di Raffaello De Masi

La struttura e la filosofia di Rosie

Rosie, ancora Rosie. Per essere onesti, questo ambiente ha determinato, probabilmente, nel campo della costruzione dei sistemi esperti, la stessa rivoluzione che il Basic ha avuto nei linguaggi convenzionali di programmazione. E questo per motivi, anzi per meriti, che, con le dovute proporzioni, sono da ritenere analoghi: ambedue adottano per la redazione del sorgente regole di scrittura e sintassi molto simili all'inglese parlato; inoltre ambedue sono estremamente interattivi e, quindi, facili da tagliare e adattare alle specifiche richieste del programmatore-utente. È questo il motivo per cui Rosie ha avuto tanto successo nel campo della costruzione dei sistemi esperti ed è stato lo shell che, più di tutti, ha subito profonde e sostanziali modifiche. Proprio per questo diamogli una occhiata più da vicino

Per definizione e per sua vocazione principale, Rosie è un tool di carattere e struttura generale per costruire sistemi esperti; come tale esso è stato disegnato e realizzato in modo da essere utilizzabile con la maggiore elasticità possibile, e adattabile a qualunque tecnica risolutiva o qualunque tipo di organizzazione di dati. Un sistema esperto ben realizzato con Rosie ha la capacità di rappresentare costrutti logici e dati in maniera molto chiara ed efficiente. Il fatto inoltre che il codice sorgente sia facilmente leggibile facilita grandemente il contatto e la comunicazione tra l'esperto e lo sviluppatore del S.E.

Come riferisce Bagdikian nel suo «The Information Machines; Their Impact on Men and the Media», Harper & Row, New York, 1971, nell'impostazione di Rosie i progettisti si trovarono di fronte al sottile problema di favorire la leggibilità del codice sulla facilità di redazione del programma, o viceversa. Il problema non era peregrino in quanto, tanto più Rosie assomigliava alla lingua parlata, tanto più il linguaggio (ovviamente) diveniva difficile da implementare. Come prevedibile, la cosa fu risolta con un compromesso che purtroppo non fu felice in quanto da una parte non si raggiunse la completa implementazione di una lingua parlata, dall'altra proprio questa mancata completezza porta l'utente a non afferrare appieno quale è il limite preciso fin dove può arrivare con l'uso, appunto, di un discorrere tipico della lingua corrente.

Ciò può portare a qualche difficoltà nella redazione di programmi costruiti con Rosie, e forse la maggiore difficoltà dell'uso di questo ambiente è proprio avere sempre a mente quello che si può o non si può fare, confusi sovente come si è dalla familiarità e facilità d'uso dello shell.

In ossequio alle più attuali tendenze del trend evolutivo dell'A.I., Rosie ha potenza da vendere nella manipolazione e nell'organizzazione di grossi database, intesi come collezioni di dati ma anche di concetti. Esso infatti accetta:

- aggettivi, frasi condizionali e preposizionali, inclusioni, e esclusioni, e comunque clausole descrittive e propositive di set di argomenti.
- Classi di strutture, di concetti e di condizioni.
- Frasi redatte con schemi grammaticali anche abbastanza complessi, con verbi transitivi e intransitivi, con stringhe alfanumeriche significative, con predicati e complementi. Tanto per intenderci, Rosie consente di costruire programmi che comprendono frasi del tipo «Cosa si mangia oggi?» o «Che vediamo stasera in televisione?».

Come dicevamo, Rosie ha subito numerosi upgrading nel tempo; tra i più interessanti la possibilità che diversi programmi accedano e modifichino lo stesso database contemporaneamente, la possibilità di definire demon (letteralmente demoni, geni, ma per traslato folletti benefici), programmi precostruiti capaci di eseguire operazioni predeterminate in caso di eventi prefissati, e, ancora la capacità di costruire programmi con capacità di autoriferimento (possibilità di usare, modificare e cercare strutture sintattiche e linguistiche).

Un sistema esperto costruito con Rosie, come la maggior parte dei casi, consiste essenzialmente di un blocco di regole e di un database (si veda quanto abbiamo più volte detto nella rubrica dedicata al Prolog). Il blocco di regole rappresenta il motore inferenziale, la conoscenza prescrittiva che fornisce al sistema le regole intelligenti di cui ha bisogno per risolvere il problema. La base di conoscenza, invece, contiene la conoscenza descrittiva, i fatti in base ai quali può essere analizzato e risolto il particolare problema. In aggiunta, il programma può usare anche database multipli, sebbene sia consentito l'accesso ad uno solo alla volta.

Nel caso più semplice, un programma usa il gruppo di regole abbinato con il database attivo; in casi più complessi, può incorporare altri database, inattivi. La necessità di ulteriori database presenti può essere motivata da diverse

considerazioni; ad esempio, possono contenere dati relativi a punti di vista diversi dello stesso problema. Nel campo dei S.E. dedicati, per esempio, alla pianificazione industriale, uno può rappresentare dati relativi al leader del mercato, e altri quelli relativi alle caratteristiche dei concorrenti. Tanto più è efficiente e ben costruito il programma, tanto più esso è capace di passare da database a database, analizzare punti di vista e prospettive, valutare e quantizzare probabilità e valori euristici. Un altro punto di vista, un'altra tecnica, può essere rappresentata dall'analisi di una conoscenza generale, mentre altri blocchi di conoscenza rappresentano rami dell'albero principale in cui indirizzare eventualmente la ricerca. Un esempio di tal genere sono i sistemi esperti relativi alle discipline mediche, dove la grande messe di dati deve per forza di cose essere organizzata in basi di conoscenza interdipendenti, pena la impossibilità di analizzare in tempi brevi il labirinto di informazioni che starebbero in un database generale. Il principio è che dal sistema principale, attraverso opportune chiavi, vengono attivate opzioni relative ai database satelliti, con economia e rapidità di esercizio.

Rosie: alcune notizie storiche

La prima idea di Rosie nacque nei laboratori Rand nel 1979; fin da allora il principio che animò il tutto fu quello di costruire un linguaggio dalla elevata leggibilità (cosa che certo non si può dire di Lisp, allora imperante). Dotato di semplici regole semantiche, con una grammatica molto simile alla lingua parlata, e capace di esprimere concetti molto articolati senza eccessive complessità.

Per la verità Rosie non è nato dal nulla; esso discende dalle esperienze eseguite con Rita (Rand Intelligent Terminal Agent; simpatico questo tentativo di adottare nomi di donna). Rita implementava già in sé tutte le regole allora nascenti dei linguaggi rule-oriented, con il loro netto orientamento verso rappresentazioni di conoscenza attraverso ampi e articolati database. Rita applicò tali regole cercando di inserirle in un contesto programmatorio che possedesse tecniche molto vicine alla lingua parlata, capaci quindi di creare codici facili da leggere e da capire. Le regole, in Rita, erano definite usando una sintassi english-like, con una serie di opzioni piuttosto ristrette. Il database specifico manipolabile da Rita era piuttosto rigido, costituito da tre elementi fissi; oggetto, attributo, valore. Sebbene i risultati risentissero molto di queste restrizioni, è merito di Rita l'aver mostrato che una serie di forme, anche stilizzate, di strutture simili alla lingua parlata possono rappresentare in maniera adeguata una base di conoscenza.

Rosie non poteva, né fece a meno delle esperienze di Rita e, almeno all'inizio, si presentò come una logica evoluzione dei concetti propri di questa. Come prevedibile, nei propositi di realizzazione assumeva particolare importanza il superamento dei limiti sintattici e linguistici di Rita. Rita era stato sviluppato in linguaggio C, usando un PDP 11/45; la ridotta potenza di questa macchina e la sua modesta memoria rappresentò sempre la barriera contro cui le peraltro non eccezionali risorse di Rita si trovarono a cozzare. Ancora, per superare la difficoltà di maneggiare enormi masse di regole, Rosie introdusse il concetto di blocco di regole (ruleset).

Rosie fu inizialmente sviluppato usando il linguaggio Interlisp su una macchina DEC 20, adottando molte delle tecniche già presenti in Rita, tra cui il particolarmente efficiente blocco di routine di I/O; fu inoltre estesa la espressività linguistica del precedente linguaggio. Nel 1981 la versione definitiva di Rosie era completa, cosa che consentiva di cominciare a costruire le prime applicazioni. Inoltre la stessa versione (1.0) fu disponibile presso la clientela.

Dopo poco tempo ci si rese conto che le cose non andavano come previsto, visto che si ripresentavano, talora in maniera più sottile, i problemi del precedente linguaggio. Inoltre, in applicazioni avanzate, Rosie raggiungeva facilmente i limiti del DEC 20. Tutto questo portò, l'anno successivo, a far rivedere il tutto, hardware e software. Fu adottato un più potente sistema Vax 11/780, con linguaggio Vax-Interlisp, e, in alternativa, un potente Xerox SIP 1100 con linguaggio Interlisp-D. Tutto ciò portò a disporre di un amiente molto più potente, e Rosie passò rapidamente alla versione 2 che, sfrondata di alcuni bus e diversi orpelli ereditati dal vecchio Rita, fu distribuito all'utenza nella versione 2.3; si aveva così a disposizione un linguaggio davvero nuovo, molto compatto, con funzionalità e caratteristiche davvero elevate. Da questo punto in poi Rosie si avviò verso una brillante carriera (che dura tutt'oggi); gli sforzi per migliorarlo avanzarono su due fronti; da una parte si cercava di spostare l'asse di Rosie verso la portabilità, adottando il PLS (Portable Standard Lisp), disponibile su diverse macchine, e lo sviluppo di un compilatore, in linguaggio C. Su suggerimento dell'utenza furono aggiunti diversi miglioramenti, tra cui i metaelementi, i database condivisi, e i demoni.

Le caratteristiche del linguaggio

Parlare di Rosie è più semplice attraverso la voce degli implementatori principali, così come di esso ebbero modo di parlare in più riprese su una serie di riviste specializzate fin dal 1981. Lo scopo principale dei progettisti del linguaggio fu finalizzato a diversi obiettivi; il primo fu quello di creare un linguaggio che incoraggiava a scrivere un codice molto ben leggibile. Questo fu ottenuto, in linea di principio, adottando due criteri di disegno; la minimalità e la completezza. Apparentemente, e anche in effetti, i due termini sono in antitesi; da una parte la minimalità impone chiarezza e semplicità di espressione, evitando strutture di definizione complesse e articolate, dall'altra, l'esigenza di completezza non può rinunciare a un codice non proprio semplicissimo (non si di-

A.I. e scacchi

Come i lettori avranno notato, generalmente abbiniamo alla trattazione sui sistemi esperti qualche considerazione, aneddoto od episodio particolarmente interessante sull'argomento, anche per rendere un poco più leggero il tutto, che diverrebbe altrimenti un mattone da leggere. Si tratta talora di argomenti a favore, talora a sfavore delle tesi di efficienza che l'intelligenza artificiale sta tentando di guadagnarsi nel pur ampio campo dell'informatica avanzata, e che, speriamo, possano, per il loro carattere discorsivo, sfatare l'aureola di estraneità che queste tecniche, forse in parte a ragione, si portano appresso.

Nel campo dell'A.I. il tema degli scacchi rappresenta uno dei temi principe per sviluppare strategie e per studiare a fondo le strutture ad albero. Perché tutto ciò? Rispondono con un esempio D. Mitchie e R. Johnston, nel loro volume «Intelligenza artificiale e Futuro dell'uomo», opera già diverse volte citata su queste pagine. Nel campo dell'A.I. gli scacchi equivalgono alla Drosophila Melanogaster nella biologia. Questo piccolo insetto, dotato di una grande capacità di riprodursi e di un ciclo vitale di appena undici giorni, permette di studiare l'effetto di modifiche genetiche su generazioni successive senza dover attendere tempi eccessivamente lunghi. Gli scacchi, nel settore della simulazione e del problem solving, rappresentano una palestra insostituibile, racchiusa in un ambiente notevolmente circoscritto. Ma, sempre a proposito di aneddotistica, sentite, a proposito degli scacchi questo che segue.

Ad una convention della International Federation for Information Processing, organizzata a Toronto nel 1977, Ken Thompson (chi se non lui) dei laboratori Bell presentò un programma che giocava una partita di scacchi ridotta, rappresentata da una finale di donna e re contro re e torre.

Secondo la teoria degli scacchi e secondo una banale analisi della potenza dei pezzi in gioco, il giocatore di re e donna avrà, in un tempo più o meno lungo, la meglio su giocatore di re e torre. Si tratta di un assioma che, anche solo matematicamente, per effetto della semplice valutazione della potenzialità dei pezzi, è incontrovertibile. Thompson aveva istruito la sua macchina ad analizzare le posizioni dei pezzi sulla scacchiera, istruendola ad eseguire la mossa successiva in base a una tabella che raccoglieva tutte le possibili varianti del gioco, in quella posizione; questa tabella, dalle dimensioni enormi, era organizzata come un database e costruita secondo la tecnica «se i pezzi si trovano in questa posizione, esegui questa mossa». Il programma era costruito in modo da cercare di vincere, se giovava con lo schieramento di donna, e di sottrarsi quanto più possibile alla sconfitta, se giocava dalla parte della torre.

Alla convention erano presenti due campioni di scacchi, Hans Berliner, ex campione mondiale per corrispondenza, e Lawrence Day, campione canadese. Thompson sfidò i due campioni a giocare contro la sua macchina, ovviamente adottando per loro lo schieramento di donna. Con molto stupore e imbarazzo nessuno dei due riuscì a battere la macchina. Sebbene non avessero mai perso, durante la partita, la loro posizione di vantaggio.

Ambedue i campioni riferirono poi in vari scritti di essere rimasti sconcertati dalla strategia adottata dalla macchina stessa. Quasi mai questa adottava disposizioni e sviluppo di gioco logico (per gli scacchisti). Essi affermarono di aver visto condurre la difesa in un modo così bizzarro da non aver addirittura compreso cosa effettivamente la macchina avesse intenzione di fare. Ad esempio, chi gioca a scacchi sa che torre e re vanno sempre tenuti accanto in quanto l'uno difende l'altra e viceversa contro la eccezionale mobilità della regina avversaria. Berliner e Day si trovarono di fronte ad una macchina che, talvolta, divideva senza eccessivi problemi i suoi due pezzi, avendo trovato un intricato cammino di mosse che, senza mettere in pericolo il re, consentiva di procrastinare, attraverso questa tortuosa via, più a lungo l'inevitabile (ma che poi non ci fu) sconfitta.

I due campioni, da persone sportive e di spirito, chiesero a Thompson di far giustificare alla macchina certe sue mosse apparentemente incomprensibili. Alla domanda «Perché hai spostato la torre in quel punto?» essa rispondeva immancabilmente «Perché così dice la tabella». Mancava cioè alla macchina qualunque struttura concettuale, rappresentata in termini di scopi, opportunità, linee d'azione. La macchina, così come strutturata, non «rischiava», contrapponendo a tecniche euristiche le sue rigide strutture algoritmiche. Se la macchina fosse invece stata chiamata a battersi contro se stessa (o contro una costruita allo stesso modo) ci sarebbe stata l'inevitabile sconfitta finale dello schieramento torre-re.

L'argomento, comunque, è troppo allettante per lasciarlo cadere qui, ne parleremo ancora la prossima volta.

mentichi che si sta lavorando su sistemi esperti, ben più raffinati di un codice algoritmico). Il risultato fu, come al solito, dettato da un compromesso, onorevole tra le due parti, che portò ad un linguaggio di programmazione abbastanza ben articolato, capace di soddisfare numerose esigenze, cosa che neppure gli implementatori si aspettavano da un linguaggio destinato a scopi così particolari.

«Abbiamo usato la lingua inglese come la guida principale alla struttura grammaticale e sintattica del linguaggio» sono le parole di Henry Sowizral e James S. Kipps, i padri dell'ultima release di Rosie (così come riferiscono in R3246-ARPA, The Rand Corporation, Luglio 1985). Ciononostante, la lingua inglese (e, ovviamente, qualunque altra lingua) ha certe caratteristiche e costrutti che ben difficilmente possono essere acquisiti da un linguaggio di programmazione. Ciononostante l'eccezionale impegno profuso dagli implementatori permise l'accettazione di un gran numero di costrutti grammaticali e sintattici, come aggettivi e articoli, preposizioni, predicati nominali e verbi ausiliari. La vera limitazione nell'adozione totale delle tecniche della lingua parlata è rappresentata dalla impossibilità di afferrare il contenuto, il significato, di parole come i sostantivi ambigui o gli avverbi, di cui non afferra nemmeno la funzione. Inoltre Rosie possiede solo una conoscenza superficiale dell'inglese, nel senso che l'utente ha la responsabilità e il compito di assicurare la buona fusione della struttura elementare del linguaggio in possesso del sistema agli scopi che si prefigge di raggiungere.

Nel tentativo di raggiungere la massima leggibilità, gli implementatori non ebbero alcuna remora a modificare alcune delle strutture ritenute basilari nell'informatica. L'esempio più eclatante è la modifica del monolitico costrutto «IF condizione THEN azione ELSE azione alternativa», a favore della struttura «IF condizione, azione altrimenti azione alternativa», non proprio caratteristica di un linguaggio di programmazione. Ma al di fuori dell'esemplificazione, che pur faremo, il tutto si può compendiare nel principio: «Un programma va scritto come se fosse raccontato in una lingua parlata». Vedremo come ciò sia stato, nel caso, reso semplice.

Se te ne servissero 10.000 in un'ora...

.....Prova a contattarci.
Da diversi anni importiamo e distribuiamo supporti magnetici e data cartridge, soltanto delle migliori produzioni mondiali, in tutti i formati esistenti:
Floppy da 2.8'', 3'', 3.5'', 5.25'', 8''.
Data cartridge da 10 a 150 MB.

MEDIA DISK

di L. Antonelli

SONY PROLOK
Microforum Dysan
Verbatim Nashua
Central Point Software INC 3M

Specializzato in forniture a enti pubblici - scuole - università software house - computer shop.

ORARIO: 9-19 sabato 9-13

SPEDIZIONI ESPRESSE IN TUTTA ITALIA

00162 ROMA • VIA CIOCIARIA, 12 • TELEFONO 06/42.40.379 C.C.I.A.A. 653620 • MICOMEX 7012371