ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA
SCUOLA DI INGEGNERIA E ARCHITETTURA

Corso di Laurea in Ingegneria Elettronica,Informatica e delle Telecomunicazioni

ESPERIMENTI DI DESIGN PATTERN DI ISPIRAZIONE BIOLOGICA

Elaborata nel corso di: Sistemi Distribuiti

Tesi di Laurea di:
RICCARDO MENCUCCI

Relatore:
Prof. ANDREA OMICINI

Co-relatori:
Ing. STEFANO MARIANI

ANNO ACCADEMICO 2014-2015
SESSIONE II

PAROLE CHIAVE

Auto-organizzazione

Sistemi MAS

Design Pattern

TuCSoN

ReSpecT

A tutte le persone care: parenti e amici

Indice

Introduzione

L'informatica, assieme alle sue innovazioni tecnologiche, offre al mondo d'oggi uno scenario in continuo sviluppo evolutivo che permette di facilitare alcune necessità dell'essere umano. Con la nascita di internet e dei nuovi dispositivi cellulari, la comunicazione è stata resa più malleabile e immediata. Tuttavia, le nuove tecnologie utilizzano infrastrutture complesse che non sempre sono ampiamente sfruttate a causa delle loro esigenze quali scalabilità, risposte in tempo reale, o tolleranza. Per far fronte a queste caratteristiche, una nuova tendenza del software è quella di fornire autonomia e pro-attività alle entità nel sistema in modo da incrementare la loro interazione. Queste caratteristiche permettono di responsabilizzare i soggetti rendendo il sistema *auto-organizzato*, con una migliore scalabilità, robustezza, e quindi riducendo le esigenze di calcolo di ciascuna entità.

Lo studio dei sistemi auto-organizzanti è stato ispirato alla natura, e in particolare, ai sistemi biologici. Questi sistemi mostrano le caratteristiche interessanti per gli scenari pervasivi, poiché sono robusti e resistenti, in grado di adattarsi al contesto ambientale e quindi reagiscono a determinate modifiche che si verificano nell'ambiente comportandosi di conseguenza.

L'ingegneria dell'auto-organizzazione ha il compito di simulare e testare questi comportamenti presentando uno schema progettuale completo che permetta di presentare soluzioni ricorrenti a problemi noti. Tale schema è definito in termini informatici *design pattern*.

Le entità, definite agenti, per interagire e comunicare tra di loro hanno bisogno di coordinarsi tramite un modello specifico. Nel nostro caso è stato scelto TuCSoN, poichè riesce a separare uno spazio dedicato allo scambio di informazioni da uno spazio dedicato alle specifiche che permette di descrivere delle politiche di comportamento per *sistemi MAS* implementati nell'opportuno linguaggio di programmazione ReSpecT.

I capitoli del mio lavoro di tesi sono stati strutturato in questo modo:

1)descrizione delle caratteristiche di un design pattern, che descrive comportamenti ispirati alla biologia, riportando anche un catalogo che mostra tutti i pattern (da quelli di base fino ad alto livello),

2) descrizione delle caratteristiche del modello di coordinazione TuCSoN

e del linguaggio di programmazione ReSpecT,

3) implementazione, simulazione e testing dei comportamenti dei pattern di base (spreading, aggregation, evaporation e repulsion) tramite ReSpecT utilizzando l'infrastruttura TuCSoN.

L'obiettivo principale è quello di fornire un servizio che descriva comportamenti per sistemi MAS auto-organizzanti.

Capitolo 1

Design Pattern

In questo primo capitolo introduttivo si cerca di dare una rappresentazione chiara e concreta di cosa sono i design pattern, del perché vengono utilizzati e in che modo viene sfruttato il loto utilizzo. L'approccio a questo primo capitolo vede la presentazione di un concetto di base di partenza; prima di andare a rappresentare i comportamenti di ispirazione biologica presentati come design pattern è necessario introdurre il concetto di "auto-organizzazione" (presentato nella prima sezione sia nella vita reale che nell'informatica). Nelle sezioni successive vengono presentati cosa sono i design pattern, il modello computazionale utilizzato per la loro descrizione, il modello di progettazione nei sistemi auto-organizzanti, il catalogo dei modelli utilizzati fino ad arrivare alla descrizione dei tipi di design pattern attraverso un catalogo, partendo da quelli di base, passando fino a quelli composti (aggregato di più design pattern di base), fino ad arrivare ai design pattern di alto livello (a loro volta rappresentano un aggregato di design pattern di base e composti).

1.1 Auto-organizzazione e sistemi auto-organizzanti

L'auto-organizzazione è un approccio molto interessante per l'ingegneria dei sistemi complessi e dei sistemi distribuiti . Generalmente, viene intesa come una forma di sviluppo del sistema attraverso influenze ordinanti e limitative provenienti dagli stessi elementi che costituiscono il sistema oggetto di studio e che permettono di raggiungere un maggior livello di complessità. Questo concetto lo riscontriamo inizialmente in natura quando andiamo ad osservare i sistemi biologici. Questi sistemi mostrano caratteristiche interessanti per gli scenari pervasivi, poiché sono robusti e resistenti, in grado di adattarsi al contesto ambientale e all'evoluzione degli eventi che si verificano nel sistema attraverso dei comportamenti conseguenti. L'auto-organizzazione sembra essere il prerequisito per la capacità di evoluzione, poiché genera i tipi di struttura che possono trarre vantaggio dalla selezione naturale.

Soltanto quei sistemi che sono capaci di auto-organizzarsi spontaneamente possono essere in grado di evolvere ulteriormente. La selezione utilizza principalmente proprio ciò che è auto-organizzato e robusto perché le caratteristiche auto-organizzate sono quelle che possono venire modellate più rapidamente. Le specie viventi sono esempi classici di auto-organizzazione: possono essere organizzate in milioni di generi di animali e piante attraverso gli stadi dell'evoluzione biologica, a partire dal molto semplice fino ad arrivare al complesso. Anche gli Stati nazionali possono essere considerati come emergenti da un processo di integrazione di elementi differenziati come villaggi, cittadine, città, contee. Nell'universo, poi, le particelle elementari formano gli atomi, le molecole, gli elementi e i composti. Su scala cosmica le nuvole gassose si condensano per dare luogo alle stelle, che poi portano alle galassie, ai gruppi di galassie, ecc. Il cervello umano è un altro splendido caso di auto-organizzazione, in quanto può essere visto come un sistema integrato di elementi differenziati - i neuroni - che da soli non hanno coscienza. La coscienza è dunque una proprietà emergente dal processo di auto-organizzazione. Gli stormi di uccelli sono auto-organizzati. I movimenti fluidi dello stormo sembrano essere guidati da una coreografia predefinita, ma in realtà molti stormi non hanno un leader. Uno stormo agisce in modo armonioso perché ogni singolo uccello segue un insieme di regole di base. Le formiche si comportano in modo analogo: se le si mettono in un gruppo che interagisce, emerge un formicaio. Poiché il formicaio emerge da interazioni dinamiche bottom-up e non è il prodotto di una pianificazione top-down, si dice che c'è auto-organizzazione. In natura vi è davvero poco spazio per ciò che è rigidamente pianificato e pertanto immutabile. Negli esseri umani circa il 10-12% del patrimonio genico è deputato a gestire la quotidianità, la quota restante è orientata alla risoluzione di situazioni di crisi o impreviste. Noi sopravviviamo grazie a questa ridondanza. È molto comune, osservando le organizzazioni tradizionali, notare come non esista ridondanza: il peso della gestione della quotidianità è spesso molto maggiore di quello della gestione o della creazione di imprevisti. La complessità esorta uomini ed organizzazioni a superare i modelli tradizionali e quindi si ritiene che le auto-organizzazioni hanno maggiori probabilità di evolvere. Auto-organizzazione non significa assenza di leadership ma bensì significa una nuova leadership complessa: imprenditori e manager sono chiamati a prestare attenzione alle dinamiche emergenti dal basso, dai propri collaboratori, dal personale di linea, da attori esterni come fornitori, clienti, consumatori, ecc. Il cambiamento non è da poco: si tratta di passare dalla tradizionale logica top-down ad una logica anche bottom-up. Il focus della leadership dovrebbe essere quello di delegare pur nell'ambito di alcune semplici regole di base, allo scopo di fare emergere all'interno delle organizzazioni l'intelligenza distribuita, ovvero le capacità (intellettuali, operative, innovative, emozionali) in rete degli esseri umani. Le dinamiche che regolano questo

equilibrio instabile sono di cooperazione e competizione sia all'interno che all'esterno dell'impresa.

L'auto-organizzazione è raggiunta favorendo la creazione di una rete interna all'organizzazione, dove i nodi sono costituiti dall'intelligenza delle singole persone, e le connessioni sono rappresentate dalle interazioni anche informali tra le stesse (intelligenza distribuita); e di una rete esterna all'organizzazione, che coinvolga tutti gli attori potenzialmente rilevanti, come avviene nei distretti italiani (alleanze strategiche). Secondo Vicari (1998), l'auto-organizzazione interna è visibile nella capacità di risposta senza ricorrere a gerarchia o a meccanismi di coordinamento. Gli elementi che la determinano possono essere i singoli oppure gruppi formali o informali. Quello che conta è che collaborino e competano tra di loro. Contano, dunque, le persone e le interconnessioni tra di loro. Non si tratta di lasciare libertà assoluta, ma di favorire un contesto in cui possa nascere auto-organizzazione. I riferimenti sono presi da un articolo di Luca Comello su Internet [4] che ho trovato molto interessante.
[1]

1.2 Sistemi auto-organizzanti e Design Pattern

L' idea di rappresentare sistemi di auto - organizzazione in ingegneria ha attratto molti ricercatori sin dal 2004. Si è cercato di presentare un insieme di primitive ispirate alla biologia che descrivano come i principi organizzativi dei microrganismi multi- cellulare potrebbero essere applicati ai sistemi multi-agente (sistemi MAS). Risulta, però, difficile usarli in modo sistematico per l'ingegneria dei sistemi auto-organizzanti artificiali. Tuttavia come possiamo fare per riuscire a rappresentare e a modellare questi sistemi auto-organizzanti nella maniera più semplice e definita? Quali sono i problemi che ogni meccanismo può risolvere? A quale soluzione contribuisce ogni modello? Quali sono i principali compromessi da considerare nell'implementazione? Per rispondere a queste domande alcuni autori si sono concentrati sulle descrizioni proponendo dei meccanismi di auto-organizzazione sotto forma di modelli software di progettazione . L'idea del modello di struttura è resa sotto forma di design, facilitando l'identificazione del problema che ciascun meccanismo può risolvere. Gardelli propone una serie di modelli di progettazione per sistemi di auto-organizzazione che sono connessi con il comportamento della colonia di formiche, con l'idea che un meccanismo può essere composto da altri meccanismi [5]. Sulla base della serie di questi meccanismi proposti si può mostrare come essi interagiscono tra di loro nella

[1] *http://www.complexlab.it/Members/lucacomello/articoli/complessita-e-organizzazione-ovvero-verso-le-auto-organizzazioni*

programmazione di sistemi MAS. Questo scenario può essere naturalmente modellato e raffinato per riuscire a rappresentare in maniera più semplificata e schematica il problema, i vantaggi, gli svantaggi, gli obiettivi, i casi d'uso ecc..

1.3 Modello computazionale dei Design Pattern

Questa sezione presenta il modello computazionale che permette di descrivere la dinamica di rappresentazione dei diversi soggetti che vengono modellati e rappresentati in ogni pattern. Il modello proposto è chiaramente ispirato dalla biologia ma specializzato per il mondo artificiale in cui i modelli saranno costruiti. Nei sistemi biologici, si possono osservare due entità principali:

-*gli organismi* che vivono nel processo biologico (ad esempio formiche, pesci, api, cellule, virus, ecc)

-*l'ambiente* inteso come uno spazio fisico in cui si trovano gli organismi. L'ambiente fornisce risorse che gli organismi possono utilizzare (per esempio cibo, riparo, materia prima).

Si può anche osservare che gli organismi posso anche essere condizionati dagli eventi che si verificano sull'ambiente stesso, riscontrando quindi dei cambiamenti nel sistema (ad esempio tempeste, alluvioni, nevicate tuoni o incendi). Un altro aspetto fondamentale che si verifica all'interno del sistema rappresenta la comunicazione; gli organismi possono infatti comunicare tra di loro, percependo delle informazioni dall'ambiente stesso agendo su di esso (ad esempio le formiche lasciano feromoni sull'ambiente per comunicare con altre formiche). La comunicazione tra gli organismi può essere:

- *diretta* : (ad esempio, i delfini inviano degli ultrasuoni attraverso l'acqua, i castori emetteno un suono per avvisare una presenza di predatore, etc.)

- *indiretta* utilizzando l'ambiente stesso come modo per depositare delle informazioni che altri organismi sono in grado di percepire.

Gli organismi,che vivono nel sistema, sono esseri autonomi e che vanno ad agire in maniera attiva nell'ambiente. Il sistema agisce sulle risorse e negli organismi in maniera dinamica (ad esempio gli organismi possono essere uccisi, si possono distruggere le risorse, modificare la topologia dello spazio

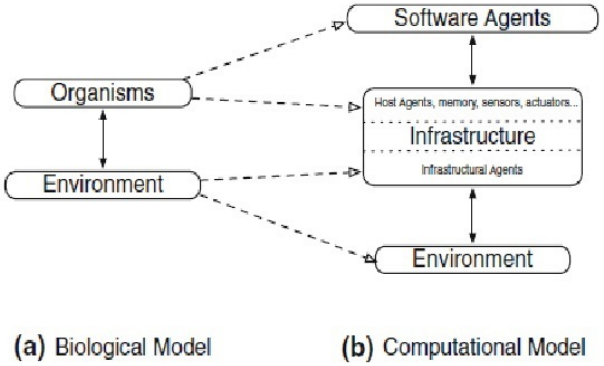


Figura 1.1: Modello computazionale

in cui vivono gli organismi, cambiare la posizione del cibo, togliere il cibo, aggiungerne del nuovo, ecc.).

Nella figura 1.1 vengono mostrati sia il modello biologico che il modello computazionale (rispettivamente figura a) e figura b)).

Partendo dal modello biologico possiamo arrivare a rappresentare un modello computazionale. La sostanziale differenza rispetto al modello appena rappresentato sta nel fatto di dover aggiungere un livello che si frapppone tra ambiente e le entità attive nel sistema. Questo nuovo livello, chiamato livello di infrastruttura, è necessario perché, in un progetto di sistema, l'agente software deve essere ospitato in un dispositivo con potenza computazionale in modo da fornire agli agenti la capacità di interagire con l'ambiente (rilevando informazioni nell'ambiente attraverso sensori o andando ad agire in esso tramite attuatori) e comunicare con altri agenti. Le entità proposte nel modello computazionale sono:

- *gli agenti*: che sono entità software autonome in esecuzione in un nodo (nel nostro caso un nodo TuCSoN)

- *l'infrastruttura*: che contiene nodi con potenza di calcolo, sensori e attuatori .

Essa è composta da un insieme di nodi connessi tra di loro e da agenti infrastrutturali (interni all'infrastruttura). Un nodo è un'entità con potenza computazionale, che permette di comunicare grazie anche a sensori e attuatori. Un nodo è in grado di fornire servizi agli agenti. Un agente infrastrutturale è un soggetto autonomo e pro-attivo, che agisce sul sistema a livello di infrastruttura. Esso può essere incaricato di svolgere comporta-

menti presenti in natura, come nel nostro caso la diffusione, l'evaporazione, l'aggregazione e la repulsione.

-*l'ambiente*, lo spazio reale dove l'infrastruttura si trova.

Gli eventi sono fenomeni che si verificano nello spazio circostante e ne cambiano lo stato (quindi agiscono sull'ambiente), l'evento viene rilevato dagli agenti utilizzando dei dispositivi (sensori). Ogni agente ha bisogno di un nodo per essere eseguito, per comunicare con altri agenti, per rilevare eventi o per agire nell'ambiente. Così, l'infrastruttura fornisce agli agenti tutti gli strumenti necessari per simulare il comportamento degli organismi e fornire quindi un luogo, dove poter conservare informazioni che possono essere lette da altri agenti. Nella maggior parte dei processi biologici, l'ambiente svolge un ruolo fondamentale, grazie alla capacità di agire dell'agente su di esso, permette un continuo cambiamento di stato nel sistema. Per affrontare questa abilità, ogni nodo nelle infrastrutture ha un software incorporato, chiamato agente infrastrutturale (IA). Sia l'IA che i comportamenti dell'agente devono essere progettati per seguire modelli auto-organizzanti. Gli IA giocano un ruolo importante quando gli agenti possono muoversi liberamente. Ad esempio, le valutazioni d'impatto possono essere responsabilizzate per la gestione delle informazioni depositate in ogni nodo. La Figura 1.2 mostra i vari strati computazionali: il modello e le loro interazioni corrispondenti. Lo strato superiore è rappresentato dagli agenti software all' interno del sistema. Gli agenti utilizzano il livello di infrastruttura per ospitare se stessi, reciprocamente per comunicare con altri agenti, agiscono sull'ambiente depositando informazioni che altri agenti sono in grado di leggere. Ci sono due varianti nel modello: quando gli agenti possono muoversi liberamente tra i nodi (ad esempio agenti mobili) o quando vengono accoppiati tra i nodi stessi (ad esempio sciami di robot). La separazione tra lo strato di agenti e l'infrastruttura consente di coprire una più ampia varietà di scenari. Da un lato, l'agente software può essere mobile. D'altra parte invece, l'infrastruttura può essere fissata (cioè con nodi fissi) o può essere mobile. Nodi locali cellulari possono essere controllati dagli agenti (ad esempio un robot) o meno (ad esempio movimenti sotto il controllo del suo proprietario). Questo è tipico di scenari pervasivi dove diversi dispositivi mobili, come ad esempio computer portatili, o telefoni cellulari si trovano in uno spazio fisico comune (ad esempio, un centro commerciale, un museo, ecc, formando quella che viene solitamente indicata come infrastruttura opportunistica, dove i nodi sono in movimento secondo i movimenti dell'utilizzatore che li trasporta, e gli agenti possono passare liberamente da un nodo all'altro.

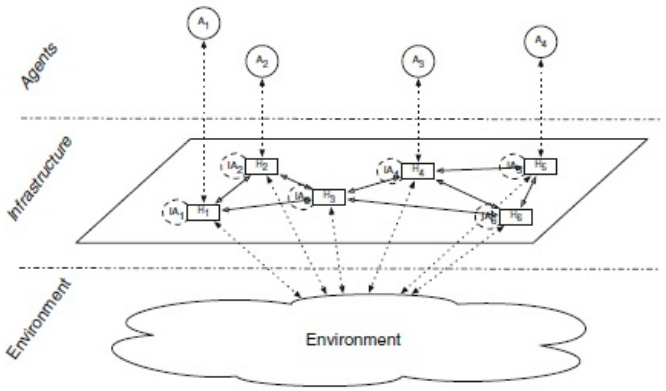


Figura 1.2: Il modello computazione

1.4 Modello di progettazione dei Design Pattern nei sistemi auto-organizzanti

Le metodologie attuali per produrre sistemi auto - organizzanti seguono le fasi tipiche della metologia appresa nell'ambito dell'ingegneria del software: requisiti, analisi, progettazione, attuazione, verifica e test. Questo perché per partire con la progettazione del lavoro è necessario ricevere dei requisiti da un committente, una volta ricevuti i requisiti lo step successivo è quello di inquadrare i requisiti e il problema facendo un analisi ben dettagliata prima di iniziare la fase di progettazione a livello di design. Questo perchè *non c'è codice senza progetto, non c'è progetto senza analisi del problema, non c'è problema senza requisiti.* Di conseguenza i design pattern auto-organizzanti sono sfruttati durante la fase di scelta di design per la metologia prescelta. Occorre ,quindi effettuare una scelta dei modelli di progettazione, da effettuare durante una prima fase di progettazione . I design pattern auto-organizzanti hanno il principale obiettivo di *identificare il problema da risolvere, determinando la soluzione appropriata per risolvere il problema. In particolare, aiutano a definire i confini di ogni problema, fornendo una soluzione corrispondente proposta dal pattern. Il loro uso risulta fondamentale, poichè si riesce ad astrarre il problema ancora prima di formulare algoritmi.* I Pattern sono dotati di una descrizione dei principali parametri legati al modello e il loro effetto sul comportamento risultante. Questa fase è quindi cruciale per la determinazione dei valori dei parametri. Queste dinamiche appena descritte possono essere riassunte nella figura 1.3.

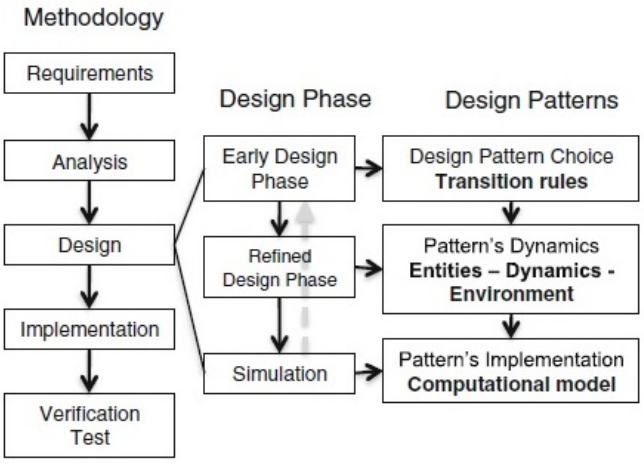


Figura 1.3: Fasi di progettazione

1.5 Catalogo dei modelli

Per creare il catalogo dei modelli si sono analizzati i comportamenti dei sistemi auto-organizzanti, classficandoli in diverse categorie. Il processo di classificazione è avviato selezionando quei meccanismi di alto livello che sono ben noti e sono stati applicati con successo a differenti sistemi . Analizzando il loro comportamento, è stato identificato che meccanismi comuni possono essere di livello inferiore (e quindi in forma base o in forma atomica).Partendo da scenari inferiori, aggregando tra loro comportamenti di base si raggiunge un livello secondario definito pattern composto, mentre a loro volta una combinazione di aggregazione di scenari di base con scenari composti definisce un pattern ad alto livello. Ricapitolando, come risultato, quindi abbiamo classificato i modelli in tre strati . I meccanismi di base che possono essere utilizzati singolarmente a un livello primario. Allo strato centrale, ci sono i meccanismi formati dalle combinazioni dei meccanismi di livello inferiore (quindi design composti). Mentre al terzo e ultimo strato (o strato superiore) spiccano modelli di livello superiore che mostrano diversi modi di sfruttare valori di meccanismi di base e composti(nel nostro caso andiamo a implementare solo i modelli di base). Per costruire un catalogo sistematico di modelli, è necessario descrivere ogni modello rispetto ad un noto schema. Lo schema è molto semplice. Esso consiste nel rappresentare: *nome, problema, soluzione e le conseguenze*. Come sottolineato, i modelli per sistemi MAS dovrebbero essere descritti usando programmi specifici. In particolare, uno schema candidato dovrebbe riflettere le peculiarità del target del meta modello MAS. L'obiettivo principale è di caratterizzare nel

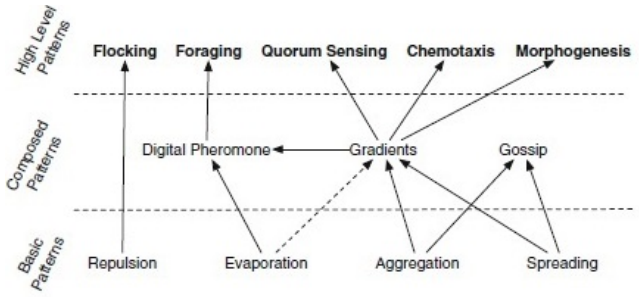


Figura 1.4: Catalogo dei modelli

miglior modo possibile dei comportamenti auto-organizzanti tramite design pattern per sistemi MAS, rispetto a uno schema di modello di riferimento [6]. Ne vale la pena notando che, poiché i modelli descritti codificano meccanismi di base, possono sembrare troppo generici per essere utili: tuttavia, crediamo che costruire modelli complessi potrebbe consentire una più profonda comprensione della dinamica dei sistemi, migliorando quindi la controllabilità. Si cerca quindi, di isolare i modelli di base che, se correttamente combinati, producono schemi di sistemi di auto-organizzazione complessi: per esempio i modelli di base sono l'evaporazione, l'aggregazione, la diffusione e la repulsione. La Figura 1.4 mostra i diversi modelli di progettazione raccolti nel catalogo e le loro relazioni . Le frecce indicano come i modelli sono composti. La freccia tratteggiata indica che l'utilizzo del pattern è facoltativo (ad esempio, il modello di gradiente può usare l'evaporazione , ma essa non è necessaria per implementare gradienti) . Tale classificazione si propone di quotazioni di meccanismi esistenti dalla letteratura, identificando i propri confini (cioè quando un meccanismo si ferma e quando un altro inizia). Per esempio, il Gossip è stato applicato a diversi pattern in diversi modi, ma tutte le implementazioni in comune permettono di dire che esso è un processo composto da meccanismi di diffusione e aggregazione. Il catalogo fornito in questo documento permette di allargare i propri confini aggiungendo a esso a nuovi meccanismi di base, una volta identificati e descritti sotto forma di schemi . Analogamente, qualsiasi nuova combinazione di modelli di base o di livello superiore può essere così aggiunta al catalogo.

Il catalogo è presentato nell'immagine 1.4. Per ogni pattern, oltre al suo nome e altre denominazioni note, sono chiaramente identificati il problema richiesto e la soluzione fornita . Il comportamento dei modelli è descritto attraverso la transizione di regole utilizzando la notazione riportata in seguito. Ogni informazioni viene modellata come una tupla $< \mathrm{L}, \mathrm{C} >$.

Le regole di transizione sono reazioni ispirate alla chimica che lavorano su modelli di tuple. Essi sono del tipo:

$$\text{name} :: < \text{L1,C1} \ ,\ldots < \text{Ln,Cn} > \rightarrow < \text{L1',C1'} >,\ldots < \text{Lm',Cm'}$$

dove il lato sinistro (i reagenti) permette di specificare quali tuple sono coinvolte nella regola di transizione (e quindi saranno rimosse come effetto della regola di esecuzione), mentre il lato destro (i prodotti), invece, specifica quali tuple sono quindi da inserire nella posizione specificata: potrebbero essere nuove tuple, di trasformazione, di uno o più reagenti o in alcuni casi i reagenti addirittura possono rimanere invariati. Le regole sono quindi dotate di una serie di politiche di transizione che determinano le variabili lato destro come funzioni del lato di sinistra. Tali funzioni possono essere soggette a condizioni e vincoli, che saranno specificate, unitamente con la reazione.Si nota che tali funzioni possono essere:

1. Parametri fissati dal modello di sistema che abbiamo;

2. Estratti direttamente dai reagenti della funzione dove possono essere applicati;

3. Specificati nella regola di transizione

1.6 Pattern

In informatica, nell'ambito dell'ingegneria del software, un design pattern (ovvero uno schema progettuale, schema di progettazione, schema architetturale) è un concetto che può essere definito come una soluzione progettuale generale ad un problema ricorrente. Si tratta quindi, di una descrizione di un modello logico da applicare per la risoluzione di un problema che può presentarsi in diverse situazioni durante le fasi di progettazione e sviluppo del software, ancor prima della definizione dell'algoritmo risolutivo della parte computazionale. È un approccio spesso efficace nel contenere o ridurre il debito tecnico. I design pattern object-oriented tipicamente mostrano relazioni ed interazioni tra classi o oggetti, senza specificare le classi applicative finali coinvolte, risiedendo quindi nel dominio dei moduli e delle interconnessioni. Ad un livello più alto sono invece i pattern architetturali che hanno un ambito ben più ampio, descrivendo un pattern complessivo adottato dall'intero sistema, la cui implementazione logica dà vita a un framework.

Come già anticipato nei capitoli precedenti un design pattern è strutturato in questo modo:

• *il nome*, costituito da una o due parole che siano il più possibile rappresentative del pattern stesso;

• *il problema*, ovvero la descrizione della situazione alla quale si può applicare il pattern. Può comprendere la descrizione di classi o di problemi di progettazione specifici, come anche una lista di condizioni perché sia necessario l'utilizzo del pattern;

• *la soluzione*, che descrive gli elementi costitutivi del progetto con le relazioni e relative implicazioni, senza però addentrarsi in una specifica implementazione. Il concetto è di presentare un problema astratto e la relativa configurazione di elementi adatti a risolvere il problema stesso;

• *le conseguenze*, i risultati e i vincoli che derivano dall'applicazione del pattern. Sono fondamentali in quanto possono essere l'ago della bilancia nella scelta dei pattern: le conseguenze comprendono considerazioni di tempo e di spazio, possono descrivere implicazioni del pattern con alcuni linguaggi di programmazione e l'impatto con il resto del progetto. L'uso di pattern nella descrizione di altri pattern dà origine ai cosiddetti linguaggi di pattern. Nelle sotto-sezione successive verranno descritte tutte le caratteristiche dei comportamenti sotto forma di pattern, partendo da quelli atomici sino ad arrivare a quelli ad alto livello [5].

1.6.1 Pattern atomici

I modelli di Pattern di base sono modelli atomici, utilizzati per comporre più modelli complessi che compaiono al livello centrale e allo strato superiore. Questi modelli descrivono meccanismi fondamentali che sono stati frequentemente utilizzati(senza di essi non possono esistere pattern di livelli superiori). I pattern rappresentati sono lo spreading, l'aggregation, l'evaporation e il repulsion. Al terzo e ultimo capitolo verrà fornita un implementazione nel linguaggio ReSpecT di questi comportamenti.

Spreading Pattern

Lo spreading pattern o modello di diffusione si basa sulla comunicazione diretta tra agenti per l'invio progressivo di informazioni nel sistema. La diffusione di informazioni nei sistemi multi-agente permette agli agenti di incrementare la conoscenza globale del sistema. La Figura 1.5 mostra le varie fasi del processo di diffusione:

(a) un agente avvia il processo di diffusione (black node);
(b) le informazioni si propagano su rete;
(c) il processo termina quando le informazioni raggiungono tutti i nodi della rete.

-*Alias*: Spreading pattern (o pattern di diffusione) è anche conosciuto come pattern di espansione di informazioni

-*Problema*: nei sistemi, dove gli agenti eseguono solo interazioni locali,è necessario conoscere informazioni globali sul sistema

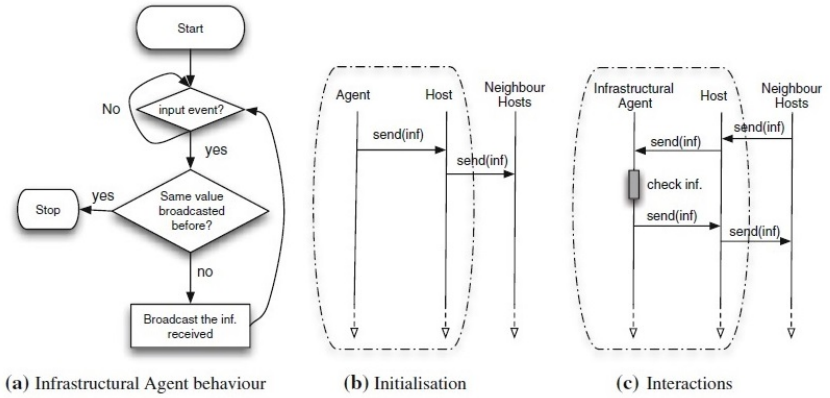
-*Soluzione*: una copia delle informazioni (ricevute o detenute da un agente) viene inviata ai vicini e propagata in rete da un nodo all'altro. La diffusione di informazioni nel sistema riduce progressivamente la mancanza di conoscenza degli agenti mantenendo il vincolo dell' interazione locale.

-*Ispirazione*: in natura, la diffusione è un processo fatto dall'ambiente. Lo spreading è un modello di base esteso o sfruttato dalla maggior parte degli altri modelli presentati in questo catalogo. Esso appare in processi importanti, come Morfogenesi, Chemiotassi o Quorum Sensing.

Vantaggi: se si verifica la diffusione ad alta frequenza, le informazioni si diffondono rapidamente attraverso la rete ma il numero di messaggi aumenta. Una diffusione rapida è desiderata quando l'ambiente è in continuo cambiamento e gli agenti devono conoscere i nuovi valori e adattarsi aggiornandosi continuamente. E 'stato dimostrato che non è necessario inviare le informazioni su tutti i nodi adiacenti al fine di garantire che ogni nodo ha ricevuto le informazioni (Birman et al. 1999).

Dinamiche entità-ambiente: le entità coinvolte nel processo di diffusione sono i nodi, gli agenti, e gli agenti infrastrutturali. Il processo di diffusione viene avviato da un agente che prima diffonde le informazioni nel nodo. Quando queste informazioni arrivano ai nodi vicini, l'agente infrastrutturale è incaricato di inviare nuovamente l'informazione ai nodi vicini, producendo la diffusione delle informazioni sull'intero sistema. Ogni agente infrastrutturale trasmette i dati ricevuti per un determinato numero di vicini e fino al numero specificato. La dinamica è di solito estesa per evitare loop infiniti e le consegne di duplicati sprecati (ad esempio, quando un agente riceve le stesse informazioni che ha inviato prima, l'agente non si risente tali informazioni).

La regola di transizione dello spreading pattern è descritta in modo più formale qui sotto.



(a) Infrastructural Agent behaviour **(b)** Initialisation **(c)** Interactions

Figura 1.5: Spreading

Spreading :: $<L,C> \rightarrow <L1,C1>,\ldots.. <Ln,Cn>$

where $<L1,\ldots.Ln> = v(L),(C1,\ldots..Cn) = \theta(C,L)$

La funzione v(L) ha il compito di determinare la sequenza di luoghi e i luoghi vicini a L a cui l'informazione in ingresso deve essere diffusa . L'insieme di tali luoghi non può essere vuota , non può essere composta da solo L, ma può essere composta da tutto il vicinato di L compreso L stesso.

La funzione θ(C, L) è data per calcolare il nuovo contenuto dell' informazione, che può cambiare all'interno del processo di diffusione.

Implementazione: l'algoritmo più comune utilizzato per diffondere le informazioni ai vicini è l'algoritmo di trasmissione. È ben noto che la trasmissione provoca quello che viene chiamato il Broadcast Storm Problem. Il Broadcast Storm Problem appare quando il raggio di azione del segnale si sovrappone su molti nodi. Così, una semplice trasmissione dalle inondazioni, si tradurrà in una sovrabbondanza in rete provocando una collisione al sistema. Col passare del tempo, ci sono state nuove proposte per un efficace modalità di diffusione delle informazioni. Questo lavoro presenta una implementazione di base per illustrare come diffondere l'informazione presentato nell'immagine 1.5. La figura 1.5a mostra il diagramma di flusso in cui le informazioni si diffondono dopo il ricevimento. La Figura 1.5b, invece mostra come inizia l'interazione. La figura 1.5c, infine rappresenta le interazioni quando l'informazione arriva ad un nodo vicino.

Impieghi noti: il meccanismo di diffusione l si riscontra spesso nel coordinamento dei giochi e nell'ottimizzazione del problema.

Conseguenze: quando si applica il modello di diffusione, gli agenti sono in grado di ricevere informazioni oltre il loro rilevamento locale. Una volta applicato lo spreading ci può essere un sovraccarico di rete (nei messaggi e nella memoria). Questo incremento diventa estremo quando l'ambiente è molto dinamico e gli agenti devono mantenere aggiornate le informazioni appena possibile.

Modelli correlati:La diffusione è utilizzato nei modelli di livello superiore come Gradiente, Morfogenesi o Chemiotassi.

Aggregation Pattern

Il pattern di aggregazione è un meccanismo di rinforzo da adottare per il sistema che si sta considerando. Questo comportamento, se si vuole, è anche osservabile nei comportamenti umani . Un classico esempio possibile lo si riscontra durante la navigazione in Internet e viene notato un fatto interessante e quindi il lettore, desideroso di intervenire con il proprio pensiero, può lasciare un "rinforzo" ovvero un commento che è in genere anonimo e quindi di fatto rappresenta un informazione aggregata automaticamente con i commenti di altri utenti . E' quindi evidente che l'aggregazione è guidata dal programma utente . Il modello di aggregazione è un modello di base utilizzato per le informazioni di fusione. Lo spreading di informazioni in sistemi di grande scala può produrre sovraccarico di rete e di memoria. L'aggregation pattern, invece, è stato introdotto come un modo per ridurre la quantità di informazioni nel sistema e per sintetizzare informazioni significative concentrandosi su quelle necessarie superando alcune problematiche del pattern precedente.

Alias: l' aggregazione è conosciuta anche come fusione.

Problema: in grandi impianti, l'eccesso di informazioni prodotte dagli agenti, può generare sovraccarichi di rete e di memoria. Le informazioni devono essere trattate in maniera distribuita per ridurre la quantità di informazioni e per ottenere informazioni significative.

Soluzione: l'aggregazione consiste nell'applicare un operatore di fusione per via locale per elaborare le informazioni e sintetizzarle in macro-informazioni. Questi operatore di fusione possono prendere molte forme (quelli trattati nella Tesi sono il filtraggio, l'unione , aggregazione, e trasformazione).

Ispirazione: in natura, l'aggregazione consente a una colonia di formiche di trovare il percorso più breve per il cibo, scartando percorsi più lunghi. (Cioè due profumi insieme creano un campo attraente più grande di un singolo profumo). In natura, l'aggregazione è un processo eseguito dall'ambiente.Infatti, anche quando ci sono agenti presenti nel sistema, l'ambiente continua a eseguire il processo di aggregazione.

Vantaggi: l'aggregazione si applica a tutte le informazioni disponibili localmente o solo su parte di tali informazioni. Il parametro in questione è la quantità di informazioni che sono fuse.

Dinamiche entità-ambiente: l'aggregazione viene eseguita sia dagli agenti infrastrutturali che da un agente esterno all'infrastruttura. In entrambi i casi gli agenti aggregano le informazioni presenti localmente. Le informazioni possono provenire dall'ambiente o da altri agenti. Le informazioni provenienti dall'ambiente sono tipicamente lette da sensori (come temperatura, umidità, ecc.) Secondo il modello presentato, l'aggregazione è eseguita da un agente che riceve l'informazione. Tale nodo è capace di leggere informazioni dall'ambiente o una comunicazione da un dispositivo ramite un sensore, che ricevere le informazioni dal nodo vicino. L'aggregazione può essere applicata a qualsiasi agente che riceve informazioni indipendentemente dall'infrastruttura sottostante. Il processo di aggregazione non è ripetitivo e termina quando un agente esegue la funzione di aggregazione. La regola di transizione per l'aggregazione è la seguente :

l'informazione in ingresso (o eventualmente un insieme di informazioni) è trasformata in un nuovo insieme di informazioni con minor cardinalità:

aggregation :: <L1,C1 >,,,,,, <Ln,Cn > → <L,C' >,. . . ., <L,C'm >

where C'1,. . . .,C'm = α(C1,. . . .,Cn)

Implementazione: le informazioni disponibili prendono la forma di un flusso di eventi. L'aggregazione di informazioni può assumere diverse forme: da un operatore semplice (somma, moltiplicazione o media). Gli operatori relativi all'aggregation pattern sono classificati in quattro gruppi diversi :
1)*Filtro*: questo operatore seleziona un sottoinsieme di eventi ricevuti (ad esempio il sensore richiede 10 misure al secondo, ma l'applicazione ne processa solo 1 al secondo) e li mantiene nel centro di tupla.(Come vedremo nel nostro caso si può scegliere se mantenere l'informazione più recente o più vecchia);
2) *Trasformatore*: questo operatore cambia il tipo di informazioni ricevute in ingresso (ad esempio vengono passate coordinate GPS e le uscite sono i paesi dove le posizioni si trovano).

3) *Fusione*: questo operatore unifica tutte le informazioni ricevute ed emette tutte le informazioni ricevute come un unico pezzo di informazione (se io ho più di un informazione uguale, si mettono più informazioni in una sola tupla).
4) *Aggregatore*: data una o più informazioni in ingresso viene applicato l'aggregator per fare dei calcoli sull'informazione stessa (es max, min o avg) e fornire come output il risultato in una tupla.
Il diagramma di flusso 1.6a mostra che il processo di aggregazione inizia quando l'agente riceve l'informazione in ingresso (un evento). Una volta arrivato l'input, viene applicato l'operatore di fusione inviando le informazioni aggregate al nodo. La Figura 1.6b mostra come l'agente, o l'agente infrastrutturale, utilizza l'interfaccia organizzata dai nodi per avere i dati; viene applicato un operatore di fusione per depositare i dati aggregati nel nodo.

Impieghi noti: l' aggregazione è stata utilizzata nell'algoritmo ACO (Dorigo 1999)consentendo di aggregare feromoni,e quindi emulando le concentrazioni più elevate quando due o più feromoni sono vicini l'uno all'altro.

Conseguenze: l' aggregazione aumenta l'efficienza in rete, riducendo il numero di messaggi,e aumentando quindi, la durata della batteria e della scalabilità del sistema. Inoltre fornisce un'aggregazione per il meccanismo di estrazione di macro-informazioni su sistemi di larga scala, come l'estrazione di informazioni significative dei dati ottenuti da letture di diversi sensori. Pertanto, la quantità di memoria utilizzata dal sistema è ridotta.

Pattern correlati: il modello di aggregazione può essere realizzato in collaborazione con modelli di evaporazione e modelli di gradiente per formare pattern di feromoni digitali (Digital pheromone pattern). L'evaporazione (come vedremo nella sotto-sezione successiva)può essere utilizzata assieme all'aggregazione per aggregare informazioni recenti raccolte dall'ambiente.

Evaporation Pattern

L'evaporazione è un modello che si occupa di gestire ambienti dinamici ,in cui le informazioni sono utilizzate dagli agenti e in alcuni casi possono diventare obsolete. Negli scenari del mondo reale, le informazioni cambiano con il tempo e la previsione o rimozione, a causa del cambiamento, è generalmente costosa o addirittura impossibile. Così, quando gli agenti devono modificare il loro comportamento, tenendo conto delle informazioni dell'ambiente, devono considerare solo l'informazione più recente ed escludere le informazioni più antiche. L'evaporazione è un meccanismo che riduce progressivamente la pertinenza delle informazioni. Così,informazioni recenti diventano più ri-

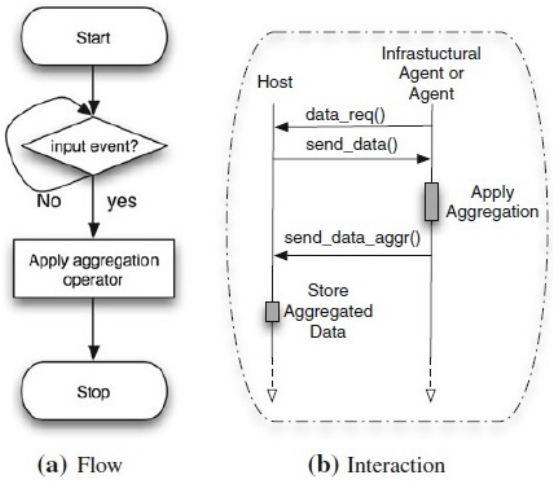


Figura 1.6: Aggregation

levanti di informazioni elaborate qualche tempo addietro. È stato proposto di un modello di evaporazione come un modello di progettazione per sistemi multi-agente di auto-organizzazione.

Alias: l'evaporazione è anche nota come decadimento.

Problema: una volta superata l'informazione non può essere rilevata e deve essere rimossa, o il suo rilevamento comporta un costo che deve essere evitato. Le decisioni dell' agente si basano sulla freschezza delle informazioni presentate nel sistema,consentendo un corretta risposta a ambienti dinamici.

Soluzione: l'evaporazione è un meccanismo che riduce periodicamente il campo delle informazioni presenti nel sistema. Così, le informazioni recenti, diventano più attuali delle informazioni più vecchie favorendo una sorta di "aggiornamento".

Ispirazione: l'evaporazione è presente in natura. Per esempio,in una colonie di formiche, quando una formica deposita dei feromoni nell'ambiente, questi feromoni tendono ad attrarre altre formiche e le guidano dal nido al cibo e vice-versa. L'evaporazione agisce sui feromoni riducendo la loro concentrazione nel tempo fino a scomparire. Questo meccanismo permette alle formiche a trovare il percorso più breve per il cibo, anche quando nell'ambiente si verificano cambiamenti (ad esempio, nuovi luoghi alimentari o ostacoli nel percorso). Le formiche sono in grado di trovare i nuovi percorsi più brevi scartando quelli più vecchi.

Vantaggi: l'evaporazione è controllata da dei parametri che definiscono sia il fattore di evaporazione (cioè quanto le informazioni sono evaporate),che la frequenza di evaporazione (cioè ogni quanto tempo viene eseguita l'evaporazione), utilizzata per decrementare la rilevanza delle informazioni. Il fattore di evaporazione deve affrontare le dinamiche dell'ambiente: se l'evaporazione è troppo veloce, possiamo perdere informazioni.Se invece l'evaporazione è troppo lenta, le informazioni possono diventare obsolete e sviare il comportamento degli agenti. Un più alto fattore di evaporazione rilascia la memoria, ma riduce anche le informazioni disponibili nel sistema per gli agenti. Quando l'evaporazione viene applicata a ricerca collaborativa o di ottimizzazione di algoritmi, il fattore di evaporazione controlla l'equilibrio tra esplorazione e sfruttamento: un alta evaporazione può ridurre i tassi di conoscenza degli agenti per l'ambiente, aumentando l'esplorazione e la produzione veloce di adattamento ai cambiamenti dell'ambiente. Tuttavia, un maggior fattore di evaporazione diminuisce le prestazioni quando si verifica un cambiamento nell'ambiente.

Dinamiche entità-ambiente: l'evaporazione può essere applicata a qualsiasi informazione presente nel sistema. Periodicamente,la sua rilevanza decade nel tempo. Così, le informazioni recenti diventano più rilevanti delle informazioni elaborate in tempi più antichi. L'evaporazione è eseguita dall'agente o dall'agente infrastrutturale in continuazione. La regola di esecuzione viene applicata in questo modo:

evaporation :: $<$L,C$>$ $\rightarrow$ $<$L,C'$>$

where C' = ϵ(C)

La regola influisce sul valore di rilevanza del contenuto in C.

Implementazione: il pattern di evaporazione viene eseguito da un agente che deve aggiornare la rilevanza delle informazioni al suo interno. I casi di applicazione possono esser due: nel primo approccio, un agente incapsula l'informazione e decade la propria rilevanza. In questo caso,lo si può riscontrare osservano gli schemi di flusso e interazione (corrispondentemente 1.7a e 1.7b). Nel secondo approccio, l'informazione viene depositata in un nodo da un agente. A questo punto interviene l'agente infrastrutturale che interagisce con il nodo stesso attraverso un'interfaccia per leggere le informazioni e cambiare il valore di rilevanza. In questo caso, l'interazione tra l'agente infrastrutturale e il nodo viene mostrato in Fig. 1.7c.

Impieghi noti: l'evaporazione è stata utilizzata principalmente nell'otti-

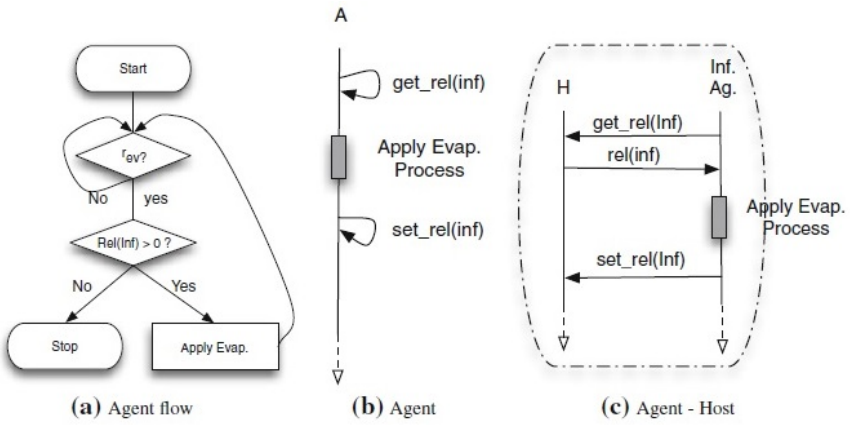


Figura 1.7: Evaporation

mizzazione dinamica. Esempi di algoritmi che utilizzano evaporazione sono ACO (Dorigo 1999) e Quantum Swarm Optimization di evaporazione (QSOE).

Conseguenze: l'evaporazione permette l'adeguamento dei cambiamenti ambientali. Tuttavia, l'uso di scenari statici nell'evaporazione, può diminuire le prestazioni, grazie alla perdita di informazione associati a questo meccanismo. L'evaporazione del modello offre la possibilità di auto-adattamento a cambiamenti ambientali, aumentando la tolleranza al rumore.

Pattern correlati: il pattern di evaporazione è usato da modelli di livello superiore come digital pheromone pattern o gradient pattern.

Repulsion Pattern

Il repulsion pattern è un modello base per il coordinamento del movimento in larga scala di sistemi MAS. Esso attiva gli agenti per ottenere una distribuzione uniforme di una determinata area e permette di evitare la collisione tra di loro. Utilizzando la repulsione gli agenti possono ottimizzare il sistema distribuendo l'informazione in maniera equa.

Alias: nessuno di nostra conoscenza.

Problema: i movimenti degli agenti devono essere coordinati in un sistema decentrato al fine di ottenere una distribuzione uniforme ed evitare collisioni tra loro.

Soluzione: il repulsion pattern crea un vettore di repulsioni che permette agli agenti di spostarsi, da una regione con alta concentrazione, a una regione con concentrazioni più bassa. Così, dopo poche iterazioni, gli agenti raggiungono una distribuzione uniforme nell'ambiente.

Ispirazione: il meccanismo di repulsione appare in una vasta gamma di processi auto-organizzanti biologici (come il processo di diffusione in fisica, l'affollamento di uccelli o branchi di pesci). Consideriamo come esempio, il processo di diffusione di particelle che ,attraverso un moto casuale,permette di passare da regioni di maggiore concentrazione a regioni di concentrazione più bassa. Questa dinamica è riportata nella figura 8. innanzitutto, una concentrazione di inchiostro viene depositata nel vetro d' acqua. Lo stato iniziale (stato (a)) permette di osservare la concentrazione delle particelle in un angolo del vetro. L'angolo con le particelle, quindi, contiene una maggiore concentrazione di particelle di inchiostro. In un secondo luogo, le particelle iniziano a muoversi nel processo di diffusione, dalle regioni di maggiore concentrazione verso regioni di concentrazione più bassa (stato (b)). Più particelle sono all'angolo, maggiore è la concentrazione, in modo da creare un cosiddetto gradiente di concentrazione. Questo gradiente è fornito dalla differenza fra la concentrazione di particelle vicine. Infine, si osserva che, il processo di diffusione si è mosso in modo casuale attorno a tutte le particelle dentro l'acqua, producendo una distribuzione uniforme delle particelle. A questo punto le diverse concentrazioni di inchiostro scompaiono. All'interno di un contenitore, le particelle raggiungono una distribuzione uniforme dopo il processo di diffusione. Tuttavia, in uno spazio aperto, il processo di diffusione diffonde particelle fino a quando la concentrazione è così bassa che è considerata trascurabile. Come mostra la figura 8, il processo di diffusione termina quando le particelle raggiungono una distribuzione uniforme, cioè quando il gradiente di concentrazione diventa zero. Il meccanismo di repulsione è anche alternativamente presentato e ispirato dalla teoria dei gas. Nel caso della teoria del gas, il tempo per raggiungere una concentrazione uniforme è più breve rispetto al caso del processo di diffusione.

Vantaggi: i principali parametri legati al repulsion pattern sono la frequenza di repulsione (quanto frequentemente viene usata la repulsione) e il raggio di repulsione (cioè quanto forte è la repulsione). Un alta frequenza di repulsione comporta una più veloce diffusione degli agenti e un veloce adattamento quando l'area di diffusione cambia. Tuttavia, aumentano il numero di messaggi, perché il repulsion pattern richiede informazioni sulla posizione dei vicini. Il raggio di repulsione dovrebbe essere limitato al campo di comunicazione degli agenti, perché non rende senso di muoversi

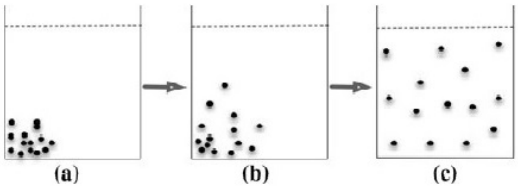


Figura 1.8: Diffusione

ad una posizione in cui la concentrazione di agenti è sconosciuta e anche perché gli agenti non possono saltare a un nodo che non è nel raggio di comunicazione. Così il movimento di un agente, in ogni fase di repulsione, deve essere limitato alla sua gamma di comunicazione.

Dinamiche agente-ambiente: la repulsione può essere applicata nei sistemi in cui gli agenti sono residenti in nodi mobili (ad esempio, sciami robotici) o in agenti software,dove hanno libertà di muoversi liberamente in una rete composta da nodi (fissi o meno). In entrambi i casi le dinamiche tra loro sono le stesse. Quando la repulsione è applicata, l'agente che esegue la repulsione, invia una posizione a tutti i suoi agenti vicini. Dopo di che, l'agente si mette in attesa di ricevere la posizione dai nodi vicini, una volta arrivata la risposta viene calcolata la posizione desiderata spostandosi in quella posizione. Quando l'ambiente è non continuo invece, come nel caso di agenti mobili, l'agente si muove verso il nodo più vicino dalla posizione desiderata. Per applicare il repulsion pattern, ogni agente dovrebbe sapere la sua posizione e la sua zona. Le regole di comportamento di transazione della repulsione sono espresse qua sotto:

Repulsion :: <L,C >, <L1,C1 >,..... <Ln,Cn > $\rightarrow$ <L,C' >, <L1,C1 >,... <Ln,Cn >

Where L' = ρ(<L,C >, <L1,C1 >,..... <Ln,Cn >)

La funzione ρ(<L,C >, <L1,C1 >,..... <Ln,Cn >) è data per calcolare la nuova posizione delle informazioni dell'agente, secondo la distribuzione spaziale dei vicini e della sua posizione effettiva. La stessa funzione dipende anche dai valori degli attributi contenuti in C (per esempio la concentrazione di particelle in ogni posizione).

Implementazione : una possibile implementazione raggiunge una distri-

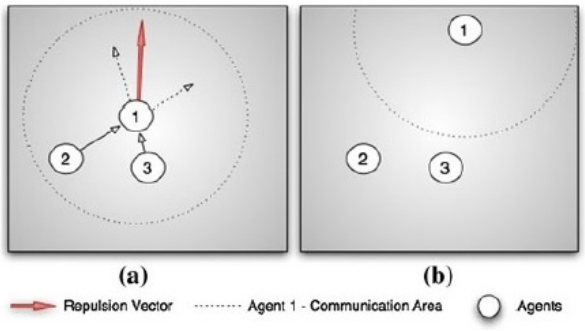


Figura 1.9: Area di repulsione

buzione uniforme , comportando una regola di transizione che calcola un vettore di repulsione tra le particelle, che sono inversamente proporzionali alla distanza tra di loro. La regola di transizione viene quindi calcolata come segue. Sia R il raggio di repulsione; d(i) la distanza fra un dato nodo e il suo nodo più vicino, p la posizione del nodo e p(i) la posizione data del vicino nodo i . La posizione p(t+1) dell'agente al tempo t +1 e il vettore spostamento m sono date da :

$p(t+1) = p(t) + m$

$m = \sum i\ (p-p(i)) / di\ (R-d(i))$

La Figura 9 mostra come l'agente 1 viene respinto dagli agenti 2 e 3 quando viene applicato il meccanismo di repulsione. In Fig. 1.9a l' agente 1 esegue l'algoritmo soprastante per creare la repulsione vettoriale. In Fig. 1.9b l' agente si muove seguendo il vettore di repulsione.

La Figura 1.10a mostra il comportamento di un agente che è l'esecuzione del repulsion pattern. All'inizio, gli agenti inviano una richiesta di posizione a tutti gli altri agenti all'interno del raggio di comunicazione. Quando le loro posizioni vengono ricevute, il vettore di repulsione viene calcolato. A questo punto se il sistema è composto da uno sciame di robot, il robot che è l'esecuzione nel repulsion pattern si sposta in una posizione desiderata. Se il repulsion pattern è in esecuzione, l'agente si sposterà verso il nodo vicino nella posizione desiderata. La Figura 1.10b mostra il diagramma di interazione tra l'agente che sta eseguendo la repulsione, il nodo in cui l'agente è in esecuzione e nei nodi vicini.

Impieghi noti: la repulsione non è stata proposta come un modello finora. Molte applicazioni hanno utilizzato il meccanismo di repulsione nell'ambito della robotica. La repulsione è utilizzata per coordinare la posizione dei

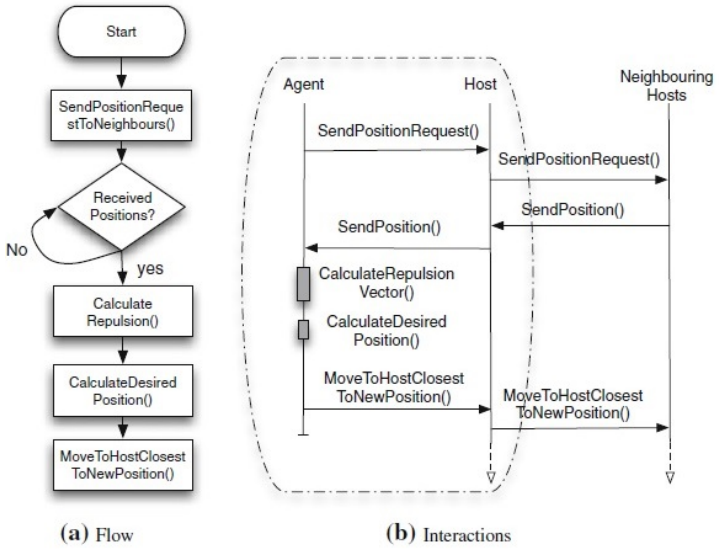


Figura 1.10: Repulsion

pezzi di informazioni garantendo l'accessibilità a tali informazioni in una specifica area di interesse utilizzando la minima memoria possibile.

Conseguenze: la repulsione non implica la replica, non vengono quindi creati nuovi agenti, contrariamente come avviene nella diffusione. La Repulsione è un processo continuo che produce una distribuzione uniforme degli agenti nel sistema.

Pattern correlati: la repulsione viene utilizzata nel Floacking Pattern.

1.6.2 Pattern composti

Questa sezione è realizzata per discutere i pattern composti ovvero pattern che derivano dall'aggregato di pattern atomici di base,creando di conseguenza pattern di livello secondario che a loro volta verranno utilizzati nei pattern ad alto livello. I pattern trattati in questo paragrafo sono il gradient pattern, il digital pheromone pattern e gossip pattern.

Gradient Pattern

Il gradient pattern rappresenta un'estensione dello spreading pattern in cui le informazioni vengono propagate in modo che vengono fornite ulteriori

informazioni a distanza dal mittente: aggiungendo sia le informazioni di distanza che informazioni di valori di concentrazione ,se il livello di concentrazione è elevato significa che il mittente è più vicino, come nei feromoni di formiche. Inoltre, il gradient pattern utilizza il modello di aggregazione per unire diversi gradienti creati da agenti differenti per unire gradienti provenienti dallo stesso agente ma attraverso diversi sentieri. Si possono verificare diversi casi: o vengono mantenute solo le informazioni con la distanza più breve del mittente, o la concentrazione delle informazioni aumenta.

Alias: il gradient pattern è un particolare tipo di campo di calcolo (cioè simile ai campi che vediamo in fisica basati sulle astrazioni).

Problema: gli agenti appartenenti ai grandi sistemi soffrono di mancanza di conoscenza globale per stimare le conseguenze delle loro azioni o le azioni eseguite da altri agenti oltre il loro raggio di comunicazione.

Soluzione: le informazioni si estendono dalla posizione iniziale e vengono depositate aggregando altre informazioni quando le si incontrano. Durante la diffusione, sono fornite ulteriori informazioni sulla distanza e sulla direzione del mittente, sia attraverso un valore che determina la distanza stessa (incrementata o decrementata), che attraverso informazioni che definiscono una concentrazione (concentrazione più bassa quando le informazioni sono più lontane e vice-versa). Così, gli agenti che ricevono gradienti hanno informazioni che provengono da oltre la loro comunicazione, aumentando la conoscenza del sistema globale non solo con informazioni di gradienti, ma anche con la direzione e la distanza della sorgente dei dati. Durante il processo di aggregazione, un operatore di filtro mantiene solo informazioni con la più alta (o più bassa) distanza, in altri casi viene modificata la concentrazione. Il Gradient pattern è in grado di affrontare modifiche della topologia di rete. In questo caso le informazioni si diffondono periodicamente.A questo punto viene applicata l'evaporazione che permette di ridurre la sua rilevanza nel tempo, consentendo quindi ai gradienti di adattarsi ai cambiamenti della topologia delle reti.

Vantaggi: l'adattamento ai cambiamenti ambientali è più veloce quando le frequenze di aggiornamento sono elevate, aumentando così il sovraccarico in rete.Le frequenze di aggiornamento più basse riducono il sovraccarico, ma può portare a valori obsoleti quando si verificano cambiamenti ambientali. C'è un trade-off tra il raggio di diffusione (numero di salti) e il carico di rete. Un raggio di diffusione più elevato porta informazioni più lontano dalla sua sorgente, fornendo una guida a distanza agli agenti. Tuttavia, aumenta il carico e può sopraffare la rete (Bea 2009).

Dinamiche entità-ambiente: i soggetti che agiscono nei gradient pattern sono gli agenti, i nodi e agenti infrastrutturali. Analogamente al modello di diffusione, quando si crea un gradiente, esso viene trasmesso ai suoi vicini assumendo che ogni tupla contiene un attributo D che rappresenta la distanza dalla corrente del nodo sorgente del gradiente.

spreading :: <L,[D,C] > → <Lk,[D+-ΔD,C] >

Where Lk = random(L1,. . . ..Ln)

aggregation :: <L,[D1,C] >,. . . .. <L,[Dn,C] > → <L,[Dn,C] > → <L,[D',C] >

where D' = min/max(D1,. . . ..,Dn)

Implementazione: gli agenti iniziano il processo inviando informazioni a tutti i loro vicini, come mostrato in Fig. 1.11b nel caso di valore di distanza. Quando un agente riceve l'informazione viene incrementato l'attributo di distanza o viene ridotto di conseguenza il valore di concentrazione delle informazioni, e inoltra il gradiente di nuovo a tutti i suoi vicini (diffusion pattern) come mostrato sul diagramma di flusso Fig. 11a e in fig. 11b per il caso con valore con la distanza. Quando un nodo riceve il gradiente, gli agenti infrastrutturali lo diffondono ulterioriormente. Si noti che questo modello può essere eseguito anche dagli agenti stessi. Quando un agente riceve più di un gradiente, essa impiega l'aggregazione (aggregation pattern) come mostrato nel diagramma di sequenza di Fig. 11c.

Impieghi noti: il modello di gradiente è stato utilizzato nei problemi come il coordinamento di sciami di robot, coordinamento degli agenti in videogiochi o routing in reti ad-hoc (Perkins 1999).

Conseguenze: il gradient pattern aggiunge un extra informazione (a distanza). La distanza può essere utilizzata per limitare il numero di salti durante il processo di diffusione.

Modelli correlati: il gradient pattern è una composizione del modello di spreading e di aggregation, esteso con il valore di distanza o di concentrazione di informazioni. Si verifica di solito nel morphogenesis pattern, chemotaxis pattern e nel quorum sensing pattern. Il modello di Gradiente può essere combinato con il modello di evaporazione per creare gradienti attivi per sostenere l'adattamento quando gli agenti cambiano le loro posizioni o le modifiche alla topologia di rete.

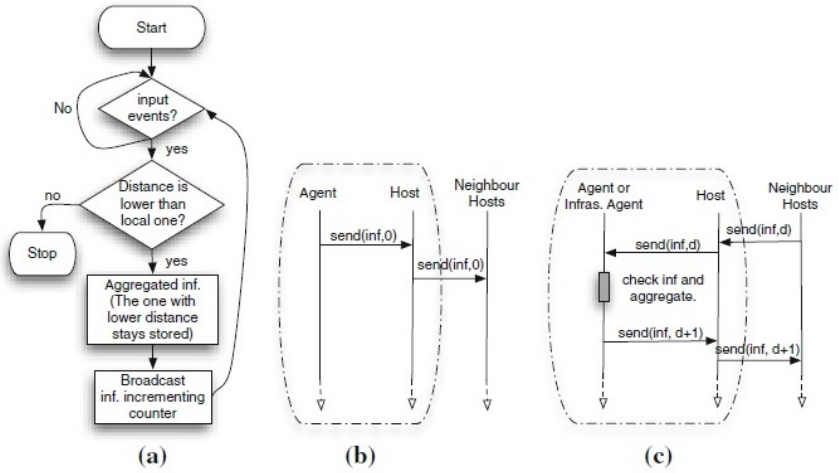


Figura 1.11: Gradient

Digital Pheromone Pattern

Il digital pheromone pattern è un modello di coordinazione basato sulla comunicazione indiretta. In questo modello, gli agenti depositano feromoni digitali nei nodi. Un feromone digitale è un marchio che diffonde un gradiente sull'ambiente e persiste in esso per un istante, scomparendo con il tempo. Altri agenti al di là della comunicazione possono quindi, ricevere informazioni trasmesse dai feromoni digitali.

Alias: nessuno di nostra conoscenza.

Problema: il coordinamento di agenti in ambienti di grandi dimensioni utilizzando la comunicazione indiretta.

Soluzione: un feromone digitale fornisce un modo per coordinare il comportamento dell'agente con la comunicazione indiretta ad alti ambienti dinamici.I feromoni digitali creano a loro volta dei gradienti che si sviluppano sull'ambiente, portando informazioni sulla loro distanza e direzione. Così, gli agenti possono percepire i feromoni dalla distanza e la crescita della conoscenza del sistema. Inoltre, col passare del tempo i feromoni digitali evaporano, fornendo adattamenti a cambiamenti ambientali.

Ispirazione: il pheromone digital pattern è stato ispirato osservando i comportamenti di colonie di formiche. Le colonie di formiche sono in grado di trovare il percorso più breve dal nido alle fonti di cibo utilizzando inte-

razioni locali grazie ai feromoni (favorendo la comunicazione indiretta). I feromoni sono depositati nell'ambiente dalle formiche per segnare il percorso che stanno seguendo dal nido alla fonte di cibo e viceversa. I feromoni evaporano rapidamente quindi devono essere rilasciati continuamente per mantenere le informazioni del percorso. Le colonie sono in grado di adattarsi ai cambiamenti dell'ambiente (ad esempio, nuovi ostacoli, nuovo cibo fonti, fonti di cibo che si svuotano, ecc..).

Vantaggi: ll pheromone digital pattern prevede l'implementazione del gradient pattern e dell'evaporation pattern al fine di creare un gradiente attivo. La differenza principale tra gradienti attivi e feromoni digitali è data dal modo in cui viene effettuata la comunicazione. Il pheromone digital pattern comunica attraverso feromoni tramite l'ambiente (comunicazione indiretta), mentre un gradiente propaga l'informazione da agente in agente. Inoltre come per il gradient pattern, il digital pheromone pattern è composto dall'aggregation pattern e dallo spreading pattern. La diffusione tramite feromoni riesce a raggiungere distanze lontane, permettendo a più agenti di ricevere le informazioni, ma comporta un consum di memoria e di banda maggiore.

Dinamiche agenti-ambiente: gli agenti sono le unica entità che possono depositare i feromoni. I feromoni sono depositati nei nodi, quindi gli agenti infrastrutturali applicano i meccanismi di diffusione, di aggregazione, e di evaporazione . Una volta che i feromoni sono diffusi e aggregati in ogni nodo, quando due o più informazioni di feromoni arrivano, evaporano fino a scomparire. La regola di transizione per il digital pheromone pattern è ottenuta componendo i tre modelli di base: spreading, aggregation e di evaporation, come mostrato sotto:

spreading :: < L,[PhV,C] > → <Lk,[PhV - Δ PhV,C] >

where Lk – random(L1,. . . .,Ln)

aggregation :: <L,[PhV1,C] >,. . . . . <L,[PhVn,C]→ <L,[PhVi,C] >

where PhVi – max(PhV1,. . . .Phvn)

evaporation :: <L,[PhV,C] → <L,[PhV',C] >

where PhV' = PhV *Evfactor

Implementazione: i feromoni digitali sono generalmente implementati utilizzando l'evaporazione statica (vale a dire lo stesso fattore di evapo-

razione viene utilizzato periodicamente per le informazioni del feromone). Indipendentemente dai modelli utilizzati per implementare il pheromone digital pattern,i feromoni possono essere depositati nei nodi, (cioè seguendo il modello proposto). Nel pheromone digital pattern, gli agenti appena depositano feromoni consentono agli agenti di diffondere,aggregare e evaporare i feromoni.

Impieghi noti: i feromoni digitali sono stati utilizzati principalmente nel coordinamento autonomo di suoni UV.

Conseguenze: l'implementazione di feromoni digitali fornisce i seguenti problemi al sistema:

(1)semplicità, rispetto alla logica necessaria in un centralizzato approccio;
(2) la scalabilità, poichè il digital pheromone pattern lavora in maniera totalmente decentrata, cioè applicabile in larga scala MAS;
(3) la robustezza, dovuta al decentramento e al continuo processo di autoorganizzazione digitale che fornisce feromoni(alcuni agenti possono fallire ma il sistema rimane abbastanza solidale per superare questi fallimenti).

Pattern correlati: il pheromone digital pattern è composto dagli schemi di gradiente ed evaporazione. Quest'ultimo si compone dai modelli di aggregazione e diffusione ma coinvolge anche i modelli di spreading e di evaporation. Il pheromone digital pattern è utilizzato dall' ant foraging pattern ad alto livello.

Gossip Pattern

L'obiettivo del gossip pattern è quello di ottenere un accordo condiviso sulle informazioni che vengono distribuite nel sistema,occorre quindi organizzare il sistema in maniera decentrata . Tutti gli agenti nel sistema collaborano per raggiungere progressivamente questo accordo : tutti contribuiscono con la propria conoscenza aggregando le conoscenze proprie con quelle dei vicini, diffondendo e aggregando a loro volta, le informazioni. Così, l'aggregation pattern aumenta la conoscenza e riduce l'incertezza di un singolo agente tenendo conto della conoscenza che possiedono altri agenti.

Alias : nessuno di nostra conoscenza.

Problema : in impianti di grandi dimensioni , gli agenti hanno bisogno di raggiungere un accordo , condiviso tra tutti gli agenti.

Soluzione : le informazioni vengono diffuse ai nodi vicini(spreading), dove vengono aggregate le informazioni locali(aggregatation)che poi verranno diffuse ulteriormente (di nuovo spreading).

Ispirazione: Il gossip pattern è ispirato dal comportamento umano legato alla diffusione di voci. Ad esempio le persone aggiungono le loro informazioni personali a informazioni ricevute da altre persone, aumentando le proprie conoscenze e diffondendo a sua volta una nuova conoscenza. Quando il processo viene ripetuto diverse volte,la gente inizia a condividere la stessa conoscenza che deriva dalla condivisione della conoscenza di diverse persone.

Vantaggi: il gossip pattern è composto dai modelli di diffusione e di aggregazione. Si presenta quindi lo stesso compromesso. Come nella diffusione, il principale problema di gossip è il sovraccarico della rete che viene prodotto dalla trasmissione continua realizzata dagli agenti. Per ridurre il sovraccarico di rete, la trasmissione deve essere ottimizzata per poi può essere applicata (ad esempio non tutti i vicini ricevono informazioni). Il numero di vicini di casa che ricevono l'informazione è il trade-off di questo modello. Più i vicini ricevono le informazioni, più il sistema è robusto in caso di guasti, ma produce ancora più sovraccarico in rete.

Dinamiche entità-ambiente: le entità coinvolte nel meccanismo di gossip sono gli agenti, agenti infrastrutturali e i nodi di rete. Le dinamiche tra le entità sono praticamente le stesse che troviamo sia nell'aggregazione che nella diffusione. Analogamente alla diffusione, solo un agente può avviare il processo. Quando un agente desidera avviare un processo di gossip, vengono inviate le informazioni (ad esempio i parametri e valori) per un sottoinsieme di vicini; se un agente è presente in un nodo, esso riceve le informazioni,le aggrega a quelle che già possiede. Le informazioni ricevute con le proprie informazioni vengono aggregate e re-inviate a un sottoinsieme di nodi vicini. Lo stesso comportamento è prodotto dagli agenti infrastrutturali quando nessun agente è presente in un nodo e il nodo stesso riceve un'informazione, in questo caso l'agente infrastrutturale riceve informazioni aggregando tutte le informazioni ricevute e le re-invia. Un agente o un agente infrastrutturale termina il processo di gossip quando le informazioni ricevute e le informazioni precedentemente inviate coincidono,il che significa che non ci sono informazioni aggiuntive da aggregare.

Implementazione: per quanto riguarda l'implementazione, la trasmissione ottimizzata può essere applicata. È stato dimostrato che eseguire un gossip (broadcast) con una probabilità tra 0.6 e 0.8 è sufficiente a garantire che quasi ogni nodo riceve il messaggio in quasi ogni esecuzione. Questa ottimiz-

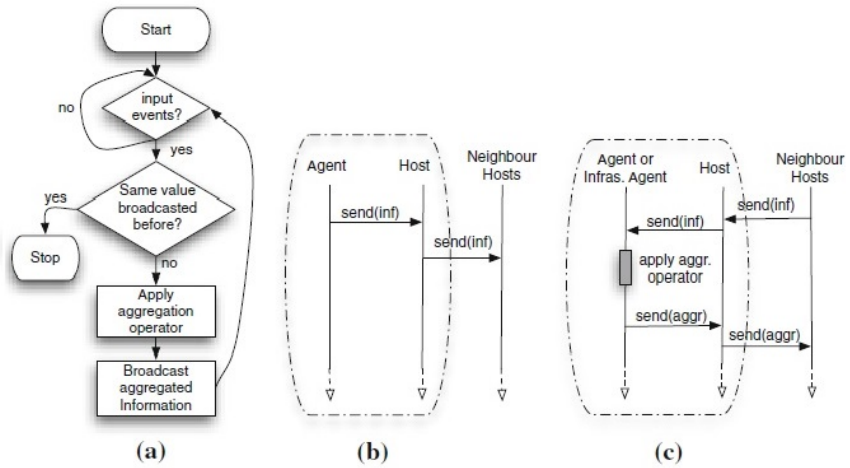


Figura 1.12: Gossip

zazione decrementa il numero di messaggi del 35%. La Figura 12a mostra il diagramma di flusso per il meccanismo di gossip , in cui le informazioni si diffondono attraverso la trasmissione. La figura 12b mostra l'interazione tra l'agente che avvia il processo di gossip e il nodo in cui l'agente è in esecuzione con i nodi vicini. Una volta che il gossip ha iniziato il processo , gli agenti e gli agenti infrastrutturali seguono il comportamento presentato in Fig. 12c.

Impieghi noti: lo riscontriamo in un semplice protocollo basato per il calcolo delle somme, medie e altre funzioni aggregate.

Conseguenze: il vantaggio principale del gossip è la robustezza. Anche in presenza di guasti, il modello è in grado di raggiungere l'accordo. Inoltre, fornisce un gossip di adattamento continuo quando nuovi valori arrivano nel sistema.

Pattern correlati: il pattern di a gossip si compone dei pattern di spreading e aggregation.

1.6.3 Pattern di alto livello

Questa ultima sotto-sessione, che rappresenta la parte finale di questo primo capitolo , è volto alla descrizione dei pattern che vengono utilizzati ad alto livello e che quindi forniscono una descrizione e una visione molto completa, adatta a sistemi auto-organizzanti complessi. Presentiamo qui solo quei

modelli che sono stati ampiamente accettati e utilizzati come meccanismi. I pattern in questione sono: ant foraging pattern , chemotaxis pattern , morphogenesis pattern , quorum sensing pattern , flocking pattern.

Ant foraging Pattern

L'ant foaging pattern rappresenta il design pattern ad alto livello che permette di descrivere l'attività in cui una serie di formiche collabora per ricercare del cibo. Questo pattern presenta una ricerca collaborativa decentrata. Principalmente, l'ant foraging pattern è stato applicato a problemi di ottimizzazione e in robotica.

Alias: Ant Colony Optimization (Dorigo 2002).

Problema: i grandi problemi di ottimizzazione di scala possono essere trasformati nel problema di trovare il percorso più breve su un grafo pesato.

Soluzione: l' ant foraging pattern fornisce regole per esplorare l'ambiente in modo decentrato permettendo lo sfruttamento delle risorse che riserva l'ambiente (sfruttando il pheromone digital pattern).

Ispirazione: si ispira al comportamento dell'ant colony optimization. Le formiche all'interno di una colonia, per trovare il percorso più breve dal nido al cibo (e viceversa), adottano un comportamento coordinato sfruttando l'ambiente. Colonie di formiche usano una comunicazione stigmergica (modificando l'ambiente e depositando un feromone). Il feromone, come abbiamo visto nel digital pheromone pattern, è depositato nell'ambiente per attrarre altre formiche.Quindi seguendo la più alta concentrazione di feromoni, le formiche trovano il percorso più breve dal nido al cibo, e adattano questo percorso quando ci sono gli ostacoli o quando il cibo è esaurito.

Vantaggi: ogni formica ha una probabilità di seguire il gradiente prodotto dai feromoni. Quando una formica non segue il gradiente, cammina casualmente nell'ambiente alla ricerca di nuove risorse (e quindi si può dire che va in esplorazione). Quando la probabilità di esplorazione è elevata (e quindi quando non segue il gradiente), le formiche si adattano più velocemente ai cambiamenti dell'ambiente , ma sono più lenti nel raggiungere le risorse. Considerando che, con un basso livello di esplorazione (ovvero seguendo il gradiente),le formiche sono in rapido sfruttamento delle risorse, poiché la maggior parte delle formiche seguono la risorsa del percorso. Tuttavia, a causa della mancanza di esplorazione, quando la risorsa è esaurita le formiche spendono più tempo del previsto per trovare nuove risorse e di conseguenza l'adattamento ai cambiamenti dell'ambiente diventa più lento.

Inoltre, l' ant foraging pattern presenta la stesso vantaggio del pheromone digital pattern. Se la velocità di evaporazione del feromone è troppo bassa, il profumo del feromone non evapora abbastanza in fretta. L'ambiente si riempie con feromoni e lo sfruttamento non è efficiente. L'alto tasso di evaporazione invece consente al feromone di evaporare prima che le formiche possano costruire un percorso e mantenerlo, riducendo quindi lo sfruttamento e incrementando l'esplorazione.

Dinamiche entità-ambiente: le entità coinvolte nell'ant foraging pattern sono gli stessi che agiscono nel digital pheromone pattern. Quando un agente rileva la presenza di un feromone digitale, decide di seguire il gradiente o di spostarsi in modo casuale . La regola di transizione descrive il comportamento di foraggiamento delle formica qua sotto

up_move :: <L,[PhV1,C] >,. . . . . , <L,[PhVn,C] > → <Li,[PhVi,C] >

where Phvi = max(PhV1,. . . .,PhVn)

random_move :: <L,C > → < Li,C >

where Li = random(Li,. . . .,Ln)

Il primo modello regola un agente che rileva i valori del feromone nella sua posizione e nel quartiere, e quindi segue la direzione del massimo valore di gradiente per trovare cibo. Il secondo modello regola un agente che si muove in modo casuale. Entrambe le regole sono soggette a un tasso che regola lo sfruttamento dell'attività di esplorazione.

Implementazione: secondo alcune probabilità di esplorazione,gli agenti o seguono delle entità scout (cioè dei soggetti reclutati per cercare cibo), o eseguono qualche ricerca casuale. Nel caso di formiche "scout", i feromoni sono depositi nel loro ambiente, per poi essere percepiti da altre formiche per trovare fonti di cibo. La Figura 13a mostra il comportamento generale di formiche, mentre la Fig. 13b mostra il comportamento delle formiche in cerca di cibo, a seguito di un sentiero o prendendo un percorso casuale, infine fig. 13c viene visualizzato il ritorno al nido, cadere feromone, una volta un pezzo di cibo è stato trovato.

Impieghi noti: il modello dell'ant foraging pattern è stato principalmente applicato come già detto nell' Ant Colony Optimization.

Conseguenze: il sistema raggiunge prestazioni di alta qualità a problemi di ricerca NP-hard.

Pattern correlati: Il pattern sfrutta il digital pheromone pattern e utilizza l'evaporazione, la diffusione e l'aggregazione.

Chemotaxis pattern

Il chemiotaxis pattern fornisce un meccanismo per eseguire il coordinamento del movimento in sistemi su larga scala di sistemi MAS. Il chemiotaxis pattern estende il gradient pattern:gli agenti identificano la direzione del gradiente per decidere la direzione dei loro prossimi movimenti.

Alias: nessuno di nostra conoscenza.

Problema: la coordinazione dei movimenti decentralizzata che mira a individuare le fonti o confini di eventi.

Soluzione: gli agenti avvertono localmente informazioni di gradiente e lo seguono in una direzione specifica (cioè seguono i valori più elevati di pendenza, i valori più bassi di gradienti.

Ispirazione: in biologia ,la chemiotassi è il fenomeno in cui gli organismi di singole o pluricellulari dirigono il loro movimento secondo alcune sostanze chimiche presenti nel loro ambiente. Ad esempio in natura le cellule dei leucociti si muovono verso una regione di un'infiammazione batterica o batteri che migrano verso alte concentrazioni di nutrienti.
Si noti che in biologia, la chemiotassi è anche un meccanismo di base della morfogenesi. Essa guida le cellule durante lo sviluppo in modo che possano essere messe in giusta posizione finale. In questo lavoro, in seguito (NAGPA 2004), il termine chemiotassi è usato come coordinazione del movimento seguendo i gradienti, mentre il termine viene usato per morfogenesi innescando specifici comportamenti sulla base di posizioni relative determinate attraverso un gradiente.

Vantaggi: il chemiotaxis pattern sfrutta il Gradient Pattern. Nel Chemiotaxis pattern la comunicazione gioca un ruolo importante. Quando il campo di comunicazione è lungo, gli agenti si muovono più velocemente seguendo i gradienti. Questo, tuttavia, causa problemi di fonti.

Dinamiche entità-ambiente: la concentrazione di gradiente guida i movimenti degli agenti in tre diversi modi, come mostrato in Fig. 15:
(1) il movimento attraente, quando gli agenti cambiano le loro posizioni seguendo l'alta concentrazione di valore di gradienti;

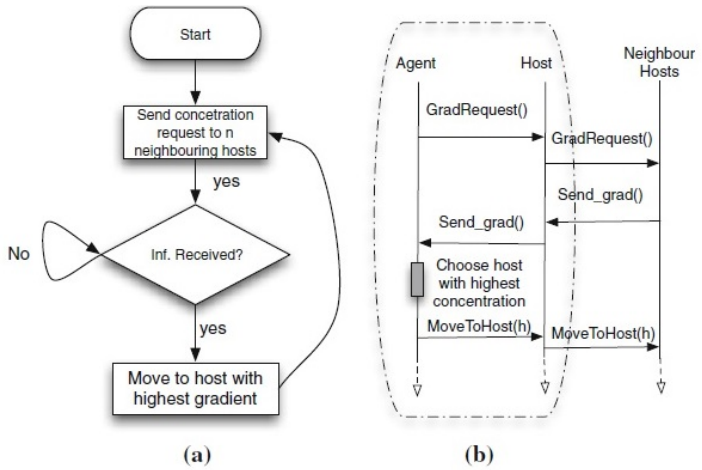


Figura 1.13: Chemotaxis

(2) il movimento repulsivo, quando gli agenti seguono valori di gradiente inferiore, incrementando la distanza tra l'agente e la sorgente del gradiente; (3) il movimento equipotenziale, quando gli agenti seguono pendenze tra soglie.

Data la regola di transizione che crea il gradiente. La regola di transizione determina il movimento agente verso il più alto, più basso, o il valore di gradiente di potenziale(a seconda dei casi).

Move :: <L,[D1,C]¿,. . . . . <L,[Dn,C] > → <Li,[Di,C] >

Where Di = min/max/equals (D1,. . . .,Dn)

Implementazione: il chemotaxis pattern può essere implementato in due modi diversi. In primo luogo, utilizzando gradienti esistenti nell'ambiente per coordinare le posizioni dell'agente o con movimenti attraenti per rilevare fonti di eventi diffuse attraverso un approccio multi-agente su una infrastruttura di rete di sensori.

In secondo luogo, vengono utilizzando campi di gradienti generati dagli agenti (ad esempio, utilizzando un approccio basato su gradiente per coordinare la posizione. I diagrammi 14 a e b mostrano un caso particolare di applicazione,dove gli agenti ottengono informazioni su gradienti vicini, prima di prendere una decisione su dove andare al prossimo. Come mostrato nel diagramma 14a, ogni agente sceglie un nodo vicino casuale e invia loro un gradiente a grande richiesta di concentrazione. L'agente sceglie il nodo

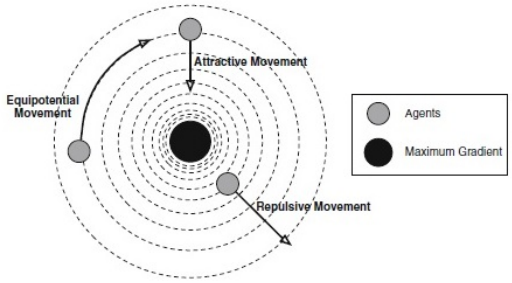


Figura 1.14: Chemotaxis-Agent

vicino che ha un gradiente di concentrazione più alta e lo sposta lì. Ripetendo questo processo l'agente è in grado di trovare la fonte di gradiente.

Impeghi noti: il chemotaxis pattern è utilizzato per coordinare la posizione di uno sciame di semplici robot mobili. Inoltre viene applicata per instradare i messaggi in scenari di calcolo.

Pattern correlati: chemotaxis pattern estende il gradient pattern.

Morphogenesis Pattern

L'obiettivo del morphogenesis pattern è quello di selezionare differenti comportamenti dell'agente a seconda della posizione in cui esso si trova nel sistema. Anche il morphogenesis pattern sfrutta il gradient pattern per valutare alcune informazioni sulla posizione attraverso una o più fonti di gradiente generati da altri agenti presenti. La Morfogenesi è stata scelta come esempio di rappresentazione di meccanismi auto-organizzanti. Il processo di morfogenesi in biologia è stato considerato come fonte di ispirazione per i campi di gradiente.

Alias: nessuno a nostra conoscenza.

Problema: in larga scala di sistemi decentrati, agenti possono decidere sul loro ruolo o pianificare le loro attività sulla base della loro posizione spaziale.

Soluzione: Gli agenti sono in grado di valutare le loro posizioni nel sistema per calcolare la loro distanza relativa alla fonte morfogenetica di gradiente.

Ispirazione: nel processo di morfogenesi in biologica alcune cellule creano e modificano le molecole (attraverso l'aggregazione), si diffondono (attraverso la diffusione andando quindi a creare, attraverso il gradiente, delle molecole).

Vantaggi: i vantaggi presentati in questo modello sono uguali a quelli del Gradient Pattern.

Dinamiche entità-ambiente: le entità coinvolte nel processo di Morfogenesi sono gli agenti,gli agenti infrastrutturali e i nodi. All'inizio, alcuni degli agenti diffondono uno o più pendenze di Morfogenesi, utilizzando il modello di gradiente. Altri agenti utilizzano il gradiente morfogenetico per calcolare la loro relativa posizione. A seconda delle loro posizioni relative, gli agenti possono adottare ruoli diversi e coordinare le loro attività in modo da raggiungere gli obiettivi di collaborazione. Le regole di transizioni sono espresse qua sotto:

state_evolution :: $<$ L,[D,State,C] $\rightarrow$ $<$ L,[D,State',C] $>$

where State' = Π (D);

L'obiettivo della funzione Π(D) nella morfogenesi è quello di cambiare le variabili di stato dell'agente,evoluzionando il suo stato in base alle informazioni localmente percepite nell'ambiente.

Implementazione: il diagramma 16 mostra come gli agenti stimano la loro posizione in base alla risposta alle informazioni gradiente propagata dai nodi vicini.

Impieghi noti: il morphogenesis è usato per implementare tecniche di controllo per modulare robot auto-configurabili.

Conseguenze: Il morphogenesis equipaggia gli agenti con un meccanismo per coordinare le loro attività sulla base delle loro posizioni relative. Come altri meccanismi precedentemente presentati,la robustezza e scalabilità sono proprietà garantite da questo modello.

Pattern correlati: il morphogenesis pattern estende il gradient pattern. Il modello di morfogenesi può essere combinato con il digital pheromone pattern dove il ruolo e il comportamento degli agenti dipendono dalle distanze delle fonti di feromoni.

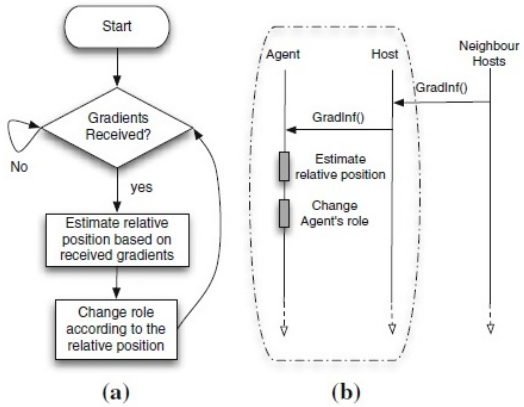


Fig. 16 Morphogenesis: agent behaviour (a), agent interaction (b)

Figura 1.15: Morphogenesis

Quorum Sensing Pattern

Il Quorum sensing Pattern è un processo decisionale per la coordinazione del comportamento e per prendere decisioni collettive in un modo decentralizzato. L'obiettivo del quorum sensing pattern è quello di fornire una stima del numero di agenti (o di la densità degli agenti) nel sistema usando solo interazioni locali . Il numero di agenti nel sistema è cruciale in quelle applicazioni in cui un numero minimo di agenti sono necessari per collaborare su compiti specifici.

Alias: Nessuno a nostra conoscenza.

Problema: le decisioni collettive in larga scala di sistemi decentrati, richiedono un numero di soglia di agenti o di stima della densità di agenti in un sistema, utilizzando solo interazioni locali.

Soluzione: il quorum sensing pattern permette di prendere decisioni collettive, attraverso una stima dai singoli agenti di densità (valutare il numero di altri agenti con cui interagiscono) e per determinare un numero soglia di agenti necessari per prendere la decisione.

Ispirazione: il quorum sensing pattern in uso si ispira al processo quorum sensing (QS), che è un tipo di segnale intercellulare utilizzato dai batteri per monitorare la densità cellulare per una varietà di scopi. Questi batteri si auto-organizzano e, il loro comportamento, permette di produrre luce solo quando la densità dei batteri è sufficientemente elevata. Un altro esempio

interessante è dato dalle colonie di formiche che deve trovare un nuovo posto per il nido. Una piccola porzione delle formiche ricerca nuovi potenziali posti di nidificazione e valuta la loro qualità. Quando ritornano al vecchio nido, aspettano un certo periodo di tempo prima di assumere altre formiche (valutazioni superiori producono periodi di attesa inferiori).Le formiche reclutano altre formiche per visitare il posto di un potenziale nido e fare una propria valutazione sulla qualità che ooffre un nuovo nido per poi tornare al vecchio nido e quindi ripetere il processo di reclutamento. A causa del tempo di attesa in alcuni periodi, il numero di formiche presenti nel migliore nido tendono ad aumentare. Quando le formiche in questo caso incontrano altre formiche ,superando una particolare soglia, viene raggiunto il numero legale. Altri sciami, come api o vespe, usano la stessa tecnica per il loro nido.

Vantaggi: il quorum sensing pattern utilizza gradienti e quindi presenta gli stessi parametri del gradient pattern. La soglia, indicando che il numero massimo è stato raggiunto, innesca un comportamento di collaborazione. Il quorum sensing fornisce una stima della densità di agenti nel sistema. Tuttavia, questo pattern non fornisce una soluzione per calcolare il numero di agenti necessari per svolgere un compito collaborativo (cioè per identificare il valore di soglia).

Dinamiche entità-ambiente: le entità coinvolte nel quorum sensing pattern sono le stesse che troviamo nel gradient pattern.Vale a dire, agenti, nodi, ed agenti infrastrutturale. La concentrazione è stimata dalla aggregazione del gradiente. La regola di transizione per il pattern può essere modellata attraverso regola di transizione espressa qui sotto:

$\Pi(D) =$ State if $D \leq$ threshold

State' if $D >$ threshold

Implementazione: non vi è alcuna implementazione specifica per il Quorum Sensing Pattern. Tuttavia, i sistemi biologici presentati sopra possono darci qualche idea su come implementare il modello. Qui vi proponiamo due diversi approcci per implementare il modello di Quorum Sensing:
(1)per utilizzare il modello di Gradiente per simulare gli auto-induttori come nei batteri bioluminescenti. In questo caso le concentrazioni di gradiente forniscono agli agenti una stima della Densità degli agenti;
(2) come nei sistemi di formiche , degli agenti della densità può essere stimata attraverso la frequenza a cui gli agenti sono nel raggio di comunicazione. L'uso di gradienti fornisce stime migliori rispetto l'uso delle frequenze.

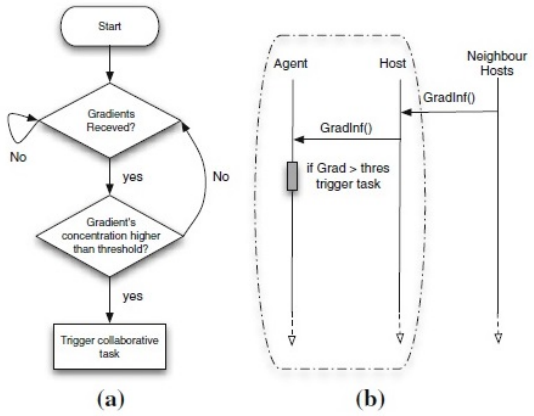


Figura 1.16: Quorum Sensing

Tuttavia, è più costoso e richiede computazionalmente più le comunicazioni di rete.
I diagramma 17 a e b mostrano come gli agenti identificano il gradiente di concentrazione e eventualmente se ha raggiunto la soglia. In caso affermativo, le risposte vengono propagate alle informazioni del gradiente propagato da nodi vicini.

Impieghi noti: il quorum sensing pattern viene utilizzato per aumentare il risparmio energetico in reti di sensori wireless. Un altro noto esempio è il coordinamento delle autonome Swarm Robots.

Conseguenze : ogni agente è in grado di stimare la densità dei nodi o la densità di altri agenti nel sistema usando solo informazioni locali ricevute dai vicini , anche quando il sistema è molto grande e gli agenti sono anonimi.

Pattern correlati: il quorum sensing pattern , a seconda sulla sua applicazione , usa il modello di gradiente.

Floacking Pattern

Il flocking pattern è una sorta di coordinazione dinamica .Il comportamento è analogo a quello di un branco auto-organizzato di animali di grandi dimensioni, spesso in movimento in massa o in migrazione e nella stessa direzione e con un obiettivo comune di gruppo. Il flocking patttern è in grado di controllare le formazioni di pattern dinamici e spostare gli agenti sull'ambiente mentre mantengono la loro omogeneità, le interconnessioni tra essi e cercano di evitare le collisioni.

Alias:nessuno di nostra conoscenza

Problema: la coordinazione dinamica

Soluzione: il flocking pattern fornisce un insieme di regole per gruppi di agenti in movimento sull'ambiente mentre mantengono la formazione e le interconnessioni tra di loro.

Ispirazione: questo modello si ispira al comportamento di un gruppo di uccelli quando sono in volo in gruppo oppure è ispirato a un insieme di pesci quando cercano di evitare un attacco di un predatore. Ad esempio, quando un insieme di pesci è sotto un attacco di un predatore, il movimento del primo pesce di rilevamento è fondamentale perchè produce un movimento veloce per allertare altri pesci da onde di pressione inviate attraverso l'acqua.
E quindi poi il gruppo di pesci cambia la loro formazione per la prevenzione all'attacco da predatori, recuperando la formazione iniziale dopo il tentativo di pesca da parte del pescatore in nave.

Vantaggi: parametri come, a distanza di evitare, massima velocità e accelerazione massima devono essere sintonizzate per conseguire il coordinamento di movimento desiderati.

Dinamcihe entità-ambiente: i soggetti partecipanti al Flocking Pattern sono solo agenti che usano la comunicazione diretta. Fondamentalmente,gli agenti rilevano la posizione del loro nodo e mantengono una costante distanza desiderata. Quando la distanza cambia a causa di perturbazioni esterne, ogni agente risponde in modo decentralizzato per controllare la distanza e recuperare il pattern di formazione originaria. La figura 18 rappresenta due agenti che coordinano il loro comportamento secondo il primo termine (coesione e separazione):
(A)Gli agenti sono attratti l'uno dall'altro, perché sono situati in una zona in questi settori;
(B) Gli agenti si respingono perché sono troppo vicini;
(C)Gli agenti sono nella zona neutra in cui il termine diventa zero.

Impieghi noti: la prima applicazione del flocking pattern è stata la modellazione del comportamento animale per i film. Specificamente,è stato utilizzato per generare realistiche mosse di folla.

Conseguenze: l'affollamento cerca di generalizzare il comportamento di floacking, indipendentemente da individui (uccelli, pinguini,pesce, ecc). Il

suo comportamento non dipende dai metodi utilizzati per la generazione delle traiettorie degli agenti. Il flocking pattern fornisce robustezza e di auto-guarigione proprietà quando di fronte a fallimenti di agenti e problemi di comunicazione.

Pattern correlati:ll flocking pattern estende il repulsion pattern . Infatti,la repulsione può essere vista come una semplificazione del flocking pattern dove l'incompatibilità del vettore viene presa in considerazione per il calcolo della posizione successiva.

Capitolo 2

TuCSoN-ReSpecT

I sistemi distribuiti sono sistemi situati. Sistemi che sono immersi in un ambiente che risulta reattivo a eventi di qualsiasi tipo. Quindi questo presuppone che l'interazione con l'ambiente avviene in maniera proficua e continua in maniera coordinata. I linguaggi e modelli di coordinazione negli ultimi anni hanno riscosso notevoli risultati nei sistemi complessi MAS (multi-agente). Questo secondo capitolo di tesi è volto alla presentazione del modello di coordinazione e del rispettivo linguaggio utilizzato. Nel primo capitolo sono stati esplicitati chi sono i soggetti (Design Pattern)che andiamo a trattare,il loro comportamento, le eventuali interazioni e l'ispirazione presa dalla biologia. Per simulare e testare comportamenti auto-organizzanti è necessario lavorare, in maniera coordinata, in uno spazio di risorse condiviso, che separi lo spazio di informazione dallo spazio di specifica; questo spazio viene definito centro di tupla. In questo capitolo vengono spiegate tutte le nozioni fondamentali dell'infrastruttura (architettura,middleware,tools ecc..). Nella seconda parte invece, vengono spiegate tutte le caratteristiche del linguaggio ReSpecT, utilizzato nell'infrastruttura TuCSoN, in modo da da fornire tutti i requisiti necessari per un' implementazione dei comportamenti di un sistema auto-organizzante tramite design pattern nel capitolo successivo.

2.1 Dagli spazi di tupla ai centri di tupla

Prima di fornire la presentazione del modello TuCSoN e del linguaggio ReSpecT, è necessario andare a spiegare in maniera più approfondita il concetto di centro di tupla (che cos'è e da dove nasce) [2]. La tecnologia di sistemi multi-agente (MAS) sta diventando un paradigma chiave e onnipresente in rete Internet. In particolare risulta sempre più importante andare a definire come gestire uno spazio di coordinazione e il modo in cui gli agenti interagiscono in esso attraverso un modello di coordinazione basato sulle tuple. I primi modelli di coordinazione che si basano sulle scambio di tuple provengono da Linda , e sono stati inizialmente utilizzato nel campo di

programmazione parallela. Il loro utilizzo è stato adatto anche per il coordinamento di sistemi aperti, distribuiti e eterogenei. Le informazioni vengono depositate dagli agenti nello spazio chiamato "spazio di tuple", nel quale avviene uno scambio di informazioni gestito dagli agenti. Quindi, in questo spazio gli agenti cooperano,sincronizzano e comunicano tra loro memorizzando,leggendo e consumando tuple in modo associativo. La coordinazione nello spazio di tupla è gestita in maniera a information-driven dove gli agenti sincronizzano e cooperano tra di loro in uno spazio condiviso scambiando informazioni.Tuttavia, i modelli di coordinazione basati sullo spazio di tupla (ad esempio Linda) rappresentano dei limiti perchè non c'è modo di rappresentare in maniera separata come l'informazione viene rappresentata e, come essa viene percepita nella comunicazione agente. E' stato necessario quindi, riuscire a superare questo limite definendo oltre a uno spazio di tupla anche uno spazio di specifica in cui inserire delle politiche di coordinazione in risposta agli eventi di comunicazione.Questo concetto è stato rappresentato come centro di tupla . La definizione di centro di tupla infatti è definita come uno spazio di tupla arricchito da uno spazio di specifica di comportamento, esplicitando un comportamento da adottare in risposta agli eventi di comunicazione. I centri di tupla vengono programmati attraverso il linguaggio di programmazione ReSpecT che utilizza TuCSoN come modello di coordinazione in rete.

2.2 TuCSoN

Nel paragrafo precedente è stato spiegato cosa rappresenta un centro di tupla e come è nato questo concetto. Adesso invece andiamo a analizzare qual' è il modello di coordinazione che sfrutta il centro di tupla per effettuare la coordinazione e quali sono i suoi aspetti principali. TuCSoN (Tuples centres spread over the network) ,la quale traduzione è espressa come " centri di tuple che si espandono in rete", rappresenta un modello di coordinazione per processi distribuiti, nonché di entità definite come agenti autonomi,intelligenti e mobili. Il capitolo suddivide in sotto-sezioni le caratteristiche chiave della piattaforma TuCSoN specificando modello e arichitettura, il Middleware e i tools utilizzati. Riferimenti presi dalla guida di TuCSoN[3] di Omicini e Mariani.

2.2.1 TuCSoN: modello e architettura

Come già accennato la piattaforma TuCSoN consente di coordinare le richieste degli agenti in maniera efficiente senza creare intralci al sistema. Gli agenti nell'ambito di TuCSoN sono quindi entità che sono coordinabili; cioè entità in grado di cooperare, sincronizzare e competere sulle base di tuple

all'interno di uno spazio di coordinazione che in TuCSoN è formato da tanti nodi che corrispondono ai server collegati in rete andando a definire una topologia di rete. Ogni nodo può contenere più centri di tupla, ciascuno identificato da un nome logico. Gli agenti,i nodi e i centri di tupla rappresentano un unico sistema TuCSoN. Ogni nodo in rete viene identificato tramite una coppia di valori:

< NetworkId,PortNo >

- *NetworkId* rappresenta il numero Ip per l'indirizzamento del nodo.

-*PortNo* rappresenta invece il numero di porta dove vengono ascoltati ed eseguiti i servizi di coordinamento.

La sintassi quindi per rappresentare un nodo ospitato in un dispositivo di rete è rappresentata cosi:

netid : portno.

Una volta identificato l'indirizzo di rete NetId e il numero di porta, è necessario associare un nome ammissibile per il centro di tupla. Dal punto che per ogni nodo si può associare al massimo un nome ammissibile per il centro di tupla,è importante andare a definire un nome che sia univoco,con lo scopo di riuscire a identificare un sistema TuCSoN (tname @ netid:portno). E' molto importante anche, andare a identificare un nome ammissibile per gli agenti che entrano nel sistema TuCSoN; appena entrano a far parte del sistema stesso vengono identificati universalmente tramite un UUID (universally unique identifier), in modo da essere rappresentati tramite la coppia aname:uuid . Le operazioni di coordinazione nello spazio sono definiti come primitive di coordinazione. Le operazioni in TuCSoN sono invocate da un'agente sorgente e ognuna di esse è suddivisa in due fasi:

-*invocazione*: viene fatto una richiesta dall'agente sorgente portando con sé tutte le informazioni sull'invocazione;

-*completamento*: la risposta torna dal centro di tupla attraverso l'agente sorgente,portando con sé tutte le informazioni necessarie per l'esecuzione.

La sintassi per indicare un operazione è espressa dall'abbreviativo op.

Es. tname @ netid : portno ? op

Le primitive di cui facevamo riferimento prima per costruire un linguag-

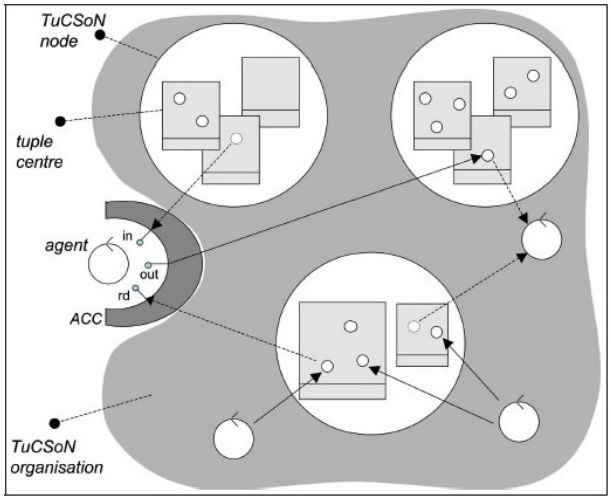


Figura 2.1: Archit.TuCSoN

gio di coordinamento sono le seguenti:

-out,rd,in

-rdp,inp

-no,nop

-get,set

In breve per inserire una tupla nello spazio di tuple viene utilizzata la primitiva di coordinazione out. Mentre per andare ad accedere a una tupla che si trova su uno spazio di tupla si usano le primitive rd e in. La sostanziale differenza sta nel fatto che la prima legge la tupla mentre la seconda la rimuove dallo spazio di tuple cercando tra il tamplate di tuple. Se la tupla non è disponibile,si attende che un'altra primitiva (tramite un operazione di out) sblocchi l'operazione. Mentre le primitive rdp e inp hanno un comportamento non bloccante, ovvero se la tupla non è disponibile allora l'esecuzione fallisce. È' possibile accedere e modificare lo spazio di tuple attraverso le primitive get e set cambiandone anche il suo stato. Le politiche delle operazioni svolte dagli agenti è dato dall'ACC (Agent coordination context) che rappresenta se è possibile effettuare un operazione o meno. Per programmare centri di tupla con degli spazi di tupla e spazi di specifica adatti, TuCSoN utilizza ReSpecT come linguaggio di programmazione per programmare politiche di coordinamento tramite primitive. La figura sottostante stabilisce la vista architetturale di TuCSoN.

2.2.2 TuCSoN Middleware

Nel middleware di TuCSoN possiamo vedere:

-Api Java per l'estensione delle primitive di TuCSoN all'interno del package *alice.tucson.api **

-Classi Java per programmare agenti TuCSoN in Java all'interno del package *alice.tucson.api.AbstractTuCSoNAgent*, che rappresentano dei thread che usano direttamente le primitive TuCSoN.

-Librerie Prolog per l'estensione dei programmi tuProlog per le primitive di coordinazione TuCSoN all'interno del package *alice.tucson.api.TuCSoN2PLibrary*. Esso permette, tramite librerie tuProlog, di usare le primitive TuCSoN.

Per prima cosa, per andare a interagire con la piattaforma TuCSoN, è necessario effettuare una connessione,lanciando la classe *TuCSoNNodeService* dove un agente TuCSoN richiede di connettersi al nodo TuCSoN per interagire con il centro di tupla specificando *"tname,@netid,portno"* (se si vuole lavorare in locale si mette il nome del centro di tupla di default,l'indirizzo IP locale e la porta di default 20504). Una volta effettuata la connessione alla porta, è possibile interagire direttamente con l'agente che richiede i servizi attraverso delle richieste di operazioni per essere eseguiti. L'agente TuCSoN viene creato con un opportuno identificativo ID (classe *TuCSoNAgentId*). In seguito, una volta definito, ad ogni agente viene associato un Meta-ACC (*TuCSoNMetaACC*) indispensabile per ottenere un ACC che permette all'agente di interagire con il centro di tupla (rappresentato dalla classe *TuCSoNTupleCentreId*, anch'esso associato con un identificativo). Nell'interfaccia *ITuCSoNOperation*, sono rappresentate le operazioni eseguite,al suo interno è possibile vedere se esse hanno avuto successo o no.

La classe *AbstractTuCSoNAgent* è una classe astratta che permette di assegnare a un *TuCSoNAgentId* un ACC,per interagire con il centro di tuple e invocare le primitive espresse anche in prolog (*TuCSoN2PLibrary*)

2.2.3 TuCSoN tools

Una volta effettuata la connessione al nodo è possibile andare a interagire con esso ed è possibile osservare cosa realmente succede all'interno del centro di tupla. La piattaforma TuCSoN dispone di una *CommandLineInterpreter* attraverso la quale l'agente ,se la connessione è stabile,interagisce direttamente con il centro di tupla. Invocando il comando "help" è possibile

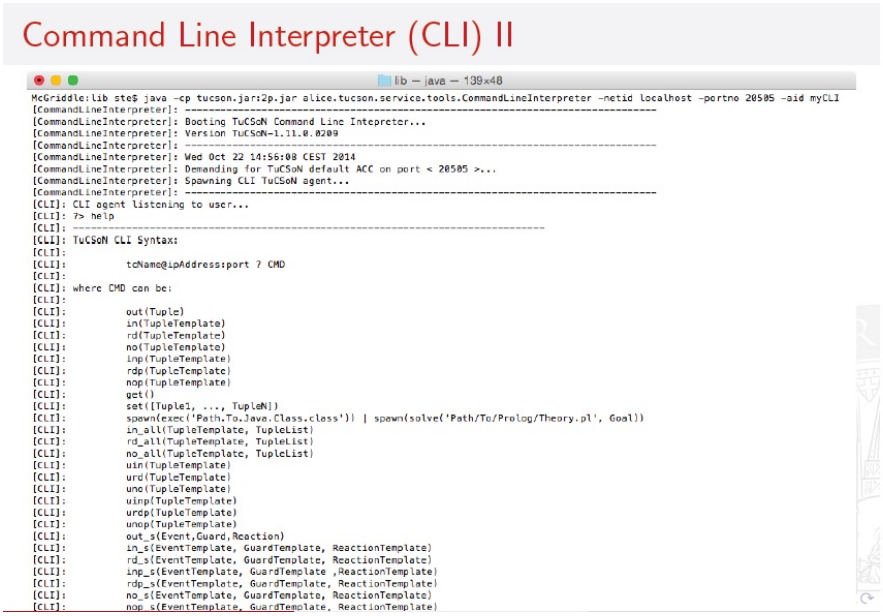


Figura 2.2: Command Line Interpreter

vedere quali sono le primitive disponibili da effettuare (riferimento a figura 2.2).

Ad esempio, se io da CommandLine eseguo la primitiva "out(hello)" non faccio altro che scrivere nello spazio di tuple la tupla "hello". Inoltre è possibile ispezionare varie caratteristiche del centro di tupla, attraverso l'interfaccia *TuCSoNInspector*, in modo da rendere visibile in maniera chiara e concisa al programmatore (agente) cosa realmente accade all'interno del centro di tupla.

Una volta selezionato il centro di tupla attraverso i dati (nome del centro di tupla,indirizzo IP e porta), controllando che ci sia una connessione disponibile, è possibile osservare quattro aspetti essenziali in un centro di tuple:

-*Tuple Space*: nello spazio di tuple sono rappresentate tutte le tuple che vengono inserite nel centro di tuple a runtime.

In questa finestra si possono osservare i cambiamenti in maniera proattiva o in maniera semplicemente reattiva osservando solo quando richiesto.

-*Specification Space*: spazio che, al suo interno, contiene spazi di specifica implementati in ReSpecT.Al suo interno vengono inserite le reaction in modo da assegnare a ogni evento una reazione in modo da definire delle specifiche di comportamento da adottare nei centro di tupla. E' possibile caricare un file ReSpecT, cambiare e settare un centro di tupla corrente,

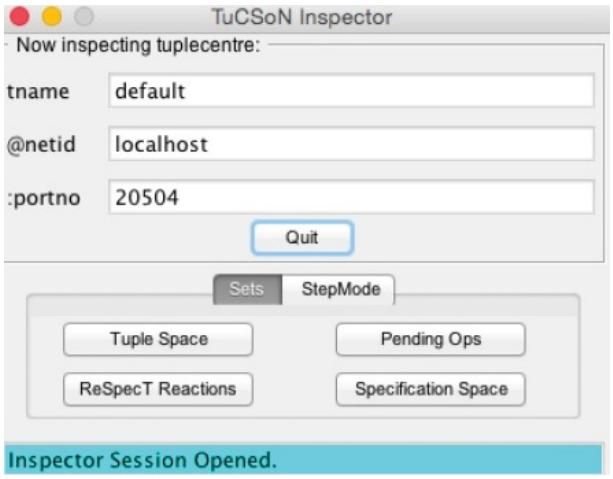


Figura 2.3: TuCSoN Inspector

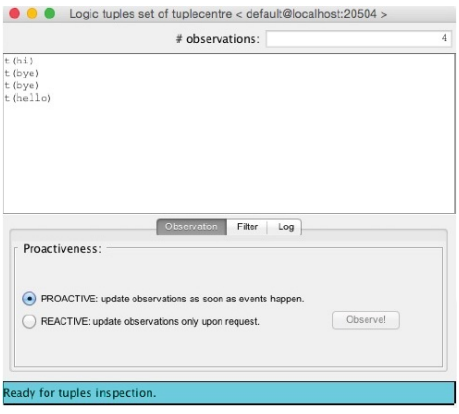


Figura 2.4: Spazio di tuple del centro di tuple

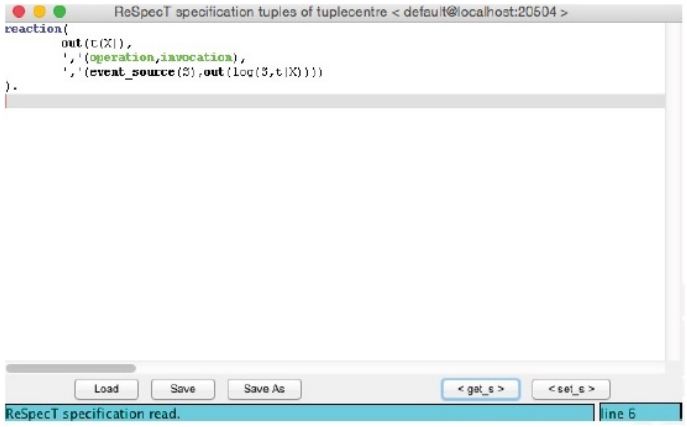


Figura 2.5: Spazio di specifica del centro di tuple

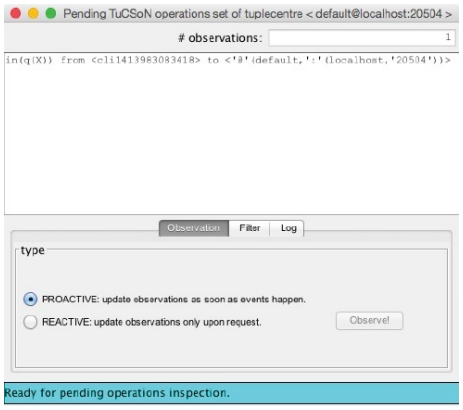


Figura 2.6: Spazio delle operazioni in attesa di completamento

prendendo le caratteristiche di base e salvare le sue specifiche.

Questo spazio rappresenta il fulcro chiave della nostra tesi perché al suo interno come vedremo nel capitolo successivo verrano specificate le reaction in formato ReSpecT dei design patterns.

-*Pending Ops*: In questa finestra vengono esplicitate tutte le operazioni che sono in sospeso e quindi sono in attesa di essere completate ,specificando chi ha richiesto l'operazione e da quale centro di tupla.

Si può osservare, tramite un timestamp, quale centro di tupla, specificando il tempo, che ha invocato l'operazione.

-*Reaction* ReSpecT: rappresenta la finestra di tutte le reaction ReSpecT triggherate e si può osservare se un operazione è andata a buon fine o meno.

Anche in questo spazio si può osservare tramite un timestamp quale

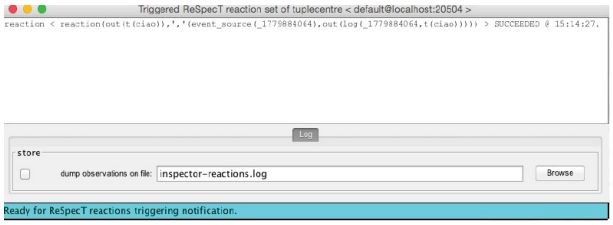


Figura 2.7: Spazio delle reazioni Respect eseguite

centro di tupla, specificando il tempo, che ha invocato l'operazione.

2.3 ReSpecT

ReSpecT è un linguaggio di programmazione basato sulla logica. Linguaggi basati sulla logica hanno già dimostrato di essere efficaci per consentire la comunicazione tra agenti in sistemi multi-agente . Inoltre , i centri di tuple hanno messo in luce che i linguaggi basati sulla logica possono essere efficacemente sfruttati per governare la comunicazione tra gli agenti in modo da costruire comportamenti sociali. In questo lavoro , abbiamo formalmente definito la nozione di centro di tuple logiche, nonché un centro di tupla che rispetta la semantica operazionale della lingua ReSpecT, basato sulla logica per la specifica di comportamento dei centri di tupla . A questo scopo, sfruttiamo un quadro formale per sistemi asincroni, permettendo supporti di coordizione di essere rappresentati in modo separato e indipendente rispetto ai soggetti coordinati. In questo modo si dimostra che un approccio basato sulla logica può essere efficacemente sfruttato per la coordinazione delle operazioni eterogenee degli agenti di diversi tipi e tecnologie.

2.3.1 Centri di tupla ReSpecT

Nel paragrafo precedente abbiamo discusso di cosa sono i centri di tupla, qual è il loro scopo e del modello di coordinazione che li sfrutta. Qui invece andiamo a esplicitare come i centri di tupla vengono implementati nel linguaggio di programmazione ReSpecT e quali sono le caratteristiche di base. Centri di tupla ReSpecT sono centri di tupla logici dove la comunicazione e le reazioni specifiche di linguaggio sono basate sulla logica. Il linguaggio ReSpecT consente di esplicitare un linguaggio di specifica di comportamento. Esso consente di:

• definire delle computazioni all'interno di un centro tuple, chiamate reazioni;

• associare delle reazioni agli eventi di comunicazione che si verificano in un centro di tuple.

Quindi ReSpecT contiene, sia una parte dichiarativa, che una parte procedurale . Una specifica di linguaggio consente a eventi di comunicazione di essere associati in maniera dichiarativa a reazioni per mezzo di tuple logiche specifiche, chiamate tuple specifiche, la cui forma consiste nella reaction che contiene la tripla (E , G, R) . In breve questa semantica è definita in questo modo: se si verifica un evento E e la guarda G permette di eseguire l'evento allora applico la reaction R. Come linguaggio di reazione , ReSpecT consente reazioni procedurali definiti in termini di sequenze di obiettivi di reazione logica. Una reaction nel suo complesso va a buon fine se tutti i suoi obiettivi di reazione hanno successo, altrimenti e quindi fallisce. Ogni reaction quindi può accedere e modificare lo stato di un centro di tupla mentre ogni evento di comunicazione può triggherare una molteplicità di operazioni eseguite localmente (porta di default 20504). Quindi i centri di tupla logici ReSpecT sono in gradi di:

-se lo spazio di specifica di comportamento è vuoto allora non ho reazioni da triggherare indipendentemente dalla comunicazione degli eventi;

-dal punto di vista dell'agente, il risultato delle invocazioni delle primitive, è la somma degli effetti delle primitive e di tutte le reazioni triggherate.

Queste specifiche ci consentono di superare i limiti che troviamo negli spazi di tupla Linda. Tutte le reazioni innescate da un evento di comunicazione vengono eseguite prima di servire qualsiasi altro evento: così, ogni agente percepisce il risultato di una reazione ed esegue tutte le reazioni associate complessivamente come singola transizione dello stato di centro di tupla . Parlando in generale, poiché ReSpecT ha dimostrato di essere un linguaggio che possiede le capacità computazionali equivalenti a quelle di una macchina di Turing, ogni legge di coordinazione che sia computabile può essere in linea di principio contenuta in un centro di tupla ReSpecT. Come già accennato un centro di tupla ReSpecT è strutturato in due parti:

-*spazio di tupla*: dove vengono inserite tutte le query primitive relative alla comunicazione (simile allo spazio di coordinazione Linda),
-*spazio di specifica* :dove vengono esplicitate le reaction spiegate poco fa.

2.3.2 TuCSoN-ReSpecT nella prospettiva A&A

Una volta definito qual' è il modello di coordinazione utilizzato per la programmazione di centri di tupla e il relativo linguaggio di programmazione,andiamo a discutere quale sarà il modello di interazione adottato per la coordinazione nei centri di tupla di sistemi MAS. Il meta-modello utilizzato per l'interazione è quello A&A dove i sogetti principali sono gli agenti e gli artefatti [1]. Gli agenti sono le entità attive di controllo nel sistema con il compito di costruire una risposta ai vari comportamenti nei sistemi MAS. Gli artefatti, invece, sono delle entità passive che forniscono servizi e fanno da mediatori tra gli agenti per la comunicazione, permettendo agli agenti stessi di lavorare insieme favorendo la collaborazione generale all'interno del sistema. Infatti, gli agenti comunicano tra di loro usando gli artefatti che, a loro volta, si collegano tra loro. Ricollegandosi quindi a TuCSoN - ReSpecT usufrendo, TuCSoN fornisce agli agenti una molteplicità di artefatti distribuiti(che rappresentano i centri di tuple), che contengono sia la conoscenza condivisa che la logica del coordinamento espressa in termini di tuple logiche . Adottando la prospettiva A&A, si propone una visione più articolata nello spazio di interazione MAS . Le uniche fonti reali di eventi in un sistema MAS sono gli agenti e l'ambiente. Tuttavia, gli artefatti,collegandosi tra di loro,sono reattivi l'un con l'altro influenzandosi a vicenda . Come prima conseguenza, la causa diretta di un evento da parte di un artefatto, può essere anche un eventuale richiesta di collegamento di un altro artefatto. Come meta-modello per il calcolo distribuito, la prospettiva A & A promuove il disaccoppiamento di controllo degli agenti e degli artefatti, in questo modo:

(i) gli artefatti legati dovrebbero essere pienamente disaccoppiati.

(ii) gli agenti dovrebbero essere lasciati.

Significa quindi di essere liberi di scegliere autonomamente le primitive (che siano sincrone o asincrone), mentre il comportamento di artefatti rimane invariato. Come risultato, ogni operazione (o link) su un artefatto, deve avere una struttura richiesta / risposta: qualsiasi invocazione (richiesta), una volta servita, implica sempre un messaggio di completamento. In caso di un collegamento invocato da un altro artefatto, il completamento deve essere trattato in modo del tutto asincrono. Nel caso in cui, un'operazione vene invocata da un agente, il completamento dovrebbe essere trattato sia in maniera sincrona che in modo asincrona, a seconda dell'agente scelta autonoma.

2.3.3 Comportamenti dei centri di tupla ReSpecT

Lo stato di uno spazio logico di tupla può essere espresso come una coppia <T, W >, dove T rappresenta l'insieme di tuple logiche nello spazio di tupla , e W è l'insieme di query in sospeso in attesa di essere serviti. Per quanto riguarda lo spazio di tuple, lo stato di un centro di tupla contiene anche le reazioni innescate nella forma di coppia (E, R), quindi associando a ogni evento una reazione è necessario associare a sua volta una specifica di comportamento σ. Quindi lo stato di un centro di tupla ReSpecT può essere espresso come una tripla <T , W , Z > σ dove:

-T rappresenta l'insieme di tutte le tuple logiche nello spazio di tuple logiche ordinario,

-W è l'insieme delle richieste pendenti, rappresentando le reaction triggerate dagli agenti in attesa di essere servite o ascoltate,

-Z infine rappresenta l'insieme delle reaction triggherate in attesa di essere eseguite,

infine σ rappresenta il comportamento di specifica ReSpecT che determina l'evoluzione (e quindi il cambiamento) di stato del centro di tupla.

Il comportamento di un centro operativo di tuple ora, può essere modellato in termini di un sistema di transizione con tre tipi di transizioni ammissibili :

• listening ($\rightarrow$ l) , prendendo gli eventi di comunicazione triggherati di agente come ingressi,

• speaking($\rightarrow$ s) , tornando risposte agli agenti come uscite,

• reacting($\rightarrow$ r) , la manipolazione dell'esecuzione della reaction.

Nella prospettiva A& A di un centro di tupla ReSpecT invece, lo stato del centro di tupla viene espresso come una quadrupla etichetta InqQ <Tu, $\sum$, Re, Op >OutQ,dove Tu e $\sum$ rappresentano rispettivamente gli insiemi della tuple specifiche e ordinarie nel centro di tupla. Re è l'insieme delle reazioni innescate in attesa di essere eseguite,Op invece è l'insieme delle richieste in attesa di una risposta. InQ e OutQ sono rispettivamente le code di eventi in entrata e in uscita. InQ è una coda che è automaticamente estesa quando gli eventi in entrata interessano un centro di tuple e le transizioni speciali sono necessarie per gli eventi in arrivo. Dualmente, OutQ viene svuotata automaticamente emettendo gli eventi in uscita, senza biso-

gno di nuove transizioni speciali. Il comportamento operativo di un centro di tupla ReSpecT nella prospettiva A & A , il cui stato è InQ<Tu, $\sum$, Re, Op >OutQ , può essere modellato in termini di un sistema di transizione con quattro tipi di diverse sotto-transizioni, in ordine di priorità decrescente:

-reaction: quando Re = $\varnothing$,vengono innescate reazioni in Re e vengono eseguiti attraverso una reazione di transizione ($\rightarrow$ r).

-time: quando Re = $\varnothing$ e temporizzato (n, $\sum$) = $\varnothing$, reazioni a tempo possono scatenare nuove reazioni attraverso una transizione di tempo ($\rightarrow$ t).

-service: quando il servizio Re = timed (n,$\sum$) =$\varnothing$, e sat (Op, Tu, $\sum$) = $\varnothing$, le richieste in attesa di una risposta possono essere servite attraverso una transizione di servizi ($\rightarrow$ s).

-log: quando Re = timed (n, $\sum$) = sat (Op, Tu, $\sum$) = $\varnothing$, e INQ =$\varnothing$, cioè richieste in coda in INQ, possono essere registrate da un centro di tuple attraverso una transizione log ($\rightarrow$ l).

Capitolo 3

Implementazione dei Design Pattern in TuCSoN-ReSpecT

In questo terzo e ultimo capitolo di tesi viene analizzata più a fondo la parte lavorativa . Infatti se nel primo e secondo capitolo si sono spiegate le caratteristiche dei design patterns(che cosa sono,come vengono usati e le diverse categorie partendo dai pattern di base fino ad arrivare a quelli ad alto livello tramite un catalogo) e le principali caratteristiche dell'infrastruttura TuCSoN e del linguaggio che utilizza (ReSpecT) per andare ad agire sui centri di tupla. Questa parte mette insieme le caratteristiche principali di entrambi i capitoli permettendo quindi di andare a vedere a lato pratico come vengono implementati e similati comportamenti auto-organizzanti, tramite design pattern, sfruttando il linguaggio ReSpecT e appoggiandosi sull'infrastruttura TuCSoN citee.

3.1 Spazio di specifica → Design Patterns atomici in ReSpecT

Come abbiamo spiegato nel capitolo precedente,lo specification space o spazio di specifica è la parte del centro di tupla dove vengono gestiti i comportamenti delle specifiche di coordinazione ,in modo da poter associare a ogni evento una reazione da eseguire. L'obiettivo principale di questa tesi è di implementare (e quindi simulare e testare),nella maniera più simile,le carattestiche dei comportamenti atomici di un sistema MAS auto-organizzante tramite pattern. Come già accennato nel primo capitolo, i pattern di base sono *spreading,aggregation,evaporation e repulsion.*Alcune parti di codice sono state tagliate per problemi di immagine, il codice sorgente completo è possibile trovarlo nel repository di TuCSoN

3.1.1 Spreading Pattern

Lo spreading,come ben spiegato nel capitolo 1, permette di diffondere l'informazione in maniera distribuita a tutti i nodi della rete,copiando l'informazione e distribuendola a tutti i nodi vicini. Se poi l'informazione ricevuta è uguale a quella di partenza il processo di diffusione termina. Il processo di spreading è stato implementato per qualsiasi combinazione di topologia di rete anche per combinazioni complesse(non è detto che tutti i nodi sono collegati tra loro). Come esempio si considera una topologia composta da tre porte (20504,20505 e 20506),prendendo come default il centro di tupla con la porta 20505,supponendo che ad esempio la porta 20505 sia collegata sia con la 20504 che con la 20506, però la 20504 non sia collegabile con la 20506.

La sequenza transazionale delle cose da fare per attivare il processo di spreading è il seguente:

1)E' necessario ,come prima cosa,attivare i centri di tupla lanciando i tre nodi TuCSoN (*TucsonNodeService*) specificando le tre porte diverse.

2)Una volta creati ad esempio tre nodi TuCSoN,è necessario creare una connessione tra i centri di tupla definendo una connessione bilaterale in questo modo:

- :20504 ? out(neighbour(default@localhost: 20505))

- :20505 ? out(neighbour(default@localhost: 20504))
- :20505 ? out(neighbour(default@loclahost: 20506))

- :20506 ? out(neighbour(default@localhost: 20505))

3)Una volta definita la topologia di rete, è necessario andare a settare la spazio di specifica di ogni centro di tupla con le reactions che permettono di attivare il processo di spreading riportate sotto.

4)A questo punto siamo pronti, abbiamo tutto il necessario per attivare lo spreading: l'agente (in questo caso l'utente) manda l'informazione di spreading al nodo di default (out(spread(t(hello))) dove hello è la tupla inserita nello spazio di tuple. In questo modo la tupla non viene inserita solo nel nodo 20505 ma viene anche diramata ai suoi vicini (nodi 20504 e 20506), che a loro volta manderanno altre informazioni. Il processo quindi termina quando l'informazione di partenza del mittente coincide con l'informazione ultima del ricevente. Riferimenti in Figura 3.1 e Figura 3.2.

```
% Reify agent request.
reaction(

  out(spread(INFO)),
  (operation, invocation),
  (

    event_source(SRC),
    event_time(TIME),
    event_target(TARGET),
    out(spread(INFO, SRC, TIME, TARGET)),
    current_time(NOW),
    CHECK is NOW + 10000,
    out_s(

      time(CHECK),
      (internal),
      (
        in_all(spread(INFO, SRC, TIME, _), _)
      )
    )
  )
).
% Delete garbage tuple.
reaction(
  out(spread(INFO)),
  (operation, completion),
  (
    in(spread(INFO))
  )
).
% Serve agent request.
reaction(
  out(spread(INFO, SRC, TIME, TARGET)),
  (internal),
  (
    out(INFO),
    rd_all(neighbour(_), NBRS),
    multicast(NBRS, spread(INFO, SRC, TIME, TARGET))
  )

).
```

Figura 3.1: Spreading

```
% Forward (new requests only).
reaction(
  out(spread(INFO, SRC, TIME, TARGET)),
  (link_in, invocation),
  (
    no(spread(INFO, SRC, TIME, TARGET)),
    out(INFO),
    event_source(N),
    rd_all(neighbour(_), NBRS),
    delete(N, NBRS, N_NBRS),
    multicast(N_NBRS, spread(INFO, SRC, TIME, TARGET)),
    current_time(NOW),
    CHECK is NOW + 10000,
    out_s(
      time(CHECK),
      (internal),
      (
        in_all(spread(INFO, SRC, TIME, _), _)
      )
    )
  )

).
% Multicast to neighbours.
multicast([], _).

multicast([neighbour(N)|NBRS], ITEM):- N ? out(ITEM), multicast(NBRS, ITEM)
).
```

Figura 3.2: Spreading

3.1.2 Aggregation Pattern

Il pattern di aggregazione ,come gia spiegato, è un pattern che ha il compito di ridurre l'ammontare di informazioni nel sistema in modo da far aumentare la conoscenza globale sull'ambiente applicando localmente degli operatori di aggregazione definiti tramite specifiche. Quelli implementati in questa sotto-sezione sono:

-*filter:* permette di mantenere nel centro di tupla solo l'informazione più recente o più vecchia (a seconda da quella scelta dall'utente: 'neweset' o 'oldest').

-*merge:* date due o più tuple con la stessa informazione è possibile inserire all'interno di una sola tupla tutte le tuple con le stesse caratteristiche.

Non è necessaria la connessione e lo scambio di informazioni con altri centri di tupla. La procedura per attivare l'aggregation è molto semplice:

1) Attivare il nodo TuCSoN (*TucsonNodeService*).

2)Settare lo spazio di specifica con le reactions di aggregation.

3)L'agente fa partire il processo di aggregazione scegliendo un operatore da applicare.
Riferimenti in Figura 3.3.

```
%  Timestamp new items from agents.
reaction(
  out(aggregate(filter(F), INFO)),
  (operation, invocation),
  (
    (F = 'newest' ; F = 'oldest'),
    event_time(TIME),
    out(aggregate(filter(F), INFO, TIME))
  )
).
% Handle 'merge' filter
reaction(
  out(aggregate(filter(merge(N)), INFO)),
  (operation, invocation),
  (
    no(merged([INFO|T])) ->
      (
        out(merged([INFO]))
      )
    ;
      (
        rd(merged(L)),
        length(L, M),
        M < N ->
          (
            in(merged(L)),
            out(merged([INFO|L]))
          )
        ;
          true
      )
  )
).
```

Figura 3.3: Aggregation

3.1.3 Evaporation Pattern

Anche in questo caso la procedura è molto semplice:

1) Attivare un nodo TuCSoN (*TucsonNodeService*).

2) Settare lo spazio di specifica con le relative reactions di evaporation.

3) L'agente inserisce per un notevole numero di volte le stesse tuple nello spazio di tupla (facendo ad esempio out(evaporate(hello)) e out(evaporate(world))).

4)L'agente definisce il tempo di delay da applicare al pattern: esempio con 3 secondi out(delay(3000).

4) A questo punto l'agente fa partire il processo di evaporazione tramite la primitiva out(start_evaporation) consentendo di andare ad aggiornare il sistema ogni tre secondi eliminando la tupla più vecchia inserita per ordine di tempo. Una reaction fa partire l'evaporazione mentre l'altra esegue il processo. Riferimenti in Figura 3.4

```
reaction(
  out('$evaporation'),
  internal,
  (
    in('$evaporation'),
    current_time(NOW),
    rd(delay(D)),
    CHECK is NOW + D,
      out_s(
        time(CHECK),
        internal,
        (
          urd(evaporate(T)) ->
          (
            in(evaporate(T)),

            out('$evaporation')
          )
          ;
          (
            out('$evaporation')
          )
        )
      )
  )
).
```

Figura 3.4: Evaporation

3.1.4 Repulsion Pattern

Il pattern di repulsione permette di passare da una zona di alta concentrazione a una zona di più bassa concentrazione. Quindi permette di ridistribuire l'informazione in rete in maniera omogenea in modo da avere lo stesso numero di informazioni negli stessi nodi. La procedura di applicazione è molto simile a quella dello spreading però a differenza dello spreading l'informazione non viene replicata ma viene semplicemente trasmessa. Quindi consideriamo come esempio la topologia di rete utilizzata precedentemente, è necessario quindi:

1)Attivare i tre nodi specificando le porte (assumendo come prima per de-

fault la porta 20505).

2)Settare gli spazi di specifica dei tre centri di tupla con le reactions relative dello repulsion patterns.

3)Stabilire la topologia di rete facendo in modo che la porta 20505 abbia come vicini sia la 20504 che la 20506, mentre loro due non sono comunicanti:

- :20504 ? out(neighbour(default@localhost: 20505))

- :20505 ? out(neighbour(default@localhost: 20504))
- :20505 ? out(neighbour(default@loclahost: 20506))

- :20506 ? out(neighbour(default@localhost: 20505))

4)Inizio il processo di repulsione tramite la primitiva out(repulse(t(hello))) nel centro di tupla di default. A questo punto eseguendo la primitiva più volte l'informazione viene distribuita in maniera omogenea consentendo di passare a zone con lo stesso equilibrio di densità di informazione. Riferimenti in Figura 3.5

```
% Whoever is the requestor (agent/tc), start repulsion.

% If global density will balance, move item, else do nothing.
reaction(
  rd_all(repulse(INFO), THERE),
  (link_out, success),
  (
    length(THERE, LT),
    rd_all(repulse(INFO), HERE),
    length(HERE, LH),
    LT2 is LT + 1,
    LH2 is LH - 1,
    LT2 =:= LH2,
    event_target(DEST),
    in(repulse(INFO)),
    DEST ? out(repulse(INFO))
  )
).

% "Sense" information "density" in the neighbourhood.
multiread([], _).
multiread([neighbour(N)|NBRS], ITEM):- N ? rd_all(ITEM, _),
multiread(NBRS, ITEM).  multiread(NBRS, ITEM).
```

Figura 3.5: Repulsion

Conclusione

Il lavoro appena presentato fornisce un servizio ottimale che descrive comportamenti di sistemi auto-organizzanti atomici ispirati alla biologia. Grazie alle specifiche di comportamento implementate in ReSpecT, appoggiandosi all'infrastruttura TuCSoN, è stato possibile testare e simulare i comportamenti atomici, e quindi di base, espressi nel catalogo dei modelli. Una volta definiti tali modelli (spreading, aggregation, evaporation e repulsion) , è possibile:

1) allargare i confini di questo lavoro creando specifiche ci comportamento che si trovano a livelli più alti:

-*pattern composti:* aggregando tra loro specifiche di comportamento di più pattern atomici. Includendo quindi, in un solo pattern, caratteristiche comuni di base (gradient,pheromone digital e gossip),

-*pattern ad alto livello:* aggregando, a loro volta, pattern atomici con pattern composti. Includendo quindi ,in un solo pattern, caratteristiche comuni di pattern di diverso tipo (morphogenesis, chemiotaxis, quorum sensing, ant foraging e floacking.

2) creare delle librerie in modo da fornire un servizio completo che può essere di grande prospettiva per l'ingegneria dell'auto-organizzazione.

Ringraziamenti

Volevo ringraziare tutte le persone che mi sono state vicine in questi anni e che mi hanno dato la forza per riuscire a raggiungere questo primo grande traguardo della vita,aiutandomi e incoraggiandomi nei momenti di difficoltà. Un ringraziamento speciale va quindi alla mia famiglia (ai miei genitori Leonardo e Cristina che non mi hanno fatto mancare niente e a mio fratello Giacomo che è sempre presente e disponibile),ai miei amici di studio che mi hanno aiutato e hanno collaborato assieme a me nel raggiungimento di questo obiettivo,ai miei amici di compagnia che sono stati sempre al mio fianco durante questi anni. Volevo anche ringraziare il mio prof. relatore Andrea Omicini che è stato disponibile durante lo svolgimento del lavoro e soprattutto il mio co-relatore Ing.Mariani che mi ha seguito e aiutato durante tutto lo svolgimento della tesi. Un ringraziamento finale va all'università che mi ha dato i requisiti necessari per raggiungere questo primo grande traguardo della vita, preparandomi al meglio per studi e lavori futuri.

Bibliografia

[1] E. D. Andrea Omicini. Formal respect in the a&a perspective. Electronic notes in theoretical computer science 175 (2007) 97-117.

[2] E. D. Andrea Omicini. From tuple spaces to tuple centres. Science of computer Programming 41 (2001) 277-294, june 16, 2000.

[3] S. M. Andrea Omicini. The tucson coordination model & tecnology: a guide. june 15, 2015.

[4] N. Antonucci. Complexlab.it, 2005. `http//:www.complexlab.it/Members/lucacomello/articoli/complessita-e-organizzazione-ovvero-verso-le-auto-organizzazioni`.

[5] S. M. Jose Luis Fernandez-Marquez, Giovanna Di Marzo Serugendo. Description and composition of bio-inspired design patterns:a complete overview. Springer Science+Business Media, 1 May 2012.

[6] A. O. Luca Gardelli, Mirko Viroli. Design patterns for self-organizing multiagent systems. 2007.