di Raffaello De Masi

Le tecniche e i campi di utilizzo di un sistema esperto

Dopo aver descritto le parti più importanti e la struttura costruttiva e operativa di uno dei più complessi sistemi esperti, Rosie, vediamo in base a quali qualità e meriti esso è divenuto in breve tempo uno dei più efficienti e imitati SE presenti sul mercato.
Rosie è stato usato come linguaggio base per implementare applicazioni del tutto diverse l'una dall'altra. A titolo di esempio citeremo solo alcuni dei più interessanti e complessi sistemi esperti generati (e volutamente escluderemo da questa elencazione i SE di medicina, che la fanno da padrone in questo campo, e che non citeremo proprio per non cadere nel luogo comune, piuttosto diffuso, che SE=ambiente di analisi medica)

Esempi di applicazione di Rosie sono ArtStat, un modello decisionale applicato ad ambiente legale (Waterman e Peterson, 1981), TATR (Tactical Air Target Recommender, letteralmente consulente di attacco aereo tattico) e, cosa molto più complessa, la progettazione di Adept, una workstation sperimentale destinata a cooperare con analisti di decisioni tattiche in operazioni di combattimento terrestre.

Il primo caso è particolarmente interessante in quanto evidenzia la possibilità di usare tecniche di strategia d'uso di regole informali in un campo piuttosto specializzato e ristretto come quello delle consulenze legali. Nel secondo caso TATR, progettato su richiesta e con esigenze specifiche della Tactical Air Force, pianifica l'uso di aerei tattici in una azione d'attacco. Il sistema verifica e determina due scelte principali; esso indica quali aerei utilizzare e quali bersagli attaccare in base a parametri iniziali e opzioni diverse inseriti dall'operatore e permette di prevedere inoltre le perdite e di adottare le migliori strategie per ridurre i danni prodotti dal nemico.

La workstation Adept, infine, sviluppata dalla Rank in collaborazione con il sistema di difesa TRW è stato forse il progetto più ambizioso (e riuscito) per aiutare gli analisti militari e prevedere le attività nemiche, usando rivoluzionarmente una tecnica d'analisi quanto più possibile vicina a una struttura intelligente.

Si è adottato, in questo sistema, un avanzato uso di tecniche euristiche, e di basi di conoscenza costruite su di esse. Il progetto, più che realizzare la workstation stessa, aveva lo scopo di verificare la fattibilità stessa dell'hardware e del software dedicato, e di dimostrare la utilizzabilità di tecniche di intelligenza artificiale in questo dominio piuttosto atipico. Il progetto, prima della sua realizzazione, mise a punto una dimostrazione basata su un esempio di lavoro rappresentato da circa 200 regole programmate mediante una accorta combinazione di tecniche derivate da Rosie e dal linguaggio Lisp.

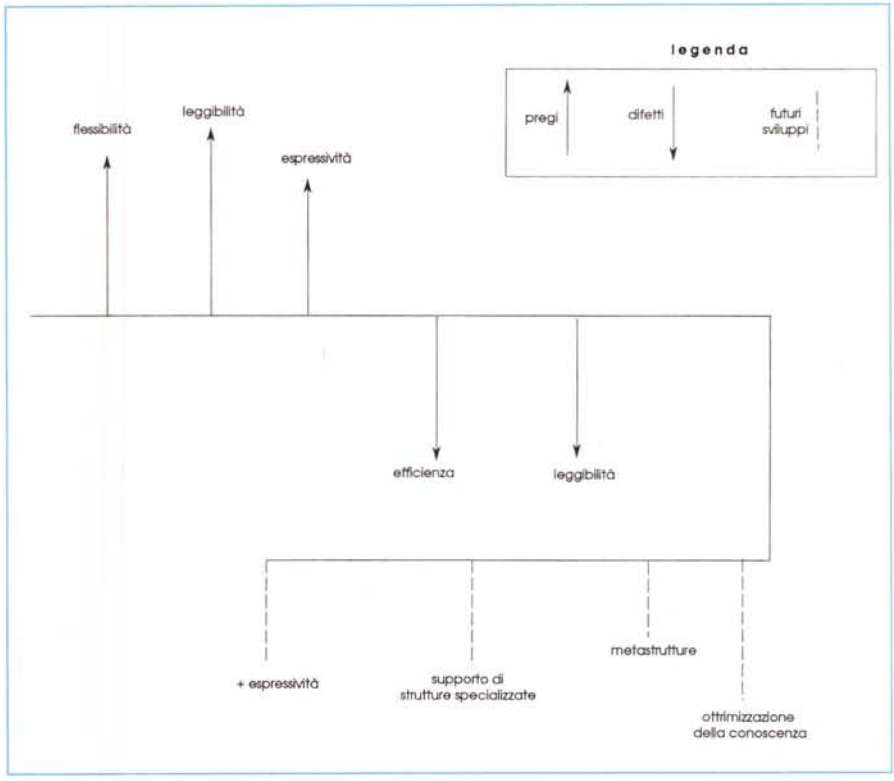
Vantaggi e svantaggi della implementazione di Rosie

Rosie, nella sua implementazione originale, fu sviluppato in un periodo di quattro anni, dall'idea originale alla release 1.0. Tutto questo tempo fu però sufficiente a creare un prodotto ben articolato, abbastanza libero da errori, veloce, soprattutto, ben noto, per quanto attiene ai vantaggi e agli svantaggi del tipo di implementazione stessa.

Dei vantaggi del tipo di implementazione Rosie, abbiamo parlato più volte; riassumendoli qui essi possono essere così riuniti: leggibilità, flessibilità e facilità d'espressione.

Il maggior vantaggio, lo abbiamo detto diverse volte, è rappresentato dalla leggibilità; un ingegnere della conoscenza può scrivere una applicazione in modo che un utente appena specializzato in questa area applicativa può leggerla in maniera immediata e facile; questo però, più che un merito del linguaggio stesso è frutto anche di uno sforzo del programmatore stesso; è altrettanto facile scrivere un codice sorgente del tutto incomprensibile, così come lo è in altri linguaggi che godono fama di facilità, comprensibilità e immediatezza, come Basic, Pascal o Logo. Dalla facile leggibilità discende in maniera pressoché immediata la possibilità di scrivere applicazioni facilmente espandibili, con aggiunte al codice di base dirette e immediate, ben integrate nel corpo principale che rappresenta il codice di struttura su cui innestare le aggiunte necessarie. È appunto merito di un buon ingegnere della conoscenza implementare un codice di base ben scritto, regolare e ben strutturato, cui fare aggiunte non costi più di un piccolo sacrificio in termini di tempo e sforzo applicativo. In altri termini una struttura di base scritta in maniera Rosie è, se ben redatta, elastica tanto da accogliere in maniera estremamente efficiente modifiche diverse, magari applicate a basi di conoscenza differenti.

Che la leggibilità e la facile comprensibilità del codice sorgente sia una delle



Le caratteristiche e le nuove prospettive di Rosie.

caratteristiche più importanti è sempre stato un cruccio dei programmatori di sistemi esperti. Ogni tool o metodologia che migliora le tecniche di comunicazione tra la struttura di base del programma e i domini che esso maneggia rappresenta lo scopo principale, ben più che le assurde e talora insulse corse, che si sono viste negli ultimi tempi, sulla velocità. Una facile leggibilità e una certa elasticità nel redigere lo schema e il file sorgente attenua molto le difficoltà di implementare strutture sintattiche e algoritmiche molto complesse. Inoltre, l'avere a disposizione un background idiomatico facilmente leggibile anche da un non esperto consente l'accesso, alla redazione del programma, di esperti di diverse branche, appena a conoscenza di un poco di tecnica e struttura programmatoria.

Il secondo vantaggio è la flessibilità; i modelli di rappresentazione tendono a essere usualmente scritti, nei linguaggi tradizionali di programmazione, usando strutture logiche fortemente nidificate e strutture di dati costruite per servire queste in maniera abbastanza univoca. Tutto questo porta a gravi problemi, rappresentati essenzialmente da notevoli difficoltà quando si desidera effettuare modifiche all'algoritmo di base o comunque alla struttura fondamentale del programma della base di dati. Al contrario Rosie (e tutti i successivi linguaggi destinati alla produzione di sistemi esperti) permette, e anche incoraggia lo sviluppo di programmi altamente modulari che non impediscono una considerevole flessibilità quando su di essi si interviene per consentire una espansione ragionata.

La struttura principale dei programmi scritti in Rosie è rappresentata da un set di regole condizionali (come non potevano essere in un sistema esperto?); la differenza che Rosie determina è rappresentato da un blocco di poche regole strutturate a grappolo, più complesse da realizzare, leggere e, eventualmente correggere o modificare, e da un'ampia messe di regole «piatte», composte di semplici fatti o di condizioni poco complesse. Questo tipo di struttura permette di individuare facilmente i punti di innesto di nuove regole, destinate ad ampliare e modificare la struttura di base; il risultato finale è una struttura di base molto «scheletrica», che permette di modificare o ridisegnare completamente un programma senza «buttar via» nulla, senza, cioè, disperdere lavoro (routine, oggetti, regole, fatti) che possono essere riutilizzati (è un poco il concetto della programmazione object oriented, trattata su altre pagine di questa rivista).

L'ultima dote, la capacità espressiva, si basa sulla caratteristica principale di Rosie: ricchezza e complessità del linguaggio stesso. Di pari passo con la chiarezza, un comando o una serie di regole possono essere espressi in modo diverso pur portando allo stesso risultato finale. È come dire che è più facile, espressivo e piacevole parlare in lingua umana che non in linguaggio di programmazione; non solo, ma una particolare facility presente in Rosie permette addirittura di «inventare» nuove espressioni e modi di dire, del tutto simmetrici ed equivalenti a quelli implementati come patrimonio di base del linguaggio.

Come per tutte le cose della vita, anche in Rosie non tutto è rose e fiori; esistono degli svantaggi che gli stessi implementatori, con grande chiarezza e serietà professionale non mancano di evidenziare: il primo difetto di Rosie è legato alla sua efficienza, il secondo alla sua «scrivibilità».

Rosie è scritto in Interlisp, una vecchia versione di Lisp piuttosto lenta e non priva di limiti; tutto questo porta a rendere difficoltoso e talora virtualmente impossibile lo sviluppo e il test di modelli complessi. Rosie fu sviluppato, per la verità, all'inizio come un esperimento; come tale gli implementatori non tennero in nessun conto la velocità e la grandezza del codice; il loro principale obiettivo era quello di creare una «macchina» per lo sviluppo di sistemi complessi. Questa mancanza di attenzione ai due parametri precedentemente evidenziati portò a modelli il cui sviluppo, debug e messa su strada risultava estremamente frustrante e tedioso. L'influenza di Interlisp, che già di per sé è un sistema esperto, e che qui doveva a sua volta generare un successivo sistema esperto e pesante in ogni momento della costruzione del programma, dalla costruzione dello scheletro ini-

ziale fino alla stesura del codice finale.

Klahr e Waterman, gli autori di Rosie, consapevoli dei limiti della loro creatura, riscrissero completamente in linguaggio PSL (Portable Standard Lisp), la loro creatura; e questo fu un bene, non solo in termini di accresciuta facilità d'uso del programma stesso, ma anche perché questo linguaggio era ed è molto più diffuso dell'altro. Vantaggio non trascurabile, alla fine, fu anche un certo miglioramento nella velocità di esecuzione del programma stesso.

Il secondo difetto è come dicevamo, la ridotta scrivibilità; questo sembrerebbe una contraddizione in termini con quanto detto in precedenza circa la leggibilità, ma l'affermazione va considerata solo in funzione della difficile ricerca nel trovare appropriate forme linguistiche per alcuni concetti. Forse la sua notevole somiglianza con la lingua inglese, a volte, si trasforma in un difetto; l'utente, spinto dalla errata convinzione che tutto, sotto forma di frasi simili alla lingua parlata, gli è permesso, è spinto a lasciar andare un poco le briglie delle strutture linguistiche, con il pericolo di trovarsi con frasi che Rosie non capisce. Alcune regole presentate dal linguaggio stesso e facili da ricordare sono state introdotte per limitare questa eccessiva «libertà» di espressione nella programmazione, questo ha portato alla fine a rendere un poco più rigido il problema, che se da una parte hanno limitato un poco l'espressività del linguaggio stesso, hanno dall'altra imposto una certa rigidità, e conseguente necessità d'ordine, che non ha affatto nuociuto alla leggibilità del sorgente, ma ha migliorato di molto le tecniche di scrittura del programma stesso.

I futuri sviluppi di Rosie

Dopo la prima release, accolta con interesse e successo nel mondo dell'A.I. Rosie è stato oggetto di una serie di migliori e upgrading di notevole qualità e livello. Gli sforzi espressi dagli implementatori sono stati orientati verso quattro categorie: ancora espressività, supporto di strutture specializzate, metacomandi (metalinguistici, metastrutturali e di metacontrollo), e ottimizzazione della conoscenza di base.

Riguardo alla espressività si trattava di portare avanti lo sforzo già realizzato in parte; si trattava infatti di rendere per quanto più possibile, accettabili le forme della lingua inglese, sintattiche e grammaticali. Ovviamente nessun linguaggio informatico può coprire tutta la lingua parlata. Così alcune forme linguistiche, come i verbi passivi, sono esclusi dal vocabolario e dalla comprensione del linguaggio Rosie. Si tratta quindi di costruire forme accessorie che, senza adottare la forma passiva, siano facilmente comprensibili da Rosie stesso, e non modifichino il significato iniziale dell'ordine o della frase. Inoltre frasi non esprimenti concetti certi, come «A po-

Sulla debolezza della mente umana

Chi usa correntemente il Macintosh si sarà sicuramente chiesto come fanno programmi come Draw, MiniCad, o meglio Claris Cad a contenere, in un file di pochi K, una messe anche estesa di figure complesse. La deduzione più logica è che nel file sia rappresentata tutta la pagina, punto per punto, dell'intero foglio di lavoro (così come, in effetti fa, uno scanner quando legge una pagina di scritto o di disegno); ma poi ci viene un dubbio; riapriamo il documento e sbattiamo un elemento del disegno stesso in quarta pagina; secondo la logica, se il programma non fa che registrare tutti i punti del documento, lo stesso disegno dovrebbe occupare quattro volte spazio; con lo stesso ragionamento, un foglio pieno di figure e uno contenente una sola, magari occupante la intera pagina, dovrebbero avere la stessa grandezza, o meglio, lo stesso «peso».

Invece non è così, per nostra fortuna. D'altro canto, se così fosse, non saremmo più nella possibilità di modificare e spostare oggetti sulla pagina alla riapertura del file stesso. Ci deve quindi essere una differente forma di cofidica.

Il lettore ci perdoni questa digressione, che forse gli sembra banale (e magari orientata a uso e consumo del mio amato Mac), ma volevo solo evidenziare che una macchina, quando salva un file di tal fatta, non lo fa certo per pixel; usa invece una codifica, uno schema di memorizzazione, che sia nel contempo meno dispendioso in termini di memoria e molto più efficiente. La macchina usa (ed è una delle rare volte) una forma di codifica e rappresentazione molto simile a quella di un cervello umano.

L'uomo memorizza la forma degli oggetti senza tener conto della loro effettiva dimensione fisica ma applicando se così si può dire, tecniche di database agli oggetti che sta analizzando; il cervello umano quando guarda un oggetto o una figura, sia essa un semplice quadrato, o Mozzafiato Mahoney che segue Dick Tracy, riempie una serie di campi in una scheda immaginaria (un record) che automaticamente si crea in mente; così un record «Ferrari» sarà rappresentato dal campo modello (che poi comprenderà altri record, come posizione dei fanali, numero delle porte, dimensione delle ruote, ecc), colore, accessori (magari compresa Mozzafiato!), e così via. Un semplice rettangolo sulla carta, al contrario sarà rappresentato dalla disposizione della figura (TL, BR, per usare una notazione cara ai linguaggi di programmazione) e dalla forma; il resto è inutile e forse sovrabbondante. È così che i vari oggetti sono conservati nel file MacDraw; efficienza, rapidità e scarse possibilità d'errore.

La forma con cui l'informazione visiva viene memorizzata dall'uomo è tutto fuorché la semplice immagine stampata sulla retina; ogni oggetto presente nella nostra mente è il risultato di un lavoro di codifica istantaneo ancorché immediato, che viene eseguito talora inconsciamente. A dimostrare tutto questo concorre un esperimento eseguito da Jean Hayes (così come riferito da Michie e Johnston, opera diverse volte citata) su una classe di un asilo infantile.

Hayes mostrò a una bambina di tre anni e mezzo un quadrato, orientato a 45° e chiese alla bambina di ridisegnarlo. Questa disegnò, la prima volta, qualcosa di simile alla figura a1, ma dopo qualche minuto, ad una anloga richiestra, tracciò il disegno di figura a2; alla richiesta del motivo per cui avesse disegnato questa nuova forma, la bambina si giustificò dicendo testualmente di aver dimenticato, nel primo disegno, di mettere «le cose dure», le «cose che vanno su e giù» e i pezzettini verticali.

Sono rimasto estremamente affascinato dall'esperimento e ho provato a fare lo stesso con mio nipote Francesco, della stessa età; devo dire di essere stato impressionato dalla somiglianza dei risultati (anche se ovviamente non ha usato le stesse parole per chiamare i lati e gli spigoli). Anche lui, al terzo tentativo del disegno (sono giustificato dal fatto che non sapevo quanto tempo era intercorso, nell'esperimento originale, tra il primo e il secondo disegno), dopo circa due ore, ha tracciato il disegno di figura b3, che, con le dovute approssimazioni, può essere ritenuto simile a quello del test di Hayes

Per tutto ciò Hayes fornisce una spiegazione del tutto plausibile, ma che dimostra come certi schemi rappresentativi e strutturali siano del tutto innati nella mente umana; egli racconta che il bambino, sollecitato a giustificare il motivo della sua rappresentazione abbia lasciato stranamente intendere come il suo disegno sia la rappresentazione schematica del disegno in termini dei suoi elementi costitutivi, non solo sotto il punto di vista fisico, ma anche per così dire semantico; così «le cose che vanno su e giù», secondo la ricostruzione di Hayes, sono i lati, le «cose dure» sono gli angoli e i «pezzettini laterali» sono i lati. Nel disegno di Checco il risultato è lo stesso, se si tien conto dello scopo che ogni pezzo del disegno ha nel contesto generale. Fatto sta che quando ho parlato con mio fratello delle analogie tra il mio Mac e suo figlio, mi ha guardato con fare strano, dan-

trebbe amare B» o futuri, come «A amerà B» non sono facilmente implementabili in ambiente Rosie.

Per quanto invece attiene alle strutture specializzate, il primo scopo perseguito nelle successive release di Rosie è stato e sarà quello di comprendere strutture specifiche di diverse discipline, che possano essere agevolmente adottate e utilizzate dal linguaggio. Un notevole sforzo è stato adottato, ad esempio, per permettere la utilizzabilità di tabelle decisionali, o di implementare situazioni in termini di oggetti (vedi puntate del giugno e luglio '90), proprio per permettere di raggiungere quella compattezza di codice che come abbiamo detto, Rosie, fin dall'inizio, non si è preoccupato di avere.

Una terza area di sviluppo è senza dubbio quella dei metaoggetti, strutture metalinguistiche che permettono di migliorare l'efficienza e, talora, la espressività del codice. Inoltre i costrutti metalinguistici permettono di disporre di un potente tool di programmazione, l'autoriferimento. I costrutti metalinguistici permettono, appunto l'autoriferimento... Il risultato finale di questa filosofia è la possibilità di disporre di linguaggi autoconfigurantisi e automodificantisi. Al limite, un buon metalinguaggio permette all'utente di scrivere nuovi monitor e di controllare motori inferenziali sulla base di un set di regole precostituite.

L'ultima area di sviluppo potrebbe essere la ottimizzazione delle basi di conoscenza, eterna croce dei sistemi esperti. Rosie, in certe applicazioni che adottano basi di dati piuttosto estese, rallenta in maniera esasperante; l'adozione di metacomandi aggiunge nuovo peso, rallentando ancora di più l'andatura, già certamente non veloce del carrozzone. Per correggere questo difetto, Waterman & C. hanno adottato la tecnica di sviluppare metodi che non solo ottimizzano il codice interno (in termini di velocità ed efficienza del costrutto), ma anche il codice che l'ambiente genera quando Rosie compila un programma in PSI (che, al contrario del classico LISP, è un linguaggio compilato). Un'altra tecnica, quella della valutazione simbolica, permette di articolare e trasformare le chiamate a funzioni che hanno almeno una chiamata a un argomento costante in una equivalente chiamata a una nuova funzione, che non ha più bisogno dell'argomento costante. Questo processo, noto appunto come valutazione simbolica, genera un codice eseguibile molto più rapidamente, anche se talvolta a danno di una maggior quantità di memoria occupata. Tutto sta, quindi ad adottare un uso giudizioso in questa tecnica, incrementando la velocità di esecuzione senza sacrificare molto la disponibilità di memoria.

La sola valutazione simbolica non può permettere, da sola, grandi vantaggi in termini di velocità; può invece farlo se abbinata in maniera efficiente con le tecniche precedentemente descritte (primi tra tutti i metacomandi). Ma qui non interviene più la potenza di linguaggio, ma solo le potenzialità e l'abilità del programmatore.

Conclusioni

Termina qui il nostro discorso su Rosie, un linguaggio da sistema esperto non nuovo, ma che tuttora rappresenta un punto di riferimento per chi si avventura sulla ardua strada dei sistemi esperti. Abbiamo visto come i suoi punti di forza siano una struttura linguistica molto simile alla lingua inglese, che permette di costruire prototipi e ambienti di sviluppo di grande potenza ed efficienza. Merito di Rosie è stato quello di mostrare come le tecniche di analisi sintattica possano permettere di «umanizzare» linguaggi di programmazione anche estremamente specializzati. Con questo sistema anche argomenti non semplici divengono alla portata di chi possiede basi addirittura non specialistiche. Infine il maggior difetto di Rosie, la sua lentezza, pur rimanendo il suo vero punto debole, è stato affrontato in diverse occasioni, e, oggi, il problema è parzialmente risolto, in parte migliorando la struttura stessa del linguaggio, in parte ricorrendo a tool programmatori più sofisticati e potenti.

Abbiamo così concluso il nostro iter in uno dei più vecchi ma ancora efficienti linguaggi-ambiente per Sistemi Esperti. Ancora oggi Rosie rappresenta un punto di riferimento per chi desidera affrontare la programmazione e la costruzione di un sistema esperto.

La prossima volta parleremo di un altro linguaggio, tanto più specialistico quanto più Rosie era generale; «Ross», un ambiente fatto apposta per maneggiare simulazioni.

MC

do chiari segni di non essere molto soddisfatto.

Somiglianza della macchina all'uomo? Sembrerebbe di sì se, da come vediamo, sembra emergere che il cervello memorizza secondo schemi e modelli estremamente simili a quelli con cui giocherellano gli ingegneri della conoscenza nei loro sistemi esperti (ricordate le forme elementari di cui dicemmo negli articoli «See, per vedere?»); ma quali sono le aree del cervello deputate alla schematizzazione e alla conservazione di queste informazioni? Lo vedremo la prossima volta.

◀ *Figura a - Rappresentazione di un quadrato (da Haynes e Piaget).*

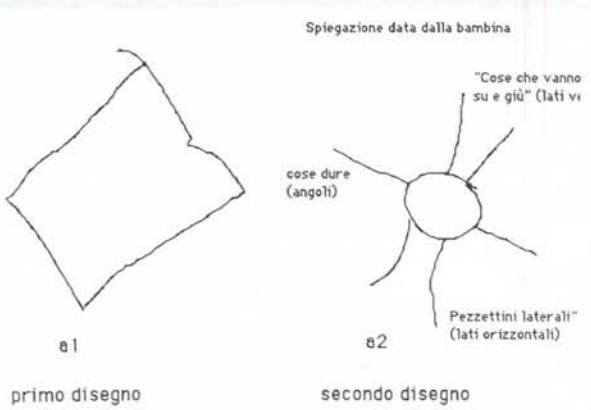


Figura b - Sequenza di disegno eseguita su un esperimento dell'autore; il terzo disegno è stato tracciato dopo circa due ore dal primo. ▼