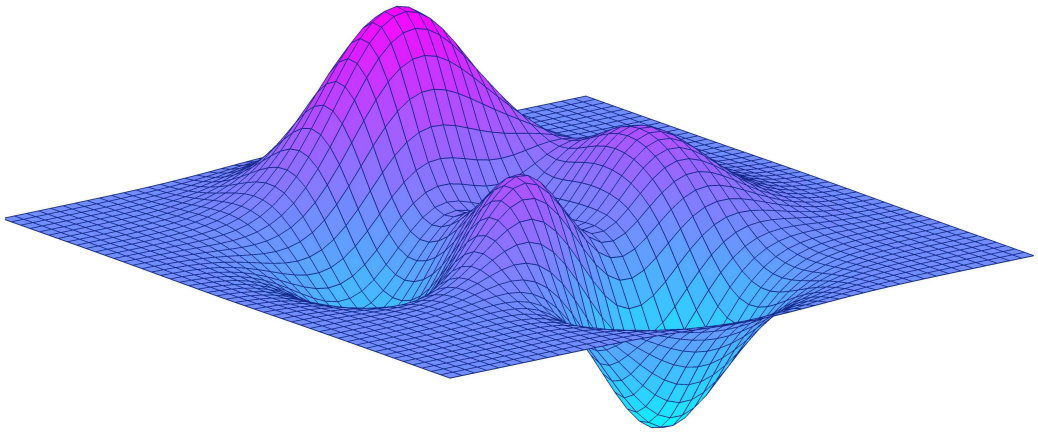
Università degli Studi di Bologna - II Facoltà di Ingegneria
Laurea Specialistica in Ingegneria Informatica
Sistemi Multi-Agente LS
A.A. 2008/'09



Co-Field & TOTA

una risposta al problema della coordinazione in sistemi multi-agente

Lucio Tudisco

16 Marzo 2009

1 Introduzione

La coordinazione - *the glue that binds separate activities into an ensemble* [1] - è una dimensione fondamentale dei sistemi concorrenti e distribuiti. L'esigenza di (meta) modelli di coordinazione è avvertita in particolare nell'*agent oriented software engineering*; ambito nel quale si manifestano in modo diffuso scenari che richiedono di ingegnerizzare il comportamento sociale di più entità autonome (*agenti*) cosicché esse possano condurre a termine collettivamente un desiderato task computazionale.

Affinché un modello di coordinazione possa essere efficacemente applicato in un sistema multi agente (MAS), esso dovrà supportare le proprietà di *apertura*, *dinamismo*, *complessità*, *eterogeneità tecnologica*, *distribuzione* ed *adattabilità* tipiche di tali sistemi.

I *Co-Field* - acronimo di *computational field* - rappresentano una tra le molteplici ed eterogenee risposte che la ricerca scientifica ha fornito al problema della coordinazione. *TOTA* - acronimo di *Tuples On The Air* - è un middleware applicativo che implementa il modello dei *Co-Field*.

Il modello dei *Co-Field* ed il middleware *TOTA* sono stati entrambi sviluppati in Italia nell'ambito del gruppo di ricerca *Agents and Pervasive Computing Group*[2] dell'Università degli studi di Modena e Reggio Emilia sotto la guida dell'Ing. M. Mamei e del Prof. F. Zambonelli. Un'completa descrizione di entrambi è disponibile nel testo [10].

La relazione introdurrà, senza alcuna pretesa di esaustività, gli aspetti peculiari del modello di coordinazione promosso dai *Co-Field* (*paragrafo 2*) e dal middleware *TOTA* (*paragrafo 3*). La discussione sarà arricchita da semplici esempi applicativi.

2 Campi computazionali

2.1 Il modello di coordinazione promosso dai *Co-Field*

I *Co-Field* sono strutture dati distribuite nello spazio fisico/logico popolato dagli agenti di un MAS che ne influenzano le percezioni e le azioni con l'obiettivo di soddisfare un requisito di coordinazione *application-specific*.

I *Co-Field* traggono ispirazione dal mondo della fisica ed, in particolare, dalla modalità con cui i campi di forze agiscono sul moto delle particelle. Si pensi, ad esempio, all'azione di un campo gravitazionale su una massa o a quella di un campo elettromagnetico su una carica.

Un *Co-Field* specifica per tutti i punti dello spazio in cui è definito un valore numerico ed una regola di propagazione. Il valore numerico rappresenta l'intensità del campo in quel punto, la regola di propagazione, invece, stabilisce la modalità con cui il campo si propaga nello spazio.

I *Co-Field* possono essere *statici* o *dinamici*. Un *Co-Field* è detto statico se, una volta propagatosi nello spazio, esso rimane immutabile nel tempo. Un *Co-Field* dinamico, al contrario, cambia nel tempo per effetto dei movimenti degli agenti o di regole di propagazione tempo varianti.

Secondo il modello dei *Co-Field*, ogni agente del MAS calcola una combinazione lineare dei campi computazionali che percepisce. Tale combinazione è detta *campo di coordinazione* dell'agente.

Gli agenti si muovono nello spazio seguendo il profilo della superficie del proprio campo di coordinazione. La figura 1 rappresenta un esempio di campo di coordinazione. A sinistra un agente si muove nel campo seguendo la direzione ed il verso del gradiente. Al centro, invece, l'agente si muove nel campo nella stessa direzione ma in verso contrario al gradiente. A destra, infine, l'agente si muove lungo una delle linee equipotenziale del campo. Quelle appena citate sono le tre principali modalità con cui gli agenti si spostano all'interno dei campi di coordinazione.

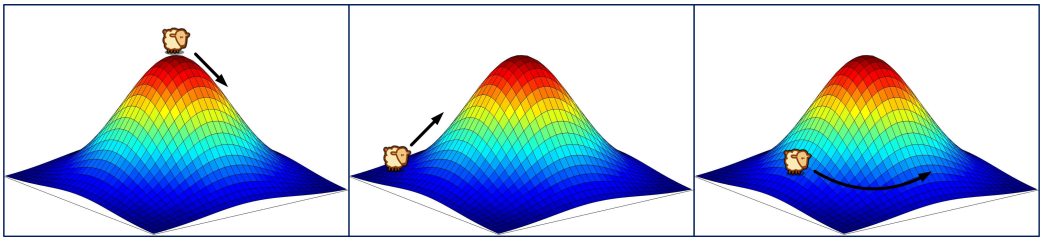


Figura 1: Movimento di un agente all'interno di un *Co-Field*. A sinistra l'agente si muove lungo la direzione crescente del campo, al centro lungo la direzione decrescente, a sinistra lungo una delle linee equipotenziali.

Muovendosi in un campo di coordinazione dinamico, gli agenti inducono dei cambiamenti nel campo che influenzeranno, a loro volta, i successivi mo-

vimenti degli stessi agenti. Si innesca in questo modo un ciclo di feedback che porta a fare emergere nel sistema meccanismi di autorganizzazione che consentono il raggiungimento dei task di coordinazione desiderati.

Alla luce delle precedenti considerazioni emerge con chiarezza la caratteristica dei *Co-Field* di promuovere una coordinazione di tipo stigmergico [4], ovvero, una coordinazione mediata da un ambiente condiviso che i singoli agenti percepiscono e modificano attraverso le loro azioni, influenzando reciprocamente i propri comportamenti. Gli agenti, pertanto, realizzano i propri obiettivi non in virtù di capacità individuali ma parchè parti di una collettività (auto) organizzata.

La natura autorganizzante della coordinazione - alternativa a quella a controllo centralizzato - rende l'applicazione del modello dei *Co-Field* particolarmente efficace in contesti aperti, dinamici ed impredicibili quali sono i sistemi di *pervasive computing* sempre più diffusi ai nostri giorni.

Evidenziamo che l'impiego dei *Co-Field* non è limitato a scenari che richiedono il movimento coordinato di agenti in spazi fisici o logici. Esso, tutt'altro, permette potenzialmente di coordinare gli agenti nell'esecuzione di qualunque tipo di azione. I *Co-Field*, infatti, si propongono come *framework* generale attraverso cui poter esprimere la totalità dei sistemi di *swarm intelligence*. Una prospettiva futura molto interessante, a questo proposito, è quella di sviluppare una metodologia che permetta di mappare una desiderata politica di coordinazione in uno specifico campo computazionale.

La dinamica di coordinazione promossa dal modello dei *Co-Field* può essere compiutamente descritta in modo analitico attraverso un sistema di equazioni differenziali alle derivate parziali.

Indichiamo con $x_1^i(t)$, $x_2^i(t)$,..., $x_n^i(t)$ le coordinate al tempo t di un generico agente $i-esimo$ in uno spazio $n-dimensionale$ e con $CF_i(X_1, X_2, , X_n, t)$ il suo campo di coordinazione. Il comportamento di tale agente $i-esimo$ è modellato analiticamente dal seguente sistema differenziale:

$$\frac{dx_j^i}{dt} = \pm v \frac{\partial CF_i(X_1, X_2, ..., X_n, t)}{\partial X_j} \qquad j = 1, 2, ..., n$$

Il termine v indica la velocità dell'agente. Il segno a destra del simbolo d'uguale discrimina, invece, la scelta dell'agente di muoversi lungo la direzione del gradiente del suo campo di coordinazione o in direzione opposta.

La possibilità di esprimere in modo analitico la dinamica di un sistema autorganizzante permette di analizzarne adeguatamente il comportamento e di regolarne opportunamente i parametri. La simulazione basata sui modelli analitici, d'altra parte, è - quando percorribile - la più efficace tra le tecniche di simulazione disponibili.

Notiamo, per concludere, come la dinamica di coordinazione che emerge in un sistema basato sul modello dei *Co-Field* sia completamente deterministica. I sistemi autorganizzanti, tuttavia, presentano notoriamente elementi probabilistici che ne garantiscono la robustezza e la flessibilità. È possibile arricchire il modello dei *Co-Field* in senso probabilistico prevedendo che gli agenti si muovano all'interno del proprio campo di coordinazione secondo la direzione concorde (o opposta) al gradiente con un'opportuna funzione di probabilità e/o prevedendo che all'interno del campo di coordinazione si propaghi del rumore che ne alteri l'intensità puntuale in modo aleatorio.

2.2 Caso di studio: Accerchiamento di una preda

Per chiarire le modalità di utilizzo dei *Co-Field* consideriamo il problema di coordinazione che consiste nell'accerchiamento di una preda da parte di un branco di n predatori.

In natura la strategia seguita dai predatori è quella di muoversi in direzione della preda cercando di mantenere al contempo un'opportuna distanza dagli altri componenti del branco.

È possibile modellare questa strategia nei termini dell'astrazione dei *Co-Field*. Supponiamo, in particolare, che la preda generi il campo $PreyField$, che ciascun predatore $i-esimo$ generi il campo $PredatorField_i$ e che tali campi siano aggiornati in tempo reale in funzione dei movimenti della preda e dei predatori.

Indicando con (X_m, Y_m) le coordinate della preda, il campo $PreyField$ da essa generato è espresso dalla seguente equazione:

$$PreyField(x, y, t) = -k_m e^{-h_m((x-X_m)^2+(y-Y_m)^2)} \qquad k_m, h_m > 0$$

In modo analogo, indicando con (X_{wi}, Y_{wi}) le coordinate del predatore

$i-esimo$, il campo $PredatorField_i$ da esso generato sarà espresso dall'equazione:

$$PredatorField_i(x, y, t) = k_w e^{-h_w((x-X_{wi})^2+(y-Y_{wi})^2)} \qquad k_w, h_w > 0$$

Le figure 2 e 3 rappresentano i campi computazionali generati rispettivamente dalla preda e da un generico predatore.

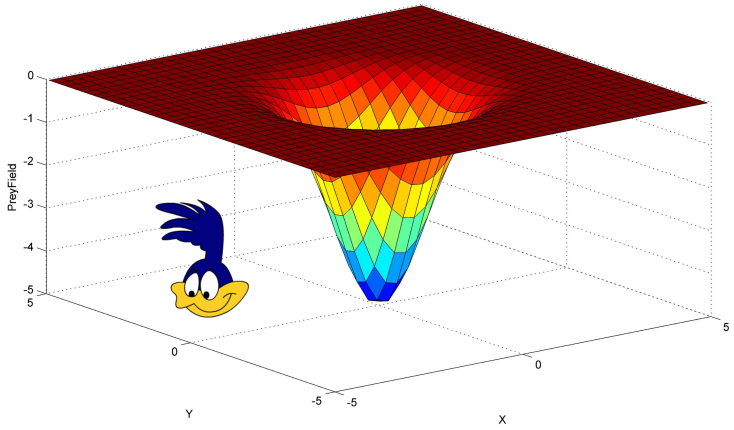


Figura 2: Campo $PreyField$ ($X_w = 0$, $Y_w = 0$, $k_m = 5$ e $h_m = 0.5$)

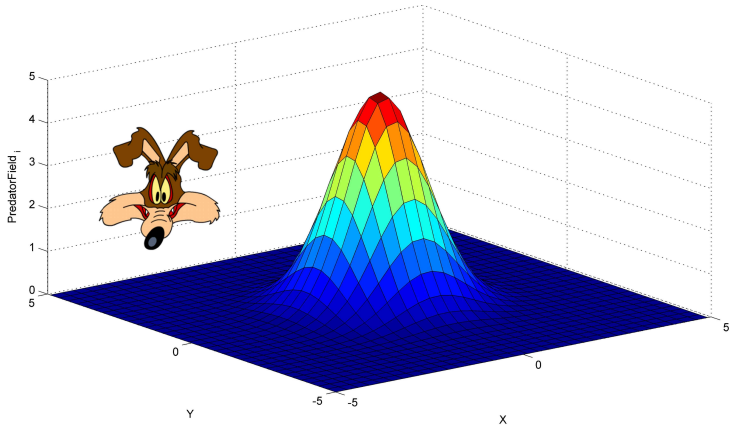


Figura 3: Campo $PredatorField_i$ ($X_{wi} = 0$, $Y_{wi} = 0$, $k_w = 5$ e $h_w = 0.5$)

Il campo di coordinazione della preda $PreyCoordinationField(x, y, t)$ è la combinazione lineare di tutti i campi dei predatori.

$$PreyCoordinationField(x, y, t) = \sum_{i=0}^{n} PredatorField_i$$

La preda fuggirà via dai predatori scendendo verso la regione a potenziale minimo del proprio campo di coordinazione.

Il campo di coordinazione $PredatorCoordinationField_i$ del generico predatore $i - esimo$ consiste, invece, nella combinazione lineare tra il campo della preda ed i campi degli altri predatori del suo branco.

$$PredatorCoordinationField_i(x, y, t) = PreyField + \sum_{\substack{j=0 \\ j \neq i}}^{n} PredatorField_j$$

I predatori accecheranno la preda scendendo verso la regione a potenziale minimo del proprio campo di coordinazione.

Quello appena discusso rappresenta solo uno tra i possibili esempi d'impiego dei *Co-Field*. Essi, infatti, possono essere agilmente applicati per modellare numerosi pattern di comportamento animale, tra questi citiamo gli stormi di uccelli, la ricerca di cibo e la suddivisione del lavoro delle formiche, la costruzione dei nidi delle termiti, etc. .

Le metafore naturali sono, in generale, rappresentative di problemi in ambito informatico ed ingegneristico. I comportamenti sociali di insetti ed animali costituiscono un ricco bacino di modelli e strategie a cui l'ingegneria attinge per la realizzazione di sistemi di swarm intelligence. Si pensi, ad esempio, all'utilizzo degli algoritmi di *Ant Colony Optimization* [5] (ACO) - ispirati al comportamento delle formiche alla ricerca di cibo - per la soluzione di problemi di network routing e, più in generale, di ottimizzazione combinatoria.

Ispirandosi ad alcune di queste metafore naturali ed adottando il modello di coordinazione dei *Co-Fields*,l'*Agents and Pervasive Computing Group* ha proposto alcune interessanti applicazioni in cui gruppi di individui dotati ciascuno di un dispositivo PDA si muovono in modo coordinato, ad esempio, per visitare un museo o per evitare ingorghi di traffico.

Per conludere il caso di studio, possiamo, come preannunciato nel precedente paragrafo, descrivere analiticamente la dinamica dell'accerchiamento

attraverso il sistema di equazioni differenziali alle derivate parziali riportato di seguito.

$$\begin{cases} \frac{dx_m}{dt} = \frac{\partial PreyCoordinationField}{\partial x_m} \\ \frac{dy_m}{dt} = \frac{\partial PreryCoordinationField}{\partial y_m} \\ \frac{dx_{w_i}}{dt} = \frac{\partial PredatorCoordinationField_i}{\partial x_{w_i}} \quad i = 1, ..., n \\ \frac{dy_{w_i}}{dt} = \frac{\partial PredatorCoordinationField_i}{\partial y_{w_i}} \quad i = 1, ..., n \end{cases}$$

È possibile risolvere tale sistema non lineare di equazioni differenziali alle derivate parziali ricorrendo ad opportune tecniche di integrazione numerica. Il grafico in figura 4 riporta le traiettorie nel piano (x-y), nel caso di una preda e di tre predatori, risultanti dalla soluzione del sistema. Tale grafico evidenzia, in particolare, come i predatori raggiungano la preda da tre direzioni diverse, proprio come desiderabile in una strategia di accerchiamento coordinata.

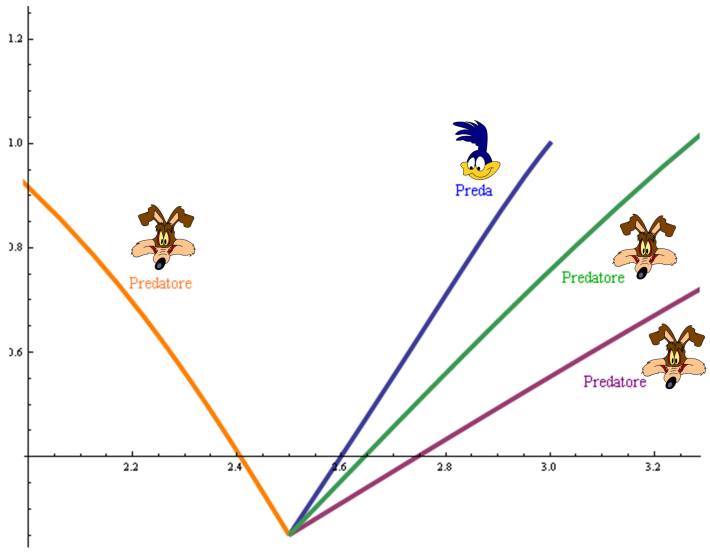


Figura 4: Dinamica dell'accerchiamento di una preda da parte di un branco di 3 predatori. Si noti come i predatori raggiungano la preda da tra distinte direzioni in modo coordinato.

3 Il middleware TOTA

3.1 Overview Generale

Il modello astratto dei *Co-Field* può essere potenzialmente mappato su implementazioni concrete tecnologicamente eterogenee. Ad una infrastruttura software che implementi il modello dei *Co-Field*, in particolare, sono richiesti la capacità di *storage* per la memorizzazione dei valori del campo ed un meccanismo di *event notification* per informare gli agenti sui cambiamenti dei campi per i quali hanno manifestato interesse tramite una preliminare procedura di *subscription*.

TOTA è il middleware di coordinazione svilupoato dall'*Agents and Pervasive Computing Group* come possibile implementazione del modello di coordinazione dei *Co-Field*.

TOTA - acronimo di *Tuples On The Air* - coniuga il paradigma *event-driven* con un'architettura basata su spazi di tuple. TOTA, in particolare, rappresenta i campi computazionali sotto forma di tuple in grado di diffondersi all'interno di una rete di nodi *peer-to-peer* secondo opportuni pattern di propagazione. L'interazione degli agenti in una rete TOTA si riconduce, quindi, all'inserimento ed alla percezione di tuple.

Ciascuno dei nodi della rete esegue un'istanza locale del middleware TOTA. Ogni nodo, inoltre, mantiene un riferimento ad un insieme di nodi vicini. Le relazioni di vicinato dei nodi determinano la struttura della rete. TOTA supporta strutture di rete dinamiche, ovvero, è in grado gestire i fenomeni di connettività ad intermittenza caratteristici di reti con nodi mobili e/o suscettibili a guasti.

Una tupla TOTA T è una tripla: $T=(C,P,M)$

- C rappresenta il contenuto informativo della tupla. Tale contenuto può variare da un semplice valore numerico fino a strutture dati arbitrariamente complesse.
- P è una regola di propagazione che definisce la modalità con cui la tupla si diffonde nella rete TOTA. Una regola di propagazione, in particolare, specifica la massima distanza fino a cui la tupla può propagarsi, la direzione della propagazione e le modifiche che subirà il contenuto informativo della tupla nel corso della sua propagazione.

Quando nuovi nodi vengono aggiunti alla rete, TOTA riverifica automaticamente le regole di propagazione delle tuple già presenti, propagandole eventualmente ai nuovi nodi.

Un semplice esempio di regola di propagazione è quella che prevede di propagare una tupla in tutte le direzioni, incrementando il valore numerico in essa contenuto ad ogni *hop*. Tale regola di propagazione permette di diffondere nella rete un campo che esprime la distanza di ogni nodo dalla sorgente del campo medesimo.

- M è una regola di mantenimento che specifica le reazioni della tupla agli eventi che occorrono nell'ambiente. Tra gli eventi ai quali una tupla potrebbe reagire citiamo lo scadere di allarmi temporali piuttosto che cambiamenti della struttura della rete TOTA

3.2 Architettura TOTA

Il middleware TOTA è interamente realizzato in Java, come tale, esso può essere utilizzato in qualunque dispositivo computazionale equipaggiato con una JVM. TOTA, inoltre, supporta sia le tradizionali reti *wired* che le più recenti reti *wireless*. L'estrema portabilità di TOTA ne garantisce e ne promuove l'applicabilità in contesti tecnologicamente eterogenei come quelli del *pervasive computing*.

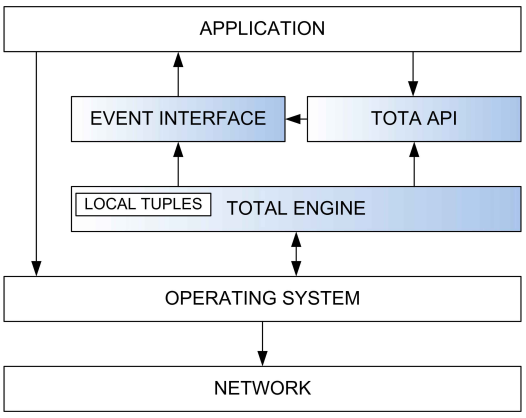


Figura 5: Architettura del middleware TOTA.

Dal punto di vista architetturale il middleware TOTA si compone di tre elementi principali: le API TOTA, la EVENT INTERFACE ed il TOTA ENGINE.

Le API TOTA, che approfondiremo nel paragrafo successivo, costituiscono l'interfaccia tra lo strato applicativo ed il middleware.

La EVENT INTERFACE è il componente responsabile di notificare allo strato applicativo eventi quali l'arrivo di nuove tuple in un nodo o la connessione/disconnessione di un nuovo nodo nella rete.

Il TOTA ENGINE, nucleo del middleware TOTA, è responsabile della propagazione delle tuple (gestisce i socket di comunicazione per l'invio e la ricezione di tuple), del loro mantenimento e della gestione della rete TOTA (mantiene il riferimento ai nodi vicini). Il TOTA ENGINE, inoltre, mantiene in un *tuple space* locale le tuple che hanno raggiunto quel nodo nel corso della loro propagazione. Il TOTA ENGINE, per concludere, monitora lo stato della rete e l'arrivo di nuove tuple aggiornando e ripropagando le tuple presenti all'interno del suo *tuple space* in relazione alle loro regole di propagazione e mantenimento.

3.3 Programmazione TOTA

Nel proseguio di questo paragrafo forniremo una breve panoramica delle API di interazione fornite da TOTA e delle modalità di specifica delle tuple TOTA con riferimento, in particolare, alle loro regole di propagazione e di mantenimento.

Le primitive di interazione esportate dal middleware TOTA sono le seguenti:

```
public void inject(TotaTuple tuple);
public Vector read(Tuple template);
public Vector readOneHop (Tuple template);
public Tuple keyrd(Tuple template);
public Vector keyrdOneHop(Tuple template);
public Vector delete(Tuple template);
public void subscribe(Tuple template,ReactiveComponent comp, String rct);
public void unsubscribe(Tuple template,ReactiveComponent comp);
```

Figura 6: API TOTA

La primitiva `inject` inserisce la tupla specificata come parametro nella rete TOTA. Una volta inserita, la tupla inizia a propagarsi secondo la regola di propagazione specificata nella sua definizione.

La primitiva `read` accede al tuple space TOTA locale restituendo l'insieme di tuple che unificano con il tuple template passato come parametro.

La primitiva `readOneHop` restituisce l'insieme di tuple presenti nel vicinato one *hop* del nodo che unificano con il tuple template passato come parametro. Tale primitiva è utilizzata, ad esempio, per consentire ad un agente di muoversi lungo la direzione del gradiente del campo presente nella rete TOTA.

Le primitive `keyrd` e `keyrdOneHop` permettono l'accesso alla tupla sulla base dell'id passato come parametro.

La primitiva `deleate` rimuove dal *tuple space* locale tutte le tuple che unificano con il tuple template passato come parametro.

La primitiva `subscribe` permette di associare l'esecuzione di un'azione da parte di un agente all'occorrenza di un evento che faccia match con il tuple template specificato come parametro. L'occorrenza di tale evento, in particolare, determina l'esecuzione automatica del metodo `react` dell'agente che ha eseguito la sottoscrizione, passando come parametro la stringa reaction e l'evento stesso.

Le tuple TOTA sono rappresentate sottoforma di oggetti Java. Lo stato dell'oggetto modella il contenuto della tupla. Le regole di propagazione e mantenimento, invece, sono incapsulate, rispettivamente, nei metodi `propagate` e `react`. La classe astratta `TotaTuple`, riportata di seguito, fornisce un modello generale per la definizione delle tuple TOTA.

```
public abstract class TotaTuple {

  protected TotaInterface tota;

  public void init(TotaInterface tota){
    this.tota = tota;
  }

  public abstract void propagate();

  public void react(String reaction, String event){}

}
```

Figura 7: Struttura generale delle tuple TOTA

Quando una tupla è inserita in un nodo della rete TOTA, essa riceve un riferimento all'istanza locale del middleware TOTA in esecuzione su quel nodo.

Lo stesso middleware, quindi, invoca i metodi `init` e `propagate` della tupla. Qualora nel corpo del metodo `propagate` sia presente l'invocazione del metodo `move` del middleware, la tupla è inviata a tutti i vicini *one-hop* ed i suoi metodi `init` e `propagate` saranno rieseguiti in modo ricorsivo dai middleware TOTA presenti in tali nodi. Nel corso della migrazione di una tupla all'interno della rete, il suo stato è serializzato quindi ricostruito all'arrivo presso il nuovo nodo.

Una tupla permane in uno stato attivo anche dopo l'esecuzione del metodo `propagate` da parte del middleware TOTA. In particolare, la tupla è riattivata, in accordo alla sua regola di mantenimento, all'occorrenza degli eventi per i quali notifica il suo interesse attraverso la procedura di sottoscrizione `subscription` messa a disposizione dalle API TOTA .

Un programmatore può definire una tupla estendendo la classe astratta `TotaTuple`. Pur caratterizzandosi per l'estrema flessibilità, la scelta di definire ex-novo una tupla a partire da una classe astratta potrebbe rilevarsi troppo complessa. TOTA, pertanto, fornisce una libreria di tuple che il programmatore può estendere per creare specifiche tuple di suo interesse astraendo dai dettagli di basso livello della loro propagazione e del loro mantenimento. La figura 8 rappresenta la gerarchia di tuple fornite da TOTA.

La classe `StructureTuple` articola il metodo `propagate` nei seguenti quattro sottometodi:

- `decideEnter`: definisce le condizioni che devono essere soddisfatte affinché la tupla sia inserita all'interno di uno specifico nodo.
- `makeSubscription`: definisce gli eventi rilevanti ai quali la tupla reagisce in accordo alla sua regola di mantenimento.
- `changeTupleContent`: incapsula le modifiche al contenuto della tupla.
- `decidePropagate`: definisce le modalità di propagazione della tupla.

La classe `MessageTuple` è utilizzata per realizzare messaggi che si diffondono nella rete e che sono automaticamente eliminati allo scadere di prefissati intervalli di tempo.

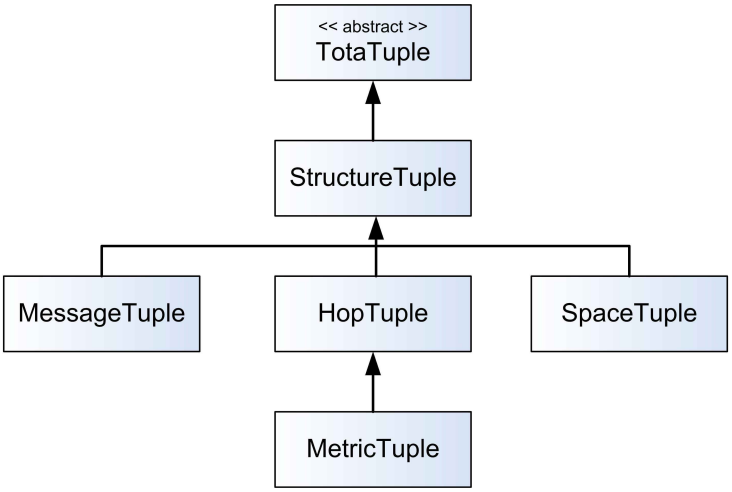


Figura 8: Gerarchia delle tuple TOTA

La classe `HopTuple` modella tuple che si diffondono nella rete in modalità *breadth-first* e che incapsulano un *hop* counter intero incrementato ad ogni *hop*. Tali tuple, inoltre, reagiscono alle modifiche topologiche della rete così che il loro *hop* counter rimanga sempre consistente con la distanza dal nodo in cui esse sono state inserite.

Le classi `MetricTuple` e `SpaceTuple` sono del tutto analoghe alla classe `HopTuple` fatta eccezione per la loro gestione della distanza in termini fisici piuttosto che in termini di numero di *hop*. Prerequisito all'utilizzo di tali tuple è la disponibilità di un sistema di localizzazione geografica (si pensi, ad esempio, ad un sistema GPS) in ciascun nodo della rete. Una tupla di classe `MetricTuple`/`SpaceTuple`, pertanto, si diffonde nella rete realizzando un sistema di coordinate condivise centrato nel nodo in cui la tupla è inserita.

3.4 Caso di studio: muoversi in formazione

Per chiarire le modalità di impiego del middleware TOTA consideriamo uno scenario in cui sia richiesto di coordinare il movimento di un gruppo di agenti affinché essi si spostino in formazione, ovvero, mantenendo una specifica distanza l'uno dall'altro. Una possibile concretizzazione di tale scenario potrebbe essere, ad esempio, la coordinazione di una pattuglia di agenti addetti a vigilare alla sicurezza di un edificio.

Per realizzare questo task di coordinazione è possibile ispirarsi, ancora una volta, ad una metafora naturale di *swarm intelligence* e, nello specifico, al comportamento degli stormi di uccelli. Le straordinarie coreografie degli stormi di uccelli, in particolare, emergono a partire dal semplice comportamento dei singoli uccelli che consiste nel mantenere una distanza costante dai propri vicini ed, al contempo, nel muoversi con la loro stessa velocità.

È possibile modellare questo comportamento attraverso l'astrazione dei Co-Fields supponendo che ciascun agente *i-esimo* generi un campo $Flock_i$ (*figura 9*) la cui intensità sia minima in corrispondenza della distanza inter-agente che si desidera mantenere e che tali campi siano aggiornati in tempo reale in funzione dei movimenti degli agenti. Ciascun agente, quindi, si muoverà in formazione seguendo la direzione decrescente del campo di coordinazione risultante dalla combinazione lineare dei campi $Flock_i$ generati dagli altri agenti del gruppo.

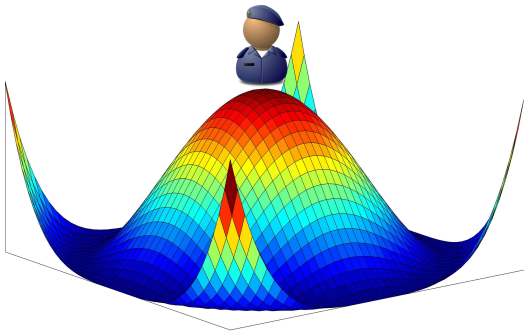


Figura 9: Campo $Flock_i$ generato dal generico agente *i-esimo*

Per generare il proprio campo computazionale $Flock_i$, ciascun agente *i-esimo* inietta nella rete TOTA una tupla di tipo `FlockingTuple`.

La classe `FlockingTuple` estende la classe `HopTuple` ridefinendo il metodo `changeTupleContent` con l'obiettivo di generare un campo che abbia il suo valore minimo in corrispondenza dei nodi posti a distanza `RANGE` dal nodo correntemente occupato dall'agente che l'ha iniettata. La regola di mantenimento della classe `FlockingTuple`, ereditata da `HopTuple`, garantisce l'aggiornamento in tempo reale dei campi generati da ciascun agente coerentemente ai loro movimenti all'interno della rete TOTA.

Il comportamento seguito da ciscun agente consiste semplicemente nell'iniettare una `FlockingTuple` all'interno della rete TOTA, quindi, nel muo-

```
public class FlockingTuple extends HopTuple {

  private int RANGE, value;

  public FlockingTuple(int RANGE) {
    this.RANGE = RANGE;
    value = RANGE;
  }

  protected void changeTupleContent() {
     super.changeTupleContent();
     value = (hop <= RANGE)?value-1: value+1;
  }

}
```

versi seguendo la direzione decrescente del campo di coordinazione generato dalle `FlockingTuple` degli altri agenti del gruppo.

```
public class FlockingAgent extends Thread implements AgentInterface {

   private TotaMiddleware tota;

   public void run() {

      FlockingTuple ft = new FockingTuple ();
      ft.setContent(peer.toString());
      tota.inject(ft);

      while(true) {
        FlockingTuple query = new FlockingTuple();
        Vector v = tota.read(query);
        GenPoint destination = getDestination(v);
        this.move(destination);
      }

   }

}
```

La *figura 10* rappresenta il profilo ideale del campo di coordinazione percepito dagli agenti. Tale figura permette di evidenziare, in particolare, come nel sistema emerga una formazione regolare di agenti che si spostano in modo coordinato nello spazio.

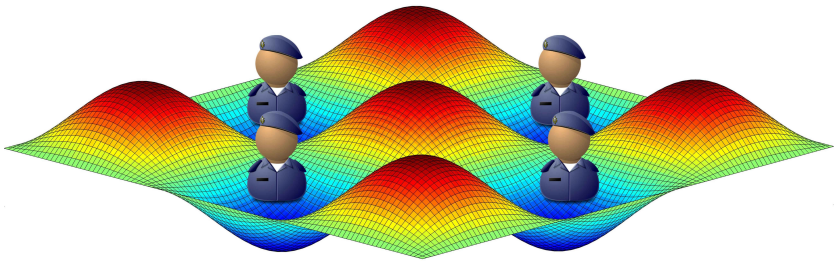


Figura 10: Campo di coordinazione ideale generato a seguito della propagazione nella rete delle *FlockingTuple*

Riferimenti bibliografici

[1] D. Gelernter e N. Carriero., "Coordination languages and their significance", *Communications of the ACM*, Volume 35 , Issue 2, pp. 97-107, 1992.

[2] Sito ufficiale dell'*Agents and Pervasive Computing Group*, http://agentgroup.ing.unimo.it

[3] M. Mamei e F. Zambonelli, "Field-based Coordination for Pervasive Multiagent Systems", *Springer*, 2005.

[4] P.P. Grassé, "La reconstruction du nid et les coordinations interindividuelles", *Bellicositermes natalensis et Cubitermes sp. La théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs*, pp. 6-41, 1959.

[5] M. Dorigo e T. Stützle, "Ant Colony Optimization", *MIT Press*, 2004.

[6] M. Mamei, L. Leonardi e F. Zambonelli "Co-Fields: A Unifying Approach to Swarm Intelligence", *3rd International Workshop on Engineering Societies in the Agents World (ESAW)*, LNAI 2577, Springer Verlag, pp. 68-81, 2003.

[7] M. Mamei, F. Zambonelli e L. Leonardi, "Distributed Motion Coordination with Co-Fields: A Case Study in Urban Traffic Management", *6th IEEE Symposium on Autonomous Decentralized Systems (ISADS 2003)*, IEEE CS Press, Pisa (I), 2003.

[8] M. Mamei e F. Zambonelli, "Programming Pervasive and Mobile Computing Applications with the TOTA Middleware", *2nd IEEE International Conference on Pervasive Computing and Communication*, IEEE CS Press, Orlando (FL), 2004.

[9] M. Mamei, F. Zambonelli e L. Leonardi, "Co-Fields: A Physically Inspired Approach to Distributed Motion Coordination", *IEEE Pervasive Computing*, Volume 3 , Issue 2, pp. 52-61, 2004.

[10] M. Mamei e F. Zambonelli, "Programming Stigmergic Coordination with the TOTA Middleware", *International Joint Conference on Autonomous Agents and Multiagent Systems*, ACM Press, Utrecht, NL, 2005.