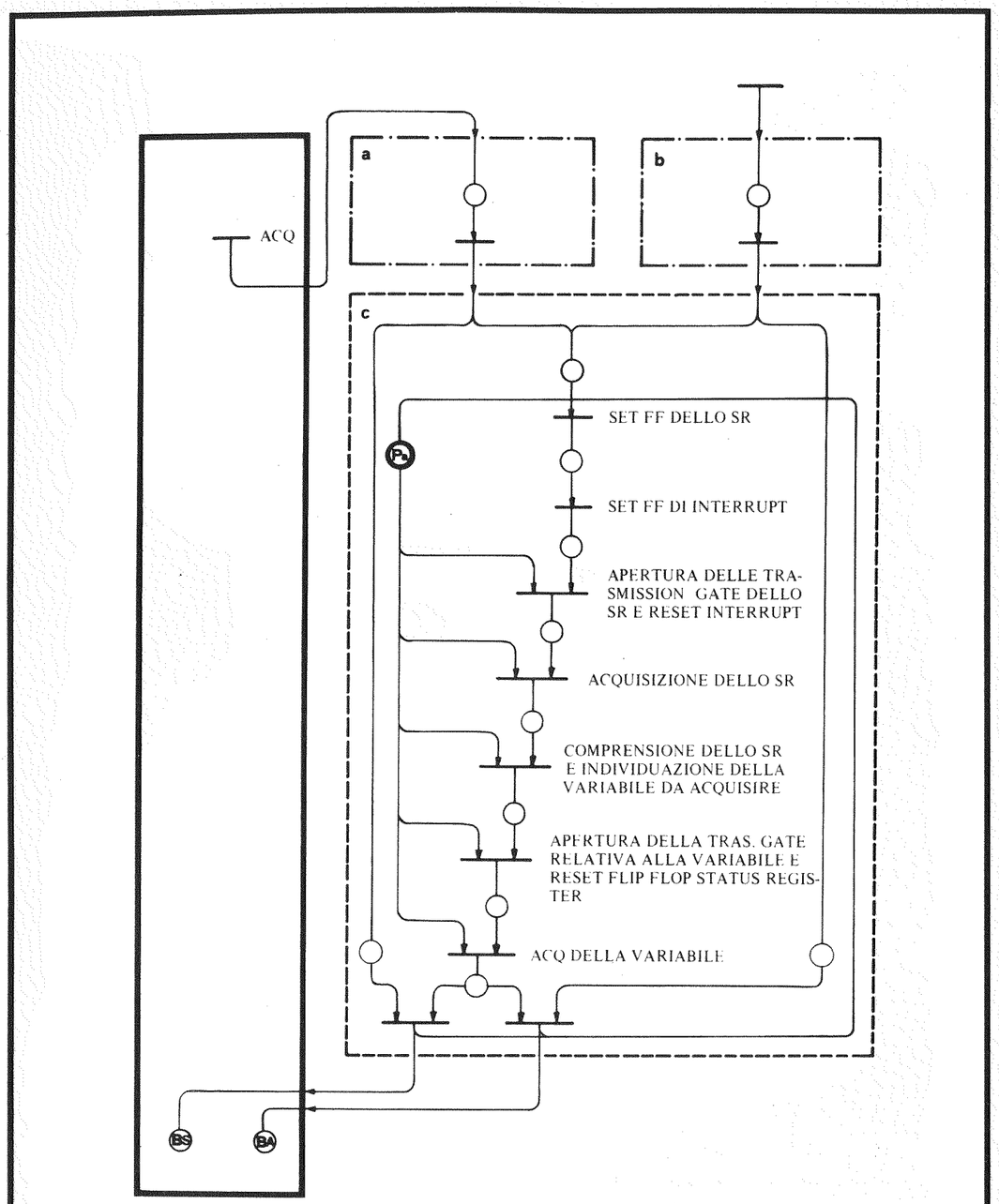


Fig. 6

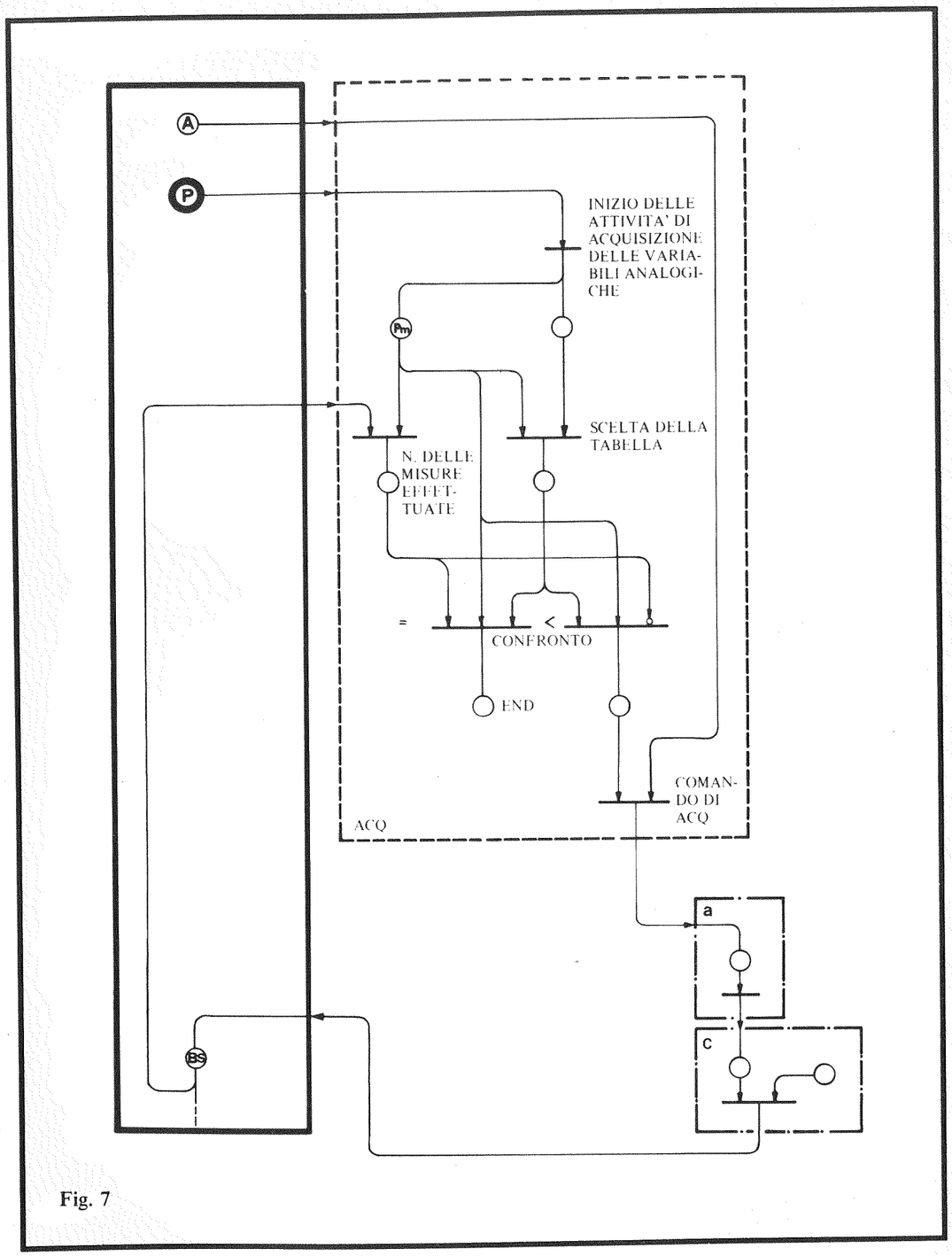

A
P
INIZIO DELLE ATTIVITA' DI ACQUISIZIONE DELLE VARIABILI ANALOGICHE
Pm
SCELTA DELLA TABELLA
N. DELLE MISURE EFFETTUATE
=
<
CONFRONTO
END
COMANDO DI ACQ
ACQ
a
c
BS

Fig. 7

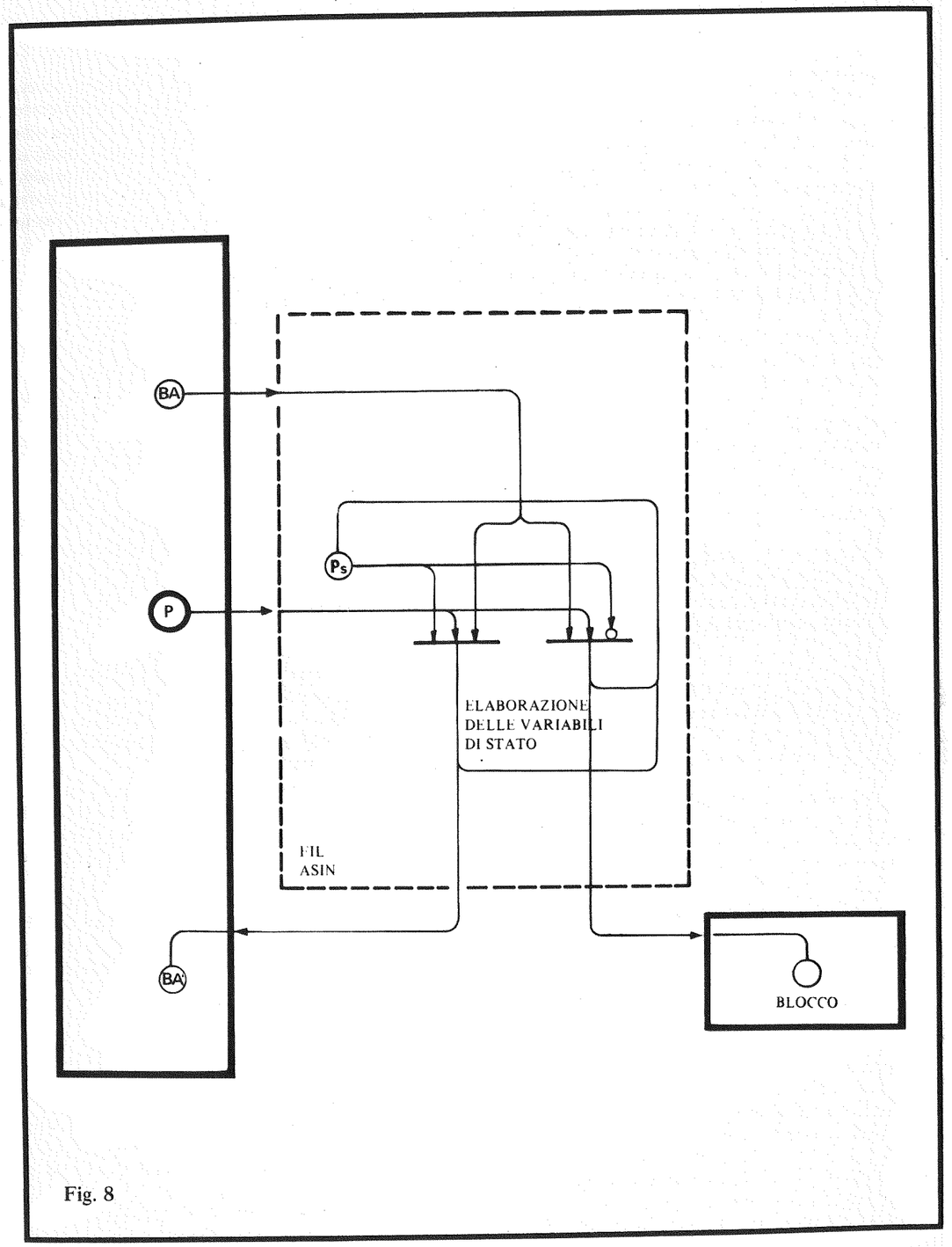

BA
P
Ps
ELABORAZIONE DELLE VARIABILI DI STATO
FIL ASIN
BA
BLOCCO

Fig. 8

stita da appositi packages, che conterranno le informazioni circa gli algoritmi matematici da applicare alle singole variabili o a gruppi delle stesse.
Chiamiamo questi packages Pf e Pe. Essi opereranno non solo sulle variabili costituenti Bs, ma che dovranno utilizzare obbligatoriamente, per la risoluzione delle equazioni di regolazione, i set point o altri parametri che verranno impostati preventivamente dall'operatore.

Precede solitamente l'attività di elaborazione il filtraggio dei dati che, salvo rari casi, è ricavato da una media aritmetica di un certo numero di letture della stessa variabile (Fig. 9).

Esistono, tuttavia, casi di filtraggio più complessi che, data la loro particolarità, non vengono considerati in questa sede.

Proseguendo, possiamo dire che un package Pf provvederà al filtraggio delle variabili e comparerà il valore mediato di quella variabile con una banda di accettabilità prestabilita preventivamente dall'operatore.

Le caratteristiche del filtraggio, depositate rigidamente in PF, provvederanno, tramite il programma principale, a comandare le ripetizioni delle misure, così da poter accumulare i dati necessari alle medie.

Una volta mediati, alcuni dati saranno linearizzati mediante l'uso di tabelle rigide di conversione per i vari tipi di trasduttori o, al limite, mediante un apposito algoritmo di linearizzazione.

Dopo essere stati linearizzati i dati verranno convertiti in unità ingegneristiche, ancora mediante l'impiego di tabelle rigide contenute sempre nel programma Pf.

Interviene, ora, il meccanismo di comparazione con limiti prefissati dall'operatore ed atti a stabilire eventuali condizioni di allarme.
Per fare ciò, oltre al programma Pf, dovranno intervenire tabelle o parametri opportuni atti a questo confronto.
L'esito sfavorevole del confronto condurrà alle condizioni di sicurezza mediante la forzatura di uno stato di blocco del sistema.
Il confronto favorevole consentirà le successive operazioni di elaborazione.

A questo punto, possiamo parlare del vero programma di elaborazione Pe che, utilizzando dati pronti per essere operati, applica gli algoritmi previsti dal programma utilizzando i set point preimpostati dall'operatore (Fig. 10).

I dati così elaborati daranno le indicazioni necessarie alle attuazioni e potranno essere ulteriormente confrontati con i limiti di errore; l'eventuale sfavorevole confronto condurrà ancora alla condizione di blocco, mentre il confronto favorevole produrrà i dati validi per le attuazioni. Di tutte le attività verrà comunque informato il programma principale.

9. ATTUAZIONI

I dati pronti per l'attuazione, espressi secondo le utilità del programma, dovranno essere operati con costanti al fine di essere tradotti in grandezze idonee ad essere inviate agli attuatori: tempi di attuazione, tensioni, correnti per i comandi analogici, e comandi ON OFF per quelli di stato (Fig. 11).

Opererà a questo fine un programma Pt unitamente ad una ulteriore tabella, o preimpostata dall'operatore o in altri casi inclusa nel sopracitato programma.

10. BLOCCO

Lo stato di blocco, come si è già accennato, è relativo alla messa in sicurezza dell'impianto. Ciò avviene mediante una rete sequenziale, costruita con tecniche che gli conferiscono la sicurezza intrinseca di funzionamento.
L'azione di blocco è normalmente fatta da questa rete sia su segnalazioni di stato provenienti dal campo o dall'operatore, come già osservato, sia in esito ad alcune elaborazioni condotte dal programma (Fig. 12).
Tale azione si attua togliendo l'alimentazione alle valvole rapide ON OFF di alimentazione degli organi da arrestare immediatamente, e con una serie di azionamenti concorrenti alla sicurezza dell'impianto.

L'elaborazione dei dati acquisiti conduce ad una serie di operazioni che preliminarmente stabiliscono l'eventualità che si debba porre il sistema in stato di blocco.
Tra le variabili discrete asincrone, alcune potranno porre direttamente il sistema in blocco, agendo sui circuiti hardware dedicati a questa operazione.

La instaurata condizione di blocco impedisce le regolazioni che fossero in corso e invia i comandi di

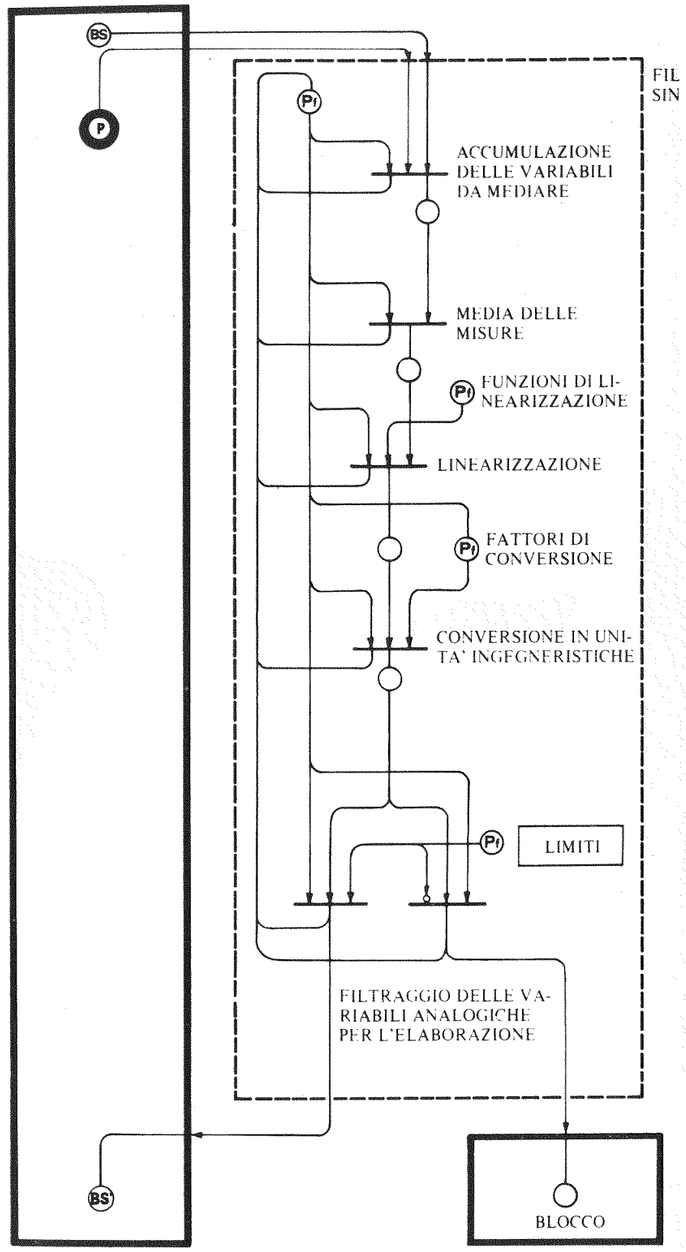


Fig. 9

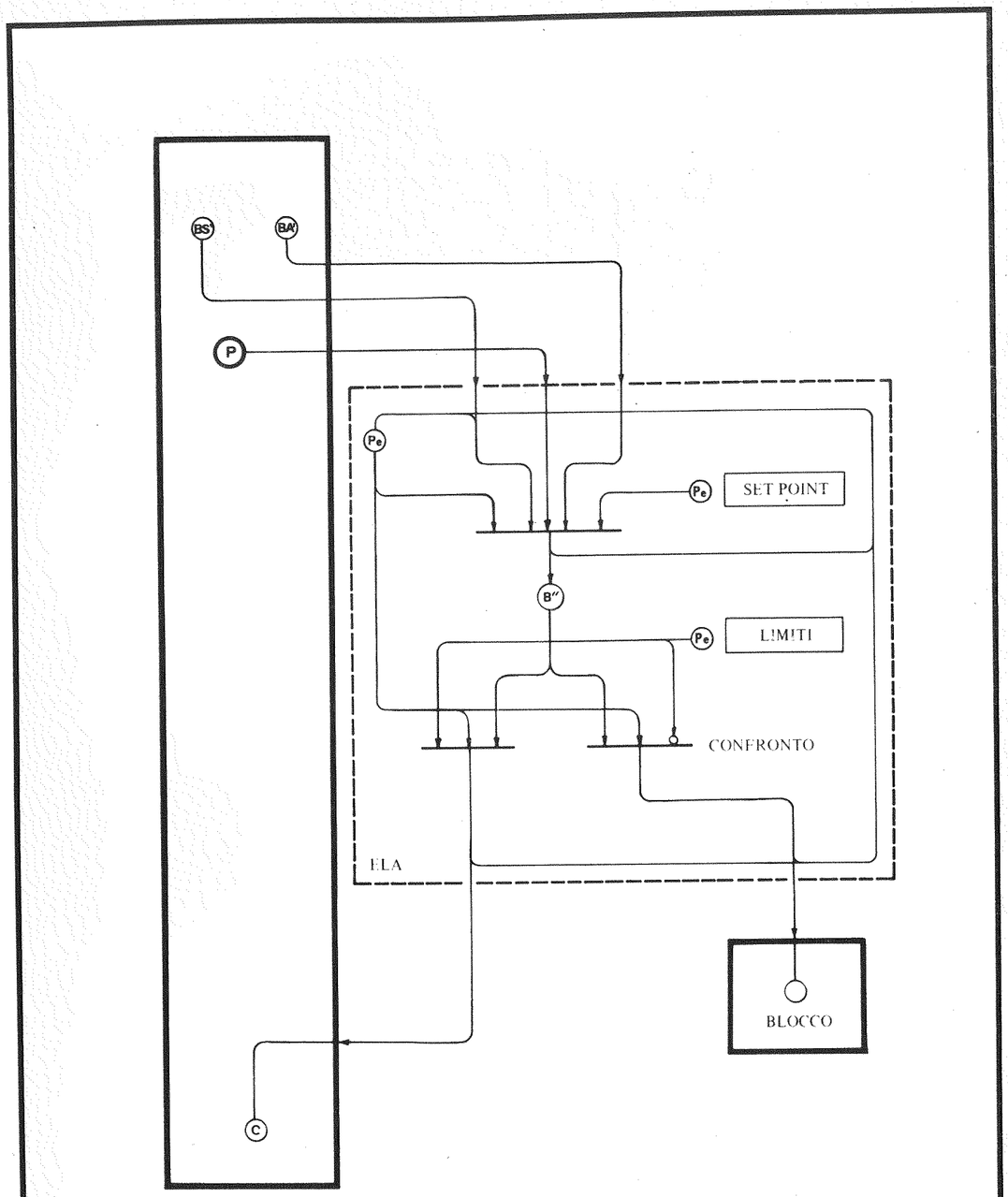

BS'
BA'
P
Pe
Pe
SET POINT
B''
Pe
LIMITI
CONFRONTO
ELA
BLOCCO
C

Fig. 10

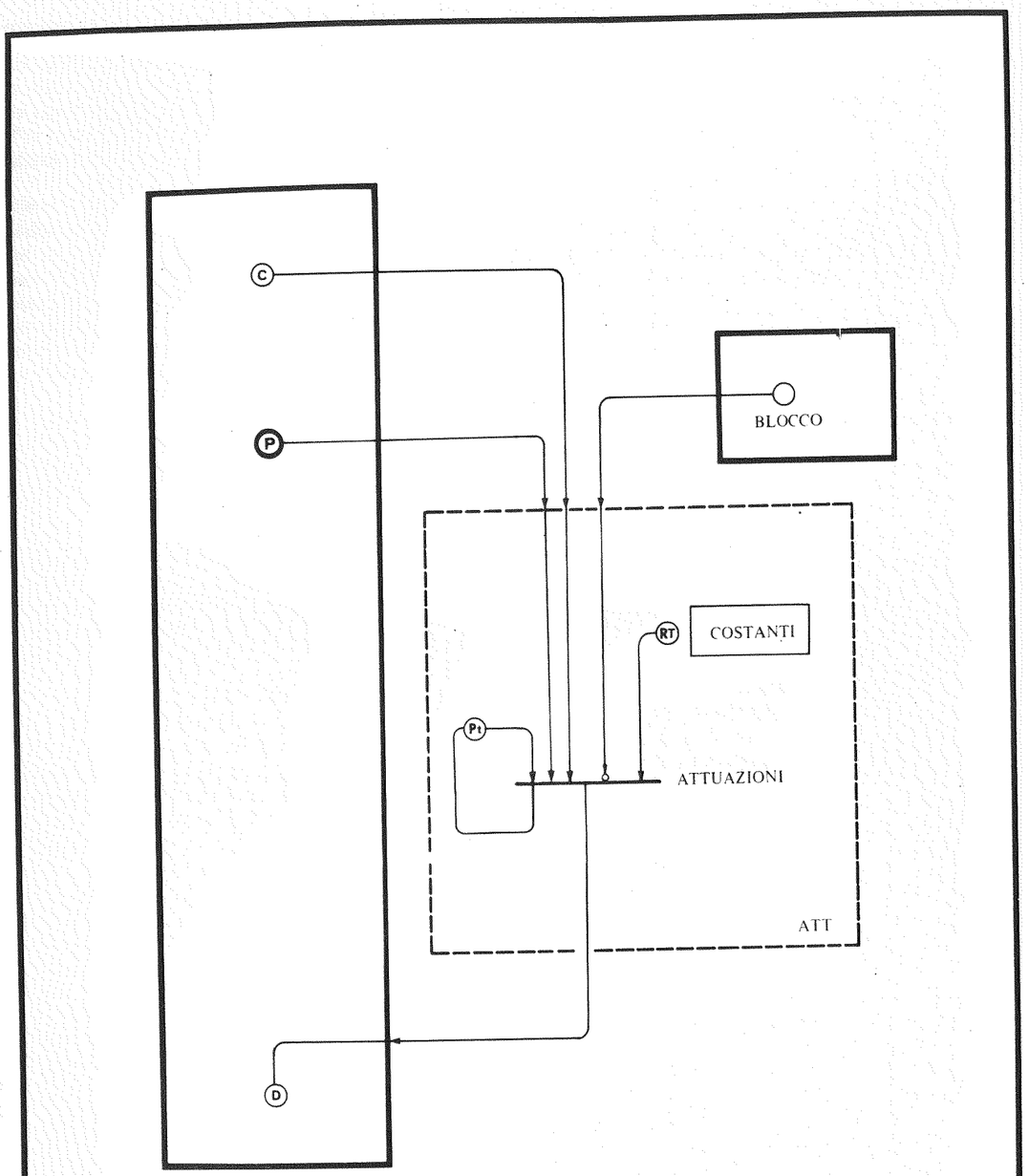

C
P
BLOCCO
RT
COSTANTI
Pt
ATTUAZIONI
ATT
D

Fig. 11

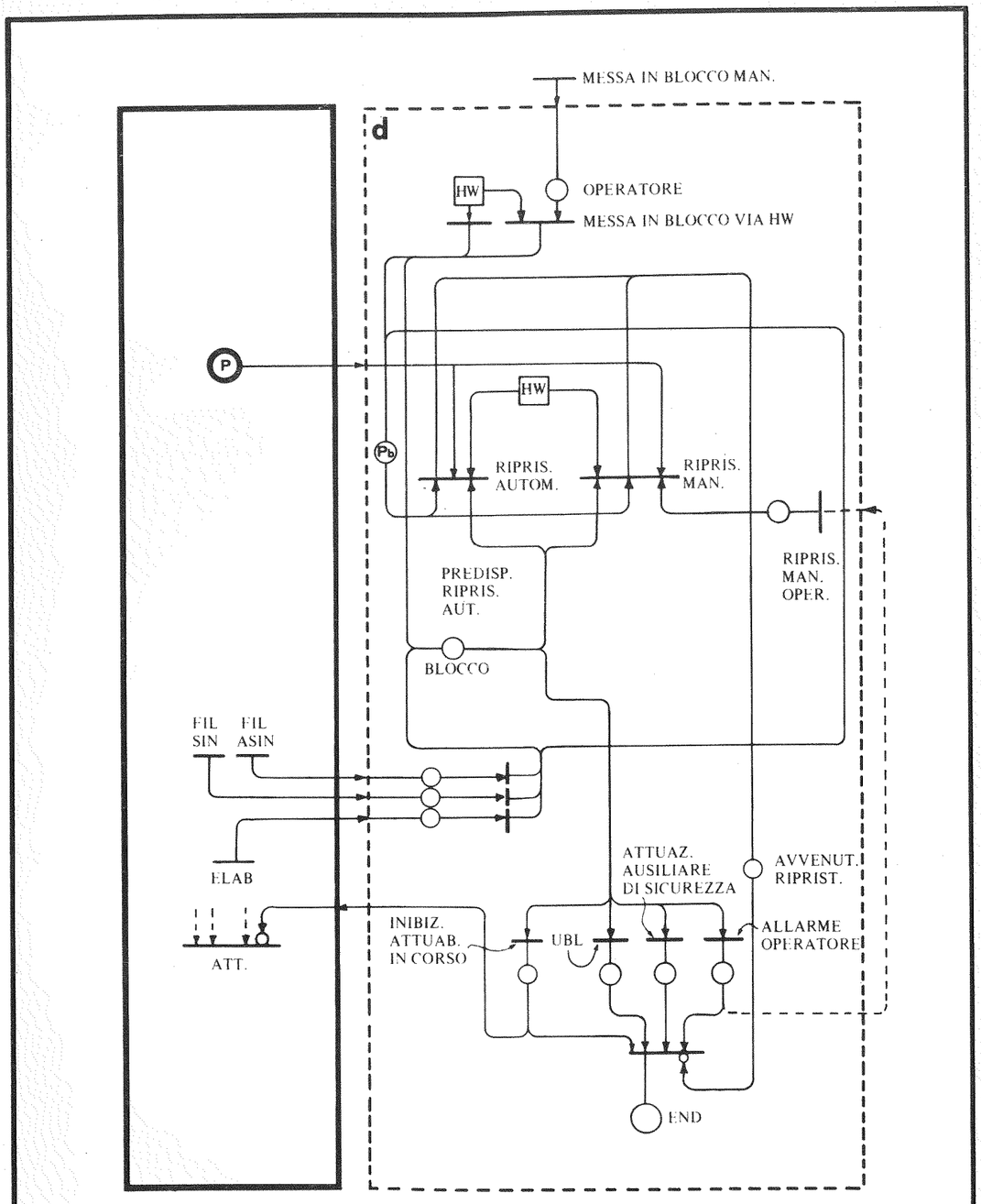


Fig. 12

messa in sicurezza del sistema UBL; agirà, inoltre, su quelle regolazioni con USBL che concorreranno a realizzare le sicurezze minori spesso utili durante la ripartenza del sistema (USC, UA).
Provvederà, infine, ad avvisare l'operatore mediante appositi allarmi.

Oltre a queste attività, l'avvenuta instaturazione dello stato di blocco farà in modo che le acquisizioni continuino ad avvenire, siano esse sincrone e non e siano magari maggiormente protese alle variazioni più significatve per quel tipo di blocco. Per arrivare a ciò, concorreranno anche aree di programma normalmente non utilizzate, che a volte condurranno alla stampa delle variabili interessate.

Continueranno non solo le attività di acquisizione, ma anche quelle di elaborazione che, qualora fosse previsto sia dal programma sia dai circuiti di blocco, condurrebbero alla ripartenza automatica.

Qualora, invece, la ripartenza fosse manuale, si avviserà ulteriormente l'operatore della avvenuta cessazione della causa di blocco, invitandolo ad effettuare il ripristino del funzionamento normale. Il ritorno alla normalità consentirà la rinizializzazione del ciclo ordinario.

L'argomento "Circuiti e stato di blocco" si presta ad una moltitudine di altre considerazioni particolari, che non vengono analizzate dettagliatamente in questa sede, ma di cui si vuole dare solamente una generale informazione. Da tenere presente, oltre alla grande interazione tra HW e SW, sono le possibilità di comando del blocco, che possono essere: da operatore, da elaborazione (sia essa applicata alle variabili di regolazione sia alle variabili di tipo diagnostico) o da supervisori remoti. Infine il comando del blocco verrà imposto da dispositivi sensori posti in campo. Per tutti questi tipi di blocco, il ripristino potrà essere manuale od automatico, oltre a ciò si dovranno considerare le possibilità di bypassare alcune o tutte le cause di blocco per consentire una gestione manuale delle emergenze.

Per concludere sull'argomento, è infine opportuno ricordare che potrà esistere una gerarchia di blocchi e di inibizioni, specialmente in impianti di notevoli dimensioni anche se realizzati con tecniche di automatica distribuita.

11. STRUTTURA DEL SOFTWARE

Nella descrizione condotta, abbiamo esaminato una serie di attività svolte dalla unità di controllo e regolate da appositi programmi.

I programmi che sono stati menzionati sono:

— Pg Sistema operativo e programmi di supervisione;

— Pa Programma di gestione asincrona delle acquisizioni;

— Pm Programma relativo alla acquisizione delle variabili analogiche;

— Ps Programma di gestione delle variabili di stato;

— Pf Programma di filtraggio delle variabili analogiche, che opererà in concomitanza ad una tabella rigida per la conversione delle variabili in unità ingegneristiche, e di una seconda tabella contenenti i limiti di accettabilità della variabili.
Tabella, questa, scrivibile e modificabile tramite apposite procedure previste in Pf;

— Pe Programma di elaborazione che opera, come osservato in precedenza, con tabelle riportanti sia i set point che i limiti delle grandezze elaborate, scrivibili e modificabili tramite apposite procedure;

— Pb Programma per le elaborazioni e le gestioni dello stato di blocco;

— Pt Programma di attuazione dei comandi, che opera assieme a costanti preimpostabili da apposite procedure.

Il funzionamento concorrente dei programmi sopraelencati può essere descritto nel seguente modo.

All'accensione (Fig. 13), il sistema si troverà nello stato di blocco; questa condizione verrà comunicata al programma principale Pg e si inizierà, così, una serie di attività che condurranno, nel caso favorevole, alla predisposizione della inizializzazione del ciclo ordinario di regolazione.
Il funzionamento normale verrà iniziato solamente se la condizione di blocco iniziale sarà manualmente rimossa mediante l'apposito comando di avviamento-ripristino, comando che potrà essere attuato qualora non esistano altre cause che riconfermino lo stato di blocco.

L'avvenuta attività di inizializzazione, comunicata a Pg, consente l'abilitazione del programma di ac-

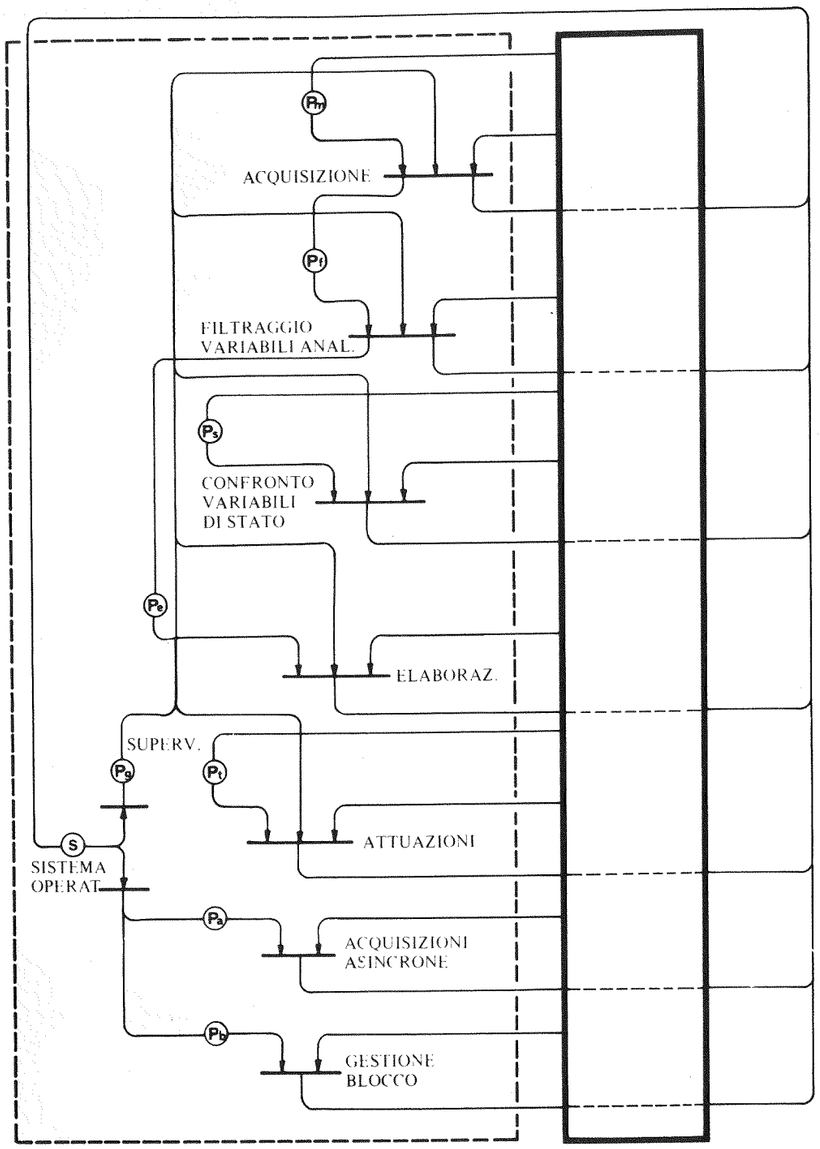


Fig. 13

quisizione delle misure delle variabili analogiche Pm, a cui seguiranno i comandi al multiplexer.

L'avvenuto comando al multiplexer produrrà una opportuna segnalazione al programma di supervisione Pg, che gestirà l'attesa della acquisizione asincrona, consentendo lo svolgimento di altre operazioni che si rendessero necessarie.

Se non accadessero mai eventi asincroni, in una simile macchina non si avrebbero mai code, ma essendo questi eventi tipici di ogni processo, si verranno a generare certamente delle acquisizioni che si sovrapporranno ad altre attività del sistema. Da ciò deriveranno, in funzione delle priorità accordate da Pg, sospensioni di alcune attività che verranno poi riprese nei momenti di attesa, che nel sistema in esame saranno decisi dal comportamento del multiplexer.

Qualsiasi evento asincrono verrà gestito, per l'acquisizione della variabile associata, dal programma Pa.

Di questi eventi, quelli relativi alle variabili analogiche andranno a costituire una tabella elaborata dal programma di filtraggio Pf.
In esito a questa attività, potremo avere l'instaurazione del blocco, oppure una ulteriore tabella atta alla elaborazione.

Il programma Pe effettuerà l'elaborazione necessaria alla regolazione con il concorso delle variabili asincrone acquisite e preelaborate da Ps, al fine di stabilire l'eventuale impostazione del blocco.

I dati così elaborati potranno essere trasformati in variabili di comando dal programma Pt, nulla ostando da parte dei circuiti di blocco.

L'avvenuta attività di attuazione produrrà la riinizializzazione del ciclo ordinario di regolazione.

Tutte queste attività verranno comunicate al programma di supervisione Pg.

Infine, l'avvenuta instaurazione dello stato di blocco produrrà una serie di azioni, tra cui l'inibizione delle attività di regolazione e la predisposizione alla ripartenza del ciclo ordinario, qualora le condizioni che imponevano il blocco siano cessate e dopo che si sia proceduto all'eventuale ripartenza manuale.
Le attività del blocco, oltre che dalla struttura HW, verranno gestite dall'apposito programma Pb.

Come si osserva, tutte le operazioni avvertono il programma Pg al momento della loro avvenuta attuazione, in modo che esso possa procedere alle altre operazioni.
Questa comunicazione avviene tramite il sistema operativo S che potrà così non solo abilitare le attività nella sequenza vista, ma anche inibirle in modo da concedere attuazione alle attività prioritarie, quali quelle gestite da Pa e Pb, che saranno sempre abilitate per la specifica caratteristica della loro attività (Fig. 14).

12. CONCLUSIONI

Nella presente trattazione si è voluto suggerire una metodologia ordinata di progetto di un sistema di automazione di processo, privilegiando gli aspetti del comportamento del sistema per poi fornire gli elementi di riferimento sulla struttura dei programmi.

Una siffatta struttura dei programmi dà luogo ad una architettura capace di esprimere una notevole modularità, in quanto molte sono le funzioni svolte dai vari packages che possono venire accomunate a più processi.

In particolare, se togliamo la parte dei programmi relativi alle tabelle, siano esse rigidamente connesse al programma siano esse da compilarsi da parte dell'operatore, osserviamo che solo i programmi Pe e Pg differiscono sostanzialmente da caso a caso.

Ricordiamo, tra le tabelle rigide, quelle di indirizzamento del multiplexer, le tabelle di confronto dei segnali di stato, quelle della conversione dei dati numerici in dati ingeneristici, etc. e, tra quelle da impostarsi, le costanti di regolazione, i set point etc.

Con ulteriore indagine, si può osservare come molti algoritmi di calcolo inclusi nel programma di elaborazione Pe sono comuni ad una infinità di processi, quali le elaborazioni proporzionali, derivative ed integrative che ancora costituiranno moduli elementari.

Consideriamo costante il sistema operativo per tutta la classe di processi interessanti a questo genere di regolazione, osserviamo che solo il programma Pg sarà specifico a quel processo.

Il programma di supervisione Pg sarà, tuttavia, un oggetto di semplice realizzazione, che consentirà lo

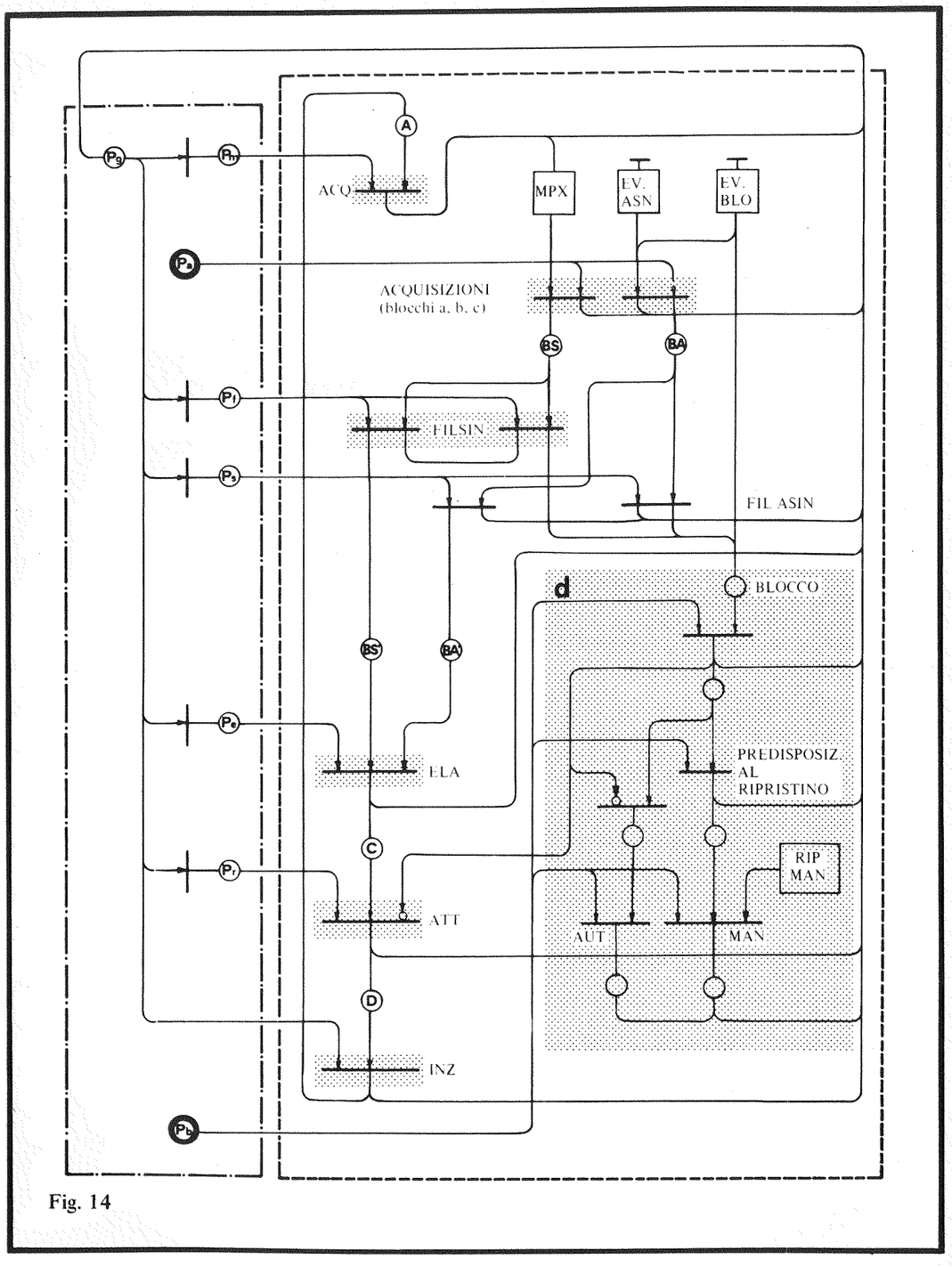


Fig. 14

svolgimento delle attività previste, con la sequenza da esso stesso impostata, da parte dei moduli di programma che collegherà al sistema operativo.

Abbiamo volutamente trascurato, in questo lavoro, alcune delle caratteristiche peculiari di un controllo digitale diretto, quali l'autodiagnostica, ritenendo che simile argomento meriti una trattazione distinta dalla presente, in quanto coinvolge contemporaneamente aspetti hardware e software, e pertanto non distinguibile da una descrizione della architettura dell'insieme di interfacce.

AVVENIMENTI

ADVANCED COURSE ON GENERAL NET THEORY OF PROCESSES AND SYSTEMS, AMBURGO 8-19 OTTOBRE 1979

Si è svolto ad Amburgo, dall'8 al 19 ottobre, un corso interamente dedicato alla "General Net Theory", organizzato dall'Università di Amburgo e dal GMD (Gesellshaft für Mathematik und Datenverarbeitung) di Bonn, cioè il gruppo diretto da C.A. Petri, l'iniziatore della Teoria (1962). Le reti di cui si è parlato nel Corso, infatti, sono note come Reti di Petri, anche se contributi a questa teoria, riconosciuta come "open-ended", provengono ormai un po' da tutto il mondo, secondo linee di ricerca e applicazioni anche molto diverse fra loro.

Alla non definitività dei risultati ed alla articolazione degli interessi e relative applicazioni lo stesso Petri ha più volte richiamato l'attenzione dei partecipanti (un centinaio), con una infaticabile attività di relazioni, seminari ed interventi, dedicati soprattutto agli aspetti concettuali e interdisciplinari della teoria generale.
Petri contrappone questa ad una teoria speciale, in quanto la seconda ha a che fare con il flusso di risorse attraverso strutture del tipo grafi, con sincronizzazioni, branching e merging, ma sempre all'interno di singole reti, mentre la teoria generale tratta di relazioni fra reti, trasformazioni di reti in altre reti, in una parola di morfismi, con conservazione, nel passaggio, di certe proprietà. L'interesse dei morfismi appare immediatamente quando si pensi alla difficoltà di esplorare il comportamento di sistemi descritti in migliaia di elementi (già una dozzina possono rendere ardua la comprensione attraverso la simulazione manuale). La possibilità di mappare le reti corrispondenti a tali sistemi in reti molto più piccole è destinata a semplificare il compito. Oltre ad occuparsi delle interconnessioni fra differenti livelli di interpretazione e rappresentazione, la teoria generale contribuisce alla soluzione di alcuni noti problemi quali la concorrenza (definita come indipendenza causale fra eventi), la limitazione, non necessariamente negativa, delle risorse (comprendendo fra queste l'informazione, l'assenza di certe entità o fenomeni, la computabilità, etc.).

L'individuazione di differenti livelli di descrizione dei processi o dei sistemi comporta, fra l'altro, la definizione dei concetti relativi a ciascun livello, dei differenti gradi di esplicitezza e di precisione, nonchè di formalizzazione. Alla base della teoria generale sono la nozione stessa di concorrenza e l'interpretazione delle reti in termini di condizioni (elementi di stato) ed eventi (elementi di transizione e regole associate). Alle Reti C-E sono riconducibili tutti i livelli superiori, per i quali ricorda Petri sono stati mutuati numerosi concetti da varie discipline oltre che dalla computer science. La teoria generale, d'altra parte, restituisce a queste discipline, come feed-back, i propri risultati, con applicazioni che vanno dalla chimica alla giurisprudenza ed alle scienze sociali. Soprattutto a questi livelli si rende necessario, per una buona tecnica del modellare, il rispetto dell'imprecisione delle misure, quando si tratta di una caratteristica inerente al processo di misura stesso od alle relazioni fra il misurante e il misurato. E da questo punto di vista i rapporti fra teoria generale delle reti e meccanica quantica, oltre che teoria della relatività, cominciano a spuntare, anche se molto ancora rimane da dire.
Inoltre, la teoria generale si propone quale strumento per colmare la frattura fra modelli discreti e modelli continui: alla base, ancora una volta, la nozione di concorrenza. È a partire dalla concorrenza, infatti, che vengono costruite le nozioni di ordine, "caso", cambiamento, processo, sistema, e simili.

Sono stati H.J. Genrich, K. Lautenbach, P.S. Thiagarajar (del gruppo di Petri che ha sede a St. Augustin) a sviluppare i temi avanzati da Petri. Essi, che hanno costituito la parte preponderante del Corso, hanno presentato la teoria generale quale strumento per esplorare i concetti e le tecniche necessari a spiegare e ad agevolare l'uso dei calcolatori, non tanto come isolati strumenti di calcolo quanto come parti integranti di sistemi organizzativi molto più vasti. In altri termini, l'interesse della teoria è rivolto innanzitutto ad analizzare e modellare sistemi di informazioni il cui compito precipuo sia stabilire uno schema di flusso dell'informazione fra una collezione di agenti umani ed artificiali. Da questo deriva la terminologia usata: nozioni come coordinamento, sincronizzazione, conflitti, concorrenza, sicurezza, etc., saranno pertanto più frequenti, nella trattazione, di quelli propri della teoria della computazione. I temi sviluppati dai tre relatori sono stati i seguenti.

- Teoria dei sistemi basata sul modello di reti C-E, con definizione formale dei sistemi C-E ed esposizione dei loro requisiti, fra cui una base formale al concetto di informazione.

- Dopo aver definito "informazione" come ciò che è necessario per prendere una decisione e ciò che consegue alla decisione presa, è stato presentato un modello formale (Information Flow Graphs) per studiarne le proprietà, basato sulle reti C-E e sulle nozioni di flusso (degli effetti delle decisioni) e di influenza (fra decisioni). Una conclusione sorprendente: la direzione delle influenze causate da un flusso è indipendente dalla direzione del flusso stesso.

- È stato presentato il modello dei grafi a "sincronizzazione bipolare" per l'identificazione di una sottoclasse di sistemi C-E, il cui comportamento è sottoposto a restrizioni, soprattutto sui tipi di conflitto e sui modi strutturati in cui conflitti e concorrenze sono connessi. La finalità è lo sviluppo di tecniche per la costruzione di sistemi che esibiscano comportamenti "buoni".

- Partendo sempre dall'interpretazione di base (reti C-E) viene sviluppata una interpretazione di livello superiore con l'aggiunta di una nuova nozione: individui e cambiamenti di loro proprietà e relazioni. Nascono le Predicate-Transition Nets, il cui uso viene illustrato in un esempio di data base distribuita.

- Una relazione fra logica e teoria delle reti è stabilita attraverso la cosiddetta "struttura enlogica", risultato del completamento di una rete nel senso delle transizioni strutturalmente costruibili. Nascono da qui i "fatti", cioè le invarianti del sistema o eventi che *non possono* accadere, e le "violazioni", cioè gli eventi che *non devono* accadere. Fatti e violazioni hanno valore sia descrittivo che normativo.

- Le relazioni generali fra logica e reti sono esplorate per quanto riguarda: la logica dei sistemi a posti e transizioni (sistemi PT); reti e predicati del primo ordine; reti e logica modale; reti e logica temporale.

- Un altro completamento delle reti C-E, questa volta nel senso delle condizioni strutturalmente costruibili, viene effettuato per analizzare e specificare sistemi attraverso misure di dipendenze reciproche fra eventi. È stato introdotto il concetto di "distanza sincronica" fra eventi per le dipendenze causate da eventi sincronizzati o per stabilire il loro grado di libertà.

- Morfismi e loro restrizioni consentono di svolgere operazioni come raffinamento, contrazione, espansione, completamento, divisione, restrizione, etc., con la conservazione di certe proprietà. Reti e morfismi di reti costituiscono ciò che in matematica viene chiamato "categoria".

- Partendo dalla nozione di evento come cambiamento elementare indivisibile e dalla composizione di eventi in configurazioni di concorrenza o di sequenza, è stata stabilita una corrispondenza diretta fra logica statica e regole di cambiamento, per giungere alla più generale nozione di "sistemi di sostituzione", da usare come riferimento nella descrizione e confronto di diversi modelli di sistemi.

Se il gruppo di Petri si è quasi interamente dedicato alla presentazione delle reti C-E, gli altri relatori hanno posto l'accento sulle Place-Transition Nets, cioè sulle reti in cui gli elementi di stato sono interpretati come posti che possono contenere un numero variabile di marche, e agli archi che congiungono posti e transizioni, e viceversa, viene assegnato un "peso". Data una marcatura iniziale e regole di transizione, il "token game" può cominciare.

Hanno presentato definizioni e teoremi per le proprietà formali delle PT Nets, M. Jantzen e R. Valk (Università di Amburgo), mentre Valk e G. Roucairol (Università di Parigi) hanno introdotto nei sistemi concorrenti, con particolare riferimento ai modelli PT, sequenze indivisibili di operazioni al fine di ridurre il grado di parallelismo.

Roucairol e G. Memmi (ECA, Parigi) studiano i principi di conservazione e stabilità come asserzioni invarianti sul comportamento di sistemi con l'uso dell'algebra lineare, che permette di: (a) trovare le proprietà delle reti che dipendono solo dalla loro struttura e sono valutabili per tutti gli stati iniziali; (b) provare la correttezza di sistemi concorrenti; e (c) valutare le loro "performances".

A problemi di "performance evaluation" sono rivolte anche le Timed Petri Nets di J. Sifakis (Lab. IMAG, Grenoble) per lo studio di proprietà di sistemi dipendenti dal tempo e in particolare per determinare i limiti di uso di un sistema in funzione delle caratteristiche dei suoi componenti (durate e probabilità di esecuzione delle azioni).

E. Best (Newcastle) introduce una notazione, dovuta a P. Lauer, le "Path expressions" che permettono al programmatore di esprimere una grande varietà di vincoli di sincronizzazione. I programmi scritti con questa notazione, o "path programs", mostrano fra le loro proprietà quella della adeguatezza, definita come la proprietà per cui ogni operazione del programma deve sempre poter occorrere, ciò che corrisponde alla proprietà di "vivezza" delle reti. In un'altra relazione, Best presenta una caratterizzazione operativa della atomicità delle operazioni (ininterrompibilità delle azioni) nelle cosiddette reti di occorrenze, considerate come modelli discreti appropriati per la descrizione dei processi non sequenziali. Vengono toccati aspetti di implementazione, di linguaggio, di "error recovery" in sistemi decentralizzati, nonchè problemi di finitezza e discretezza.

Nei modi di caratterizzare i grandi sistemi di calcolo, osserva J.D. Noe (USA), si individuano due estremi: a) i diagrammi a blocchi che rappresentano gli elementi principali e le loro interconnessioni ma non danno informazioni nè sul "dove" nè sul "quando" farne uso; (b) i diagrammi logici connessi all'hardware e i listaggi di codici e microcodici. Le reti potrebbero essere lo strumento adatto a colmare il divario, per la flessibilità ad essere adattate ai problemi e per la capacità a rappresentare insieme, da una parte, i livelli di dettaglio e dall'altra, la connessione fra le parti trattate e il resto del sistema. Per modellare processori e processi Noe suggerisce le Pro-Nets, un perfezionamento delle Evaluation Nets.

Il compito che si assume K. Zuse (Hunfeld) è invece di sanare la lacuna lasciata dalle trattazioni teoretiche, cioè il punto di vista dell'ingegnere progettista. Le sue reti sono Components-Nets, una forma speciale di reti da far corrispondere alla macchina a stati ed alla Teoria degli automi. Aggiunge alcuni simboli per le condizioni ed effetti laterali e per varie forme di alternativa, adatti per problemi di switching algebra. Fra gli esempi, in complessità crescente: la simulazione di un sistema in cui un conducente può scegliere fra due posti di parcheggio liberi; un altro sistema in cui i conducenti sono due e il posto uno; un carrello mosso da un motore; il controllo di un montacarichi.

A Zuse, noto come l'inventore del moderno elaboratore, il corso ha dedicato un Simposio di un pomeriggio, in cui l'intera sua opera è stata esposta e commentata (Zuse Colloquium).

A problemi di disegno di hardware e software R.M. Shapiro ha applicato tecniche descrittive basate sulle Reti di Petri. Si tratta di esperienze americane parallele a quelle di Petri, che si concludono con la proposta di un ipotetico Automated Design Tool che, in interazione con il progettista, svolge compiti guidati di: simulazione, verifica, "recovery", stime temporali.

S.S. Patil (MIT, Boston) ha mostrato come le reti C-E possano venire impiegate per la specificazione funzionale di circuiti integrati su larga scala (LSI), costituendo uno strumento che permette di ridurre la occupazione su silicio dei circuiti, progettati direttamente a partire dalle specifiche funzionali. In questo modo, il progetto viene semplificato e viene di molto ridotta l'importanza del progetto elettrico.

Non sono mancati gli spunti didattici. H. Oberquelle (Università di Amburgo) considera le reti un utile strumento sia nell'insegnamento che nello studio terminologico. Si riferisce, in particolar modo, a tre interpretazioni di alto livello: le reti di canali ed agenzie, le reti di mezzi ed attività, e le reti di relazioni e funzioni. L'utilità per lo studio terminologico è esemplificata con i sistemi di dialogo, ove la teoria generale delle reti si mostra strumento insostituibile nella trattazione di fenomeni di livelli concettuali molto differenti.
Janzen ha rappresentato parti di conoscenza matematica per mezzo di reti di asserzioni e dipendenze. Come esempi di applicazione, ha proposto la ristrutturazione di indici di volumi di matematica e di curricula di studio. L'uso delle reti permette, fra l'altro, la scelta di strategie personali di apprendimento.

Il materiale del Corso è stato distribuito ai partecipanti in due volumi ed una bibliografia sulle Reti di Petri aggiornata all'agosto 1979. La versione finale apparirà nella serie "Lectures Notes in Computer Science" della Springer-Verlag, nel 1980.

B. Zonta

IL SEGNALIBRO

In questa rubrica vengono segnalati libri, articoli o studi di recente pubblicazione che, a giudizio della redazione o dei lettori di NOTE DI SOFTWARE, rivestono un particolare interesse nell'ambito dei temi a cui questa rivista è dedicata.
Le citazioni fra virgolette, se non ne è indicata la fonte, sono tratte dai lavori stessi.

Il congresso AICA '79

● ATTI DEL CONGRESSO AICA, Bari, 10-13 ottobre 1979

VOL.I: INFORMATICA DISTRIBUITA; VALUTAZIONE DELLE PRESTAZIONI DEI SISTEMI. Sessioni: Sistemi funzionalmente distribuiti (19 lavori); Informatica e telecomunicazioni (12 lavori); Basi di dati distribuite (6 lavori); Valutazione delle prestazioni dei sistemi (9 lavori).

VOL. II: PRODUZIONE DEL SOFTWARE. Sessioni: Strumenti per lo sviluppo, l'analisi ed il collaudo del software (25 lavori); Tecniche e linguaggi di specifica (3 lavori); Metodi per l'implementazione dei sistemi (12 lavori); Progettazione di basi di dati (15 lavori).

VOL. III: INFORMATICA E TERRITORIO. Sessioni: Metodi e strumenti informatici per l'agricoltura (10 lavori); Telerilevamento (18 lavori); Gestione delle risorse naturali (7 lavori).

Un libro sul testing dei programmi

● Myers, G.J., THE ART OF SOFTWARE TESTING, John Wiley & Sons, 1979, pp.xi + 177

"[...] As the title implies, the book is a practical, rather than theoretical, discussion of the subject. Although it is possible to discuss program testing in a theoretical vein, the book is intended to be a practical, "both feet on the ground" handbook. Hence, many subjects related to program testing, such as the idea of mathematically proving the correctness of a program, were purposely excluded. Chapter 1 is a short self-assessment test that every reader should take before reading further. It turns out that the most important practical information that one must understand about program testing is a set of philosophical and economic issues; these are discussed in Chapter 2. Chapter 3 discusses the important concept of non-computer-based code walkthroughs or inspections. Rather than focus attention on the procedural or managerial aspects of this concept as most discussions do, Chapter 3 discusses it from a technical, how-to-find-errors point of view. The knowledgeable reader will realize that the most important component in the bag of tricks of a program tester is the knowledge of how to write effective test cases; this is the subject of Chapter 4. Chapter 5 and 6 discuss, respectively, the testing of individual modules or subroutines and the testing of larger entities, with Chapter 7 presenting some practical advice on program debugging. Chapter 8 surveys testing tools, research areas, and techniques not discussed elsewhere in the book and contains an extensive bibliography. The book has three major audiences. Although it is hoped that not everything in the book will be new information to the professional programmer, it should add to his or her knowledge of testing techniques. If the material allows one to detect just one more bug in one program, the price of the book will have been recovered many times over. The second audience is the project manager, since the book contains new, practical information on the management of the testing process. The third audience is the programming or computer-science student; the goal here is to expose the student to the problems of program testing and to provide a set of effective techniques. [...]".

Indice: 1. A SELF-ASSESSMENT TEST; 2. THE PSYCHOLOGY AND ECONOMICS OF PROGRAM TESTING: The economics of Testing; Testing Principles; 3. PROGRAM INSPECTIONS, WALKTHROUGHS, AND REVIEWS: Inspections and Walkthroughs; Code Inspections; An Error Checklist for Inspections; Walkthroughs; Desk Checking; Peer Ratings; 4. TEST-CASE DESIGN: Logic-Coverage Testing; Equivalence Partitioning; Boundary-Value Analysis; Cause-Effect Graphing; Error Guessing; The Strategy; 5. MODULE TESTING: Test-Case Design; Incremental Testing; Top-Down versus Bottom-Up Testing; Performing the Test;

6. HIGHER-ORDER TESTING: Function Testing; System Testing; Acceptance Testing; Installation Testing; Test Planning and Control; Test Completion Criteria; The Independent Test Agency; 7. DEBUGGING: Debugging by Brute Force; Debugging by Introduction; Debugging by Deduction; Debugging by Backtracking; Debugging by Testing; Debugging Principles; Error Analysis; 8. TEST TOOLS AND OTHER TECHNIQUES: Module Driver Tools; Static-Flow-Analysis Tools; Test-Coverage Monitors; Mathematical Proofs of Program Correctness; Program Correctness Provers; Symbolic Execution; Test Data Generators; Environmental Simulators; Sneak-Circuit Analysis; Virtual Machines; Testing Mathematical Software; Software Error Studies; Software-Error Data Collection; Predictive Models; Complexity Measures; Program Library Systems; Testing Experiments; Debugging Experiments; Interactive Debugging Tools; Compiler Debugging Aids; Program State Monitors; Computer Architecture.

Programmazione Cobol

● Al-Jarrah, M.M., Torsun, I.S., AN EMPIRICAL ANALYSIS OF COBOL PROGRAMS, SOFTWARE - Practice and Experience,vol. 9, n.5 (maggio 1979), 341-359

"This paper describes the results obtained from the static analysis of a large sample of typical COBOL programs written and used in several commercial and industrial organizations covering different applications, machines and standards. We first describe the data base used in the analysis and comment briefly on the static analyser system, then we outline the general program profile feature and the pattern of language used in different host machines. Next, we report and discuss the results obtained from the analys of the DATA DIVISION of COBOL programs: emphasis is placed on the type and structure of data used, and suggestions for speeding up the compilation process, in the light of these analyses, are proposed. Then we describe in detail the PROCEDURE DIVISION analysis; this includes frequency, format and pattern of COBOL verbs usage, the type of data used, the correlation between verbs frequencies and their relation to program size. The paper concludes with comments on the language, its use and possible improvements to the language and its compiling system. Suggestions for further work is also discussed."

Pascal

● Il numero di luglio 1979 di DATAMATION (vol. 25, n. 8) contiene una serie di articoli sul linguaggio Pascal ("Pascal Power", pp. 142-156)

"Although the language wasn't formally defined unitl 1971, its semplicity combined with its power for expressing complex algorithms make it a good bet for the future. Dennis Fletcher (p. 142) provides perspective from the standpoint of users, vendors and standards; Robert L. Glass (p. 146) explains how the Department of Defense is developing a single programming language for the military with Pascal as its base; Keith Shillington (p. 151) discusses structure, which he characterizes as the key to Pascal's problem-solving power; and Marvin Conrad (p. 153) finds it useful as a high-level language for micros and minis." (presentazione)

● Addyman, A.M., Brewer, R., Burnett-Hall, D.G., De Morgan, R.M., Findlay, W., Jackson, M.I., Joslin, D.A., Rees, M.J., Watt, D.A., Welsh, J., Wichmann, B.A., A DRAFT DESCRIPTION OF PASCAL, SOFTWARE -Practice and Experience, vol. 9, n.5 (maggio 1979) 381-429

"This paper provides a description of the programming language Pascal. It has been published to enable those without easy access to the official BSI 'draft for comment' to comment on the description [...].
In 1977 a working group was formed within the British Standards Institution (BSI) to produce a standard for the programming language Pascal. This working group is responsible to the technical committee on programming languages (DPS/13) and is designated DPS/13/4.
The working group first produced a list of all the known problems with the current definitions of Pascal. This was called the Attention List. The final version of the Attention List, which was produced in January 1978, ran to 17 pages. It contained contributions from members of the group, from correspondents and from published criticisms of Pascal. In April 1978 it was decided that further work on the Attention List should be suspended in favour of the production of a draft. To date there have been three working drafts. [...] The draft for public comments is an edited version of the third working draft. [...] In March 1978 a proposal that Pascal be added to the ISO program for work was sent form BSI to the International Standards Organization (ISO). This proposal was voted on and accepted. As a consequence of this decision the BSI becomes the 'sponsoring body' for Pascal. This means that the BSI draft should become the ISO draft [...]."

Documentazione

● Lano, R.J., A TECHNIQUE FOR SOFTWARE AND SYSTEMS DESIGN, TRW Series on Software Technology, vol. 3 (editor: B.W. Boehm), North-Holland Publishing Company, pp.xi + 119, 1979

"This paper describes the N^2 Chart, which is an implementation tool and methodology for the tabulation, definition, analysis and description of functional interactions and interfaces. The tool presented is not limited to any particular field, discipline, market area or system type.
The N^2 Chart is simple and easy to understand, structured and methodical, top-down in nature,communicative of the design, and forces a uniform level of design consistency. The N^2 Chart is an effective tool for the integration of all the people, products and paper that make up any given system. An initial N^2 Chart definition is provided, together with discussions relating to its various usages, formats and applications. Applications covered include interface tabulation, definition design and analysis activities. Other non-obvious applications are additionally discussed which are concerned with design description, operational procedure analysis and schedule analysis activities. Two examples are presented in detail which illustrate the applications of the N^2 Chart in interface problem solving and in top-down system design definition."

Un manuale di informatica

● Andronico, A., De Michelis, G., Di Leva, A., Reineri, M.T., Sami, M.G., Simone, C. (con il coordinamento di A. Siciliano), MANUALE DI INFORMATICA, Zanichelli S.p.A., pp. viii + 421, 1979, Lit. 11.000

Un volume che presenta i concetti di base della informatica, orientato a corsi introduttivi. Dopo avere introdotto i concetti fondamentali della programmazione utilizzando un linguaggio di tipo Pascal, il volume introduce i problemi del testing dei programmi, e delle dimostrazioni formali della loro correttezza, illustrata con vari esempi. Vengono poi trattati i concetti relativi ai tipi di dati, con un approccio vicino a quello di Pascal, e l'utilizzo di strumenti quali blocchi, procedure e funzioni per la costruzione di programmi strutturati in modo top-down. Dopo una descrizione dei componenti fondamentali di un elaboratore elettronico, vengono introdotti alcuni concetti relativi alle basi di dati, ed ai sistemi operativi (problemi di gestione della concorrenza, di allocazione della memoria, di gestione di input/output, e di controllo dei lavori). L'ultimo capitolo introduce la nozione di automa a stati finiti, e la utilizza per fornire un modello di elaboratore. Ciascun capitolo è corredato di un'ampia bibliografia per argomenti.

Indice: 1. AZIONI, PROCESSI E LORO DESCRIZIONE: Concetti fondamentali e definizioni; I modelli di composizione; Un linguaggio per la descrizione di processi: gli schemi di flusso; Un linguaggio a parole per la descrizione di processi. 2. ALGORITMI E PROGRAMMI: Algoritmi; Programmi; Identificatori e assegnamento; Espressioni aritmetiche; Predicati basici ed espressioni logiche; Esempi di costruzione di semplici programmi; Computabilità e tesi di Church. 3. TESTING E CORRETTEZZA DEI PROGRAMMI: Sintassi dei linguaggi di programmazione; La sintassi del nostro linguaggio; Generazione e riconoscimento di frasi del nostro linguaggio; Gli errori di tipo logico: la correttezza; Costruzione di programmi corretti. 4. LE STRUTTURE DEI DATI: I tipi di dati; Il tipo degli interi; Il tipo dei reali; Il tipo delle stringhe di caratteri; Vettori e matrici; Le sequenze a lunghezza variabile; Componendo dei dati di tipo diverso (i record); Dati strutturati e strutture dei dati. 5. METODOLOGIA PER LA COSTRUZIONE DEI PROGRAMMI: La programmazione top-down; Procedure; Un esempio. Il problema delle 8 regine; Funzioni; Le eccezioni e le istruzioni di salto. 6. STRUTTURE E COMPONENTI DEGLI ELABORATORI DIGITALI: Descrizione generale dei sistemi di elaborazione; Strutture di interconnessione fra le unità elementari: strutture a bus; Modalità di indirizzamento; L'unità di controllo: funzioni generali e realizzazione come rete sequenziale; Il concetto di microprogrammazione e lo schema-base dell'unità di controllo microprogrammata; Schemi alternativi di unità di controllo microprogrammate; La memoria di lavoro: tecnologie a nuclei magnetici; Memorie a semiconduttori; Memorie di massa; La gestione di ingresso/uscita: trasferimenti a controllo di programma, interrupt, accesso diretto alla memoria e

canali; L'unità aritmetica; L'aumento della efficienza della macchina: parallelismo e organizzazione a "banchi di memoria"; L'aumento dell'efficienza: gerarchie di memorie; Strutture di supporto per la multiprogrammazione; Strutture con unità di elaborazione multiple. 7. BANCHE DI DATI: Descrizione delle informazioni; Il livello concettuale: l'organizzazione logica globale; Il livello esterno: le organizzazioni logiche dei dati; Il livello interno: l'organizzazione fisica; Il sistema di gestione della banca di dati SGBD; L'approccio reticolare; L'approccio relazionale; Sintesi dello schema di una BD; Organizzazione dei record fisici; Organizzazione dei metodi di accesso: l'accesso multiplo; Organizzazione dei SGBD; 8. SISTEMI OPERATIVI: Punto di vista sistemistico-gestionale e concetti di base; Componenti essenziali di un Sistema Operativo; Processi concorrenti e condivisione di risorse; Allocazione della memoria; Metodi di allocazione della memoria; Gestione Apparecchiature di Ingresso e Uscita; Gestione lavori; Una visione d'insieme; Le situazioni di stallo. 9. AUTOMI: Macchine sequenziali; Modi di descrizione degli automi; Modi di operare di un automa; Simulazione di automi; Minimizzazione di automi; Moduli e reti modulari; Osservazioni; Algoritmi ed elaboratori; Classificazione generale delle istruzioni; Funzionamento dell'elaboratore come automa; Avvio automatico di un programma.

Un libro sull'Algol 68

● Brailsford, D.F., Walker, A.N., INTRODUCTORY ALGOL 68 PROGRAMMING, Ellis Horwood Limited (distribuzione John Wiley & Sons Inc.), pp. 281, 1979

"This book is intended for anyone who wants to learn to write programs in Algol 68. It is based on the course which we have given for some years at Nottingham to mathematics undergraduates learning their first programming language. [...] In its formative years, Algol 68 had a very bad press. Stories circulated of formidably obscure documents and of the high priests of the language deliberating in conclave on whether a particular full stop in the defining report ought to be in italics. As usual, there was a grain of truth in these stories, and a mountain of exaggeration; new initiates may rest assured that there is nothing in the use of Algol 68 (as possibly opposed to its rigorous definition) which is in any way harder than the corresponding use of Fortran (or Algol 60, Basic, Pascal, PL/1, APL or any other computing language). We are certainly not high priests of Algol 68 (not even priests - perhaps missionaries). Our concern is to write programs in the most effective way possible, and to teach others to do so; and we are quite sure that Algol 68 is currently far and away the best vehicle for writing general purpose programs simply, efficiently and correctly. We hope we can convince you. [...]"

Ingegneria del software

● De Michelis, G., Lanzarone, G.A., Polillo, R., Simone, C., Vianello, P., IL DISEGNO DEL SOFTWARE: DALL'ESPERIENZA AL METODO, supplemento alla rivista Sistemi e Automazione, n. 197, ottobre 1979, pp. 35

"Per disciplinare, guidare, supportare la progettazione del software sono state messe a punto diverse tecniche alcune delle quali hanno ormai una discreta diffusione anche in Italia. In questo fascicolo vogliamo offrire un quadro sintetico e comparato di alcune metodologie, metodi, strumenti che hanno interesse o per originalità di impostazione, o per livello di diffusione. La struttura della rassegna è composita.
Il suo cuore è rappresentato dalla tavola centrale in cui sedici tecniche sono classificate in modo comparato. Un articolo introduttivo situa le tecniche nella problematica dell'ingegneria del software. Sedici schede individuali descrivono in modo più dettagliato le tecniche e offrono una bibliografia minima per ampliare le proprie conoscenze. Un breve articolo presenta infine le prime valutazioni di una inchiesta svolta tra gli abbonati di Sistemi e Automazione sulla diffusione delle metodologie di progettazione del software in Italia."
Le sedici tecniche esaminate sono: SADT; Metodologia Metra; SREM; Metodo delle Reti di Petri; ISDOS (PSL/PSA); HIPO; Galois; SDL; Structured Design; Wellmade System Design Methodology; Metodo Warnier; Metodo Jackson; Metodo PHOS; PDL; Programmazione Modulare; Programmazione Strutturata.

● Mohanty, S.N., MODELS AND MEASUREMENTS FOR QUALITY ASSESSMENT OF SOFTWARE, ACM Computing Surveys, vol. 11, n. 3 (settembre 1979), 251-275

"Several software quality assessment methods which span the software life cycle are discussed.

The quality of a system design can be estimated by measuring the system entropy function or the system work function. The quality improvement due to reconfiguration can be determined by calculating system entropy loading measures. Software science and Zipf's law are shown to be useful for estimating program length and implementation time. Deterministic and statistical methods are presented for predicting the number of errors. Testing theory is useful in planning the program test process; as discussed in this paper, it includes measurement of program structural characteristics to determine test effectiveness and test planning. Statistical models for estimating software reliability are also discussed."

Human-Computer Interfaces

● Palme, J., A HUMAN-COMPUTER INTERFACE FOR NON-COMPUTER SPECIALISTS, SOFTWARE — Practice and Experience, vol. 9, n. 9 (settembre 1979), 741-747

"The COM teleconferencing system was designed to be easy to use for both beginners and people with much computer experience. A number of design choices in organizing the human-computer interface were considered very carefully. These design problems are not unique for teleconferencing applications, but will appear in many other developments of human-computer interfaces for non-computer specialists.
This report discusses naming conventions, menu format, user commands, help facility and the treatment of 'type ahead' from the users."