Honeywell
Honeywell Information Systems Italia

Via Laboratori Olivetti - Pregnana Milanese

UNIVERSITA'
DEGLI STUDI DI MILANO
DIPARTIMENTO DI SCIENZE
DELL'INFORMAZIONE
Via Moretto da Brescia, 9
20133 Milano - Tel. 2772233

32

NOTE DI SOFTWARE

UNIVERSITA'
DEGLI STUDI DI MILANO
DIPARTIMENTO DI SCIENZE
DELL'INFORMAZIONE

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 del Dipartimento di Scienze dell'Informazione 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 e 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 vengono revisionati dal Comitato di Redazione, che si avvale del contributo di specialisti del settore. Ogni contributo pubblicato riflette, comunque, le opinioni 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 ad aziende, Enti e specialisti del settore che ne facciano richiesta alla redazione.

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

32

Giugno 1986

Indice:

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

NOTE DI SOFTWARE

QUESTO NUMERO...

Questo numero di NOTE DI SOFTWARE si apre con un lavoro di G. Facchetti sulle caratteristiche principali di un sistema object-oriented basato sul linguaggio Smalltalk in termini comparativi con i linguaggi tradizionali: si tratta di un lavoro di carattere introduttivo a questo linguaggio a cui segue l'esempio di una applicazione sviluppata in una collaborazione tra l'Istituto di Cibernetica e l'Olivetti illustrata nel secondo articolo, di U. Corda e G. Facchetti, su un sistema interattivo per la creazione e la consultazione di documentazione.

NOTE DI SOFTWARE prosegue con un lavoro di M.E. Delpero in cui viene scritto il "dossier virtuale" uno "strato" di software che si frappone tra l'utente e il foglio elettronico, mediante l'utilizzo dello spread-sheet LOTUS 123 e un esempio di realizzazione nell'ambito dell'office automation.

Nel quarto lavoro, di P. Piardi, viene proposta una possibile classificazione della corrispondenza in base ai concetti espressi e alle parole usate per esprimerli.

G. Solaro, poi, presenta la descrizione di GESC, un sistema di consultazione che permette di costruire una base di conoscenza a partire da un testo, con un esempio di applicazione al settore medico.

Il metodo C.F.D. (Computer Figurative Design) per elaborare disegni basato su un sistema bipolare viene poi illustrato da G. Cella. Il lavoro presenta alcuni risultati grafici prodotti dall'uso di questo metodo.

Infine G. Mauri recensisce un libro di grande interesse per l'ambito degli algoritmi paralleli: si tratta di "Parallel Sorting Algorithms" di S.G. Akl di recente pubblicazione.

La Redazione

UNA INTRODUZIONE ALL'AMBIENTE SMALLTALK

Giuseppe Facchetti

Istituto di Cibernetica - Università di Milano

Ing. C. Olivetti & C. - Divisione Sviluppo Sistemi

RIASSUNTO

Questo articolo presenta le caratteristiche principali del sistema object-oriented basato sul linguaggio Smalltalk. Dopo una rassegna di punti qualificanti del linguaggio, viene fatto un confronto tra i linguaggi tradizionali e lo Smalltalk, e si analizzano le idee che stanno alla base di quest'ultimo.

ABSTRACT

This paper presents the main characteristics of the object-oriented environment based on the Smalltalk language.
It begins with a description of the language itself; then, a comparison between more traditional languages and the Smalltalk is done.
Finally, the basic ideas on which it is founded are listed.

1. SISTEMI SOFTWARE OBJECT-ORIENTED

I linguaggi object-oriented nascono quale ultima evoluzione nel campo dei linguaggi general-purpose e dei linguaggi di controllo finora costruiti. Il primo tipo di interazione uomo-macchina è quello basato sui linguaggi assemblativi: essi scontano la possibilità di sfruttamento completo ed efficiente delle risorse della macchina, con le difficoltà dovute all'assenza di strutturazione ed alla scarsa leggibilità. Il primo passo verso una maggiore facilità di programmazione è il Fortran (capace di fornire un minimo di strutturazione del codice e dei dati).
Successivamente, l'introduzione dell'Algol indica uno schema strutturale seguito da molti linguaggi nati in seguito.
L'introduzione di tipi di dati astratti o Abstract Data Types (ADT), avutasi con linguaggi quali Simula e CLU, pone infine l'enfasi sulle ampie possibilità di astrazione e modularizzazione rese possibili da queste funzionalità (in particolare dal concetto di classe, fondamentale nel caso dello Smalltalk).

La programmazione **object-oriented,** ultimo passo di questo processo evolutivo, sostituisce il concetto tipico "operatore/operando" che sta alla base dei linguaggi tradizionali, con il concetto "messaggio/oggetto" [7].
Non si esprime più il flusso delle operazioni con una seguenza di richiami di operatori che lavorano su operandi legati o meno ad un tipo: si considera invece l'ambiente di programmazione come un insieme di **oggetti** ai quali viene richiesto l'espletamento dei compiti attraverso un meccanismo di invio di **messaggi** agli oggetti stessi. La modalità di risposta al messaggio, e quindi l'effetto della "chiamata", dipendono non più dal contesto globale del programma (o della zona di programma) dove la chiamata avviene, ma vengono determinate esclusivamente dal ricevitore del messaggio (cioè l'oggetto al quale il messaggio è stato mandato) in funzione della classe, cioè del tipo, cui l'oggetto in questione appartiene.

In termini pratici, una **classe** è un modello che indica

a. la **struttura interna** degli oggetti che verranno creati come membri di quella classe, e

b. l'insieme di **compiti** che tali membri possono eseguire, cioè l'implementazione delle "procedure" che permettono loro di rispondere nel modo voluto ai messaggi che possono ricevere.

Un oggetto è costituito da:

a. una certa quantità di **memoria locale,** la cui struttura è specificata dalla classe di appartenenza, e

b. una descrizione del suo tipo, cioè della **classe** della quale è stato dichiarato membro al momento della sua creazione.

La principale differenza tra questo tipo di organizzazione e quello tradizionale sta nell'indipendenza del sistema rispetto agli oggetti che esso contiene: spesso è possibile introdurre nuovi ADT senza toccare il co-

dice che vi lavorerà sopra. Un'altra questione rilevante riguarda la possibilità di scrivere codice senza conoscere esattamente il tipo degli oggetti che vengono usati: basta sapere che rispondono ad un certo messaggio.

Oltre che nel campo dei linguaggi di applicazione generale (C++ [19], Objective-C, Clascal [7], Ada [5], InterLisp [4 cap. 4], Smalltalk [13]), questi concetti hanno trovato applicazioni in alcuni linguaggi di controllo per sistemi operativi. È il caso di Hydra (un sistema a capability [10]), sopra il quale è stato costruito Cola [18], un linguaggio object-oriented progettato per effettuare una corrispondenza tra oggetti (capabilities) del sistema, ed oggetti forniti da linguaggio.

Un altro esempio di questo tipo è l'IBM System/38 [8], le cui funzioni di sistema operativo sono fornite dal Control Program Facility (CPF). L'architettura fornisce una memoria indirizzabile di tipo object-oriented: quando un oggetto viene creato, CPF gli associa gli oggetti di sistema che lo compongono, e pone il suo nome in un "contesto", cioè un libreria o catalogo di oggetti. Le funzionalità sono le stesse di Cola, e i concetti base, opportunamente adattati, sono gli stessi che nel prossimo paragrafo si vedranno per lo Smalltalk.

2. L'AMBIENTE SMALLTALK

2.1 Oggetti e messaggi

Nel linguaggio Smalltalk, con il termine **oggetto** intendiamo una qualunque componente del sistema software nel quale ci troviamo. Gli oggetti rappresentano quindi numeri, stringhe, programmi, finestre sul video, eccetera. Un **messaggio** è una richiesta mandata ad un oggetto affinchè esegua una delle operazioni che esso può svolgere. Il **ricevitore,** cioè l'oggetto che ha ricevuto il messaggio, determina **come** eseguire il compito richiesto. L'insieme di messaggi cui un oggetto può rispondere è chiamato **interfaccia** dell'oggetto verso il sistema. Quindi l'unico modo di interagire con un oggetto consiste nel passare attraverso la sua interfaccia: osserviamo che la memoria privata di un oggetto può essere manipolata solo da una delle operazioni da esso previste, e che queste, a loro volta, possono essere attivate solo attraverso il lancio di un messaggio.

I messaggi assicurano la modularità del sistema, perchè essi specificano il tipo di operazione voluta, ma non danno informazioni su come le operazioni vanno eseguite o implementate. La costruzione di una applicazione Smalltalk, in modo analogo a quanto si fa con gli altri linguaggi, implica la preventiva scelta degli oggetti dei quali l'applicazione si servirà per svolgere il suo lavoro.

Una **classe** descrive l'implementazione di un sistema di oggetti che rappresentano lo stesso tipo di componente del sistema (in modo analogo a quanto fanno le dichiarazioni di Abstract Data Types in un linguaggio Pascal-like). I singoli oggetti descritti da una classe sono chiamati **membri** di quella classe. Le proprietà "private" di un oggetto Smalltalk sono descritte nella definizione della classe di appartenenza, e sono rappresentate da

a. un insieme di **instance variable** (*) che costituiscono la memoria privata, e

b. un insieme di **metodi** che descrivono come espletare le operazioni previste.

Ogni metodo di una classe specifica come eseguire il compito richiesto da un certo messaggio: quando quel messaggio è mandato ad un oggetto obj1, membro di quella classe, il metodo corrispondente (omonimo, per la precisione) viene eseguito.
Poichè le instance variables contengono a loro volta oggetti Smalltalk, e quindi a loro volta appartengono a classi, il codice contenuto nei metodi sarà costituito da una sequenza di messaggi mandati a tali oggetti o ad altri del sistema. Tale codice, però, potrà produrre dei cambiamenti solo sulla memoria privata dell'oggetto obj1.

Dunque il lancio di un messaggio provoca l'esecuzione di un metodo, che a sua volta lancia altri messaggi. La ricorsività di questo procedimento si ferma quando si arriva ai cosiddetti **primitive methods,** che sono metodi non espressi in Smalltalk, ma predefiniti nel sistema. Essi non possono essere cambiati dal programmatore Smalltalk, e permettono di accedere alla macchina virtuale ed all'hardware sottostante (il metodo "+" definito nella classe degli interi non deve lanciare nessun messaggio al suo interno: deve solo fare in modo che l'hardware della macchina esegua un'addizione).

2.1.1 Un esempio

Un esempio concreto di questi fatti è la classe **LinkedList,** che costituisce la struttura base su cui costruire liste concatenate. La sua memoria privata consiste di due instance variables, denominata **firstLink** e **LastLink:** esse puntano rispettivamente al primo ed all'ultimo elemento della lista, e sono anche tutto ciò che serve per lavorare su una struttura dati di questo tipo (osserviamo che questa classe dà solo la struttura della lista; si vedrà più avanti come aggiungere anche un contenuto ai suoi elementi).
All'interno di tale classe sono definiti solo due metodi, denominati **at:** e **size.** Il primo prende come parametro un indice (una posizione all'interno della lista) e ritorna il corrispondente elemento; il secondo

(*) Il termine "instance variable" indica una variabile che viene associata ad un oggetto al momento della sua creazione, e quindi è una zona di memoria che esiste solo fintanto che esiste quell'oggetto.

ritorna invece il numero di elementi contenuti nella lista. Come detto più sopra, questi metodi vengono attivati inviando ad una lista il messatto at: o il messaggio size. Inoltre, il codice di questi due metodi potrà lanciare messaggi ad altri oggetti, ma potrà modificare direttamente solo i valori delle sue due istance variables (che, in particolare, appartengono alla classe degli Integer), per esempio mediante degli assegnamenti.

La classe LinkedList considerata appartiene ad un insieme di classi che forniscono le funzionalità tipiche di un linguaggio o ambiente di programmazione tradizionali: esistono classi per le operazioni aritmetiche più comuni, per le strutture di dati più semplici, per le strutture di controllo, per l'interazione uomo-macchina, per la gestione dell'interfaccia a finestre. Questo insieme "minimo" di funzionalità può essere considerato come un insieme di "classi di sistema", anche se la definizione è del tutto impropria: ogni applicazione costruita dal programmatore verrà semplicemente aggiunta al set delle classi già presenti. Quindi di fatto, come già detto, non vi è distinzione tra utente e sistema.

2.2 Sintassi del linguaggio

Lo Smalltalk è basato su una sintassi **expression-oriented,** cioè descrive oggetti e messaggi attraverso **espressioni.** Con il termine "espressione" intendiamo sequenze di caratteri che descrivono l'oggetto corrispondente al **valore** dell'espressione. Esistono quattro tipi di espressioni:

1. **Literals,** che denotano delle costanti, come numeri o stringhe;

2. **Variable Expressions,** che descrivono le variabili accessibili;

3. **Message Espressions,** che indicano il lancio di messaggi ad oggetti riceventi. Il valore di queste espressioni è il valore ritornato dal metodo corrispondente;

4. **Block expressions,** che rappresentano pezzi di codice usati per computazioni differite o strutture di controllo.

Analizzeremo ora brevemente le quattro situazioni.

2.2.1 Literal Expressions

Tipici literals sono i numeri, che vengono scritti nella notazione solita dei linguaggi di programmazione, ed i caratteri, che vanno indicati preceduti dal simbolo $. Ad esempio $a è l'"a" del Pascal; $$ è il "$". Le stringhe sono oggetti rappresentanti sequenze di caratteri, e sono indicate tra apici, come 'tesi di laurea'. Esistono poi i Simboli, che sono sequenze di caratteri che indicano nomi di oggetti, e vengono scritti aggiungendo il prefisso "#", come in #beppe o #Smalltalk. Infine gli Array, che sono da intendere come insiemi ordinati di oggetti di classi anche diverse, se indicati esplicitamente vanno preceduti da '# (' e seguiti da ')', come # (a b c), # ('pippo' 'pluto' pape 34).

2.2.2 Variable expressions

I nomi di variabile sono il mezzo con il quale è possibile accedere agli oggetti del sistema. La memoria disponibile è costituita di variabili, accedute per nome o per indice (array). Esistono, in generale, due tipi di variabili: le **private,** alle quali può accedere solo un particolare oggetto (è il caso tipico delle instance variables di quell'oggetto), e le **condivise,** alle quali possono accedere più oggetti (per queste il linguaggio richiede un nome iniziante con lettera maiuscola). Fisicamente, un nome di variabile è un puntatore alla zona di memoria che contiene l'oggetto effettivo riferito. In Smalltalk esistono alcune variabili il cui valore non può venir cambiato: esse sono dette **pseudo-variabili**. Si tratta di **nil, true, false,** con significati ovvi, e di **self** e **super,** di cui si parletà più avanti.

2.2.3 Message expressions

Come si è visto, i messaggi rappresentano le interazioni tra i componenti del sistema Smalltalk, ed indicano di fatto una richiesta di esecuzione di operazioni da parte del ricevente. Ad esempio la scrittura

3+4

lancia il messaggio "+" all'oggetto "3", passando come parametro l'oggetto "4". Il valore di questa espressione è, ovviamente, 7. La scrittura

theta abs

indica invece il messaggio "abs" inviato alla variabile theta (probabilmente theta è un numero, e abs ne ritorna il valore assoluto; in realtà solo l'oggetto di nome theta sa a quale classe egli appartiene, e quindi in quale classe cercare il metodo "abs" da eseguire). Una message expression indica sempre **un ricevitore,** un **selettore** (cioè un nome di metodo o messaggio), e zero o più **argomenti.** I due esempi di poco fa corrispondono rispettivamente ad una **espressione binaria** ed una **unaria.** Esistono poi i cosiddetti **keyword messages,** nei quali il selettore è costituito da una o più keywords, una per argomento. Ad esempio

index between: min and: max

indica il messaggio between:and: (keyword message a due keywords; i ":" sono obbligatori) lanciato alla variabile index con parametri min e max; questa espressione restituisce true se index è compreso nell'intervallo min-max, false altrimenti.

2.2.4 Block expressions

Una block expression consiste di una sequenza di espressioni separate da punti (".") e delimitate da parentesi quadrate, come

```
[anInteger: = (–4) abs.
set add: anInteger]
```

Un blocco è a tutti gli effetti un oggetto: quando viene incontrato, non viene eseguito ma può essere assegnato ad una variabile, passato come parametro o altro. L'esecuzione viene avviata quando al blocco è lanciato il messaggio "value". Così ad esempio i due pezzi di codice:

```
index: = index +1.
```

e

```
aBlock: = [index: = index + 1].
aBlock value.
```

ottengono lo stesso effetto: l'incremento di index.

Un comportamento un po' diverso è tenuto dai blocchi quando qusti vengono usati per generare quelle che per ora possiamo chiamare **strutture di controllo** (selezione ed iterazione) del linguaggio.

Per esempio la **selezione condizionale** di un'attività è ottenuta inviando ad un booleano (cioè una condizione) il messaggio ifTrue:ifFalse: con due blocchi come argomenti.
Ad esempio la seguente espressione assegna alla variabile "greater" il valore 1 se il contenuto della variabile

```
index › 0
    ifTrue: [greater: = 1]
    ifFalse: [greater: = 0]
```

"index" è maggiore di zero, ed il valore 0 altrimenti. Si osservi che dopo la valutazione di questa espressione, la variabile greater apparterrà alla classe degli SmallInteger, cioè sarà un numero, e questo indipendentemente dalla classe di appartenenza precedente (binding dinamico di tipo).

La **ripetizione condizionale** viene ottenuta mandando ad un blocco il messaggio whileTrue: (keyword message ad una keyword) con un altro blocco come argomento. Naturalmente il primo blocco, cioè il blocco ricevente, dovrà ritornare un booleano (a questo proposito si tenga presente che un blocco ritorna il valore restituito dall'ultima delle espressioni che esso contiene). Ad esempio il seguente pezzo di codice produce la minima posizione i alla quale le due liste firsLinkedList e secondLinkedList hanno due elementi uguali.

```
[item: = firstLinkedList at: i.
(secondLinkedList at: i) = item]
    whileTrue:
        [i: = i + 1].
```

Si notino le similtudini tra queste strutture di controllo ed i "guarded commands" di Dijkstra [Dijkstra 76].

È importante sottolineare, tra le caratteristiche dei blocchi, i parallelismi esistenti tra questi oggetti ed i loro equivalenti in linguaggi come C e Pascal. Di fatto i blocchi Smalltalk possono essere visti come uno strumento di strutturazione del codice, alla stessa stregua delle procedure Pascal e dei blocchi C. In realtà nel nostro caso abbiamo uno strumento meno potente, ma l'idea di fondo e le modalità di utilizzo sono simili. Una situazione dove però i blocchi mostrano la loro utilità è quello del **trattamento di eccezioni:** è molto facile rimandare indietro segnalazioni di errore nella sequenza di attivazione generata dall'esecuzione, ed interrompere quest'ultima senza goto, effetti collaterali ed altri problemi tipici dei linguaggi tradizionali in queste situazioni.

Il problema dei **side-effect** (*) si presenta semmai in un altro modo di utilizzo dei blocchi: abbiamo infatti visto che è possibile, analogamente a quanto avviene nel C, inserire del codice nella condizione della struttura "while". Questo si traduce di fatto in un incoraggiamento all'uso di side-effect, per quanto controllati e gestiti esplicitamente. Di questi fatti ci si occuperà più specificatamente più avanti.

2.3 Sottoclassi ed Ereditarietà

Gli aspetti più importanti dello Smalltalk, che lo caratterizzano come linguaggio object-oriented, sono la sua strutturazione basata su una gerarchia di classi ed i concetti di ereditarietà introdotti da tale struttura. Questi però coinvolgono anche discorsi di visibilità ed accessibilità delle zone di memoria utilizzate, e quindi è necessario fare ancora qualche osservazione sulle diverse possibilità di gestione variabili che il linguaggio offre.

Esistono due tipi di variabili private, cioè visibili unicamente ad un oggetto:

a. **Instance Variables,** che esistono fintanto che esiste l'oggetto cui appartengono. L'insieme dei valori assunti in un certo istante dalla instance variables di un oggetto rappresentano lo **stato** di quell'oggetto. Ne consegue che oggetti più complessi avranno un maggior numero di instance variables;

(*) Il termine indica gli effetti provocati indirettamente su oggetti del sistema diversi da quelli su cui l'elaborazione formalmente dovrebbe lavorare.

b. **Temporary Variables,** che sono create per una specifica attività, cioè all'inizio di un metodo, ed esistono solo mentre quel metodo è attivo.

Tre tipi di variabili possono essere accedute da più di un oggetto:

a. **Class Variables,** condivise da tutti i membri di una singola classe;

b. **Global Variables,** condivise da tutti gli oggetti presenti nel sistema;

c. **Pool Variables,** condivise dagli oggetti di un sottoinsieme delle classi presenti (precisamente, quelle che ne hanno fatto esplicita richiesta).

Questo è quanto concerne le aree di memoria. Per quanto riguarda la loro gestione, come si è visto se ne occupano i metodi, che a questo punto è possibile definire come "sequenze di espressioni separate da punti". Infine, ogni classe può avere un solo metodo con un dato nome, anche se ovviamente classi diverse possono rispondere allo stesso messaggio. All'interno di un metodo, la pseudo-variabile **self** fa riferimento al ricevitore del messaggio. Vedremo più avanti a chi si riferisce invece la pseudo-variabile **super.**

La struttura finora descritta non accenna a possibilità di intersezioni tra classi, cioè ogni oggetto è membro di esattamente una classe. In realtà lo Smalltalk permette una situazione meno restrittiva: è permesso che una classe includa tutti i membri di un'altra classe. In questo caso si dice che quest'ultima è **sottoclasse** della prima. Ne consegue che tutti gli oggetti che sono membri di una classe B, sottoclasse di una classe A, sono anche membri di A. Si dice che A è **superclasse** di B. Praticamente una sottoclasse sta ad indicare che i suoi membri sono oggetti uguali ai membri della sua superclasse, salvo che per certe differenze specificate esplicitamente. Queste differenze, e qui sta uno dei punti chiave della struttura dello Smalltalk, consistono in nuove **instance variables,** e nuovi metodi, che si vanno ad aggiungere a quelli della/e superclasse/i.

I nomi delle nuove instance variables devono essere diversi dai nomi definiti nelle superclassi, mentre per i metodi ciò non è necessario. Anzi, una delle principali potenzialità di questa organizzazione è la possibilità di ridefinire dei metodi fornendo nelle sottoclassi comportamenti diversi da quelli delle superclassi. Viene effettuato così ciò che viene chiamato un **overloading** degli operatori (o meglio, dei messaggi), in cui il metodo esatto viene determinato in base al tipo, cioè alla classe di appartenenza, del ricevitore. I metodi delle classi inferiori effettuano un **overriding** o **shadowing** su quelli delle classi superiori. Nel momento in cui ad un oggetto viene lanciato un messaggio, il metodo corrispondente viene cercato nella classi di appartenenza; se non viene trovato, la ricerca prosegue nell'immediata superclasse, poi nella superclasse della superclasse, e così via.

La gerarchia delle superclassi è per definizione limitata superiormente dalla classe **Object,** che quindi è superclasse di tutte le altre. Ne consegue che se la ricerca del metodo vista sopra si rivela infruttuosa anche nella classe Object, ciò significa che il ricevitore del messaggio non è in grado di rispondere, e quindi si è in una situazione di errore. Se invece durante questa scalata alle superclassi si trova un metodo con lo stesso nome del messaggio lanciato, la ricerca finisce e questo metodo viene eseguito.

A questo punto è possibile chiarire la funzione della pseudo-variabile **super.** Essa, analogamente a self, riferisce al ricevitore del messaggio, ma quando un messaggio è mandato a super si ha che la ricerca del metodo corrispondente inizìa non nella classe della quale il ricevitore è membro, ma nella sua diretta superclasse. Ciò permette di accedere a metodi della superclasse anche se essi hanno subito un overriding nella sottoclasse, e questo si rivela in molti casi utile.

Questo paragrafo si conclude accennando brevemente alle **metaclassi.** Poichè ogni componente del sistema Smalltalk è un oggetto, e le classi fanno parte del sistema, ne consegue che anche le classi sono oggetti, e che quindi sono anch'essi membri di qualche classe. In questo senso il linguaggio introduce una metaclasse per ogni classe del sistema: ogni classe cioè è membro della sua propria metaclasse. Tutte le metaclassi sono membri di una classe appositamente creata che si chiama Metaclass, e che è membro di se stessa.

Allo stesso modo, tutte le classi sono sottoclassi della classe appositamente creata Class. Class e Metaclass sono sottoclassi si una classe denominata Behavior, che è sottoclasse di Object. Questo chiude definitivamente il cerchio delle ereditarietà.
Esistono perciò dei metodi che lavorano sulle classi invece che sui membri delle classi; sono metodi in particolare che si occupano della creazione ed inizializzazione dei membri, e della gestione delle class variables di cui sopra. Questi metodi prendono il nome di **Class Methods,** mentre i metodi che lavorano sui membri della classe vengono chiamati **Instance Methods.**

2.4 Tipi di dati e strutture dati

In Smalltalk non è esatto parlare, come si potrebbe fare per altri linguaggi, di tipi predefiniti. Infatti, come si è detto, ogni interazione del programmatore di trasforma automaticamente in oggetti di sistema, e quindi ogni struttura dati aggiunta diventa una struttura del sistema.
In [13] si prevede, comunque, un set di classi e quindi di tipi che devono essere presenti nel sistema inziale, per rendere disponibili le funzionalità minime del linguaggio. Esempi di classi appartenenti a questo

gruppo sono ovviamente le numeriche, quali Number, Fraction e SmallInteger.

Esiste poi una classe denominata Collection, che ha sotto di sè le principali strutture dati utilizzabili dal programmatore per le proprie applicazioni.

L'accesso alla memoria secondaria passa attraverso i membri della classe Stream e sue sottoclassi, che sono strutture molto flessibili fornite di metodi per la lettura, la scrittura e l'aggiornamento degli oggetti (tipicamente collezioni) cui vengono associate. Associando perciò delle stream a file, si hanno a disposizione tutte le operazioni normalmente previste per la gestione di questi ultimi nei linguaggi tradizionali.

A questo punto il problema è capire come sia possibile aggiungere strutture più complesse al sistema. **In Smalltalk aggiungere nuovi ADT significa aggiungere nuove classi.**
Quello che si fa, cioè, è

- decidere la struttura dell'oggetto che si vuole costruire (che può essere inteso come un record Pascal di cui vanno decisi i campi);
- dare alla struttura un nome, che sarà il nome della classe che lo definirà;
- aggiungere questa nuova classe al sistema, indicando le variabili (di classe, di membro, globali) ed i metodi (di classe e di membro) che si prevedono utili;
- utilizzare infine tutto questo nelle applicazioni, generando membri della classe così creata e lanciando loro i messaggi previsti.

Perciò per ogni struttura dati che si vuole descrivere verrà data una corrispondente definizione di classe Smalltalk.

2.4.1 Un altro esempio

Come esempio molto semplice, si considerano le liste concatenate di cui si è parlato più sopra. Come visto, le classe **LinkList** fornisce solo l'"impalcatura" su cui costruire la lista, ma non è in grado di associare un contenuto ai suoi elementi. Se si vogliono invece dei nodi che siano record costituiti da un contenuto (per esempio una stringa) ed un puntatore al prossimo elemento, si può creare una classe di nome, ad esempio, **ListNode** con due instance variables che vengono chiamate "content" e "next".

Per quanto riguarda la creazione dei membri della classe ListNode, il sistema fornisce il messaggio **new** valido per tutte le classi, che genera un membro vuoto di quella classe. È però possibile fare uno shadowing di quel messaggio aggiungendo un messaggio di classe **new** per la classe ListNode, che per esempio inizializza le variabili content e next. Se vengono definiti due instance methods per la classe in esame, che assegnano i valori alle due variabili (i commenti sono tra virgolette):

```
content: aString "nome del metodo e parametro formale"

content:= aString "assegna il contenuto della string
                   aString alla intance variable content"

next: aLinkNode "nome del metodo e parametro formale"

next:= aLinkNode "assegna a next l'oggetto aLinkNode; si ricordi che le variabili
                  sono sempre dei puntatori, e quindi aLinkNode può anche essere nil"
```

basterà allora definire un class method "new" come segue:

```
new
    ^super new content: "nulla";
               next: nil.
```

dove all'elemento vuoto creato dall'espressione "super new" vengono mandati in sequenza (";") i messaggi che inizializzano le due variabili. Si ricordi che solo gli instance methods possono modificare direttamente i valori delle instance variables. Il simbolo "^" indica che il risultato dell'operazione (un ListNode inizializzato) è il valore restituito dal metodo new così definito. A questo punto si tratterà di aggiungere alla classe LinkedList metodi per l'aggiunta, l'eliminazione o la ricerca di nodi, creando tali nodi con l'invio del messaggio "new" alla classe ListNode.

È possibile fare un ultimo richiamo al concetto di sottoclasse osservando che se si volesse una lista doppiamente linkata, basterebbe definire una sottoclasse di ListNode con una instance variable di nome (per esempio) "previous", aggiungendo magari un metodo che ritorna il "nodo precedente", puntato da questa nuova variabile. I membri della nuova classe avrebbero allora una memoria privata costituita dalle variabili content e next definite nella superclasse, e previous definita localmente. Inoltre questi elementi avrebbero a disposizione tutti i metodi di ListNode, senza bisogno di ridefinirli (ma potendo comunque farlo se ce ne fosse bisogno).

3. ANALISI DELLE SIMILTUDINI E DIFFERENZE TRA LO SMALLTALK ED ALTRI LINGUAGGI

In questa parte vengono messi in evidenza i parallelismi esistenti tra lo Smalltalk ed altri linguaggi più noti, ed allo stesso tempo vengono analizzate le differenze sintattiche e semantiche tra costrutti apparentemente simili tra loro. Quello che si vuole fare è riuscire a familiarizzare con certi costrutti Smalltalk apparentemente inusuali, e fare presenti le particolarità di altri apparentemente identici a quelli di linguaggi noti. Scopo del paragrafo è tentare per la prima volta un raffronto di questo tipo.

3.1 Interfaccia

Lo Smalltalk si presenta come linguaggio innovativo anche rispetto al tipo di interfaccia utente che esso offre.
Questa è infatti basata su finestre, menu e "browser".

Con il termine "finestra" si intende la solita zona rettangolare di video contenente informazioni (che nel caso in esame consistono in liste di nomi e pezzi di teso), fornita dei tipici meccanismi di clipping e di scolling orizzontale e verticale del contenuto. I menu sono liste di voci che compaiono sul video a richiesta dell'utente: la selezione di una di queste voci è una richiesta (fatta al sistema o all'applicazione) di espletamento di un certo compito. I browser, infine, sono zone rettangolari di video composte da più finestre collegate logicamente fra loro: essi costituiscono l'interfaccia tra l'applicazione e l'utente, nel senso che le richieste dell'uno e le informazioni fornite dall'altra passano attraverso l'uso delle finestre e dei menu a queste associati.

3.2 Sintassi

Un primo tentativo di trovare delle similitudini tra Smalltalk ed altri linguaggi sarebbe tentare di classificarlo come linguaggio procedurale o funzionale. In realtà la cosa non sembra possibile, in quanto sono presenti caratteristiche dell'una e dell'altra categoria. Infatti se da una parte il meccanismo "esecuzione di metodo - ritorno di un valore" può assomigliare al "calcolo di una funzione - ritorno del risultato finale", dall'altra la presenza dell'istruzione di assegnamento impedisce una definizione del linguaggio come funzionale puro (*).

Inoltre, il lancio di un messaggio, almeno a livello puramente formale, assomiglia abbastanza ad un richiamo di procedura. A sua volta l'assegnamento, che è una delle poche caratteristiche built-in del linguaggio, ha un comportamento molto diverso da un assegnamento C o Pascal.

(*) Il riferimento è al linguaggio FP definito da Backus

L'espressione assegnata, a meno che si tratti di un literal, richiede l'invio di un messaggio, eventualmente con il passaggio di parametri. Il contesto in cui si trova l'esecuzione, a quel punto, diventa quello del metodo corrispondente da eseguire, e quindi ha un set di class e instance variables che può essere completamente diverso da quello di partenza. Non esistono concetti di variabili globali al programma e locali ad una procedura (C e Pascal) o globali e basta (Fortran o Cobol): l'ambiente di esecuzione è determinato volta per volta dalla gerarchia delle classi e dagli eventuali parametri passati. Quindi ciò che sintatticamente sembra un semplice richiamo di procedura assomiglia in realtà molto più ad un calcolo di funzione (dato anche che un metodo ritorna un unico valore, al termine della sua esecuzione). Ma neanche questo è esatto, in quanto sono possibili effetti collaterali dovuti a modifiche sui parametri e sulle variabili globali o di classe.

Un punto di contatto tra lo Smalltalk ed altri linguaggi può essere la possibilità di definire delle variabili temporanee, all'interno delle procedure e dei metodi, che vivono soltanto durante l'attivazione del pezzo di codice in questione. In questo senso il C è più potente, in quanto permette questo meccanismo anche nel caso dei blocchi.

3.3 Blocchi

I blocchi rappresentano uno degli elementi più interessanti da questo punto di vista. È stato già accennato il loro uso per il trattamento di eccezioni; esistono vari messaggi del tipo:

```
aCollection find: anElement ifAbsent: aBlock
```

in cui per esempio si invia all'oggetto di nome aCollection il messaggio find: ifAbsent: allo scopo di ottenere

- se anElement c'è, la posizione di anElement in aCollection;
- se anElement non c'è, l'esecuzione di aBlock.

aBlock a sua volta sarà qualcosa del tipo

```
[^ self error: 'Element not found']
```

dove error: è un metodo definito sotto la classe Object, ed apre una finestra con un elenco degli ultimi messaggi lanciati, ed un messaggio di diagnostica dato dalla stringa presa come parametro (in questo caso, 'Element not found').
Questo passaggio di aBlock può attraversare anche più invii di messaggi: sarà la prima situazione di errore a bloccare il lavoro e a provocare l'esecuzione di aBlock.
Questo è un meccanismo molto più semplice di quelli previsti da certi linguaggi evoluti come CLU e Ada, ma è ugualmente potente.

Un'altra particolarità dei blocchi è il loro uso nelle strutture di controllo. Come già visto, di fatto lo Smalltalk non contiene vere e proprie strutture del tipo "if" e "while" dei linguaggi tradizionali.
Prima di parlare della struttura di selezione condizionale, è necessario fare alcune considerazioni sulle pseudovariabili "true" e "false". Benchè apparentemente uguali a ciò che si trovano ad esempio nel Pascal, esse, per uniformità con il resto del sistema, sono trattate in modo molto particolare. Esse sono gli unici possibili membri delle due classi True e False, rispettivamente, dove True e False sono sottoclassi di Boolean. Evidentemente non ha senso creare altri membri di tali classi, ed infatti il sistema lo vieta (anche se il programmatore malaccorto può modificare questa situazione). In queste due classi è definito il metodo ifTrue: ifFalse:, di cui si è già parlato, che riceve come parametri due blocchi: il primo che va eseguito se la condizione valutata ha ritornato "true", e il secondo che va eseguito se ha ritornato "false". Quindi nel primo caso viene eseguito il metodo definito nella classe True (in quanto "true" è membro di tale classe), che semplicemente contiene:

```
ifTrue: trueBlock ifFalse: falseBlock "nome del metodo"
    ^trueBlock value      "esegue il primo blocco"
```

Nel secondo caso viene eseguito il metodo definito nella classe False:

```
ifTrue: trueBlock ifFalse: falseBlock "nome del metodo"
    ^falseBlock value     "esegue il secondo blocco"
```

Questo è sicuramente un modo atipico (e molto geniale) di implementare questa scrittura di controllo: si tratta sempre di un lancio di metodi ad oggetti, anche se in questo caso si tratta di oggetti particolari, cioè di pseudo-variabili, generati da espressioni booleane.
Un discorso analogo vale per la struttura di iterazione; in questo caso il ricevitore del messaggio whileTrue: è un blocco, ed è un blocco anche il parametro passato.
L'implementazione di questo metodo è ancora più interessante in quanto sfrutta la ricorsione:

```
whileTrue: aBlock   "nome del metodo e parametro"
    self value                "il ricevitore, eseguito,
                              deve restituire un booleano"
        ifTrue: [aBlock value.
            self whileTrue: aBlock].
nil                           "non deve ritornare nulla"
```

Questo conclude la descrizione dei blocchi come oggetti del linguaggio, e come importanti strumenti di strutturazione.

3.4 Strutture dati

Alcune considerazioni meritano la possibilità e le modalità di gestione delle strutture dati in Smalltalk.
Probabilmente in questo senso è possibile trovare i maggiori punti di connessione con altri linguaggi. Il vincolo di tipo delle variabili è ovviamente dinamico, cioè deciso durante l'esecuzione. Questa situazione è facilmente gestita dal linguaggio in quanto ogni nome di variabile rappresenta fisicamente un puntatore (anche se il programmatore non gestisce esplicitamente puntatori) alla zona di memoria dove l'oggetto è memorizzato. È stato già illustrato come di fatto le classi, per quando riguarda la descrizione di un tipo, svolgano una funzione analoga a quella dei record Pascal o delle strutture C. I campi del record corrispondono alle instance variables della classe, ed il nome della classe al nome dei tipo. In realtà le classi forniscono potenzialità che vanno al di là anche dal fatto di poter definire delle operazioni direttamente associate al tipo. Queste derivano dall'esistenza delle variabili globali, di classe e di pool, che sono oggetti che vivono ed esistono con il sistema (sono cioè oggetti permanenti), e le cui modifiche apportate da un qualunque oggetto che vi abbia accesso, hanno effetto immediato e permanente, cioè sono visibili a tutto il sistema (o meglio, a tutti i componenti del sistema che "vedono" tali oggetti).

Perciò ad esempio se una classe ha bisogno che tutti i suoi membri abbiano accesso a una qualche struttura dati permanente, tale struttura può essere creata come class variable. Se i membri di una classe sono strumenti di prenotazione posti per un volo, ognuno di essi deve avere possibilità di accedere al database delle prenotazioni, che deve essere comune e non deve morire alla morte dei membri suddetti. D'altra parte, altri oggetti del sistema non devono accedere a tale database, e infatti ciò è vietato dalle regole di visibilità delle class variables.
In questo senso le classi appaiono come potenti strumenti di organizzazione e modularizzazione del sistema e delle applicazioni.

4. CARATTERISTICHE DI POLIMORFISMO DELLO SMALLTALK

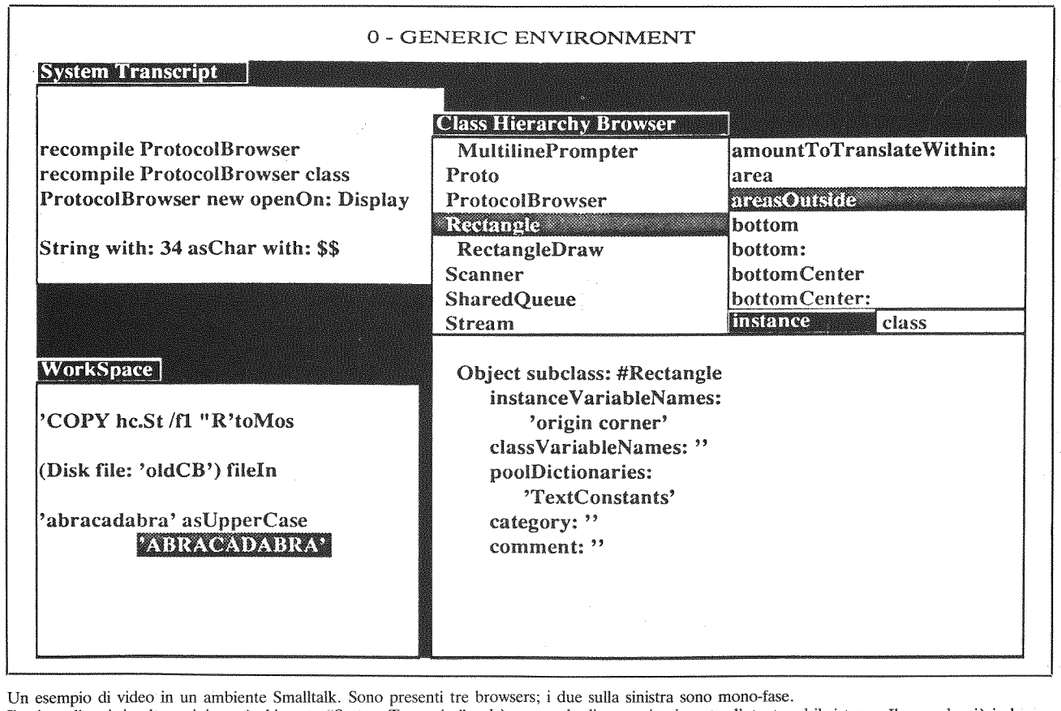
Con il termine polimorfismo, riferito ad un linguaggio, intendiamo la sua capacità di mettere a disposizione del programmatore una struttura uniforme ed allo stesso tempo flessibile e dinamica. Da questo punto di vista lo Smalltalk presenta alcuni aspetti interessanti. Una prima caratteristica rilevante è quella della **ortogonalità** delle funzionalità del linguaggio.

È stata già descritta l'utilità dei blocchi per il trattamento di eccezioni. Il fatto che questo costrutto linguistico, oltre ad essere uno strumento di incapsulamento di pezzi di codice, sia anche oggetto del sistema uguale a tutti gli altri, e quindi ad esempio passabile come parametro ed assegnabile ad una variabile, è un fatto difficilmente ritrovabile in altri linguaggi. Inoltre un blocco passato come parametro porta con sè il contesto in cui è stato costruito, e quindi una sua successiva attivazione riferisce a tale contesto (non a quello in cui viene attivato: questa è una caratteristica molto particolare ed importante del linguaggio).

Lo stesso può essere detto delle **classi,** la cui funzione non si limita ad un comportamento "passivo" di descrizione di un modello, ma può essere usata in modo "attivo" durante l'elaborazione per ottenere informazioni sulla classe stessa. Naturalmente valgono i discorsi di utilizzo completamente libero, fatti per i blocchi. Anche per i valori ritornati dai metodi valgono queste regole: possono essere ritornati numeri interi così come delle complesse strutture dati, senza differenze di trattamento. Lo stesso vale anche per gli argomenti passati ad un metodo.

È importante notare anche come di fatto il linguaggio abbia pochissime **strutture predefinite:** praticamente sono limitate all'assegnamento ed al ritorno di valore finale. Le strutture di selezione e di iterazione vengono implementate anch'esse come normale lancio di messaggi, e compilatore ed interprete infatti le trattano allo stesso modo. Altre strutture come ad esempio il "case" del Pascal possono quindi essere aggiunte molto facilmente, a patto di fissare una convenzione di chiamata compatibile con la sintassi oggetto/messaggio del linguaggio.

Un altro argomento interessante dello Smalltalk riguarda l'**allocazione dinamica** delle variabili.
Fisicamente, ogni nome di variabile rappresenta un puntatore alla zona di memoria dove l'oggetto corrispondente è allocato. Ciò nonostante, questi puntatori sono del tutto trasparenti all'utente, sia in fase di creazione sia, soprattutto, in fase di eliminazione delle strutture non più usate. Esisterà in generale un algoritmo di garbage collection che si preoccupa di liberare le aree inutilizzate, e quindi la gestione delle aree dati è del tutto sicura [17]. Questo tipo di manipolazione degli oggetti si appoggia anche sul legame dinamico esistente tra una variabile e il suo tipo. Il controllo di tipo eseguito in fase di esecuzione permette, come precisato anche in altri punti dell'articolo, una maggiore libertà di programmazione ed un migliore appoggio sul sistema object-oriented.



Un esempio di video in un ambiente Smalltalk. Sono presenti tre browsers; i due sulla sinistra sono mono-fase.
Il primo di essi, in alto a sinistra, è chiamato "System Transcript", ed è un canale di comunicazione tra l'utente ed il sistema. Il secondo più in basso, è un "WorkSpace" dove l'utente può lanciare messaggi, eseguire le proprie applicazioni, eccetera. Sulla destra, più grande e multi-finestra, è raffigurato il "Class Hierarchy Browser", che è costituito da una finestra con l'elenco delle classi definite (l'utente in questo caso ha selezionato la classe Rectangle), una finestra con l'elenco dei metodi definiti per la classe selezionata (l'utente è posizionato sul metodo areas Outside), ed una finestra inferiore contenente la descrizione della classe prescelta (come in questo caso) oppure il testo sorgente del metodo selezionato. I nomi di questi browser derivano direttamente da quelli citati in [13].

4. BIBLIOGRAFIA

[1] Balzer R., Cheatham T.E., Green C., SOFTWARE TECHNOLOGY IN THE 1990'S: USING A NEW PARADIGM, Proc. of the Computer Data Engineering Conference, Capri, May-June 1984

[2] Balzer R., Goldman N., Neches B., SPECIFICATION-BASED COMPUTING ENVIRONMENTS FOR INFORMATION MANAGEMENT, Proc. of the Computer Data Engineering Conference, Capri, May-June 1984

[3] Bandini S., I SISTEMI ESPERTI, Note di Software 23/24, Giugno 84

[4] Barstow D.R., Shrobe H.E., Sandewall E., INTERACTIVE PROGRAMMING ENVIRONMENTS, McGraw-Hill Book Company, 1984

[5] Buzzard G.D., Mudge I.N., OBJECT-BASED COMPUTING AND THE ADA PROGRAMMING LANGUAGE, IEEE Computers, march 1985

[6] Byte, August 1981

[7] Cox B.J., MESSAGE/OBJECT PROGRAMMING: AN EVOLUTIONARY CHANGE IN PROGRAMMING TECHNOLOGY, IEEE Software, January 1984

[8] Deteil H.M., AN INTRODUCTION TO OPERATING SYSTEMS, Addison Wesley, 1983

[9] Dijstra E.W., A DISCIPLINE OF PROGRAMMING, Prentice-Hall, Englewood Cliffs, N.J., 1976

[10] Fabry R.S., CAPABILITY-BASED ADDRESSING, CAMC, July 74, Vol. 17, No. 7

[11] Feather, M.S. SPECIFICATION AND TRANSFORMATION: AUTOMATED IMPLEMENTATION, Proc. of the Computer Data Enginering Conference, Capri, May-June 1984

[12] Ghezzi C., Jazayeri M., PROGRAMMING LANGUAGE CONCEPTS, J. Wiley & sons, N.Y.

[13] Goldberg A., Robson D., SMALLTALK-80, THE LANGUAGE AND ITS IMPLEMENTATION, Addison Wesley, 1983

[14] Goldberg A., SMALLTALK-80. THE INTERACTIVE PROGRAMMING ENVIRONMENT, Addison-Wesley, 1984

[15] Krasner G., SMALLTALK-80. BITS OF HISTORY, WORDS OF ADVICE, Addison-Wesley, 1983

[16] Leclerc Y., Zucker S.W., Leclerc D., A BROWSING APPROACH TO DOCUMENTATION, IEEE Computer, June 1982

[17] Lomet D.B., MAKING POINTER SAFE IN SYSTEM PROGRAMMING LANGUAGES, IEEE Trans. on Software Eng., January 1985

[18] Snodgrass R., AN OBJECT-ORIENTED COMMAND LANGUAGE, IEEE Trans. on Software Eng., January 1983

[19] Stroustrup B., DATA ABSTRACTION IN C, AT&T Belle Labs Technical Journal. Vol. 63 No. 8, October 1984

[20] Webster B., SMALLTALK COMES TO THE MICROCOMPUTER WORLD, Byte, May 1985

CONCEPT BROWSER:
A SYSTEM FOR INTERACTIVE CREATION OF DYNAMIC DOCUMENTATION

Ugo Corda

Ing. C. Olivetti & C. S.p.A. - Divisione Sviluppo Sistemi

Giuseppe Facchetti

Istituto di Cibernetica - Università di Milano

RIASSUNTO

In questo articolo viene descritto un sistema interattivo per la creazione e la consultazione di documentazione.
Lo strumento in questione, il Concept Browser, permette all'utente di creare un insieme di concetti correlati, organizzandoli secondo una struttura a rete semantica.
È possibile poi esplorare la rete così costruita, avvalendosi degli strumenti messi a disposizione dal Concept Browser stesso. Sono disponibili anche un comando di visualizzazione della struttura del documento "dinamico" che si sta esplorando, ed un comando di stampa automatica di esso o di un suo sottoinsieme.
Il Concept Browser è stato realizzato in un ambiente Smalltalk sviluppato in Olivetti, e si avvale anche di una interfaccia basata su finestre e menu.
Il Concept Browser può trovare applicazioni in svariati campi: dai sistemi per la preparazione di documenti, ai libri elettronici, alla documentazione in linea.

ABSTRACT

A system for interactive creation and browsing of dynamic documents is described.
The Concept Browser allows the user to create a semantic network of interrelated concepts and interactively navigate through the network. Outlines and printed documents can be automatically generated from the network of concepts.
The Concept Browser has been designed and implemented in a Smalltalk programming environment. An interactive, window based user interface is provided that allows the user to browse and modify the network of concepts.
Possible applications for the Concept Browser are in the areas of on line documentation, tutorial systems, document preparation systems and electronic books.

1. INTRODUCTION

The traditional method of storing information in a printed, linear form as it is done with conventional books has been demonstrated to be inadequate both to represent the complexity of information and to offer quick and flexible access to it (see [1]).

The personal computer appears to be the ideal tool to satisfy these requirements, but a simple computer-based transcription of traditional books is not the best way of taking advantage of the new functionalities offered by computers.

The purpose of this article is to describe one experiment in the design of a documentation system that can take advantage of the flexible data structures and advanced user interface provided by a Smalltalk programming environment.

A semantic network was chosen as the best way of representing the complexity of information (see Ref. [2, 6]) instead of a more traditional tree structure (see Ref. [5]).

The way semantic networks are used in the Concept Browser is different from the one adopted by Artificial Intelligence systems (see [3]). The purpose of the semantic networks of the Concept Browser is to amplify the user intelligence, not to substitute computer intelligence for human intelligence (see [4]). Therefore, processing of the semantic network is done by the user himself. The main role played by the computer is in providing a user friendly, interactive interface that can be easily learnt and used.

2. THE USER INTERFACE

Since the system has been implemented in a Smalltalk programming environment, the general ideas about the user interface and the basic tools used to implement it are directly derived from the Smalltalk user interface itself.

The Concept Browser user interface consists of two different types of browsers (a browser is a multiwindow structure intended to give visibility of different pieces of interrelated information).
The first type of browser is described in Figure 1.

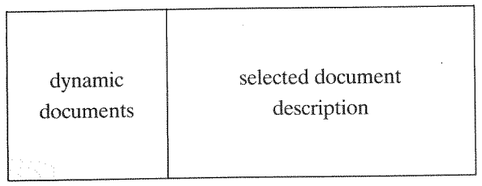


Figure 1 The list of documents

It is composed of two windows. The one on the left contains the list of all the different dynamic documents available in the system. If one entry in the list is selected, a short description of the contents of the corresponding dynamic document is dysplayed in the window on the right. A pop up menu provides commands for defining new dynamic documents and for removing existing ones.

An additional menu command opens a second type of browser on a previously selected dynamic doculent. This is the main dynamic document interface that allows the user to enter the concepts that constitute the document and to browse through them. This browser is described in Figure 2.

The window on the top contains the name of the current concept. This is the concept that is currently being created or examined; it belongs to the network of concepts which constitutes the dynamic document. The text associated with this concept appears in the window just below.

The window labelled "link types" contains the list of the different types of links that connect the current concept to other concepts in the network. Possible link types are "related concepts", "dependent concepts", "parent concepts", etc. (the user can decide wich link types he wants to define and use). When one of these link types is selected, a list of the network concepts connected to the current concept through the selected link is presented in the "linked concepts" window.

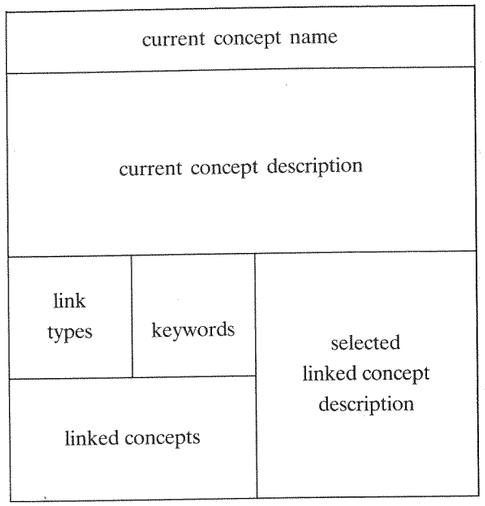


Figure 2 The actual Concept Browser

The "keywords" window contains a list of keywords associated with the current concept. If one of these keywords is selected, the list of all the other concepts that share the same keyword appears in the "linked concepts" window.

Finally, selecting a concept in the "linked concepts" window causes the text associated with this concept to be displayed in the window on the right.

3. CREATING AND MODIFYING THE NETWORK OF CONCEPTS

The Concept Browser user interface provides a set of interactive tools that help the user in the process of defining the concepts that the dynamic document is composed of and establishing different types of relationships between these concepts.

From this point of view, the Concept Browser can be seen as a generalized type of editor that is capable of handling a semantic network of concepts.

3.1 How to create a new concept

New concepts can be inserted in the network in two different ways.

The name of the concept may be entered in the top window and then a description of it may follow in the window below.

A new concept can also be created using the "add" command that appears in a pop up menu associated

with the "linked concepts" window. A prompter (a pop up window with a prompt message and a subwindow to hold the reply text) asks the name of the concept. The new concept is created and linked to the top window through the link type currently selected in the "link types" window. A description of the new concept may then be entered in the "selected linked concept description" window.

Once a concept has been created, its name can be changed selecting a menu command provided by the top window. Also, the concept description can be edited using the window editor inherited from the Smalltalk environment. The modified text is then automatically reformatted within the window where it is displayed.

3.2 How to add and remove links

A new link between two concepts can be added at any time. In order to do that the user must set the current concept (the concept defined in the top window) to be concept that the link comes from (links have a direction associated with them), and select in the "link types" window the link type that is to be assigned to the new link. Then the "add" command should be activated from the pop up menu appearing in the "linked concepts" window. A prompter is displayed that asks the name of the second concept. After this information has been provided, the new link is built and the second concept appears in the list of concepts linked to the current concept.

For some link types that are supposed to be frequently used a second link is automatically created in the opposite direction (for instance. if the link type is "dependent concepts". a link of type "parent concepts" is also created that links the second concept to the first one). The user is asked to confirm the creation of this second link.

Existing links can also be removed in a manner similar to creating a new link. Instead of selecting "add", the "remove" command from the pop up menu in the "linked concepts" window is selected. Once the link is removed, the second concept is also removed from the list of concepts linked to the current concept.

3.3 Defining keywords

A list of keywords can be associated with any concept. The concept must be selected as the current one. Then the "add" command must be activated using the pop up menu defined for the "keywords" window. The user is prompted for the new keyword name. Following the user response, the "keywords" window is redisplayed containing the newly added keyword.

An existing keyword can also be removed from a concept using the "remove" entry of the pop up menu.,

3.4 Additional network mlodification tools

To help with some complex network operations such as splitting or joining two concepts, a command associated with the top window menu creates a new window that contains a list of all the links entering the current concept together with the concept each link comes from.

This information, together with the list of the outward links provided by the "linked concepts" window, gives a complete representation of the environment that exists around the current concept. The user can then decide whether these links are to be removed or instead redirected towards different concepts.

4. NAVIGATING THROUGH THE NETWORK OF CONCEPTS

In a data structure as generalized as a network can be, there is no logically predefined path the user has to follow in order to extract the information he is interested in (think on the opposite case, the purely linear approach provided by a traditional book).

Browsing in a network is more like exploring an unknown geographical region, trying to follow the most promising routes. Concept names and descriptions, links and keywords provide the indications the user can consult during his navigation.

4.1 How to find an entry point

Finding an initial concept from which to begin browsing might be a non-trivial task, unless the user already knows the name of the concept he is interested in (in this case, typing the name of the concept in the top window is enough to establish this concept as the starting point).

To solve this information retrieval problem, different strategies can be adopted, each one supported by a particular Concept Browser feature.

The user might try to look for a concept whose name is similar to the one he has in mind. To do this, the special concept called "Concepts" can be set as the current concept in the top window. Selecting the link type "dependent concepts", an alphabetically sorted list of all the concepts defined in the network appears in the "linked concepts" window. The user can then look at this list for any name that might give him an hint, or he can select the "search" command, provided by the pop up menu, that performs a search through the list looking for a specified string or substring (this method is similar to looking at the table of contents of a book).

A different approach is to use a keyword to identify one or more concepts which share that keyword. In a way similar to the one described before, selecting the special concept "Concepts" causes the list of all the keywords defined in the network to be displayed in the "keywords" window. Again, the "search" command can be used to find an entire keyword or a keyword substring. Once a good keyword is identified, a list of all the concepts that declare this keyword can be displayed in the "linked concepts" window (this approach is similar to looking at the subject index of a book).

If also this method fails, a full text search can be requested on all the concept texts contained in the network. In this way, a group of concepts is found that might have something to do with the subject the user is interested in.

4.2 Following the links

Given a certain concept as the current one (the one defined in the top window), the user can explore the network area that is close to it. To do so, all the user has to do is to follow the links. For instance, selecting the link type "related concepts" gives a list of all the concepts that are linked to the current concept through that type of link and are therefore likely to provide information closely related to the current concept. Selecting the link type "dependent concepts" returns a list of concepts that are intended to give more details about the subject the current concept talks about. And so on, with any other link type the user has defined.

Once this list of concepts logically related to the current one is identified, the description of one particular concept in the list can be visualized in the "selected linked concept description" window. If any of these concepts looks interesting enough to deserve further examination, it can be moved to the top window and made the current concept (use the "browse" command from the pop up menu of the "linked concepts" window). All the windows of the Concept Browser are updated to reflect the status of the new current concept. This action corresponds to a move in the network of concepts.

At this point, the entire process can be repeated with the new current concept.

4.3 Keywords as special link types

For any concept defined in the top window (the current concept), the "keywords" window provide a list of the keywords associated with this concept. If one of these keywords is selected, a list of all the concepts that share the same keyword is displayed in the "linked concepts" window.

Keywords can then be thought of as implicit links connecting concepts together and therefore they can be used to move around through the network of concepts.

Once a list of concepts has been derived in the "linked concepts" window via a keyword selection, the mechanism to make any of these concepts the current one is identical to the one used with explicit links.

4.4 Moving back and forth along a path

The user can decide that the path he is following is not fruitful and he wants to back up to some concept the has already examined before. Or he just wants to look again at some previously selected concept before continuing browsing new concepts.

The menu commands "next", "previous" and "show path" help the user with this kind of requests, "previous" and "next" allow the user to move back and forth along the path already followed. "show path" displays a list of the concepts in the path along with the link type or keyword that determined each single step; selecting an item in the list makes this concept the current one.

A path can be saved and then restored later on, so that it can be followed again.

4.5 Opening multiple browsers on the same dynamic document

The Concept Browser allows the user to open more than one browser on the same dynamic document, offering in this way simultaneous views on different parts of the document. With this mechanism the reader can pursue several lines of thought or view a given object at several levels of detail.

5. CREATING A TRADITIONAL DOCUMENT

The Concept Browser can also be used to create more traditional types of documents, that is documents that are mainly used in printed from (articles, books. etc.).

Even though this kind of document only requires a linear browsing through the static printed representation, the use of the Concept Browser encourages a modular approach to the document preparation phase. The final document is obtained through the composition of self consistent single modules that represent chapters or sections. In this way better control can be reached over the process of "debugging" the single pieces and of defining the overall organization of the entire document.

5.1 Defining chapters and sections

Due to the restricted type of browsing required by a traditional document. a full network structure is not

necessary to link concepts (chapters, sections, etc.) together: a tree will do it. The general mechanisms used to create the network are still valid. Only the link types used must be chosen in such a way that a parent-child relationship is established between concepts (for example, the link type "dependent concepts" might be used).

This way, a concept defining a document chapter may be created and then linked through a "dependent concepts" link to a series of concepts representing sections of that chapter. These dependent concepts can be ordered in a particular sequence chosen by the user; this same sequence will be used when linearizing the tree into a printed document.

Even though a static kind of document is being created, the process of preparing the document can still be highly dynamic. A framework of chapters and sections can be defined before text has been entered for them (top down approach). This framework can easily be restructured during document composition. Also, parts of the doument can be written before knowing exactly where they are going to be placed in the document; adding a new link will suffice to insert the new part in a specific position (bottom up approach).
To give the user an idea of what the document will look like in its printed from, the "outline" command is provided that opens a new window containing the document outline, with chapters and sections indented and numbered.

5.2 Producing a printed document

Once the document is complete, a printed representation of the document itself can be produced using the "print doc" command in the top window pop up menu.

This utility does a walk of the tree that represents the document, starting from the current node (usually the top node) and going down the hierarchy. Each concept name appears as a chapter or section name, followed by the associated text and then by sequence of concepts directly dependent on that concept (it is a depth first tree walk).

By default the link type "dependent concepts" is used to determine the tree structure, but the user can specify other types of links. This way, a tree structure can also be extracted from a dynamic document organized as a more general network structure. Also, several different printed documents can be obtained from the same dynamic document starting from different nodes and following different links.

5.3 Making a dynamic document out of a traditional one

The tree representing a traditional document can always be augmented with non-hierarchical link types and transformed into a dynamic document. As an example, consider this article which has been prepared using the Concept Browser.
The same information it contains could be used to provide online help to the user of the Concept Browser itself. If we now add new link types like "related concepts" and a series of appropriate keywords to the article, we get a dynamic document to which browsing strategies other than the purely linear one suggested by the printed form can be applied (for instance, looking only at the concepts that deal with a specific subject like the user interface).

6. ADDITIONAL APPLICATIONS

Due to the characteristics of dynamic browsing and user friendly, interactive interface, the Concept Browser can be used to implement an electronic book or to provide on-line documentation.

Also, since a browsing path can be saved and then "played back" again, the user can create a particular path which could be used later as a tutorial about a subject contained in the dynamic document. The student would only have to load the path and follow it using the commands "next", "previous" and "show path".

Finally, the Concept Browser could also be used to create and edit generic semantic networks.

7. IMPLEMENTATION

Four new Smalltalk classes have been defined to implement the Concept Browser.

The class "Concept" generates the concepts of the network, with instance variables for the concept name, concept text, list of keywords and list of links (the concept text is actually a pointer to a location within a text file).

The class "ConceptLink" generates instances that group together links of the same type originating from one concept. Its instance variables are the link type and an OrderedCollection of Concepts.

Another two classes handle the user interface.

The data base associated with a particular dynamic document consists of two Smalltalk dictionaries.

One of them contains entries of type "conceptName ⇒ Concept"; the other one has entries of type "Keyword ⇒ Set of Concepts". The data base can be unloaded to a file.

8. CONCLUSION

The Concept Browser appears to be a flexible tool to organize information in a non traditional way.

Instead of using a hierarchical approach, a more generalized network data structure represents the connections between different pieces of information.

The window based user interface and the path mechanism seem to be adeguate tools for the user to traverse the data structure.

Further study is required to understand what kind of discipline should be imposed on the creator of the network. In fact, proper choice of link types and keywords together with a good partitioning of the entire information into manageable pieces might greatly affect the browsing process.

9. REFERENCES

[1] S.A. Weyer, SEARCHING INFORMATION IN A DYNAMIC BOOK, Xerox PARC Report number SCG-82-1.

[2] D.E. Knuth, LITERATE PROGRAMMING, The Computer Journal, vol. 27, no. 2, 1984.

[3] A. Barr, E.A. Feigenbaum (Eds), THE HANDBOOK OF ARTIFICIAL INTELLIGENCE, 3 voll, William Kaufmann Inc., Los Altos, CA, 1981.

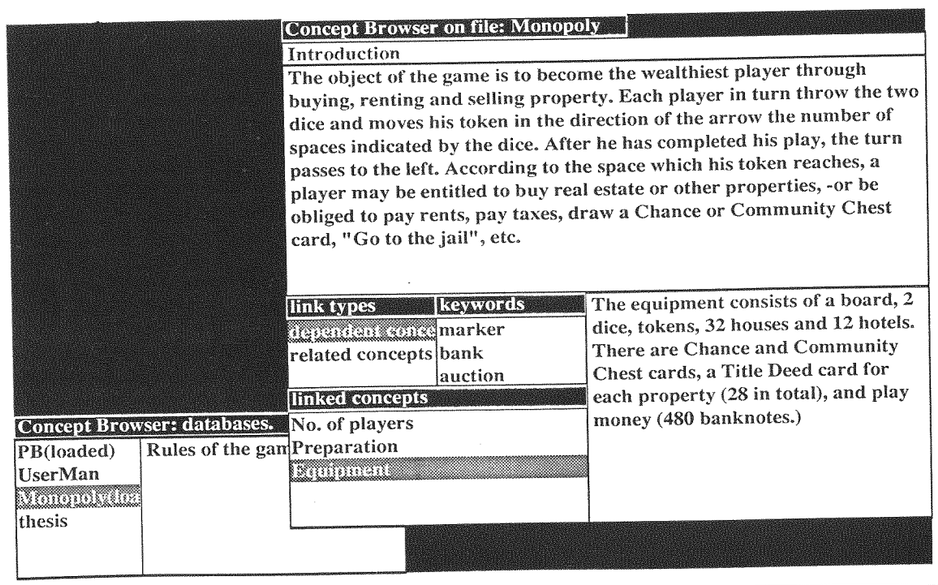
[4] P. Wegner, PERSPECTIVES ON CAPITAL INTENSIVE SOFTWARE TECHNOLOGY, International conference on advanced personal computer technology, Capri 1984.

[5] Y. Leclerc, S.W. Zucker, D. Leclerc, A BROWSING APPROACH TO DOCUMENTATION, Computer, June 1982.

[6] E. Shapiro, TEXT DATABASES, Byte, October 1984.

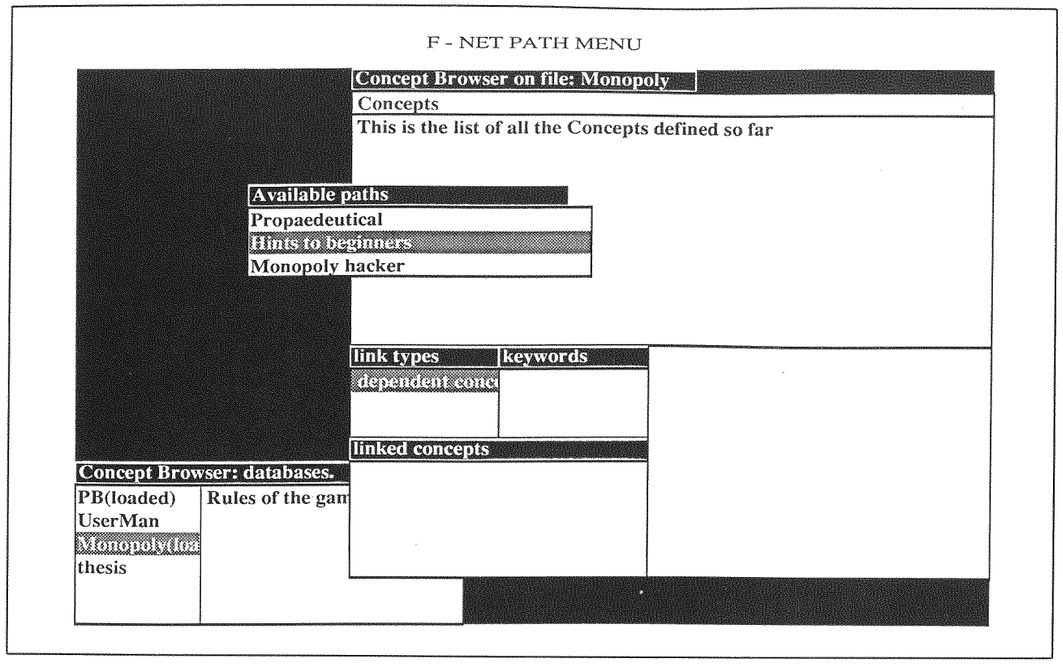
Questo lavoro è stato presentato al Convegno "Text Processing And Document Manipulation" organizzato dalla British Comp. Society a Nottingham nell'aprile 1986.

B - THE INTRODUCTION



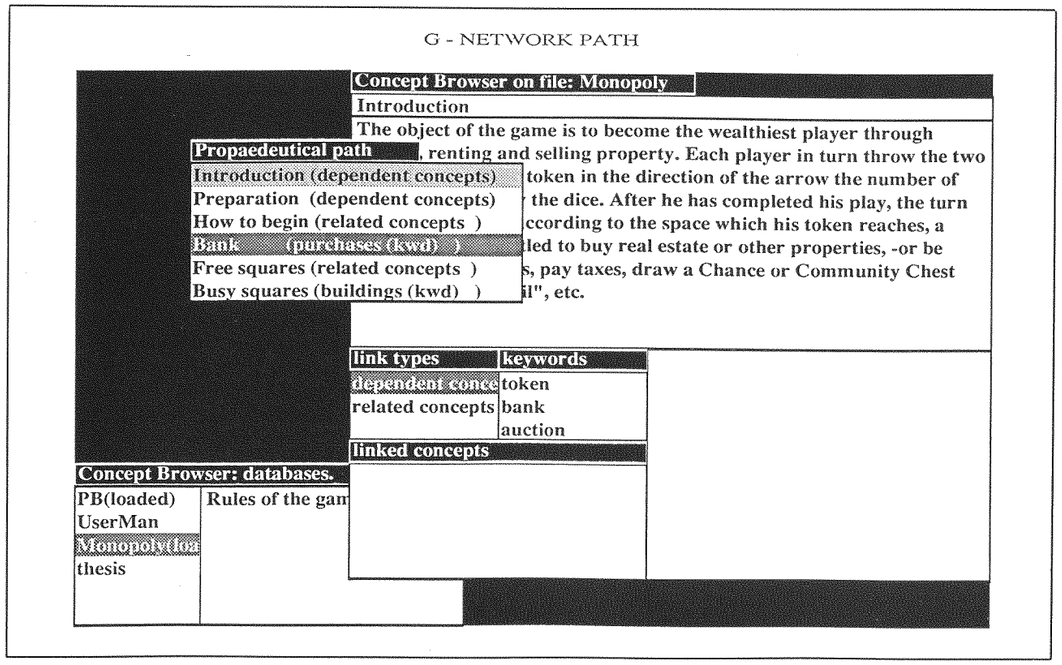
A tipical screen presented by the Concept Browser: the concept base is built using the rules of the game "Monopoly". The current concept is titled "Introduction". The concepts "Equipment" is a "dependent" of "Introduction".

F - NET PATH MENU

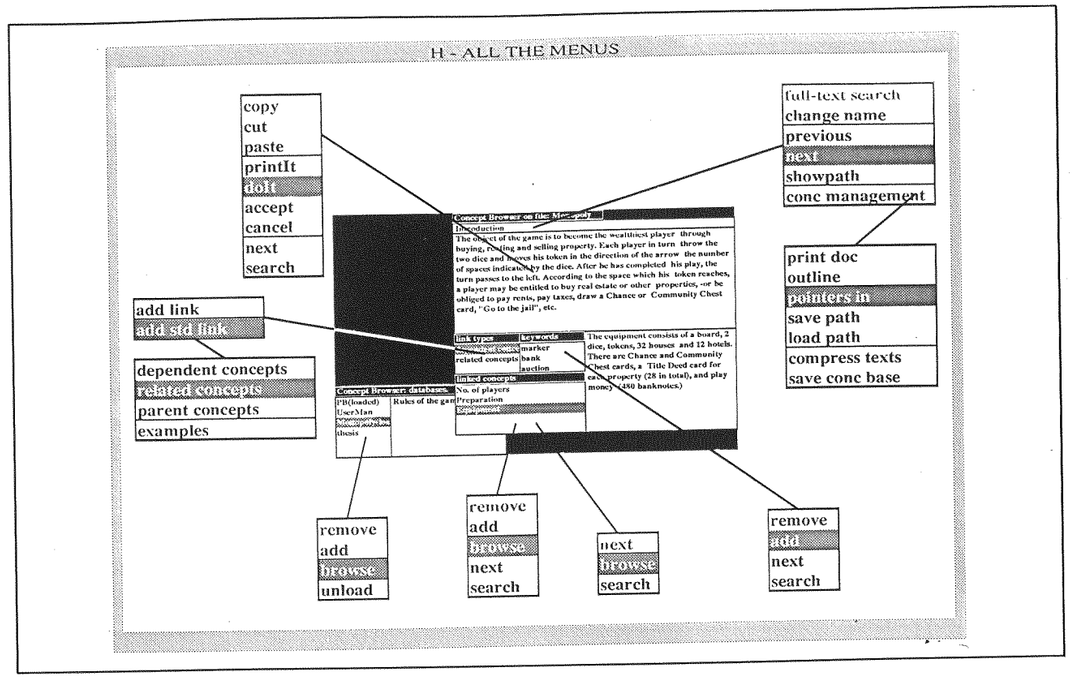


The list of available paths through the Monopoly network is shown in the small, overlapping window.

G - NETWORK PATH



The user has chosen the "Propaedenthical" path: he has been positioned on the first concept of that path, and can visualise and select another one.



The list of the menus defined for all the windows of the Concept Browser.

IL DOSSIER VIRTUALE: SOFTWARE PER FOGLI ELETTRONICI

Massimo Enrico Delpero

Etnoteam SpA - Milano

RIASSUNTO

La presente nota descrive il concetto di dossier virtuale, un nuovo strato di software che si frappone tra l'utente e il foglio elettronico, mediante l'utilizzo delle Macro dello "spread-sheet" LOTUS 123. Viene inoltre presentato un semplice esempio di realizzazione di dossier virtuale, e sono discusse le prospettive che tale concetto induce per la progettazione di applicazioni di office automation.

ABSTRACT

The present paper describes the concept of virtual dossier, a new software layer placed between the end user and the spreadsheet. The virtual dossier is implemented with LOTUS 123' Macros. It is also presented a simple example of virtual dossier realization, and the perspectives that this concept opens for the design of office automation applications are discussed.

1. INTRODUZIONE

Chiunque voglia utilizzare LOTUS 123, o un qualsiasi altro foglio elettronico, avrà a disposizione una certa interfaccia costituita dall'insieme delle funzionalità e dalla struttura del foglio stesso.

Le potenzialità di questo strumento software sono oggi ampiamente conosciute, così come la classe dei problemi e di situazioni che possono essere efficacemente affrontati. LOTUS 123, nel vasto panorama dei fogli elettronici oggi sul mercato, è di particolare interesse, non solo perchè si colloca nella fascia dei cosidetti prodotti integrati, fornendo la possibilità di utilizzo congiunto di un foglio elettronico, della grafica a colori e della gestione d'archivio, ma anche per la possibilità di utilizzo delle Macro.

La presenza delle Macro rivoluziona le possibilità e le modalità di utilizzo del foglio elettronico, fornendo l'opportunità di creare un nuovo ambiente d'utilizzo che sgrava completamente l'utente finale dalla necessità di apprendere l'interfaccia standard del foglio stesso.

In questa nota mi soffermerò sulle potenzialità indotte da questo linguaggio di programmazione a disposizione dell'utente, facendo particolarmente riferimento alla nozione di "dossier virtuale" e quindi alle implicazioni che si inducono nella fase di progettazione di una applicazione per l'office automation.

2. LOTUS 123: LE MACRO

123 consente di memorizzare nello stesso foglio elettronico sequenze di comandi; queste possono essere utilizzate durante la stessa sessione o in una successiva. Una sequenza di comandi memorizzata è chiamata Macro.

In sintesi si può affermare che una Macro costituisce un programma interpretato da 123, ogni sua istruzione viene collocata in una cella del foglio elettronico; le istruzioni da cui è costituita devono essere verticalmente contigue. Il nome mediante il quale la Macro potrà essere richiamata è quello assegnato alla cella che contiene la prima istruzione che deve essere eseguita.

In linea generale (esistono infatti alcune importantissime eccezioni che verranno esaminate in seguito) una Macro è una serie di comandi normali di 123 cui è stato associato un nome.

Seguendo questo concetto un utente potrà creare un programma per il foglio elettronico, che eseguirà automaticamente operazioni complesse e ripetitive richiamabili come un singolo comando. Per esempio, l'assegnazione dei nomi dei mesi ad alcune colonne del foglio è un'operazione frequente che può essere programmata nel seguente modo:

'GEN ~ {right} FEB ~ {right} MAR ~ {right} APR

In questa Macro il simbolo "~" rappresenta la chiave Enter, {right} sta per la freccia orientata a destra. Simboli come questi sono utilizzati per rappresentare i tasti speciali della tastiera di un personal computer IBM.

Questa Macro potrà essere assegnata ad una cella del foglio elettronico e richiamata perchè a quattro celle del foglio orizzontalmente contigue, vengano assegnati i nomi dei primi quattro mesi dell'anno. La cella a partire dalla quale ha inizio l'operazione è quella corrente.
In questo esempio la Macro può essere vista come una rappresentazione alternativa degli stessi comandi che l'utente avrebbe digitato per ottenere il risultato voluto, con l'evidente vantaggio di poter ripetere la stessa operazione un numero di volte illimitato.

Per la ragione ora esposta nel manuale di 123 ci si riferisce alle Macro come ad una "typing alternative"; in realtà le potenzialità delle Macro si estendono di gran lunga se si considerano i comandi richiamabili esclusivamente dal loro interno, cioè non utilizzabili dalla interfaccia standard di LOTUS 123.

Alcuni di questi comandi permettono, per esempio, di avere a disposizionze una struttura di controllo del tipo *if...then...else* (/xi), un'istruzione di salto incondizionato (/xg), un'istruzione per il richiamo di una subroutine e per il ritorno alla procedura chiamante (rispettivamente /xc e /xr) e un'istruzione per la creazione di menu che appaiano durante l'esecuzione di una Macro (/xm). Questi menu sono strutturalmente identici ai menu standard di 123 e permettono all'utente di operare delle scelte durante l'esecuzione delle Macro.

Utilizzando questi comandi, che sicuramente non fanno delle Macro un linguaggio estremamente potente e ben strutturato, è comunque possibile costruire delle vere e proprie applicazioni procedendo in due fasi ben distinte:

1) Progettazione del layout tabellare
2) Progettazione delle Macro

Su un piano distinto possiamo vedere le tabelle come l'implementazione del modello (dovendo identificare certi valori, organizzarli e metterli in relazione secondo opportune leggi), mentre le Macro come le funzionalità del modello.
Le funzionalità possono essere di tipo distinto:

- operativo, che implementano l'insieme delle operazioni che consentono il mantenimento della tabella, le stampe, la visualizzazione grafica, ecc.

- metodologico, che consentono l'utilizzo del modello seguendo un certo metodo. Queste operazioni, opportunamente strutturate su menu a più livelli, guidano per esempio all'introduzione di certi dati, al loro esame, all'analisi di sensitività.

Con questa tecnica di progettazione le Macro consentono l'implementazione di un modello e della metodologia per il suo corretto utilizzo.

Considerando inoltre l'opzione di "autoexecution", che permette l'esecuzione automatica delle Macro all'atto del caricamento del foglio che le contiene, si capisce che un foglio elettronico può essere ora totalmente automatizzato, nel senso che è possibile l'utilizzo di un modello (implementato su foglio elettronico) senza neppure conoscere l'interfaccia standard di 123.

A questo punto è corretto parlare di nuovo ambiente, e cioè quello fornito dall'applicazione e non più dal foglio elettronico. Grazie all'opzione di "autoexecution" si può vedere nelle Macro la possibilità di implementare un tipo di dato astratto:

- le tabelle rappresentano i tipi di dati
- le Macro le uniche operazioni mediante le quali è possibile modificare lo stato delle tabelle. L'autoexecution non permette infatti l'interazione diretta con la tabella, iniziando all'atto del caricamento l'esecuzione delle Macro.

Anche questa osservazione permette di capire che è sostanziale la differenza (e come tale va mantenuta) tra modello e sua metodologia di utilizzo: il modello è finalizzato alla descrizione della realtà ed ha scopi previsionali su di essa; la metodologia invece ad un corretto utilizzo del modello affinchè i risultati a cui si giunge siano inquadrati in un'ottica corretta.
Tale differenza è più visibile all'aumentare della complessità del modello, e cioè delle variabili e delle relazioni da cui è costituito.

3. IL NUOVO AMBIENTE: IL DOSSIER VIRTUALE

Per comprendere le implicazioni che l'utilizzo delle Macro induce da un punto di vista architetturale, è necessario introdurre il concetto di dossier virtuale.

Per l'utente finale un foglio elettronico si configura come una tabella le cui celle possono essere messe in relazione mediante formule; egli ha inoltre a disposizione dei comandi (per es. di copia e di spostamento) che ne consentono la gestione.
In questa prospettiva il dossier virtuale è descrivibile secondo la sua struttura logica e fisica:

- Logicamente, cioè dal punto di vista dell'utente finale, un dossier virtuale è una collezione di moduli, separati o interagenti, cui è associata una copertina e un indice (Fig. 1). Un modulo può essere a sua volta decomposto in un insieme di parti: una tabella a doppia entrata, un grafico che ne rappresenta i valori, un testo che descrive le relazioni (Fig. 2). Non tutti e tre gli elementi dovranno essere necessariamente presenti, o presenti singolarmente. Due o più moduli si dicono interagenti se i dati memorizzati su di uno sono calcolati utilizzando quelli presenti negli altri; due moduli si dicono invece separati se non esiste alcuna interazione tra di loro.

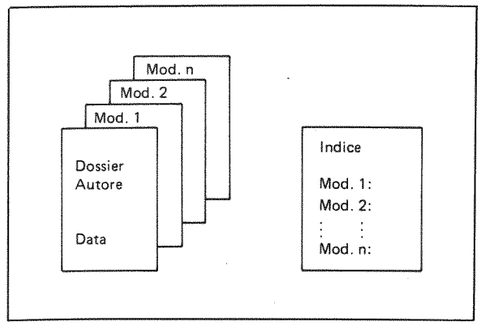


Fig. 1

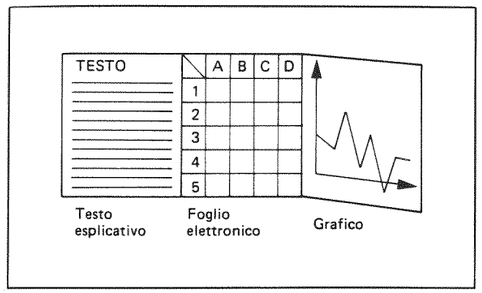


Fig. 2

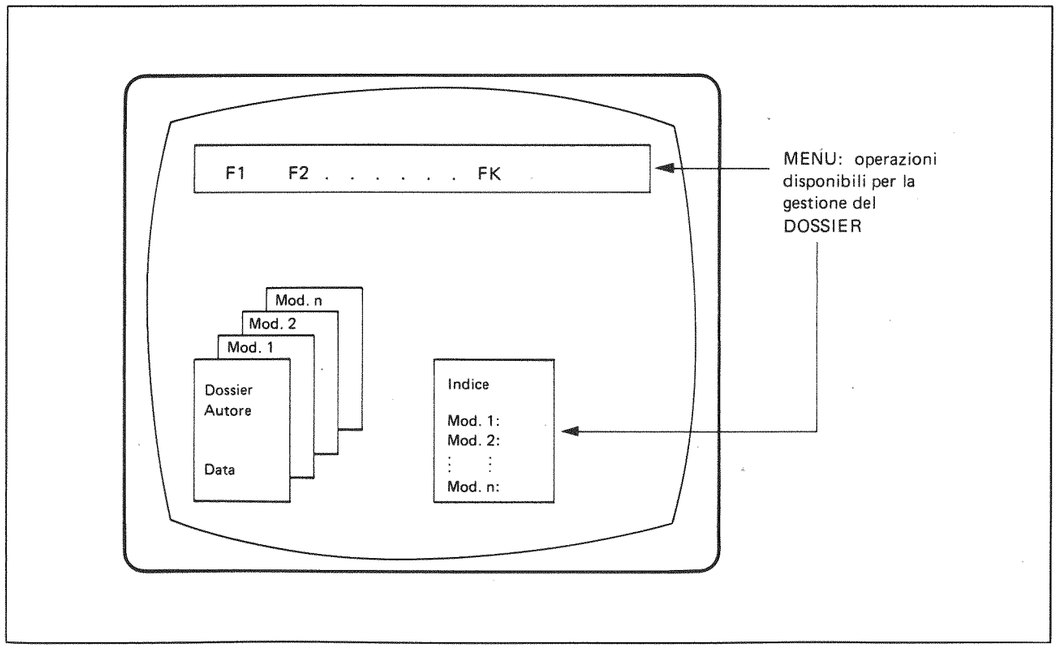


Fig. 3

L'utilizzo del dossier virtuale sarà presentato all'utente via menu; in questo modo, definite la struttura del dossier e le sue funzionalità, l'utente dovrà semplicemente conoscere il significato delle funzioni associate al dossier, e del contenuto dei moduli che lo costituiscono (Fig. 3).

- Da un punto di vista fisico si ha che il dossier virtuale è progettato utilizzando le Macro di LOTUS 123 ed è composto dai seguenti elementi:

  - le sottotabelle del foglio elettronico che rappresentano i moduli, indice e copertina
  - le Macro che implementano le funzionalità.

  È lecito così parlare di rappresentazione fisica del dossier virtuale: i singoli moduli vengono mappati su parti distinte dello stesso foglio elettronico, all'interno del quale saranno anche presenti le Macro che realizzano le operazioni di utilizzo (Fig. 4). Le Macro consentono quindi di passare dalla struttura fisica a quella logica, mascherando completamente l'esistenza della prima; le operazioni di utilizzo del dossier virtuale determinano la struttura logica della tabella e non consentono di vedere come questa sia mappata su quella fisica, agendo come un filtro che si frappone tra utente e foglio elettronico.

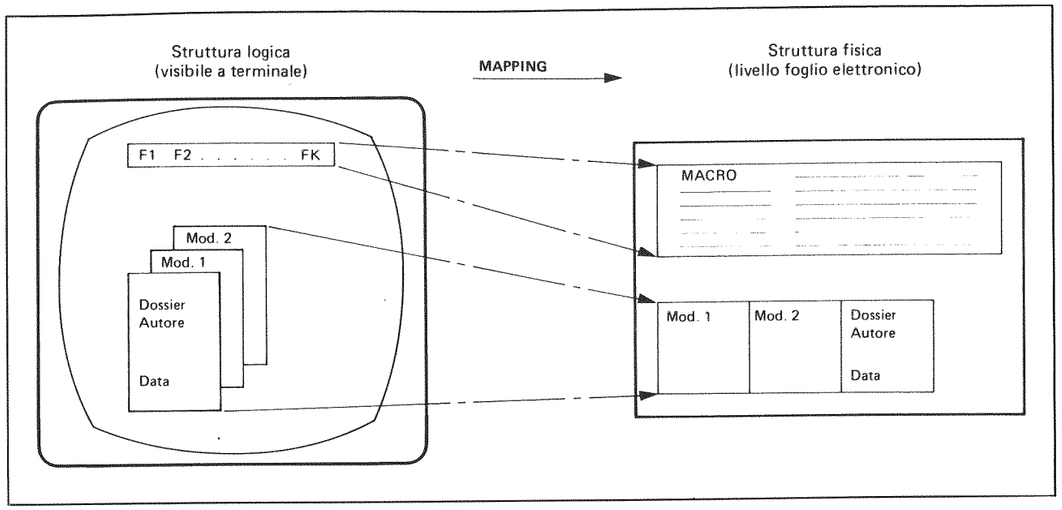


Fig. 4

Da un punto di vista delle prestazioni questo approccio è estremamente conveniente in quanto 123 lavora mantenendo l'intera tabella in memoria centrale: questo consente di passare da un modulo all'altro e di eseguire calcoli relativi a moduli separati o interagenti con estrema velocità.

La nuova prospettiva può essere valutata considerando gli strati che separano l'utente finale dall'hardware della macchina:

– Nel caso del foglio elettronico:

HARDWARE
SISTEMA OPERATIVO
FOGLIO ELETTRONICO
UTENTE

– Nel caso del dossier virtuale:

HARDWARE
SISTEMA OPERATIVO
FOGLIO ELETTRONICO
DOSSIER VIRTUALE
UTENTE

cioè: il dossier crea una nuova interfaccia utente che maschera quella del foglio elettronico.

4. IL DOSSIER VIRTUALE SU PIÙ LIVELLI

Visto che per dossier di grosse dimensioni la memoria centrale funge da limite superiore, si può approcciare il problema dell'implementazione fisica considerando più fogli elettronici di cui, ovviamente, uno solo alla volta risiede in memoria centrale. In questo caso si parla di dossier virtuali disposti su più livelli, cioè i cui moduli risiedono su più fogli elettronici separati.

Obiettivi primari per la progettazione di un dossier su più livelli, pur essendo numerose le tecniche di implementazione che possono essere efficacemente utilizzate, sono i seguenti:

– rendere completamente trasparenti all'utente le operazioni di I/O
– mantenere la stessa interfaccia del dossier virtuale disposto su un solo livello.

Nel rispetto di queste due regole è opportuno privilegiare quelle soluzioni che garantiscono prestazioni il meno penalizzanti possibile; è bene osservare comunque che i tempi di trasferimento, per tabelle di una certa consistenza, possono arrivare molto vicini al minuto.
Quindi il dossier virtuale disposto su più livelli è logicamente un dossier virtuale che sul piano fisico non mantiene in memoria centrale tutti i moduli e le funzionalità dal quale è costituito.

Un esempio è presentato in figura 5: qui si ha un dossier i cui moduli sono distribuiti su due fogli elettronici.
Nel primo sono memorizzati la copertina, un modulo e le Macro che implementano le funzionalità relative al dossier e al caricamento di dati dall'altro foglio, nell'altro i rimanenti tre moduli.

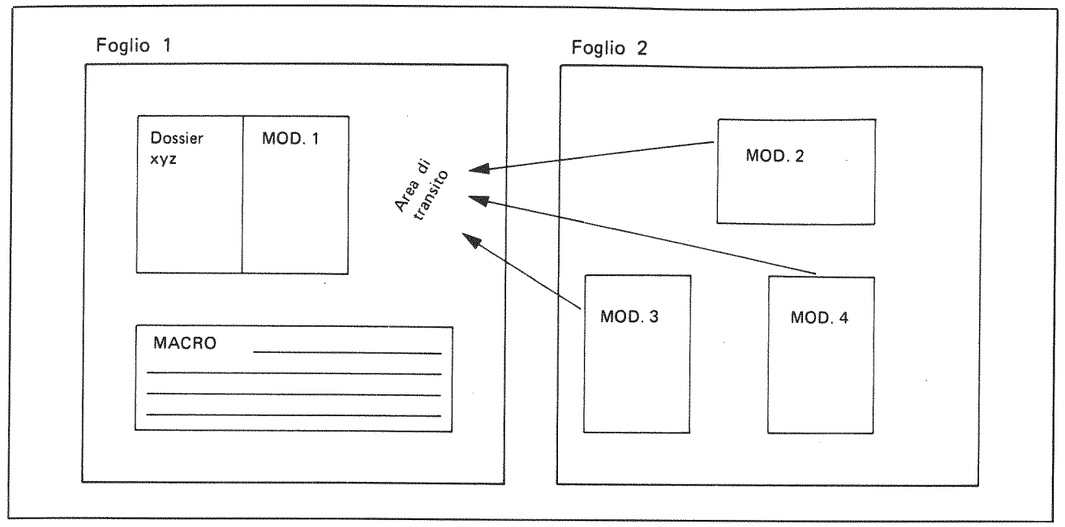


Fig. 5

Con questa soluzione, l'area libera del primo foglio può essere utilizzata come area di transito dati. Quest'area deve avere una dimensione tale da poter contenere il modulo più grosso del secondo foglio; una volta caricato il modulo selezionato questo potrà essere consultato come un qualsiasi altro modulo residente. Terminata la sua consultazione, le Macro dovranno cancellare il contenuto per lasciare l'area libera per una successiva operazione di transito.

Confrontiamo ora i comandi che le Macro implementano per la visione di un modulo rispettivamente residente e non:

- 1) Selezione del modulo
  2) Visione del modulo
  3) Ritorno al menu

- 1) Selezione del modulo
  2) Caricamento nell'area di transito
  3) Visione del modulo
  4) Pulizia dell'area di transito
  5) Ritorno al menu

Da un punto di vista logico l'interfaccia si presenta come nel primo caso perchè le operazioni di input sono del tutto trasparenti all'utente; l'unico elemento di complessità che al limite si potrebbe introdurre è dovuto alla gestione, da parte dell'utente, di un numero di dischetti maggiore di uno, oltre naturalmente a quello su cui è memorizzato 123.
Ovviamente questa considerazione non è più valida se viene utilizzato un personal computer configurato con disco rigido e se tutte le tabelle sono ivi memorizzate.

Per quanto detto, il dossier virtuale si pone come una interfaccia tra foglio elettronico e utente rendendo completamente trasparente l'insieme di comandi mediante il quale la tabella viene effettivamente gestita.
Il maggior merito di ciò è attribuibile al comando di gestione menu (/xm) che consente di costruire un'interfaccia estremamente semplice ed efficace.
La possibilità di mascheramento totale del foglio elettronico introduce necessariamente la problematica della progettazione, dello sviluppo e del mantenimento di dossier virtuali mediante Macro. Si discuterà brevemente di ciò, sulla base delle esperienze sino ad ora acquisite, dopo avere esaminato un semplice esempio di realizzazione di dossier virtuale.

5. UN ESEMPIO DI REALIZZAZIONE DI DOSSIER VIRTUALE

Viene ora presentato un esempio estremamente semplice di come possa essere effettivamente costruito un dossier virtuale e comparate la sua struttura logica e fisica.

Da un punto di vista logico il dossier è costituito da una copertina (che incorpora anche l'indice) e da tre moduli, composti ciascuno da un quadro che può contenere numeri o commenti. Ad ogni modulo può essere associato un titolo, e questo lo identifica nell'indice. La struttura di questo dossier è statica essendo composto da un numero fisso di moduli di dimensione fissa.

Le operazioni disponibili sono le seguenti:

- Copertina: serve per riempire o modificare il contenuto della copertina del dossier
- Vedere: serve per riempire, modificare, vedere un modulo
- Salvare: serve per operare il salvataggio su disco del dossier
- Cancellare: serve per la cancellazione del contenuto di un modulo del dossier
- Fine: designa la fine dei lavori.

Le cinque operazioni di gestione del dossier costituiscono il menu a disposizione dell'utente e sono implementate con l'utilizzo di Macro che ora vengono proposte e commentate.
A fianco di ogni istruzione è riportato l'indirizzo in cui è collocata.

C43 COPERTINA
C44 Inserimento o aggiornamento della copertina
C45 /riA21..H40~
C46 /cD34~L62~
C47 /cD36~L42~
C48 /cD38~L82~

C43, C44: Costituiscono rispettivamente l'operazione selezionabile da menu e la decrizione sintetica dell'operazione stessa.
C45: Limita la possibilità di input al rettangolo del foglio che ha come estremi (alto sinistra e basso destra) rispettivamente A21 e H40. In questo modo l'utente può esclusivamente vedere e modificare l'area del foglio che fisicamente contiene la copertina del dossier. Il foglio elettronico viene dunque mascherato nella sua struttura fisica globale, e l'utente vede quelle parti che i comandi implementati via macro gli consentono.
C46, C47, C48: Queste operazioni copiano rispettivamente il contenuto delle celle D34, D36, D38, che contengono il titolo dei moduli della copertina, nelle celle L62, L42, L82, che sono le celle dei moduli atte a contenere il titolo.

D43 VEDERE
D44 Inserimento, modifica e visione di un modulo
D45 {home}
D46 /xnNumero del modulo:~A46~
D47 /xiA46=1~/ri161..Y80~
D48 /xiA46=2~ri141..Q60~
D49 /xiA46=3~/ri181..O119~

D45: Sposta il cursore nella cella A1, consentendo di vedere una schermata bianca

D46: Chiede il numero del modulo che si vuole vedere e pone la risposta nella cella A46

D47, D48, D49: Fa un test sul contenuto della cella A46, e se questo è eguale a 1, 2 o 3 consente di vedere o modificare solo una parte del foglio, e cioè quella che contiene il modulo selezionato.

E43 SALVARE
E44 Salvataggio su disco del dossier
E45 {home}
E46 /fs ,?~

E46: Inizia l'operazione di File Save (salvataggio della tabella su disco) chiedendo il nome che si vuole associare alla tabella (?)

F43 CANCELLARE
F44 Cancellazione di un modulo del dossier
F45 {home}
F46 /xnNumero del modulo:~A47~
F47 /xiA47 = 1~/reJ64..X79~/reD34~
F48 /xiA47 = 2~/reJ44..P54~/reD36~
F49 /xiA47 = 3~/reJ84..N118~/reD38~

F47, F48, F49: In base al numero scelto si cancella il contenuto del modulo e il titolo riportato nella copertina (rispettivamente celle D34, D36, D38)

G43 FINE
G44 Fine dei lavori
G45 /qy~

G45: Termina la sessione dei lavori

La struttura globale del foglio è visibile in figura 6.

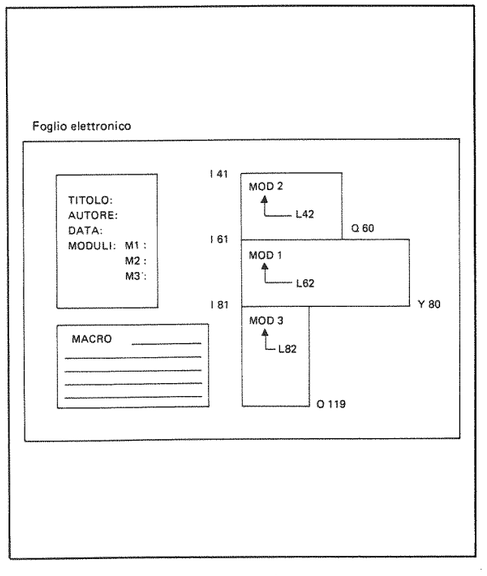


Fig. 6

6. LIMITAZIONI

Il dossier virtuale è uno strato di software che si frappone tra foglio elettronico e utente finale, creato però secondo una certa finalità specifica, e cioè quella di implementare un modello e la sua metodologia di utilizzo.
Per l'utente finale il dossier virtuale rappresenta dunque un ambiente di utilizzo creato ad hoc per le sue esigenze specifiche.

Esistono però alcuni limiti in cui si incappa utilizzando le Macro di 123 che a volte rendono difficilmente superabili ostacoli che con un comune linguaggio di programmazione non si incontrerebbero; vediamone alcuni:

- impossibilità di gestire moduli di dimensione variabile; ciò implica che all'atto della progettazione si rende necessario definire una dimensione fissa per i moduli
- impossibilità di creare per ciascun modulo un sistema di riferimento relativo, invece di utilizzare quello globale del foglio elettronico.

Osservo a questo punto che con le Macro è generalmente di una difficoltà elevatissima la gestione di strutture dinamiche, e cioè le cui dimensioni variano in funzione degli eventi, soprattutto per il fatto che è impossibile utilizzare indirizzi relativi (con un livello di indirettezza) invece che assoluti e perchè non esiste una funzione che ritorni la posizione del cursore nella tabella.

A prescindere da questo discorso è comunque vera una cosa: l'accoppiata foglio elettronico tradizionale e Macro consente di avere a disposizione uno strumento di sviluppo applicativo estremamente potente e quindi varrebbe la pena di estendere almeno la loro potenza funzionale.

7. CONCLUSIONI: LE PROSPETTIVE PER LA PROGETTAZIONE DI APPLICAZIONI DI OFFICE AUTOMATIOON CHE OFFRE IL DOSSIER VIRTUALE

In questa nota si è visto come le Macro di LOTUS 123, tramite il concetto di dossier virtuale, consentono di frapporre un nuovo strato di software tra foglio eletronico e utente finale.

Il vantaggio di questo approccio consiste nel fatto che si rende possibile lo sviluppo di una applicazione di office automation, sotto forma di dossier virtuale, che maschera completamente l'esistenza del foglio elettronico. La progettazione dell'applicazione, come detto, si compone in due fasi distinte:

- Il disegno della struttura dei moduli che compongono il dossier
- Il progetto delle Macro che implementano le funzionalità e la gestione menu del dossier

Anche se il linguaggio delle Macro risulta essere sotto taluni aspetti poco potente e scarsamente strutturato, cosa che apre dei problemi per la manutenzione e la progettazione di grosse applicazioni, consente la progettazione di applicazioni di media e piccola dimensione con tempi di sviluppo estremamente ridotti.
Inoltre, per la separazione tra il disegno del layout tabellare e delle Macro, risulta naturale l'applicazione di una tecnica di progettazione prototipale, cioè che conduce al prodotto definitivo attraverso un certo numero di prototipi intermedi.
Questo consente inoltre una facile diversificazione dell'applicazione finale in un certo numero di versioni, ciascuna destinata ad utenti che pur avendo esigenze simili strutturalmente hanno, per esempio, necessità di interfaccie e tabelle dissimili.

Per riassumere si identificano schematicamente i benefici e i problemi che si incontrano utilizzando le Macro per la progettazione di dossier virtuali:

- Benefici: Rapido sviluppo prototipale delle applicazioni. Rapida differenziazione dell'applicazione. All'utente è completamente mascherata l'esistenza del foglio elettronico.

- Problemi: Difficoltà di mantenimento dell'applicazione direttamente proporzionale alla sua dimensione. Difficoltà nel produrre documentazione finalizzata alla manutenzione, vista la scarsa strutturazione del linguaggio.

8. RINGRAZIAMENTI

A quanto espresso nella presente nota hanno contribuito in modo determinante le esperienze effettuate nell'ambito della Etnoteam SpA, nello sviluppo di applicazioni per l'office automation con l'utilizzo di LOTUS 123.

9. BIBLIOGRAFIA

[1] LOTUS 123, User Manual, Lotus Development Corporation

[2] Paulin, Polillo, Schiavo Campo, IL DOSSIER ELETTRONICO: UNO STRUMENTO DI INFORMATICA INDIVIDUALE PER L'UFFICIO, Note di Software n. 25/26, 1985

[3] Henderson, Cobb, SPREADSHEET SOFTWARE FROM VISICALC TO 123, Que Corporation

NOTE DI SOFTWARE 32

UNA INDAGINE EMPIRICA SU UN CAMPIONE DI CORRISPONDENZA

Piera Piardi

Dipartimento di Scienze dell'Informazione - Università di Milano

RIASSUNTO

Questo lavoro propone una possibile classificazione della corrispondenza in base ai concetti espressi e alle parole (verbi, sostantivi, etc.) usati per esprimerli. Le classificazioni ottenute generano attività diversificate.

ABSTRACT

This work suggests a possible classification of correspondence, according to the concepts expressed and also according to the words (verbs, nouns) used in expressing them. The classifications obtained create different procedures.

1. INTRODUZIONE

L'interesse alla automazione d'ufficio si sta gradualmente spostando verso le attività che gli operatori dell'ufficio stesso attuano davanti alla stazione di lavoro. Infatti in prospettiva, in un ipotetico ufficio del futuro, molto del lavoro si svolge leggendo su un opportuno schermo ed attuando degli interventi di varia natura con l'impiego di tastiere od altri strumenti di ingresso.

Per poter decidere quali aspetti della vita dell'ufficio così concepita possano essere supportati da procedure più o meno automatiche, è necessario effettuare delle ricerche di natura empirica.

Il presente lavoro rappresenta un primo tentativo in questa direzione, prendendo in esame la vita di un responsabile di attività in un Istituto di Ricerca. I risultati della indagine, facente parte della ricerca tuttoria in corso, sono qui esposti.
La speranza è di fornire un contributo alla costruzione di una scienza dell'ufficio basata su esperienze empiriche.

1.1 Scopo del lavoro

Scopo di questo lavoro è individuare all'interno di documenti stringhe di caratteri (patterns) di significato pragmatico il cui riconoscimento attivi procedure.
Nel paragrafo 2 viene spiegata la metodologia usata per il presente lavoro.
Nel paragrafo 3 si espongono i risultati dell'analisi cioè i raggruppamenti di insiemi di parole, più ricorrenti nella corrispondenza esaminata, che esprimono determinate situazioni.
Segue poi nel paragrafo 4 l'analisi dei risultati ottenuti.
Le conclusioni al paragrafo 6, e la bibliografia utilizzata concludono il presente lavoro.

2. METODOLOGIA UTILIZZATA

La ricerca è stata condotta seguendo una metodologia che si articola in due FASI.

2.1 Prima FASE

A - Scelta di un campione tipologico di ambiente di lavoro: l'analisi è stata effettuata in un ambiente di lavoro specifico, un Istituto Universitario.

B - Selezione dei sistemi comunicativi ed elezione di un campione per l'oggetto della ricerca: tra i vari si-

La ricerca si è svolta nell'ambito del progetto CP in collaborazione con la Honeywell Information Systems Italia (Sezione di Ingegneria, Pregnana).

stemi di comunicazione dell'ufficio è stato scelto quello epistolare.
Per campione è stata eletta la corrispondenza intercorsa tra un manager (il Direttore dell'Istituto) e sia le relazioni esterne che quelle interne (docenti, collaboratori, segreterie).

2.2 Seconda FASE

C - Studio analitico dell'oggetto (TEMI e DISCIPLINE): l'analisi del campione per il rilievo di TEMI e DISCIPLINE è stata fatta sulla corrispondenza in uscita intercorsa in un periodo di circa tre anni costituita da 322 unità tra lettere e messaggi.

Le lettere sono caratterizzate da;

- data
- destinatario (titolo di studio, nome, cognome)
- indirizzo (del destinatario)
- testo
- mittente (titolo di studio, nome, cognome)

I messaggi sono caratterizzati da:

- destinatario (nome o cognome spesso preceduti da "per")

D - Ricerca ed individuazione di composizioni di parole (stringhe di caratteri) per la classificazione in categorie: raggruppamento delle composizioni di parole per significato espresso.

E - Riporto statistico delle categorie

F - Riporto statistico del numero di concetti espresso per ogni documento.

G - Delineazione delle possibili procedure attivate dalle categorie.

3. RISULTATI DELLE ANALISI

Il riconoscimento di particolari stringhe di caratteri determina azioni differenti. L'analisi effettuata ha portato alla classificazione delle stringhe di caratteri in 14 CATEGORIE a seconda del loro significato implicito e delle procedure che attivano.

Segue un elenco delle CATEGORIE e relative stringhe di caratteri.

3.1 Richiesta informazioni

Questa categoria delimita le composizioni di parole che implicano una richiesta di informazioni. Le composizioni più ricorrenti nella documentazione esaminata sono:

Lo ritieni	possibile?
dove?	
quando?	
come va?	
mi puoi	dire?
cosa è	avvenuto?
cosa ne	pensi?
siete favorevoli?	
desidero conoscere	
puoi farmi	sapere
aspetto	notizie
aspetto	informazioni
chiedi	informazioni
cerca di	sapere
puoi	informarti
si può...?	
ho bisogno di	sapere
cerca di	sapere
mi interessa	conoscere
mi dai	informazioni?
puoi raccogliere	informazioni
puoi	dire
a che punto	siamo

Una delle procedure possibili attivate è la creazione di una scheda per l'informazione richiesta secondo il seguente schema:

INFORMAZIONE RICHIESTA

- A chi : [_____]
- Per cosa : [_____]
- Quando : [_____]
- Entro quando : [_____]
- Utilizzatore : [_____]
- Priorità : [_____]

Al ricevimento dell'informazione la scheda viene aggiornata.

3.2 Richieste oggetti

Questa categoria delimita le composizioni di parole che individuano una richiesta di oggetti:

aspetto	note
aspetto	commenti
mi dai	note?
mi dai	appunti?
fatti dare	appunti
fatti arrivare	appunti
mi impone di chiederti	lavagne
richiediamo	attrezzature
puoi farmi avere	il materiale?
predisponi	documentazione
puoi raccogliere	documentazione
puoi ridarmi	copia
ho bisogno	copia
fammi avere	rapporto

Una delle possibili procedure attivate è la creazione di una scheda per l'informazione richiesta secondo il seguente schema:

OGGETTO RICHIESTO

- A chi : [_____]
- Per cosa : [_____]
- Quando : [_____]
- Per quando : [_____]
- Utilizzatore : [_____]
- Priorità : [_____]

Al ricevimento dell'oggetto si può aggiornare la scheda ed eventualmente l'archivio.

3.3 Richieste di contatto

Questa categoria delimita le composizioni di parole che individuano una richiesta di contatto.
Segue un elenco delle composizioni più ricorrenti:

ti aspetto	
avverti	(chi concerne)
avvicina	« «
contatta	« «
avvisa	« «
fai venire	« «
fissami	appuntamento
prendimi	appuntamento
invitare a	seminario
ho bisogno di	parlare
fissa data per	incontro
cerca	(chi concerne)
mi chiami	« «
puoi chiamare	« «
senti	« «
telefona a	« «
ti prego di parlare	« «
predisponi riunione con	« «
predisponi incontro con	« «

Tra le possibili procedure se ne evidenziano 2:

. *Contatto telefonico*

Creazione di un elenco telefonate da fare secondo il seguente schema:

- A chi : [_____]
- Per che cosa : [_____]
- Priorità : [_____]

. *Incontro*

Creazione di una scheda appuntamenti secondo il seguente schema:

- Con chi : [_____]
- Per cosa : [_____]
- Quando : [_____]
- Dove : [_____]
- Priorità : [_____]

Ad incontro accordato si aggiorna l'agenda.

3.4 Richieste di organizzazione

Questa categoria delimita le composizioni di parole che individuano una richiesta di organizzazione di seminari, convegni, conferenze ecc.

Segue un elenco delle composizioni più ricorrenti:

organizzare	seminario
organizzare	conferenza
predisponi	convegno
predisponi	seminario

Tra le possibili procedure si delinea la seguente:
ORGANIZZAZIONE

- per che evento : [_____]
- 1 - quando : [_____]
- per quale argomento : [_____]
- documentazione necessaria : [_____]
- dove : [_____]
- interessati : [_____]
- convocazioni : [_____]
- conferme : [_____]

3.5 Richiesta di azioni varie

Questa categoria è molto generica, infatti comprende composizioni di parole che individuano attività molteplici generalmente eseguite dai collaboratori.
Le composizioni più frequentemente riscontrate sono:

aggiungere a	lettera
modificare	
ti prego di	mandare nota
andrebbe	informato
verifica	documentazione
fai pervenire	a ...
provvedete	a ...
leggere	
aggiungere a	ordine del giorno
aggiungere a	elenco
aggiungere a	nota
aggiungere a	lettera
mostra	catalogo
mostra	pubblicazioni
mostra	appunti
fai vedere	(oggetto)
hai provveduto	a ...?
esporre	avviso
stampare	avviso
esporre	orario

Una delle possibili procedure, per permettere al mittente di tenere traccia delle attività richieste è la creazione di una scheda del seguente tipo:

AZIONE

- Tipo : [_____]
- Richiesta a : [_____]
- Quando : [_____]
- Finalità : [_____]
- Priorità : [_____]

3.6 Segnalazioni

Questa categoria comprende composizioni di parole che individuano segnalazioni di eventi o attività e sono individuate da:

inizio a discutere	con
segnala	a
ho provveduto	a
ci hanno	attribuito
ho parlato	con
ho consegnato	
ritengo opportuno	che
ci chiede aiuto	
mi sembra indispensabile	
segnalo quanto segue	
le lezioni sono iniziate	
desidero segnalare iniziativa	
comunico a loro	che
le disposizioni	sono
il documento va bene	
annullare	
rinunciare a	
accettare	
confermo	
aderire a	
è stata fissata	data
dopo conferma	
O.K.	data
è stato approvato	
ho ricevuto	informazioni
ho saputo	
allego	catalogo
allego	lettera
allego	curriculum
allego	depliant
allego	copia
allego	biglietto
allego	documentazione
allego	proposte
allegati	
avere	telefono
telefono	numero...

Una delle possibili procedure attivate è la creazione di una scheda delle segnalazioni come segue:

SEGNALAZIONE

- A chi : [_____]
- Per che cosa : [_____]
- Quando : [_____]
- Priorità : [_____]

3.7 Collaboratori

Questa categoria comprende composizioni di parole che individuano azioni che devono essere eseguite

dai collaboratori, per la maggior parte è relazionata alle categorie "RICHIESTA AZIONI VARIE", "SEGNALAZIONI", "MEMO".
Segue un elenco delle composizioni più ricorrenti:

pensaci tu	
Puoi farlo?	
Devi	verificare
Provvedete voi	

Una possibile procedura per la traccia delle attività è la seguente:

ATTIVITÀ COLLABORATORI

- quale (rapporto di relazione con la categoria o le categorie che hanno generato "COLLABORATORI")

- A chi : [_____]
- Quando : [_____]
- Entro quando : [_____]
- Priorità : [_____]

3.8 Proposte

Questa categoria comprende composizioni di parole che individuano un suggerimento o una proposta:

Ti propongo	di ...
Siete disponibili	a ...
Ti va	di ...
La mia proposta	è ...

Tra le procedure disponibili si delinea la creazione di una scheda come segue:

PROPOSTA

- A chi : [_____]
- Per che cosa : [_____]
- Quando : [_____]
- Priorità : [_____]

3.9 Data esterna

Questa categoria contiene composizioni di parole alfanumeriche che delimitano una data nel tempo e generalmente viene utilizzata per l'archiviazione cronologica del documento.

Data espressa numericamente
Data espressa alfabeticamente
Data espressa alfanumericamente

Una possibile procedura è associare la data alle procedure delle categorie pertinenti, ad esempio:

20/5/85	-Quando
Aprile	-Quando
20 Maggio	-Quando

3.10 Data interna

Questa categoria contiene composizioni di parole che definiscono una data contenuta all'interno del documento ed è sempre relazionata ad altre categorie.
Segue una lista delle composizioni più frequenti:

In riferimento alla sua del...	(data)
In relazione alla sua del...	(data)
In data (data) è stato approvato	

Una possibile procedura è creare una relazione tra la data ed il contenuto del testo ed altri documenti a cui si riferisce.

3.11 Scadenza

Questa categoria è generata dalla "DATA INTERNA" e comprende composizioni di parole alfanumeriche che individuano una data contenuta nel testo del documento. La data espressa è il limite temporale entro cui l'evento a cui si riferisce il documento deve essere realizzato. Segue un elenco dei casi più ricorrenti:

il ... (data) sarò a Roma
rinvia l'appuntamento a ... (data)
entro il ... (data) va fatto
a fine mese ...
oggi ...
nei giorni (date) ci sarà ...

Una procedura possibile è associare questa scadenza alle schede generate dalle altre categorie. Ad esempio:

Entro il ... (data)	-Per quando (collegamento alla situazione temporale del contenuto del testo)

3.12 Priorità messaggio

Questa categoria comprende le composizioni di parole che indicano la priorità di un documento rispetto ad altri.
È individuata da:

Urgente
Immediatamente
Con urgenza
Urgentissimo

Una possibile procedura è segnalare la priorità del documento in tutte le liste attivate dal documento stesso.

3.13 Priorità dell'oggetto

Questa categoria comprende composizioni di parole che indicano la priorità delle azioni espresse nel documento stesso.
È individuata da:

Con urgenza
Urgente
Molto urgente
Al più presto
Immediatamente

3.14 Memo

Questa categoria comprende composizioni di parole che hanno la funzione di ricordare una attività o un evento, generalmente è associata ad altre categorie.
Segue un elenco delle composizioni più ricorrenti:

ti ricordo di ...
ricordiamoci di ...
ti ricordi di ...

Una delle possibili procedure è segnalare in una lista l'evidenza delle azioni da ricordare.

MEMORANDUM

- Per che cosa (rapporto di relazione con la categoria o le categorie che hanno generato la nota di MEMO)

- A chi : [_____]
- Quando : [_____]
- Entro quando : [_____]
- Priorità : [_____]

4. ANALISI DEI RISULTATI ED OSSERVAZIONI

Si nota che nella composizione di parole le procedure sono attivate dalle differenti concatenazioni create dal verbo coniugato ed i sostantivi. Questo legame reciproco genera le diverse categorie e le relative differenti procedure. Ad esempio:

a) Angela fissami una data per un incontro
- richiesta di contatto
- interesse privato
- aggiornare agenda

b) è stata fissata la data d'inizio delle lezioni
- comunicazione
- interesse pubblico
- esporre

- La categoria "RICHIESTA DI AZIONI VARIE" comprende alcuni particolari stringhe di caratteri che sviluppano azioni diverse ampliando o modificando la procedura già prevista. Segue un breve elenco con le procedure diversificate:

scrivere -	aggiornamento lista note formattazione testo correzione modifica copia archivio
lettera -	aggiornamento lista lettere ricerca indirizzi scrivere spedire
fare copia -	duplicazione
telefonare -	aggiornamento lista telefonate ricerca numero in rubrica eventuale aggiornamento
utilizzare agenda -	confronto di date/orari se eventi futuri: aggiornare se eventi passati: rimuovere
utilizzare archivio -	ricerca dati rimozione dati immagazzinamento dati

- In uno stesso documento le procedure attivate da una categoria possono essere meglio definite da altre categorie. Ad esempio:

a) Per Angela: per favore chiama ... ha telefonato numero ... è urgente.

- richiesta di contatto
- segnalazione
- collaboratore
- priorità esterna

b) ti prego di **modificare** la situazione di LUCA.
ti prego di **farmi sapere.** È **urgente.**

- richiesta di azione
- collaboratori
- richiesta di informazioni
- priorità interna

- La scadenza di un evento può essere determinata oltre che dalla categoria scadenza anche dall'uso dei verbi ausiliari e dal suffisso "**-to**.

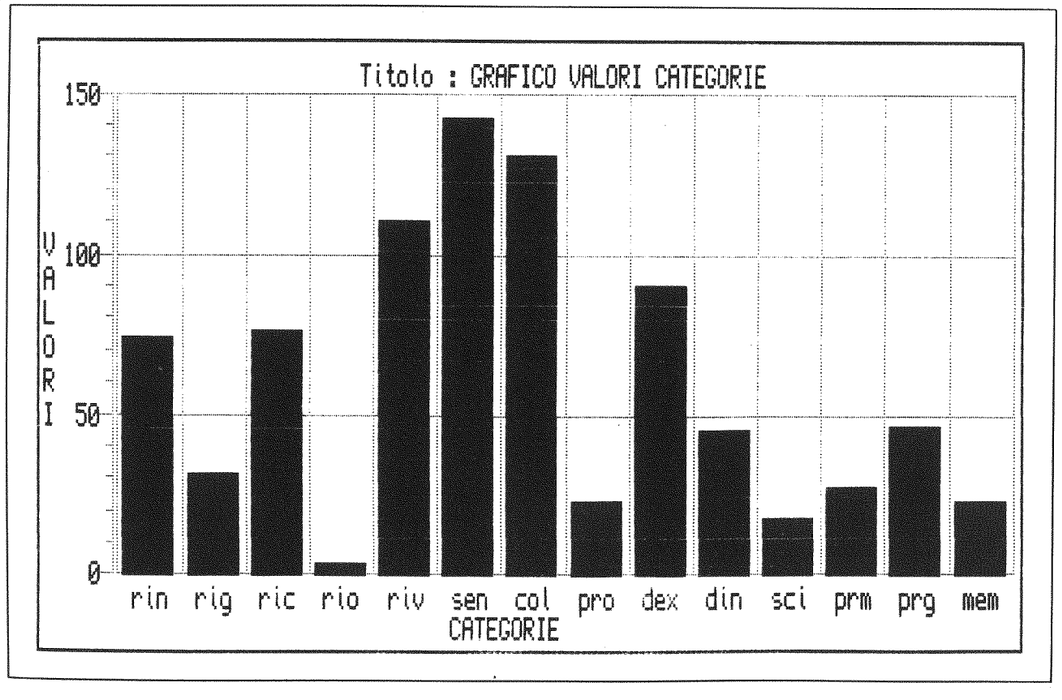
Ad esempio: "**ho** incontra**to** il prof. ..." è un avvenimento passato, si può aggiornare l'agenda mediante rimozione dell'informazione.

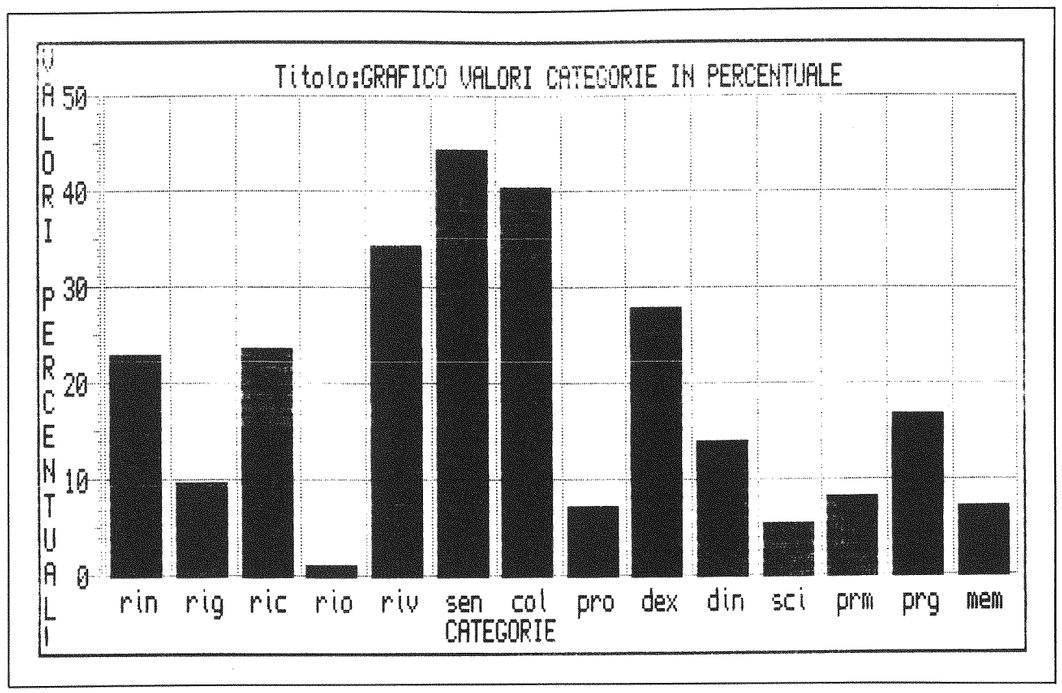
5. PROSPETTI GRAFICI

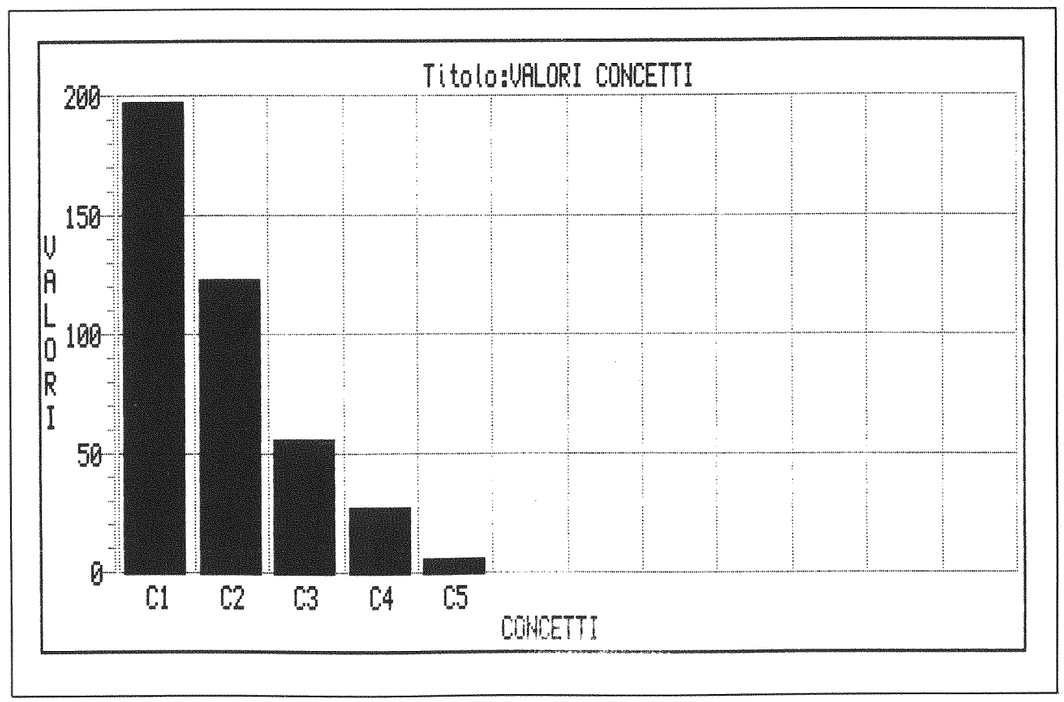
Segue una tabella che illustra la ricorrenze delle categorie e la quantità dei concetti espressi nei documenti esaminati, in valori assoluti ed in percentuali con relativa rappresentazione grafica di istogrammi.

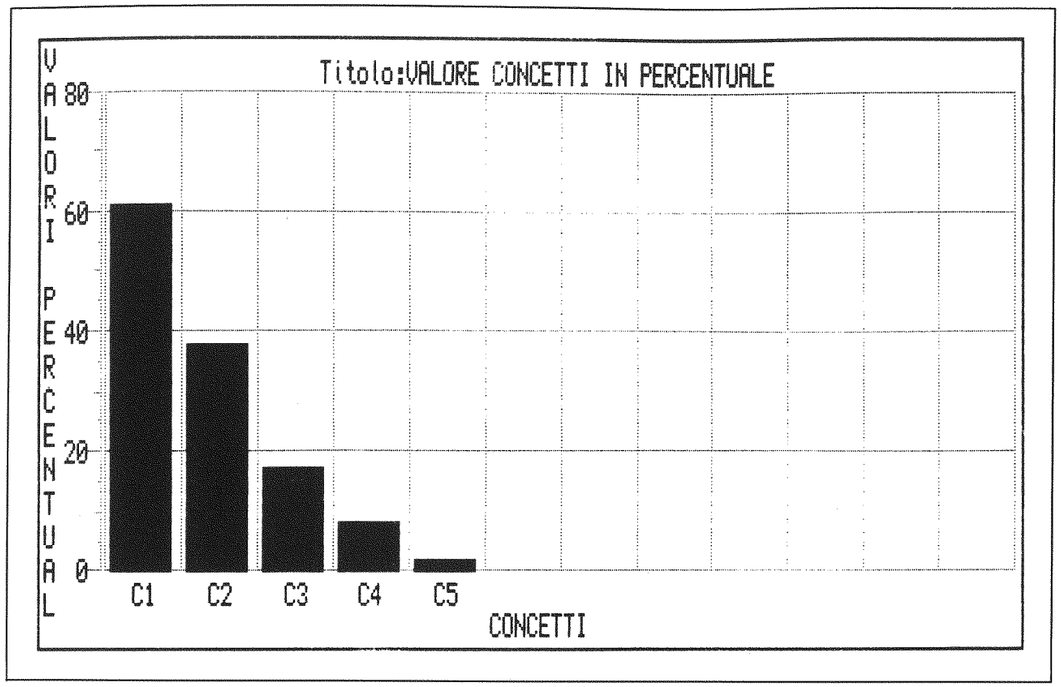
SIMB	LEGENDA	CAT.	N. unità	Valori %
rin=	richiesta informazioni	rin	75	23.29
rig=	richiesta oggetti	rig	32	9.94
ric=	richiesta contatto	ric	77	23.91
rio=	richiesta organizzazione	rio	5	1.55
riv=	richiesta azioni varie	riv	111	34.47
sen=	segnalazioni	sen	143	44.41
col=	collaboratori	col	131	40.68
pro=	proposte	pro	24	7.45
dex=	data esterna	dex	91	28.26
din=	date interna	din	46	14.29
sci=	scadenza interna	sci	19	5.90
prm=	priorità messaggio	prm	28	8.70
prg=	priorità dell'oggetto	prg	47	17.08
mem=	memo	mem	24	7.45
C1=	monoconcettuali	C1	198	61.49
C2=	biconcettuali	C2	124	38.51
C3=	triconcettuali	C3	58	18.01
C4=	tetraconcettuali	C4	29	9.01
C5=	pentaconcettuali	C5	8	2.48

I dati sopra riportati sono riferiti all'analisi condotta su 322 documenti.









6. CONCLUSIONI

La ricerca ha mostrato dati sulla vita dell'ufficio vista attraverso la corrispondenza realizzata su word-processing. I risultati mostrano chiaramente che il comportamento, per quanto vasto e variabile, è catturabile con criteri di classificazione che possono essere bene impiegati per la progettazione di interfacce utenti capaci di realizzare autonomamente tale classificazione.

7. RINGRAZIAMENTI

Si ringrazia il prof. G. Degli Antoni per aver concesso la lettura della sua corrispondenza e per i consigli e stimoli volti alla realizzazione della presente ricerca.

8. BIBLIOGRAFIA

[1] G. Degli Antoni, Le Van Hu, LA AUTOMAZIONE D'UFFICIO COME MECCANIZZAZIONE DEL RAPPORTO TRA TEMA E DISCIPLINA DELLE COMUNICAZIONI, Convegno AICA, 1982.

[2] G. Degli Antoni, MODELI PER L'AUTOMAZIONE DI UFFICIO, Rivista di Informatica, XI suppl. 4, 1982

[3] A. Celentano, P. Paolini, GESTIONE INTEGRATA DI TESTI E DATI NEI DOCUMENTI, Convegno AICA, 1982

[4] G. Degli Antoni, INTEGRAZIONE DI NORME E PROCEDURE NELLA VITA DELL'UFFICIO, Convegno FAST, "Integrazione di Informatica e Diritto", 1983

[5] G. Degli Antoni, INTERFACCIA UOMO MACCHINA, Rapporto interno, Istituto di Cibernetica della Università di Milano, 1984

[6] W. Nicolotti, ANALISI DELLE ATTIVITÀ DEL CAPO-PROGETTO, Tesi di laurea in Fisica, 1983/84

[7] E. M. Vercelli, MECCANIZZAZIONE DELLE ATTIVITÀ DEL CAPO-PROGETTO, Tesi di laurea in Fisica, 1983/84

GESC: UN SISTEMA DI CONSULTAZIONE CHE UTILIZZA COME BASE DI CONOSCENZA UN INSIEME DI ESPRESSIONI IN LINGUAGGIO NATURALE

Gabriele Solaro

Dipartimento di Scienze dell'Informazione - Università di Milano

RIASSUNTO

Il presente articolo descrive GESC, un sistema di consultazione che permette di costruire la base di conoscenza a partire da un testo, in formato libero ed in linguaggio naturale e successivamente di recuperare informazioni dalla stessa per fornire all'utente soluzioni "esperte".

ABSTRACT

The present paper describes GESC, a consultation system that allows to create the knowledge base from a text in free format and natural language, and successively to retrieve information from itself to supply the user with "expert" solutions.

1. INTRODUZIONE

In questi ultimi tempi il computer si è reso necessario in più attività lavorative per gestire l'enorme quantità di conoscenze, in continuo aumento, con cui molti professionisti hanno sempre più a che fare.
Da tale necessità nascono e si sviluppano sistemi informativi computer-assistiti.

Si tratta in gran parte di sistemi di consultazione, che contengono la conoscenza su un particolare argomento, acquisita da persone esperte. Tale conoscenza è rappresentata internamente in forma simbolica, tramite un linguaggio di rappresentazione appositamente definito e costituisce la cosiddetta base di conoscenza del sistema.
Il programma vero e proprio è costituito da una struttura di controllo che utilizza la base di conoscenza per fornire soluzioni o risposte in relazione ai dati inseriti dall'utente.

Si può fare una distinzione tra sistemi esperti e sistemi di supporto; i primi sono per sostituire un professionista in una sua particolare attività; i secondi sono sistemi di supporto alle stesse attività, che hanno come obiettivo quello di migliorare l'aiuto che un professionista, desidererebbe ricevere da varie fonti di informazioni.

Nel corso di questi ultimi anni si sono venuti a delineare alcuni problemi riguardanti sia la progettazione che l'utilizzo dei sistemi di consultazione. Tali problemi possono essere così riassunti:

1) la necessità di un esperto nel campo specifico per strutturare la base di conoscenza;
2) la necessità di un esperto informatico per la creazione e la modifica della base di conoscenza;
3) la necessità di un linguaggio il più vicino possibile al linguaggio naturale per colloquiare con l'utente.

È facile intuire come il primo problema comporti l'aumento del costo di progettazione del sistema di consultazione e come il secondo e il terzo limitino l'utilizzo del sistema da parte dell'utente.

GESC è un sistema di consultazione nato dall'esigenza di fornire una possibile soluzione ai problemi sopra esposti e per poter essere utilizzato, tramite piccole modifiche, da apportare caso per caso, come sistema di supporto ad attività lavorative relative a diversi campi, che vanno dalla medicina all'automazione d'ufficio.

La base di conoscenza che GESC utilizza è costituita da un insieme di descrizioni in linguaggio naturale, che di volta in volta sono, a secondo del campo di applicazione, descrizioni di malattie, descrizioni di operazioni commerciali, di articoli e così via.
Spesso tali descrizioni possono essere ricavate direttamente da un libro, scritto da un esperto nel campo specifico; ciò permette di non avvalersi della partecipazione diretta di un esperto alla creazione della base di conoscenza.

Inoltre le descrizioni possono essere inserite direttamente e all'interno del sistema stesso, in linguaggio naturale ed in formato libero dallo stesso utente, che è quindi in grado di creare e modificare la base di conoscenza, senza far ricorso ad un esperto informatico.

Lo stesso sistema GESC provvede ad elaborare e a strutturare le descrizioni inserite dall'utente, gestendo autonomamente sia la creazione che la modifica della base di conoscenza.
Per quanto riguarda il processo di consultazione il sistema è in grado di individuare all'interno della base di conoscenza frasi, composte al massimo da quattro parole inserite dall'utente, fornendo allo stesso il nome di tutte le descrizioni in cui sono state trovate.
GESC, ad esempio, come sistema di consultazione di diagnostica medica, è un programma che sulla base del confronto tra una lista di sintomi e descrizioni di malattie in linguaggio naturale, precedentemente elaborate e memorizzate (la cosiddetta BASE DI CONOSCENZA), fornisce all'utente un elenco delle possibili malattie.

Nel presente articolo, si cerca di evidenziare le caratteristiche di GESC, spiegando dapprima le principali funzioni fornite dal sistema, come viene costruita la base di conoscenza e come avviene il processo di consultazione; vengono poi indicati i possibili campi di applicazione di GESC e due esempi di applicazione in campi diversi.
Segue una breve descrizione degli strumenti utilizzati per la realizzazione di GESC.

2. DESCRIZIONE DI GESC

In GESC si possono individuare due sezioni ben distinte a cui l'utente può accedere separatamente:

1) una sezione che permette all'utente la creazione o la modifica della base di conoscenza;

2) una sezione che prevede l'utilizzo della base di conscenza da parte dell'utente per una consultazione "esperta".

2.1 La base di conoscenza

La base di conoscenza viene creata dal sistema, richiedendo all'utente solo descrizioni in linguaggio naturale degli elementi costituenti la conoscenza relativa al particolare campo in cui GESC è stato applicato.
Tali "elementi" saranno di volta in volta, descrizioni di malattie, contratti di lavoro, documenti e così via.

GESC richiede che la descrizione di ogni "elemento" da parte dell'utente sia divisa in parti o "contesti", suggeriti di volta in volta dallo stesso sistema. Rifacendosi ad esempio ad un'applicazione medica, la descrizione di ogni malattia sarà divisa in parti (contesti) come "nome della malattia", "etiologica", "patologica", "sintomatologica", etc.

GESC richiede all'utente di inserire la parte di descrizione relativa al contesto da lui stesso scelto, di volta in volta, tra quelli suggeriti dal sistema (vedi figg. 1 e 2).

CREAZIONE BASE DI CONOSCENZA

Contesti riconosciuti:

0 Nome della malattia
1 Terminologia alternativa
2 Etiologia
3 Sintomi
4 Indizi
5 Laboratorio
6 Raggi-X
7 Decorso/Complicazioni
8 Patologia

Numeri già inseriti: 0, 2,

Scegli un numero o RETURN se hai terminato ▶ 3

Fig. 1

CREAZIONE BASE DI CONOSCENZA

Inserisci la descrizione per il contesto: sintomi

DISCRASIA, ANORESSIA,
DISFAGIA IN GENERE... NON GRAVE;
FEBBRE DEBOLE INSISTENTE ▶ ...

Premi RETURN per cambiare contesto

Fig. 2

Dopo che l'utente ha finito l'inserimento delle descrizioni degli "elementi" della conoscenza in linguaggio naturale, il sistema prevede l'elaborazione e la classificazione di tali descrizioni, tramite una serie di programmi la cui esecuzione è gestita autonomamente dal sistema stesso.

Grazie a tali programmi, GESC crea le prime tre relazioni di cui è costituita la base di conoscenza:

1) ELEMENTI, una relazione che associa un numero di codice al nome di ogni elemento descritto nella base di conoscenza.

2) DIZIONARIO, una relazione che associa ad ogni singola parola presente nella base di conoscenza un unico codice che la individuerà univocamente nella relazione delle OCCORRENZE.

3) OCCORRENZE, una relazione che contiene una riga per ogni occorrenza di una parola del dizionario in una particolare "clausola", relativa ad un particolare contesto della descrizione di un particolare "elemento" della conoscenza.

Per "clausola" il sistema intende un insieme di parole comprese tra due segni di punteggiatura, come punto, due punti e punto e virgola. In pratica la posizione di ogni singola parola, all'interno di ogni descrizione viene individuata dal sistema facendo riferimento al numero di contesto e al numero di clausola in cui la parola stessa si può trovare.

L'utente ha anche la possibilità di creare una quarta relazione che associa termini equivalenti tra loro. Per creare tale relazione l'utente deve inserire sinonimi come frasi composte al massimo da quattro parole. Quest'ultima relazione permetterà di estendere ai sinonimi la ricerca di un'informazione nella base di conoscenza.

Il sistema GESC prevede anche la modifica della base di conoscenza, una volta che questa è stata creata. Infatti l'utente ha la possibilità di aggiungere nuove descrizioni o di cancellarne una o più tra quelle presenti nella base di conoscenza. Nel primo caso il sistema richiederà l'inserimento delle nuove descrizioni in linguaggio naturale ed in formato libero, come già avviene per la creazione della base di conoscenza; nel secondo caso il sistema richiederà il nome dell'elemento la cui descrizione deve essere cancellata.

2.2 Il processo di consultazione

Tramite la relazione delle occorrenze, il sistema è in grado di recuperare dalla base di conoscenza informazioni, inserite dall'utente e costituite da parole o frasi.
La ricerca verrà fatta parola per parola e l'informazione verrà trovata se tutte le sue parole hanno almeno un'occorrenza in comune, ovvero se sono associate almeno una volta agli stessi numeri di contesto, clausola e di elemento della conoscenza (vedi figg. 3, 4 e 5).
Appare chiaro come non abbia importanza l'ordine delle parole con cui l'informazione viene ricercata.

Vengono inoltre ricercati i sinonimi dell'informazione inserita qualora essi siano definiti nella base di conoscenza.

DIZIONARIO

Codice	Parola
.	.
.	.
.	.
4	DEBOLE
5	DISCRASIA
.	.
.	.
.	.
10	FEBBRE
.	.
.	.
.	.
15	INSISTENTE
.	.
.	.

Fig. 3

DIZIONARIO

N. Parola	N. Malattia	N. Contesto	N. Clausola
.	.	.	.
.	.	.	.
.	.	.	.
4	1024	3	3
.	.	.	.
.	.	.	.
.	.	.	.
10	1024	3	3
.	.	.	.
.	.	.	.
.	.	.	.
15	1024	3	3
.	.	.	.
.	.	.	.
.	.	.	.

Fig. 4

MALATTIE	
Codice	Nome
.	.
.	.
.	.
.	.
1024	Sindrome di Plummer-Vinson
.	.
.	.
.	.

Fig. 5

DESCRIZIONE DEL PAZIENTE

Inserisci il sintomo:
FEBBRE DEBOLE INSISTENTE...

Contesti riconosciuti:

0 Nome della malattia
1 Terminologia alternativa
2 Etiologia
3 Sintomi
4 Indizi
5 Laboratorio
6 Raggi-X
7 Decorso/Complicazioni
8 Patologia

Fig. 6

Ogni informazione ricercata e trovata nella base di conoscenza permette al sistema di consultazione GESC di fornire all'utente tutti i nomi degli "elementi" della conoscenza a cui è stata trovata associata. Infatto ad ogni informazione inserita e trovata nella base di conoscenza, il sistema associa un punteggio di "selettività" che viene calcolato secondo la formula di fig.7.

$$p = \frac{N \text{ elementi della base} - K \text{ elementi trovati}}{N \text{ elementi della base} - 1}$$

dove N = numero elementi descritti nella base

K = numero degli elementi trovati associati all'informazione inserita.

Fig. 7

Quanto più il punteggio di "selettività" si avvicina al valore "1" tanto più selettiva è l'informazione al fine dell'individuazione del singolo argomento nella base di conoscenza.
Inoltre, una volta che l'utente ha terminato l'inserimento delle informazioni, l'elenco degli elementi a cui tali informazioni fanno riferimento nella base di conoscenza, viene visualizzato dal sistema in ordine decrescente di punteggio; infatti ad ogni argomento viene associato un punteggio somma dei punteggi di "selettività" delle informazioni che lo hanno individuato.

Quante più informazioni vengono inserite e trovate nella base di conoscenza quanto più il sistema è in grado di individuare il documento, l'articolo o la malattia, che rappresentano, a secondo del campo di applicazione, il motivo per cui l'utente ha inserito le informazioni e ha consultato il sistema.
Infine il sistema prevede, a scelta dell'utente, la descrizione in linguaggio naturale di uno o più elementi, tra quelli individuati nella base di conoscenza.

3. I POSSIBILI CAMPI DI APPLICAZIONE

Il sistema GESC è nato come sistema di consultazione in tutti quei campi in cui gli elementi che costituiscono la relativa conoscenza possono essere classificati secondo contesti chiave.

Per fare un esempio la patologia medica può essere considerata come una classificazione di malattie, la cui descrizione può essere divisa in parti o contesti come "nome della malattia", "etiologia", "sintomatologia", "analisi di laboratorio" etc.

Inoltre GESC permette in molti casi di ricavare le descrizioni da inserire nella base di conoscenza direttamente da un libro.
È questo il caso delle descrizioni di malattie, che possono essere ricavate da un qualunque libro di patologia medica.
Quindi GESC può essere applicato ad esempio in campo medico, giuridico, amministrativo.

4. LA PRIMA APPLICAZIONE: IN MEDICINA

La prima applicazione di GESC è stata in campo medico come sistema di consultazione di diagnostica medica che determina una lista ordinata di malattie in base ai sintomi inseriti.

4.1 La base di conoscenza

La base di conoscenza viene creata dal sistema inserendo un testo di descrizioni di malattie, ricavate da un libro di patologia medica.
La descrizione di ogni malattia sarà divisa in parti o contesti come "nome della malattia", "etiologia" etc. (vedi figg. 1 e 2).

La base di conoscenza, in questo caso sarà costituita da un dizionario di singole parole, da una relazione delle loro occorrenze all'interno delle descrizioni di malattia e da una relazione dei nomi delle malattie descritte (vedi figg. 3, 4 e 5).

Inoltre viene previsto un dizionario dei sinonimi, che viene creato a partire da una lista di termini equivalenti. Tale dizionario permetterà di estendere ai sintomi la ricerca nella base di conoscenza dei sintomi inseriti.
Il sistema prevede anche la modifica della base di conoscenza, una volta che questa è stata creata, ovvero l'addizione e la cancellazione sia di descrizioni di malattie che di sinonimi.

4.2 Il processo di diagnosi

Il processo di diagnosi viene realizzato utilizzando la "selettività" delle parole nel linguaggio naturale come unico parametro per valutare il grado di somiglianza tra la descrizione dei sintomi di un paziente e le definizioni di malattie presenti nella base di conoscenza.

I sintomi inseriti (vedi fig. 6), frasi composte al massimo di quattro parole, vengono trovati in una o più descrizioni di malattie se le parole che li costituiscono hanno almeno un'occorrenza in comune (figg. 3, 4 e 5).

Ad ogni sintomo inserito il programma associa un punteggio di selettività che dipende dal numero di malattie presenti nella base di conoscenza e dal numero di malattie in cui è stato trovato, secondo la formula vista precedentemente (fig. 7), dove gli elementi della conoscenza sono in questo caso le malattie.

Dopo aver finito l'inserimento dei sintomi, il programma fornisce un elenco degli stessi sintomi con i relativi contesti in cui sono stati trovati e con i relativi punteggi di selettività ed una lista delle possibili malattie in ordine decrescente di punteggio (viene infatti associata ad ogni malattia la somma dei punteggi di selettività dei sintomi che la coinvolgono; vedi fig. 8).

L'utente ha inoltre la possibilità di vedere, inserendo un numero con cui una malattia compare nella lista, quali sintomi, tra quelli inseriti, hanno permesso la diagnosi di quella particolare malattia oppure la descrizione della stessa in linguaggio naturale.

L'obiettivo di tale sistema consultivo è quello di supportare l'attività del medico nel processo di diagnosi sia permettendo di verificare ipotesi diagnostiche già formulate, che suggerendo diagnosi che potrebbero essere state tralasciate per dimenticanza o perchè la malattia in questione non è comune o si presenta in una forma atipica.
La precisione con cui il sistema determina le malattie più probabili in base ai sintomi inseriti, dipende dal numero delle malattie e dall'accuratezza con cui sono

SINTOMI	(CONTESTI)	(PUNTEGGIO DI SELETTIVITÀ)			
Anemia Ipocromica	{5}	(1.000)	Anemia Perniciosa	{2}	(1.000)
Cellule Atipiche	{5}	(1.000)	Discrasia	{3}	(0.909)
Nicchia Ulcerosa	{6}	(0.909)	Stenosi Pilorica	{6}	(1.000)
Vomito	{3}	(0.727)			

LISTA DELLE POSSIBILI MALATTIE IN BASE AI SINTOMI INSERITI

6 malattie; massimo punteggio 6.545

NUMERO	PUNTEGGIO	MALATTIA
1	6.545	Carcinoma Gastrico
2	0.909	Gastrite Cronica
3	0.909	Ulcera Gastrica
4	0.727	Carcinomi Esofagei
5	0.727	Gastrite Acuta
6	0.727	Sindrome di Zollinger-Ellison

Scegli un numero di malattia nella lista o RETURN ▶

Fig. 8

state descritte nella base di conoscenza e dalla "selettività" dei sintomi inseriti.

Attualmente il sistema è in grado di suggerire diagnosi di malattie dell'apparato digerente; è però facile renderlo esperto per altri tipi di malattia inserendo ulteriori descrizioni nella base di conoscenza.

5. UNA SECONDA APPLICAZIONE

GESC è stato applicato anche nel campo dell'informazione unviersitaria, come sistema di supporto al lavoro di segreteria. In tale sistema al posto del nome della malattia si ha il "tema" dell'informazione e gli altri contesti sono sotto il nome di aula, orario, professore, esame, corso etc.
Tale sistema permette di rispondere direttamente alla richiesta di informazioni da parte degli studenti senza che questi si rivolgano alla segreteria.

6. STRUMENTI UTILIZZATI

Per la costruzione di GESC si sono utilizzati: il sistema operativo UNIX, il DBMS relazionale INGRES, il linguaggio di programmazione C e RAPID [1,2,3], uno strumento di gestione del dialogo utente-programma per sistemi informativi di tipo interattivo.

Si è scelto il sistema operativo UNIX, perchè, tra l'altro, possiede alcuni strumenti di text-processing, INGRES, perchè permette la creazione di relazioni in cui immagazzinare o da cui facilmente recuperare informazioni, il linguaggio di programmazione C perchè permette di creare dei programmi in cui siano richiamate istruzioni di INGRES o intere sequenze di comandi UNIX e infine RAPID perchè offre la possibilità di gestire il video e gli input da tastiera, collegando questi ultimi a messaggi su video o all'esecuzione di istruzioni in C, in INGRES o di comandi UNIX.

7. CONCLUSIONE

La facilità di utilizzo del sistema deriva dall'uso del linguaggio naturale per comunicare con l'utente sia in fase di creazione o modifica della base di conoscenza, sia durante il processo di diagnosi.
Inoltre l'utilizzo di RAPID, ha permesso di costruire il sistema con una grande modularità, facilitando eventuali modifiche o inserimenti, e di rendere trasparente il funzionamento del sistema per mezzo di menù e documentazione in linea.

Il sistema CESC sembra rispondere all'esigenza, che molti professionisti incontrano nella gestione dell'enorme quantità di informazioni, con cui hanno a che fare durante lo svolgimento della propria attività.
D'altra parte solo tra qualche anno si vedranno nascere nuove figure professionali: il medico-informatico, l'avvocato-informatico, ecc.

Finchè non saranno completamente definite tali nuove professionalità saranno necessari sistemi come GESC, che richiedano da parte dell'utente una minima conoscenza informatica per poter utilizzare un computer, e che gli permettano di essere il più indipendente possibile dall'assistenza di un esperto informatico.

8. RINGRAZIAMENTI

GESC rappresenta il risultato seguito al lavoro di tesi svolto sui sistemi di consultazione dallo stesso autore nel campo della medicina [5] e da Franco Cerutti in quello dell'ufficio [6].

Tale risultato è stato reso possibile grazie alla documentazione giunta dagli Stati Uniti [1,2,3,4] ed ai suggerimenti forniti da Gianni Degli Antoni e da Roberto Bertocchi durante il periodo di tesi presso l'Istituto di Cibernetica di Milano.

9. BIBLIOGRAFIA

[1] T. Shewmake, A. Wasserman, RAPID-USE REFERENCE MANUAL, University of California, San Francisco

[2] T. Shewmake, A. Wasserman, A RAPID-USE TUTORIAL, Medical Information Science, University of California, San Francisco

[3] A.Wasserman, TOOL SUPPORT FOR THE USER SOFTWARE ENGINEERING METHODOLOGY

[4] M.S. Erlbaum, THE DDX PROJECT: A METHODOLOGY AND TOOLS FOR FAST PROTOTYPING OF DIAGNOSTIC PROMPTING SYSTEMS USING NATURAL LANGUAGE DISEASE DESCRIPTIONS, Technical Report # 68, 1984

[5] G. Solaro, STRUMENTI PER LA GENERAZIONE DI SISTEMI DI CONSULTAZIONE IN LINGUAGGIO NATURALE: UN SISTEMA DI CONSULTAZIONE DI DIAGNOSTICA MEDICA, Università degli Studi di Milano, Corso di Laurea in Fisica, A.A. 1984-85

[6] F. Cerutti, STRUTTURE DI DIALOGO PER L'ACCESSO A SISTEMI CONSULTIVI IN LINGUAGGIO NATURALE: UN SISTEMA CONSULTIVO PER L'UNIVERSITÀ, Università degli Studi di Milano, Corso di Laurea in Fisica, A.A. 1984-85

IL METODO C.F.D. (COMPUTER FIGURATIVE DESIGN)

Giovanni Cella - Piacenza

RIASSUNTO

Viene presentato un metodo per tracciare disegni mediante un plotter usando il metodo CFD (Computer Figurative Design) basato su un sistema bipolare. Il tracciamento delle linee di disegno è definito da equazioni di un originale sistema di rappresentazione.
Si presta efficacemente per la ricerca compositiva della figura e a ritrovare le sue proporzioni.
Lo stesso programma è in grado di modificare le caratteristiche dell'immagine e di variare la posa come nella preparazione di cartoni animati.

ABSTRACT

In this work a method to draw by a plotter is presented. This method is named C.F.D. (Computer Figurative Design) and it is based on a bipolar method. The lines are traced by couples of coordinates that the computer estracts from equations of an original system of representation.
It is very useful to research the composition of the figure and to find out its proportions.
The same program can modify the characteristics of the image and to change its pose likewise the animated cartoon principle.

1. INTRODUZIONE

Il presente metodo per disegnare figure con l'assistenza dell'elaboratore elettronico si distingue in quanto i dati necessari alla loro composizione sono il risultato di un processo operativo che il computer svolge seguendo un originale corso di istruzioni.
Il presente metodo di disegno assume il punto quale elemento di base e definisce le proprie linee come una continuità di punti. Essi sono ricavati operando la soluzione di equazioni matematiche e il loro grafico concorre alla rappresentazione dell'immagine. Tali disegni si prestano alla resa immediata su di un supporto rigido per mezzo del plotter.

Uno dei modi concettualmente semplici e matematicamente più precisi per riunire un insieme illimitato di dati capaci di descrivere tutti i punti della generica linea, è sicuramente quello di sottoporre al computer un'equazione che, opportunamente elaborata, venga restituita sotto forma di grafico. Si tratta di applicare quanto insegna la Geometria con il sistema di rapresentazione cartesiano, o, meglio ancora, di applicare il sistema suggerito con queste note che chiamo "bipolare".

2. IL SISTEMA BIPOLARE

A differenza del classico sistema di rappresentazione cartesiano che concepisce il piano come un sistema di punti rappresentati da una coppia di numeri (le coordinate x, y riferite a misura di lunghezza), il sistema bipolare rappresenta gli stessi punti del piano ancora con una coppia di numeri, riferiti, però, alla misura di due angoli. Gli angoli (A), (B) aventi origini in due distinti punti: i poli A, B a distanza unitaria.

Fra questo sistema rappresentativo e l'elaboratore elettronico, per ragioni matematiche, sussiste un rapporto preferenziale dovuto a reciproca convenienza. Mentre la determinazione delle coordinate bipolari risulterebbe senza l'aiuto del computer, estremamente faticosa, per contro, l'elaboratore incontrerebbe grosse difficoltà a risolvere molte equazioni di grafici rappresentati con il sistema cartesiano.

Come si utilizzano le tavole logaritmiche per sostituire l'operazione di moltiplicazione con quella di addizione, in modo analogo si ricorre al sistema bipolare per rappresentare, ad esempio le coniche con equazioni di primo grado, invece di quelle di secondo grado necessarie al sistema di rappresentazione cartesiano.

In pratica, la determinazione delle coordinate dei punti nella descrizione della generica linea si svolge con l'iterazione di una delle variabili bipolari, facendole percorrere i passi seguendo l'angolo giro per rilevare i corrispondenti valori dell'altra variabile nell'esplorazione che si estende a tutti i punti del piano.

Se, per esempio, si considera un'equazione bipolare, nella osservanza della quale la somma degli angoli (A) e (B), misurati all'interno della base AB, è uguale ad una costante K, troveremo che in corrispondenza ai valori assunti da una variabile angolare, nel ciclo da 0 a 360 gradi, il suo grafico descrive il cerchio passante per i poli A e B, come, del resto, poteva essere intuito in base a precise nozioni di Geometria.

Similmente, se consideriamo un'altra equaizione bipolare per la quale la differenza fra gli angoli (A) e (B) è uguale ad una costante K troveremo che il suo grafico descrive l'iperbole equilatera, passante con i suoi distinti rami per i poli A e B.
In questo caso, invece, sarebbe stato difficile di poterlo immaginare.

Se, poi, con ulteriore diverso ciclo di iterazione vengono, di volta in volta, assegnati alla costante K valori diversi, si otterrà la descrizione di altrettanti cerchi (o, rispettivamente, iperboli) facenti parte di fasci di curve dello stesso genere tutti per i punti A e B.

Ci serviamo, cioè, della variabile ausiliare K, detta parametro, per disporre di tutte le linee costituenti il fascio. La conoscenza e l'accessibilità ad una grande varietà di fasci, offertaci dalla Geometria Bipolare, in definitiva, ci permette di estrarre da questi porzioni piuttosto consistenti di uno specifico disegno; ovvero, segmenti significativi di curve con le quali comporre una determinata figura.

Inoltrandoci nella conoscenza del procedimento in oggetto, avremo modo di constatare che esso si avvale di un contatto diretto con l'equazione matematica e che torna utile in molti frangenti una gestione della stessa più ravvicinata, senza intrmediari del genere packages grafici attualmente in uso.
Questi, infatti, servono a facilitare la stesura di grafici di altro genere, mentre per seguire il metodo proposto sarà necessario ricorrere a programmi di differente indirizzo.

Un accorgimento utile, attuato con doppia iterazione, tanto della variabile angolare quanto del suddetto parametro, ci permette di definire meglio le indagini; in particolare con la stesura di più curve dello stesso fascio entro un intervallo limitato fra due parametri si riesce ad ottenere un effetto di ombreggiature, o, eventualmente un segno di maggior spessore.

Un altro accorgimento consiste nell'invertire l'ordine delle iterazioni anzidette per ricavare una specie di seghettatura o tratteggio, in quanto, in questo caso, la linea nella sua stesura raggiunge alternativamente i punti, una volta, situati sulla curva corrispondente ad un parametro e, l'altra volta, sulla curva del medesimo fascio definita dall'altro parametro.

Un ulteriore significativa circostanza che moltiplica la duttilità del tracciato delle linee di disegno risiede nella possibilità di orientare le variazioni delle curve non soltanto nella formulazione delle loro funzioni bipolari, ma, anche, in occasione del passaggio che traduce le coordinate bipolari in coordinate cartesiane perchè siano riconosciute dalle attrezzature di output grafico.

Sfruttando questa ampia possibilità di manipolare le linee del disegno, con variazioni delle espressioni matematiche che le generano, deriva la capacità di stendere programmi interattivi. Programmi, che, aventi a corredo opportune proposizioni di input, sono abilitati a descrivere una grande varietà di immagini di un genere di figure dalle caratteristiche diverse o, anche, a riprodurre la stessa figura in pose diverse.

Alcuni esempi, volutamente semplici, serviranno a chiarire come seguire il procedimento e quali risultati si possono ottenere.

2.1 Un esempio

La figura 1 rappresenta un volto tracciato automaticamente, senza interruzione, con un comune plotter nella esecuzione di un programma in funzioni bipolari, elaborato da un personal computer.

Chiaramente si distinguono le diverse ed essenziali linee che compongono la figura. Queste sono ottenute definendo i limiti entro i quali si sviluppa l'arco che interessa la variabile bipolare e, nello stesso tempo, definendo il valore o più valori assunti dal parametro K.

Gli occhi sono descritti con il ricorso a funzioni grafiche del cerchio e dell'ellisse. L'iride, in particolare, è ricavata da un susseguirsi di cerchi concentrici con l'impiego della relativa proposizione di iterazione; con analogo procedimento il segno che sottolinea la palpebra viene estratto da segmenti di ellisse.
Le ciglia e le sopracciglia sono ugualmente parti di ellissi descritte, in questo caso, con la inversione nell'ordine delle variabili nel ciclo di iterazione, seguendo l'accorgimento precedentemente descritto.

La bocca è contrassegnata dalla funzione bipolare che istruisce il grafico della cissoide. Anche i capelli sono tracciati seguendo curve ellittiche sottoposte ad un notevole allungamento in senso verticale. Persino la sigla sotto la figura deriva da una formula matematica espressa nel sistema bipolare, con la cura di imitare la mia calligrafia nello stendere le lettere iniziali del nome.
In fig. 2 un altro esempio di resa di questo metodo.

2.2 L'identikit

Con il sistema bipolare viene usufruita e privilegiata l'equazione per meglio sovraintendere il processo elaborativo in ogni risvolto del suo sviluppo, con interventi che definiscono quelle grandezze capaci di individuare e caratterizzare le linee del disegno.

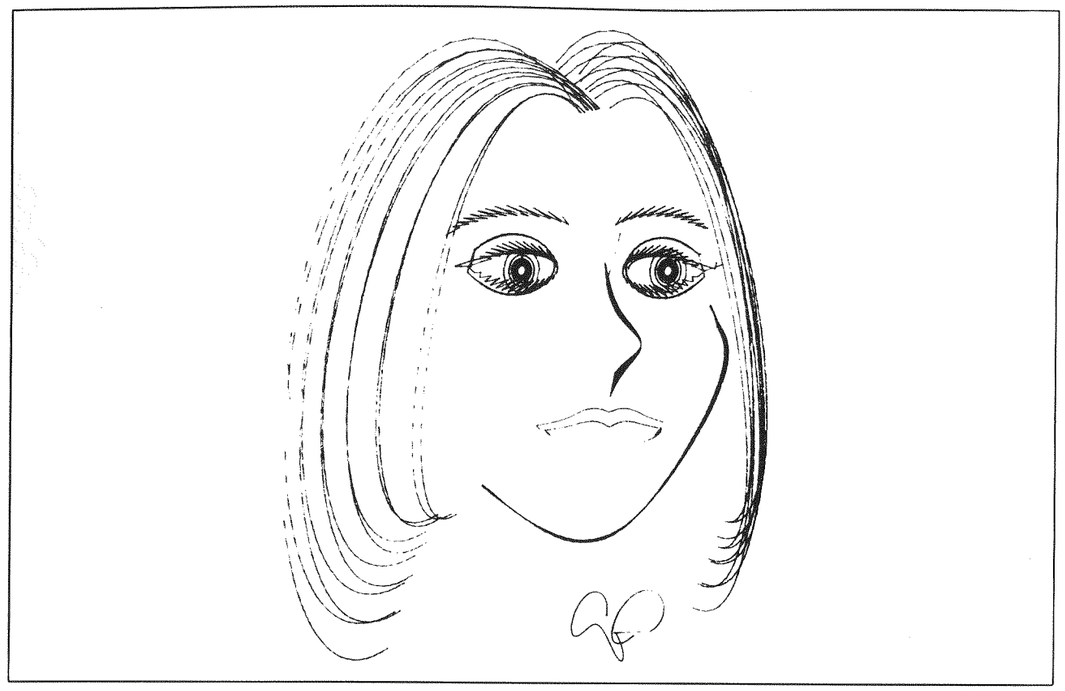


Fig. 1

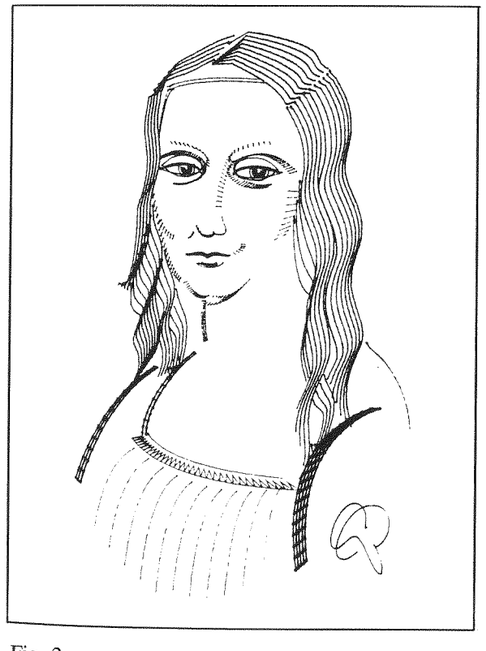


Fig. 2

Si acquisisce, in tal modo la facoltà di comporre un disegno con un insieme di linee di cui siamo, più che mai, in grado di controllarne il percorso ed, eventualmente, modificarne il tracciato. Non soltanto, come di consuetudine, con trasformazioni piane di Geometria Euclidea e variazioni operanti sulle coordinate cartesiane, ma, anche, di indurre in tutto lo sviluppo di una generica linea una diversa trasformazione. Come, per esempio, il cambiamento di valore del parametro che determina una modificazione della linea nell'ambito del fascio di appartenenza alla loro espressione matematica.

In molti casi i valori delle grandezze in gioco sono espressi come variabili che assumono, nell'ambito del programma, una serie di ben definita di quantità numeriche.
In altri casi, ad un complesso di più variabili vengono assegnati valori, di volta in volta, diversi, fuori dal programma con proposizioni corrispondenti di input.

Si affida, così, una straordinaria interattività al programma di un disegno per ottenere dal medesimo una quantità di figure suscettibili di assumere caratteristiche molto diverse.

Un esempio emblematico si presenta in un programma steso, a titolo sperimentale di impiego del computer, per definire un identikit. Tale programma si è di-

mostrato così flessibile che riesce a produrre figure capaci di subire modifiche di grande rilievo. È possibile ritrovare in esse l'illustrazione di profili diversi, quasi quanto sono diverse fra di loro le facce delle persone.

Le figure 3, 4, 5, 6 danno un'idea orientativa della estensione del campo che definisce i lineamenti di una figura. Lo stesso programma, infatti, serve a produrre, indifferentemente, immagini maschili o femminili, dall'aspetto più giovanile o meno.

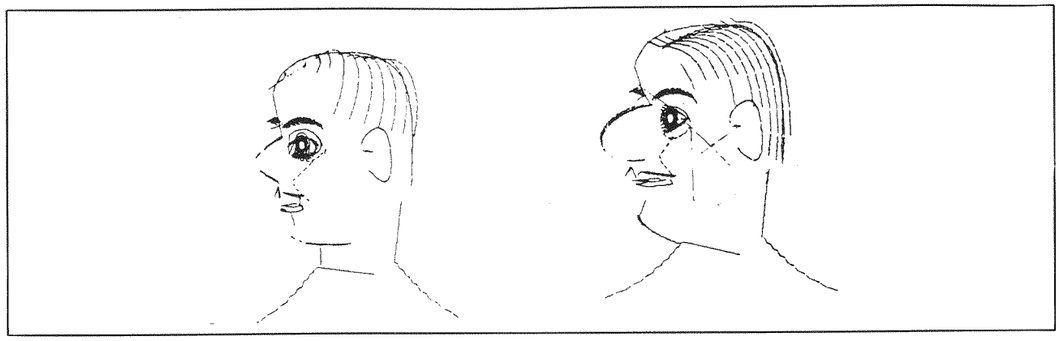


Fig. 3 - 4

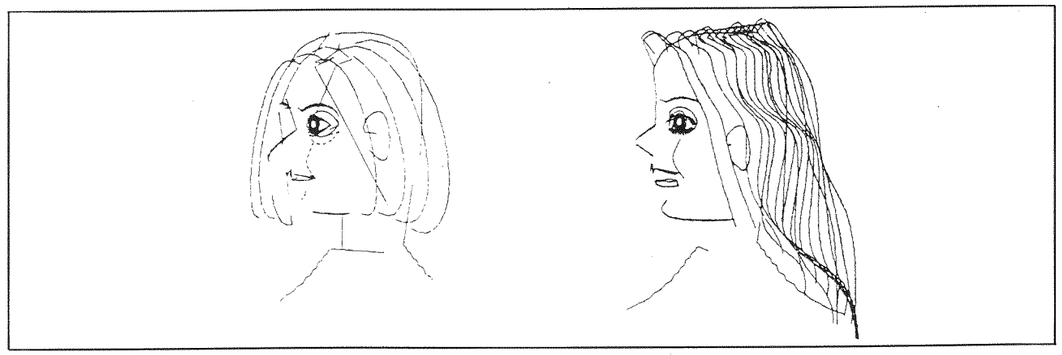


Fig. 5 - 6

Un simpatico risvolto dello stesso programma si ottiene esagerando nel disegno le proporzioni delle caratteristiche di una persona per ricavare la sua caricatura. Ho voluto fare una prova estemporanea forzando i lineamenti, che vagamente ricordavo, di due noti segretari di partito politico italiano. Il risultato, che mi ha sorpreso e divertito è illustrato con le figure 7, 8.

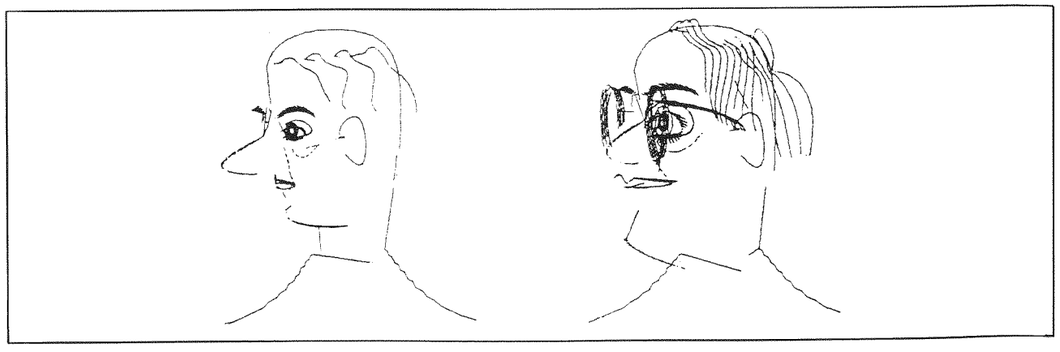


Fig. 7 - 8

2.3 Animazioni

È risaputo che la preparazione di films a cartoni animati richiede un lungo e paziente lavoro soprattutto per quanto riguarda la stesura dei disegni.

I cartoni occorrenti per tale produzione sono costituiti da un'enorme successione di immagini che, gradatamente, differiscono nelle definizioni dei particolari. Sono, appunto, queste piccole, conseguenti variazioni che nella proiezione cinematografica danno allo spettatore l'impressione del movimento.

In maniera analoga a quanto è stato programmato per ricavare i sopradescritti identikit, nel presente caso si cerca di prevedere nelle linee del disegno variazioni che modifichino alcuni particolari nel rispetto, però, dell'immagine di insieme.

Per esempio, nella configurazione di un occhio con le sue linee tipiche si fa in maniera che, con successive proposizioni di input, l'iride giri lo sguardo, la palpebra sia più, o meno, aperta, il sopracciglio sia diversamente alzato. È importante notare che questo procedimento consente di seguire trasformazioni esattamente concatenate in coerenza e armonia della figura, dove nel fluire delle sequenze anche il filo di un capello conserva una sua precisa identità.

A titolo di esempio le figure 9, 10, 11 illustrano un "Superman", in tre pose prodotte con lo stesso programma dal quale è possibile ricavare, per interpolazione degli input, tutte le pose necessarie ad "animarlo". Altri esempi, dove ogni particolare è sempre suscettibile di variazioni in modo indipendente dagli altri, sono rappresentati dalla figura 12 di un gatto e la figura 13 di un cagnolino.

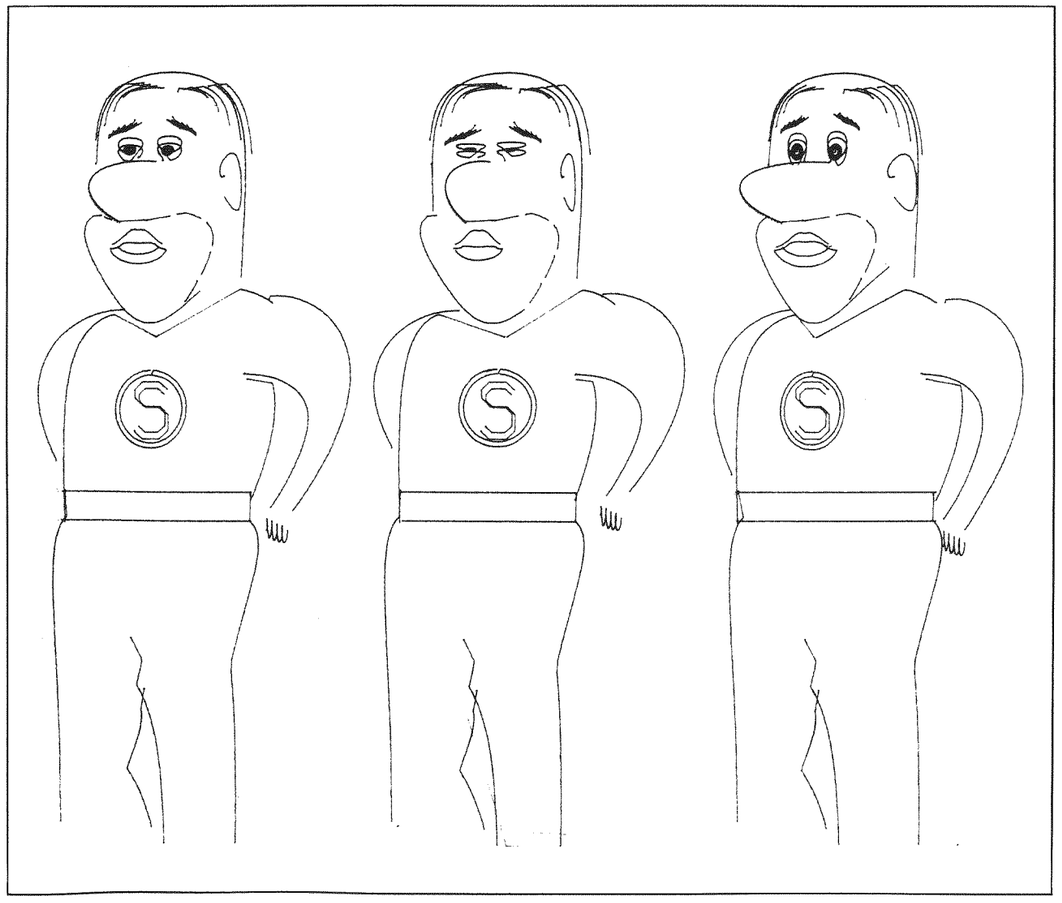


Fig. 9 Fig. 10 Fig. 11

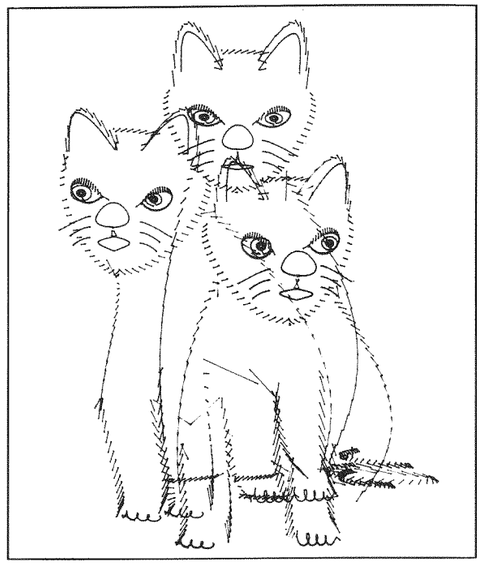


Fig. 12

APPENDICE

Seguono, a titolo di esempio, con riferimento alla figura 1, le istruzioni in linguaggio BASIC impartite ad un comune Personal Computer per ottenere su plotter (tipo PIXY-Mannesmann Tally) il grafico che descrive l'iride dell'occhio destro del volto rappresentato.

```
5 OPEN1,4
11 GOTO 101
101 REM: ********** IRIDE DESTRA
110 PRINT=1, "M"; 1600, 685
120 FORK=5TO22STEP2
130 FORA1= .1TO 360 .1STEP17
140 PRINT=1, "A1": A=A1*/180: C=TANCA
150 D=SIN(A)/(K-COS(A))
160 Y=264*(CxD)/(C+D)
170 X=-160*(C-D)/(C+D)
180 PRINT=1, "D"; X+1700, Y+685
190 NEXTA1, K
```

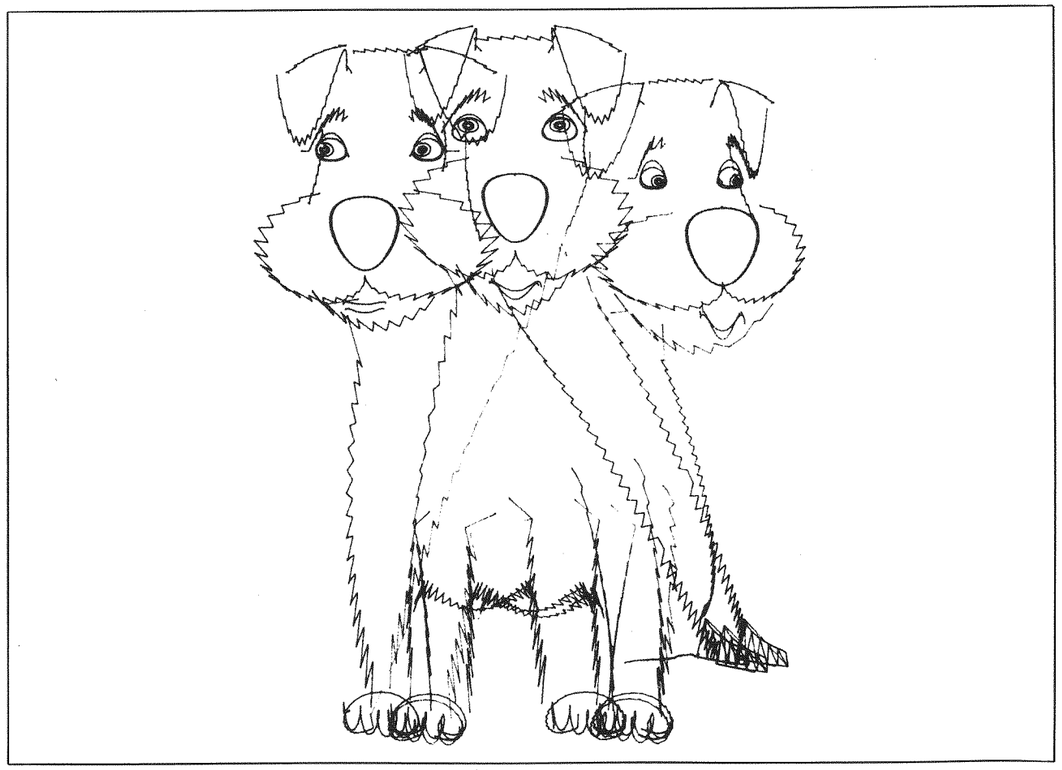


Fig. 13

RECENSIONI

Parallel Sorting Algorithms,
di Selim G. Akl, Academic Press, 1985

La riduzione del tempo necessario per risolvere automaticamente un dato problema costituisce uno degli obiettivi principali dell'informatica.
Questo obiettivo è stato perseguito sia con soluzioni tecnologiche, cioè aumentando la velocità intrinseca delle macchine, così da ridurre il tempo necessario per effettuare una singola operazione elementare a livello hardware, sia con soluzioni concettuali, cioè progettando algoritmi in grado di risolvere il problema con un numero inferiore di operazioni elementari. Oggi tuttavia sembra assai difficile, se non impossibile, ottenere, lungo queste due direttrici, successi paraganabili a quelli di qualche anno fa; da un lato, infatti la tecnologia di costruzione di elementi attivi superveloci ha ormai quasi raggiunto limiti impossibili da superare, perchè imposti dalle leggi fondamentali della fisica, dall'altro gli algoritmi noti per quasi tutti i principali problemi di base sono molto sofisticati e quasi sempre ottimali, nel senso che è dimostrato che non possono esistere algoritmi che richiedono meno operazioni. Valga per tutti l'esempio dell'ordinamento, uno dei problemi di maggior rilievo (secondo alcune stime, circa la metà del lavoro eseguito dai computer consiste nel mettere in ordine insiemi di dati): per tre decenni questo problema è stato studiato a fondo, ed oggi non solo è noto che per ordinare n dati occorre un numero di confronti tra essi proporzionale a n. $\log_2 n$, ma esistono diversi algoritmi in grado di raggiungere questo livello di efficienza.

Ma, se non abbiamo modo di eseguire meno operazioni, nè di eseguire più in fretta ogni singola operazione, abbiamo un'altra possibilità per rendere più veloce la soluzione del nostro problema: eseguire più operazioni contemporaneamente, usando processori diversi. È questa l'idea di base del calcolo parallelo, che rappresenta oggi la frontiera più avanzata dell'informatica.

Data la sua rilevanza nel calcolo automatico, cui si è già accennato in precedenza, il problema dell'ordinamento è stato ampiamente studiato anche in riferimento alla possibilità di risolverlo con algoritmi paralleli.
Questo libro di Akl offre una panoramica completa e sufficientemente chiara dei risultati di queste ricerche. Esso tuttavia non si limita ad una pura e semplice elencazione di algoritmi di ordinamento paralleli, ma fornisce, a partire da questo problema di estremo interesse, un chiaro quadro concettuale delle problematiche relative agli algoritmi paralleli ed alle tecniche di progetto e di valutazione della complessità.

A differenza dei calcolatori sequenziali tradizionali, che, pur differendo tra loro per la dimensione di memoria, il tipo di processore usato e quindi le operazioni di macchina disponibili ed altri elementi di dettaglio di questo tipo, hanno la stessa architettura di base, nel caso degli elaboratori paralleli esistono varie architetture, caratterizzate dal modo in cui sono interconnessi i processori ed in cui viene coordinata la loro attività. In particolare, si può distinguere tra macchine SIMD (Simple Instruction Multiple Data), in cui in un certo istante tutti i processori eseguono la stessa istruzione su dati diversi, in modo sincrono, e macchine MIMD (Multiple Instruction Multiple Data), che operano invece in modo asincrono, poichè ogni processore ha un proprio contatore di istruzioni. Ognuna di queste classi può poi essere suddivisa in sottoclassi in base al tipo di interconnessione. Le caratteristiche di fondo di questi modelli sono presentate in modo sintetico ma molto chiaro nel primo capitolo, insieme ai criteri di valutazione della efficienza degli algoritmi paralleli.

Il secondo capitolo descrive alcuni ciruiti di ordinamento "special purpose", cioè circuiti che implementano direttamente, per il tipo di processori usati e per il modo in cui sono collegati, particolari algoritmi paralleli di ordinamento. Gli algoritmi presentati sono l'algoritmo di enumerazione, il merge pari-dispari, il merge di sequenze bitoniche.

I successivi sei capitoli presetano algoritmi di ordinamento per macchine SIMD con sei diversi tipi di interconnessione: ad array lineare, shuffle perfetto, mesh, ad albero, a ipercubo ed infine a memoria condivisa. Agli algoritmi asincroni su macchine MIMD è dedicato il nono capitolo, mentre il decimo tratta il problema di ordinamento esterno, cioè relativo al caso (di enorme importanza pratica) in cui i dati da ordinare non possono essere trasferiti tutti insieme nella memoria centrale, ma devono essere mantenuti in una memoria di massa.

L'ultimo capitolo infine è di carattere più teorico, essendo dedicato al problema di determinare gli estremi inferiori di complessità per l'andamento in parallelo, cioè di stabilire i limiti oltre i quali non si può sperare di accelerare la soluzione del problema con nessun algoritmo ipotizzabile.

Oltre alla chiarezza espositiva, un pregio del libro è la presenza di una ricca bibliografia. Un unico neo: la scarsa attenzione prestata alla tecnologia VLSI, ed alla possibilità di implementare gli algoritmi descritti con circuiti VLSI.

Giancarlo Mauri