Linda

Linda è un linguaggio di coordinamento che estende i linguaggi tradizionali permettendone l'utilizzo nello sviluppo di applicazioni in ambiente distribuito.

Linda indipendente dall'architettura sottostante (a memoria condivisa o rete di calcolatori) e dal linguaggio sequenziale usato (C, C++, etc ...).

Linda implementa una <u>memoria associativa</u> logicamente <u>condivisa</u> da tutti i processi dell'applicazione, detta <u>*Spazio delle Tuple*</u>.

Spazio delle Tuple

Lo spazio delle tuple è un'area di memoria associativa condivisa da tutti i processi che compongono il sistema.

Si parla di memoria associativa perché le tuple vengono identificate tramite matching su una chiave piuttosto che utilizzare un meccanismo tradizionale di indirizzamento.

Sullo spazio delle tuple sono definite sei operazioni:

in(s)	rd(s)	out(t)
inp(s)	rdp(s)	eval(t)

Spazio delle Tuple
(Tuple)

Una tupla è una sequenza di campi con tipo:

(‘‘linda’’, 2, 3.14)
(sqrt(2), ‘‘a string’’, i, v)
(0, a)

Ogni campo pu contenere un valore di un qualsiasi tipo (semplice o strutturato) ammesso dal linguaggio sequenziale utilizzato.

Spazio delle Tuple
(Operazioni sulle Tuple)

out(t)	eval(t)	Generano nuove tuple
in(s)	inp(s)	Eliminano tuple preesistenti
rd(s)	rdp(s)	Leggono tuple preesistenti

Il parametro t di out() e di eval() è una *tupla*.

Il parametro s di in(), inp(), rd() e rdp() è un’*anti−tupla* (o template).

Un’anti−tupla è una sequenza di campi tipati che possono essere o dei campi valore (parametro attuale) o un campo indefinito (parametro formale) introdotto da ?

(?radix, ‘‘a string’’, ?j, 100)

Spazio delle Tuple
(Regole di Matching)

Si ha un matching tra una tupla t ed un'anti−tupla s se:

- **Il numero di campi di t e di s è lo stesso**
- **Si ha una corrispondenza tra i tipi di ogni campo di s e di t**
- **I campi valore di s sono uguali ai campi valore di t**

(?radix, ''a string'', ?j)

(sqrt(2), ''a string'', 100)

Spazio delle Tuple
(Operazioni di Generazione)

out(t):

la tupla t viene valutata all'interno del processo invocante e aggiunta allo spazio delle tuple; il processo che la ha generata continua la propria esecuzione.

eval(t):

crea implicitamente un nuovo processo (*tupla attiva*) per valutare ogni campo di t; terminata la valutazione di tutti i campi di t, t viene depositata nello spazio delle tuple.

Spazio delle Tuple
(Operazioni di Eliminazione)

in(s):

una tupla <u>t</u>, presente nello spazio delle tuple, che si accoppia con l'anti−tupla <u>s</u> viene eliminata.

Ai campi formali dell'anti−tupla <u>s</u> vengono associati i valori attuali della tupla <u>t</u>.

Se al momento della chiamata nello spazio delle tuple non esiste nessuna tupla che possa essere accoppiata all'anti−tupla <u>s</u>, il processo viene sospeso. Se esistono più tuple ne viene scelta una in modo arbitrario.

inp(s):

versione non bloccante di <u>in()</u>. Se esiste una tupla <u>t</u> che si accoppia con <u>s</u>, <u>inp()</u> si comporta con <u>in()</u> e ritorna 1. Altrimenti termina immediatamente e ritorna 0.

Spazio delle Tuple
(Operazioni di Lettura)

rd(s):

ha lo stesso comportamento di in() con la differenza che la tupla <u>t</u> selezionata non viene eliminata dallo spazio delle tuple.

rdp(s):

versione non bloccante di rd<u>()</u>.

Spazio delle Tuple
(Tuple Attive e Passive)

Lo spazio delle tuple può contenere:

- **Tuple attive**

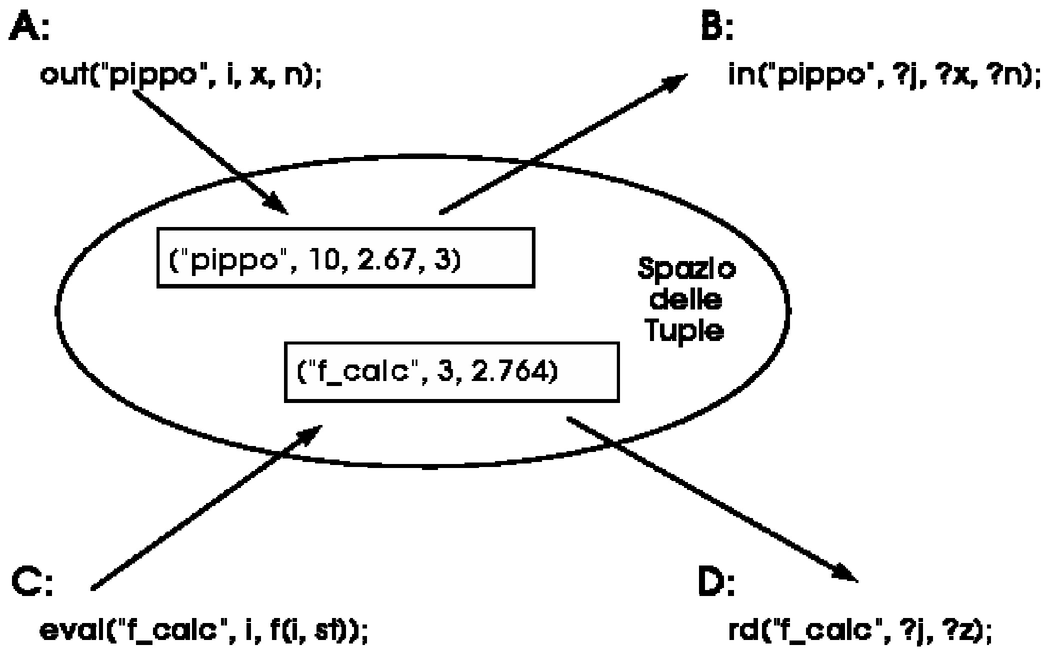
  Vengono valutate (eseguite) concorrentemente e si scambiano dati generando, leggendo e consumanto tuple passive. Una tupla attiva, quando termina la sua esecuzione diventa una tupla passiva.

- **Tuple passive**

  Dati riferiti associativamente dalle tuple attive.

Le tuple possono essere generate, lette o consumate, ma <u>mai</u> modificate. Questa proprietà semplifica l'implementazione dello spazio delle tuple.

Spazio delle Tuple



Cooperazione in Linda

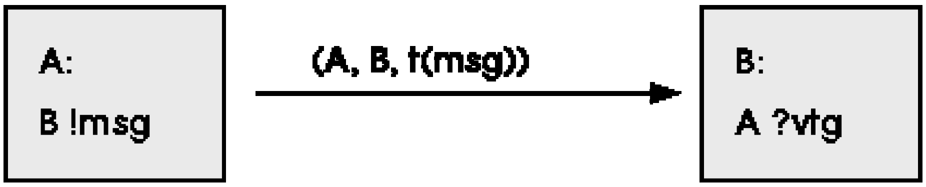
Utilizzando le sei operazioni primitive di linda è possibile emulare, attraverso lo spazio delle tuple, meccanismi di cooperazione tipici sia del modello a memoria distribuita che condivisa.

- **Semplicità di sviluppo e porting delle applicazioni**
- **Portabilità su piattaforme hardware differenti**

Scambio di Messaggi in Linda

(Comunicazione Simmetrica Sincrona)

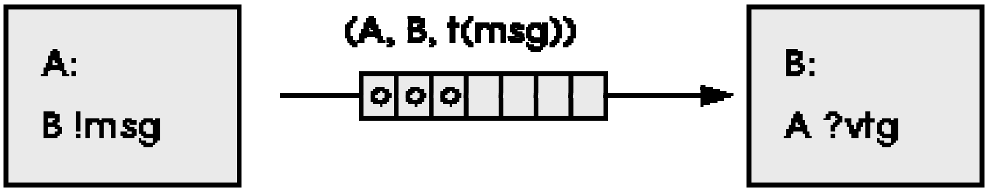
Il mittente del messaggio rimane bloccato fintanto che non viene effettivamente ricevuto dal destinatario.



A: out(msg, “A”, “B”)
in(“ack”)
B: in(?vtg, “A”, “B”)
out(“ack”)

Scambio di Messaggi in Linda
(Comunicazione Simmetrica Asincrona)

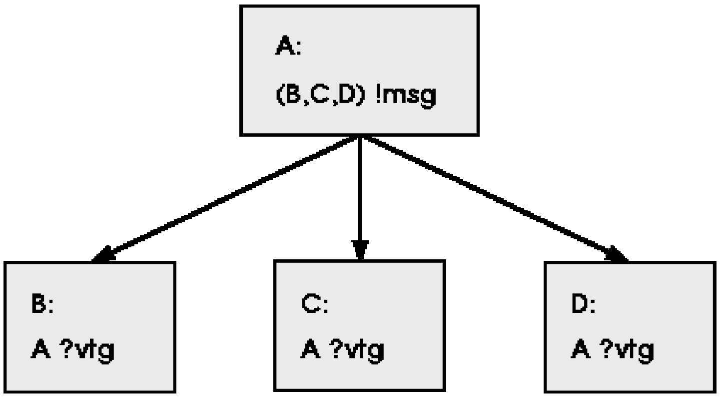
Il mittente manda il messaggio al destinatario senza attendere che effettivamente sia stato ricevuto.



A: out(msg, “A”, “B”)
B: in(?vtg, “A”, “B”)

Scambio di Messaggi in Linda
(Comunicazione Asimmetrica Sincrona 1 a n)

Il mittente manda il messaggio a più destinatari ed attende che tutti i destinatari lo abbiano ricevuto.

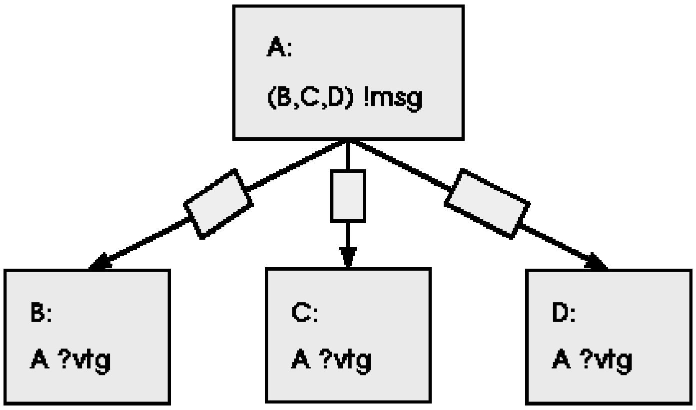


in(“ack”)
in(msg, “A”, 0)

out(vtg, “A”, m–1)
rd(?vtg, “A”, 0)
If (m==1) out(“ack”)

Scambio di Messaggi in Linda
(Comunicazione Asimmetrica Asincrona 1 a n)

Il mittente manda il messaggio a più destinatari senza attendere che di sapere che tutti i destinatari lo hanno ricevuto.



A:	out(msg, “A”)
B, C, D:	rd(?vtg, “A”)

Sincronizzazione in Linda
(Semafori)

Operazioni sui Semafori

out(“semaphore”)	Inizializzazione del Semaforo
in(“semaphore”)	Acquisizione del Semaforo (Primitiva P)
out(“semaphore”)	Rilascio del Semaforo (Primitiva V)

es. processo che usa un semaforo

A: in(“semaphore”)
<<sezione critica>>
out(“semaphore”)

Sincronizzazione in Linda
(Barriera)

Le barriere sono utilizzate per sincronizzare più processi in modo che riprendano la propria computazione simultaneamente.

```
A:      out("barrier", n−1)
        rd("barrier", 0)

Other:  in("barrier", ?val)
        out("barrier", val−1)
        rd("barrier", 0)
```

RPC in Linda

```
A:  eval("Server S", ServerS());

    ServerS() {
       int i, id;
       int s_calc(int j) { ... }
       while (true) {
           in("request", ?id, ?i);
           out("reply", id, s_calc(i));
       }

B:  int rpc_s(int i) {
       static int id = 0;
       out("request", id, i);
       in("reply", id++, ?i);
       return i;
```

Programmazione in Linda

Parallelizzazione di un loop con iterazioni indipendenti

```
for(int i=0; i<=n;i++)
    eval("this loop", something(i));

for(int i=0; i<=n;i++)
    in("this loop", ?v);
```

Array multidimensionali condivisi:

("Array Name", index1, ..., indexn, val)

Programmazione in Linda

Strutture Dati Dinamiche Condivise: Liste

("list", "head", 1)	Puntatore alla testa della lista
("list", i, element(i))	Elemento i–esimo della lista
("list", "tail", n)	Puntatore alla coda della lista

Inserire un elemento in coda alla lista:

```
in("list", "tail", ?index)
out("list", "tail", index+1)
out("list", index+1, new_element)
```

Cancellare un elemento dalla testa della lista:

```
in("list", "head", ?index)
out("list", "head", index+1)
in("list", index, ?element)
```

Cercare un valore all'interno della lista:

```
inp("list", ?index, val)
```

Programmazione in Linda
(Strutture dati vive)

Ogni elemento della struttura dati è incapsulato in un processo il cui compito consiste nel calcolare tale elemento.

```
eval("object A", compute_object(arg))
```

Effettuato il calcolo il processo (tupla attiva) termina rendendo disponibile il valore calcolato nello spazio delle tuple (tupla passiva).

```
rd("object A", ?var)
```

Crivello di Eratostene in Linda

```
main()  {
    int i, ok, limit;
    for(i=2;i<=limit;i++)
        eval("primes", i, is_prime(i));
    for(int i=2;i<=limit;i++) {
        rd("primes", i, ?ok);
        if (ok) printf("%d\n", i);
    }
}
```

```
int is_prime(int me) {
    int  i, limit, ok;
    limit = sqrt((double)me)+1;
    for(i=2;i<limit;i++) {
        rd("primes", i, ?ok);
        If (ok && (me%i == 0)) return 0;
    } return 1;
}
```

Implementazione

Le implementazioni richiedono efficienza nel pattern matching all'interno dello spazio delle tuple.

- **Preprocessing**

  A: out("x", i); B: in("x", ?j);

  x è costante durante la compilazione, quindi si mettono in contatto diretto (variabile condivisa o scambio di messaggi) i processi A e B.

  Problemi: con compilazioni separate o linguaggi interpretati.

- **Restringere la ricerca su una chiave singola**

  se le anti−tuple hanno una sola varaibile al loro interno, è possibile adottare l'hashing per la memorizzazione ed il recupero delle tuple, ottenendo l'accesso ad una tupla in tempo costante.

- **Spazi delle tuple multipli**

  lo spazio delle tuple non è centralizzato, la quantità di tuple da controllare diminuisce aumentando l'efficienza.

Riferimenti Bibliografici

- **Distributed Object−Based Programming Systems. Roger Chin and Samuel Chanson. In ACM Computer Surveys, Vol. 23(1), March 1991.**
- **Analysis of Inheritance Anomaly in Object−Oriented Concurrent Programming Language. Satoshi Matsuoka, and Akinori Yonezawa. In G. Agha, P. Wegner, and A. Yonezawa Editors Research Directions in OO Concurrent Programming. MIT Press. 1993.**