Alma Mater Studiorum · Università di Bologna

SECONDA FACOLTÀ DI INGEGNERIA
Corso di Laurea Magistrale in Ingegneria Informatica

ALGORITMI GRADIENT-BASED PER LA MODELLAZIONE E SIMULAZIONE DI SISTEMI AUTO-ORGANIZZANTI

Tesi di Laurea in Linguaggi e Modelli Computazionali L-M

Relatore:
Chiar.mo Prof.
Mirko Viroli

Presentata da:
Francesca Cioffi

Correlatore:
Ing. Danilo Pianini

Correlatore:
Dott. Ing. Sara Montagna

Prima Sessione di Laurea
Anno Accademico 2011/2012

A Massimiliano ed al nostro futuro insieme

Indice

Introduzione

I moderni sistemi computazionali continuano ad aumentare di scala. Si sta assistendo, infatti, alla crescita dei sistemi computazionali distribuiti, si pensi ai *social network*, alle reti di sensori *wireless* oppure ai sistemi *peer-to-peer*. Tali sistemi vengono definiti pervasivi (*pervasive system*) poiché comprendono un vasto numero di dispositivi (*device*) eterogenei (e, spesso, anche di proprietari diversi) connessi fra loro.

La gestione di tali sistemi computazionali solleva diverse questioni interessanti. In primo luogo, per assicurare in un sistema le proprietà *runtime* necessarie come la scalabilità, la tolleranza ai guasti, buone prestazioni e la reattività, emerge che una gestione auto-adattativa risulta essere fortemente desiderabile. Gli approcci centralizzati, infatti, per una gestione autonoma diventano velocemente impraticabili a causa delle caratteristiche di: scalabilità, complessità, forte dinamismo e dimensione non stabilita, richieste da tali sistemi. La letteratura scientifica per far fronte a tali requisiti propone i principi dell'auto-organizzazione (*self-organization*).

L'auto-organizzazione si propone come approccio risolutivo per l'ingegnerizzazione di sistemi software complessi ed è proprio tale visione il filo conduttore della presente tesi. Si ha che un sistema per definirsi auto-organizzante deve presentare un'evoluzione in una struttura spaziale/temporale che avviene senza l'intervento centrale in quanto governato solo con regole locali agli elementi, i quali interagiscono facendo emergere dinamicamente il comportamento globale.

In natura sono molteplici i sistemi con simili proprietà perciò è possibile identificare un catalogo di meccanismi ispirati alla biologia, che in chiave ingegneristica assumono la forma di design pattern, per riprodurre il funzionamento di tali sistemi.

Gli ecosistemi di servizi pervasivi (*pervasive service ecosystems*) presentano forti relazioni con gli ecosistemi naturali ed in particolare condividono con essi la necessità intrinseca di auto-organizzazione. Un ecosistema di servizi pervasivi è un sistema computazionale in cui il nostro ambiente è immerso quotidianamente, esso è costituito da attori e componenti di vario genere, che possono essere definiti con il termine di individui. L'ecosistema di servizi pervasivo è un paradigma volto ad affrontare un approccio con caratteristiche di adattività senza un controllo supervisionato e di *situation-awereness* ovvero essere consapevoli degli individui e dei processi presenti nel sistema e scegliere, di conseguenza, il comportamento da adottare. L'architettura di un ecosistema pervasivo è, infatti, composto da un insieme di individui spazialmente situate che interagiscono secondo una serie ben definita di leggi naturali applicate dall'ambiente spaziale in cui sono situate. Il comportamento degli ecosistemi è, infatti, regolato da un insieme di leggi che se progettate adeguatamente producono dinamiche auto-organizzanti. In questo modo, l'adattività (*adaptivity*) diventa una caratteristica intrinseca derivante della presenza di modelli di interazione auto-organizzanti e la struttura di tali sistemi può essere rimodellata in maniera flessibile e robusta in risposta alle perturbazioni ambientali.

La presente tesi si pone in tale contesto di ricerca ed, in particolare, è volta ad

espandere il lavoro eseguito da due tesi precedenti [12] e [16], con l'obiettivo di proporre algoritmi auto-organizzanti significativi ed utili nel contesto degli ecosistemi pervasivi. A tale scopo si sono sfruttate le funzionalità della piattaforma ALCHEMIST per modellare e simulare algoritmi.

All'interno dell'insieme di tutti gli scenari possibili negli ecosistemi di servizi pervasivi, si è scelto come caso di studio il coordinamento dei movimenti, in un ambiente bidimensionale, di un gruppo di individui in modo robusto e flessibile. L'obiettivo del coordinamento è di offrire un servizio di ricongiungimento nel punto dell'ambiente che risulta essere il baricentro rispetto alle posizioni iniziali degli individui attraverso *pattern* di coordinazione spaziali. In particolare, è stato utilizzato il gradiente computazionale ovvero una struttura di dati spaziali i cui valori indicano la distanza di tutti gli altri nodi del sistema rispetto ad un nodo detto "sorgente".

L'algoritmo di calcolo del punto baricentrico considera il fatto che l'ambiente in cui gli individui sono situati possa presentare degli ostacoli; nel caso specifico in cui il punto baricentrico non risulta essere accessibile (ad esempio per la presenza di un muro o posto in una zona molto affollata), verrà scelto un solo punto prossimo ad esso così da ottenere sempre il *join* degli individui in una sola area. Nel dettaglio, tale scopo è stato raggiunto creando un gradiente computazionale nel punto dell'ambiente eletto come baricentro e gli individui si ricongiungono effettuando una risalita di tale gradiente.

La letteratura scientifica dispone di diverse forme di gradiente che differiscono per qualità e performance. Per scegliere la forma di gradiente più adatta allo scenario del baricentro è stata effettuata un'approfondita analisi. Tale analisi ha previsto lo studio di due modalità di creazione del gradiente computazionale presenti nella letteratura scientifica: l'approccio `SAPERE` [18] e l'approccio `NBR` [3]. Dall'analisi è emerso che il gradiente base dell'approccio `SAPERE` presenta numerosi casi di *glitch* [1], tale peculiarità permette una diffusione rapida del gradiente ma al contempo lo rende instabile. A fronte di tali informazioni, è stato introdotto il gradiente `NBR` , il cui modello matematico alla base di esso è stato formalizzato da Jacob Beal, ricercatore della BBN Technologies ed affiliato della MIT CSAIL, e Jonathan Bachrach, ricercatore della MIT Artificial Intelligence Laboratory. L'analisi su tale gradiente ha prodotto che il gradiente `NBR` risulta essere stabile ma molto lento a diffondersi, poiché richiede il *firing* [2] di numerose reazioni.

Per la sua caratteristica di stabilità, il gradiente `NBR` è stato inserito nel modello del baricentro. Si è effettuata una analisi di tale modello applicando diversi *setting* sperimentali, il cui risultato è stato la conferma di robustezza dell'algoritmo in vari scenari possibili.

[1] valori che si discostano dal livello corretto per un tempo infinitesimale

[2] esecuzione della reazione

Struttura della tesi

Si descrive ora la struttura della tesi: si noti che nel capitolo 3 il contributo è stato nel test e nel collaudo del linguaggio, mentre i contributi originali sono principalmente concentrati nei capitoli 4 e 5.

Capitolo 1 - Design pattern per i sistemi auto-organizzanti Si presenta le definizioni di auto-organizzazione e di sistema auto-organizzante. Poi si discute il ruolo dei pattern di progettazione per lo sviluppo di sistemi auto-organizzanti.

Capitolo 2 - Ecosistemi di servizi pervasivi Si descrive il concetto di sistema pervasivo e si presenta un framework degli ecosistemi pervasivi che trae ispirazione dal modello chimico. Si discute il ruolo dell'auto-organizzazione nei sistemi pervasivi e del concetto di spatial computing. In particolare, si sottolinea il ruolo fondamentale dei campi computazionali negli ecosistemi pervasivi.

Capitolo 3 - Un framework di simulazione per gli ecosistemi di servizi pervasivi Prima di discutere i casi di studio, si effettua una panoramica del framework e del linguaggio impiegato, precisamente Alchemist ed il DSL per `SAPERE`.

Capitolo 4 - Gradiente `NBR` Si effettua un'analisi sull'approccio di creazione del gradiente computazionale ideato all'interno del progetto `SAPERE`. Alla luce dei risultati si introduce un nuovo approccio di creazione del gradiente computazionale ideato da Jacob Beal e Jonathan Bachrach, il quale risolve i problemi riscontrati nel primo approccio richiedendo, però, al sistema un impegno maggiore. Si noti che durante l'analisi è stato prodotto un efficiente sistema di reportistica.

Capitolo 5 - Baricentro e ricongiungimento Si modella e si simula un nuovo algoritmo basato sul gradiente `NBR` per il ricongiungimento di un gruppo di persone nel punto baricentrico rispetto alle loro posizioni iniziali. Si presentano diversi scenari in cui si mostrano le varie utilità del modello e se ne discutono i risultati.

1 Design pattern per sistemi auto-organizzanti

In questo capitolo si introducono i concetti e le definizioni fondamentali che saranno utilizzate nella tesi. Prima di tutto si fornisce una descrizione di auto-organizzazione e si discute sul ruolo dei sistemi auto-organizzanti. In particolare, si presenta la visione dell'auto-organizzazione come soluzione dello sviluppo di sistemi, che, però, necessita di essere impostata secondo l'approccio ingegneristico. Tale metodologia di costruzione del software porta alla discussione del ruolo dei design-pattern nei sistemi auto-organizzanti (come riportato in [6]).

1.1 Definizione di auto-organizzazione

L'auto-organizzazione (*self-organisation*) costituisce un approccio efficiente per affrontare la complessità dei sistemi moderni, lo testimonia il crescente interesse da parte della comunità scientifica. Un approccio auto-organizzante permette lo sviluppo di sistemi che esibiscono dinamiche complesse e che si adattano alle perturbazioni ambientali senza richiedere una conoscenza completa delle condizioni circostanti. Un sistema sviluppato seguendo i principi dell'auto-organizzazione produce pattern e dinamiche globali attraverso l'interazione locale dei suoi componenti. Molti sistemi biologici possono essere modellati efficacemente utilizzando un approccio auto-organizzante. Esempi noti includono la ricerca di cibo e la costruzione di nidi nelle colonie di formiche, i pattern geometrici negli alveari di api e comportamento aggregante nei branchi di pesci. I meccanismi auto-organizzanti osservati in natura hanno ispirato lo sviluppo di molti sistemi artificiali.

1.2 Sistemi auto-organizzanti

Le tecnologie emergenti, al giorno d'oggi , forniscono nuovi dispositivi di comunicazione che non vengono adeguatamente sfruttati dalle infrastrutture complesse a causa dei loro

requisiti, fra i quali potrebbero esserci, ad esempio, la scalabilità, la necessità di risposte real-time o la tolleranza ai guasti. Per affrontare queste caratteristiche un nuovo approccio per i software è quello di inserire nel sistema entità autonome e pro-attive e di incrementare le interazioni fra esse. L'azione di incrementare l'interazione e decentralizzare le responsabilità sulle entità, viene chiamata auto-organizzazione. I risultati sono la creazione di sistemi con una migliore scalabilità e robustezza e la riduzione della capacità computazionale di ogni entità. I meccanismi auto-organizzanti, di solito, implicano:

- la decentralizzazione, non esiste un'entità centrale che coordina la ri-organizzazione delle altre entità del sistema;
- la località, le entità individuali hanno informazioni sul loro **vicinato**, ad esempio: la lista dei nodi adiacenti, informazioni su o fornite da questi nodi), ma non l'informazione globale, poiché risulta troppo costoso mantenerla aggiornata.

Lo sviluppo di sistemi auto-organizzanti è guidato da principi diversi rispetto all'ingegneria tradizionale. Tipicamente, gli ingegneri progettano sistemi come composizione di elementi più semplici, sia nel caso di astrazioni software che di dispositivi fisici, dove le regole di composizione dipendono dal paradigma di riferimento, ma che comunque producono risultati predicibili. Al contrario, i sistemi auto-organizzanti mostrano dinamiche non-lineari e che sebbene robusti nei confronti di perturbazioni esterne, sono piuttosto sensibili ai cambiamenti di parametri interni. In particolare, l'ingegneria di sistemi auto-organizzanti pone due grandi sfide:

- come dobbiamo progettare le singole entità in modo tale da produrre il comportamento globale desiderato?
- come possiamo fornire garanzie sull'emergenza di pattern specifici?

Per iniziare a rispondere a queste domande, nella prossima sessione, si introduce un modello computazionale per lo sviluppo di sistemi auto-organizzanti.

1.3 Dal modello biologico al computazionale

La computazione nei sistemi auto-organizzanti a livello delle entità individuali (micro-livello) implica l'esecuzione di relativamente semplici regole, o comandi, rispetto ai risultati complessivi che queste computazioni raggiungono quando considerate su una macro-scala. Le caratteristiche chiave di questi meccanismi sono la robustezza e l'adattamento ai cambiamenti delle condizioni dell'ambiente. Tipicamente i meccanismi auto-organizzanti sono quelli che usano la stigmergia, si pensi al comportamento coordinato delle formiche alla ricerca di cibo, al raggruppamento per movimenti coordinati ed ai sistemi basati sui gradienti. I meccanismi auto-organizzanti sono solitamente ispirati dalla natura, in particolare dai sistemi biologici.

Nei sistemi biologici si osservano due entità principali:

- gli **organismi** che collaborano nel processo biologico;
- l'**ambiente**, uno spazio fisico dove gli organismi sono situati.

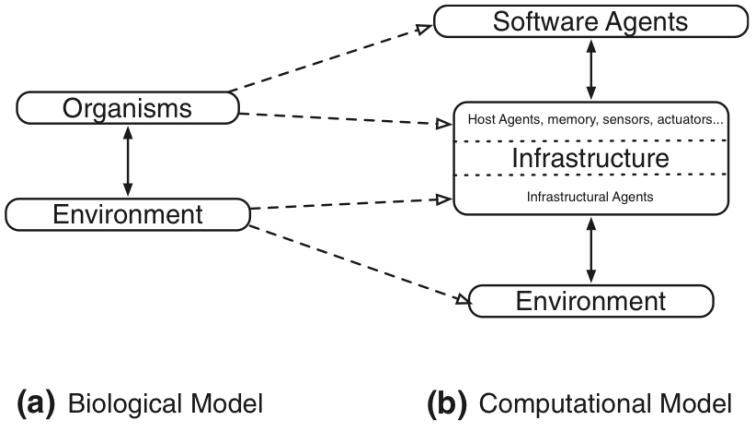


Figura 1.1: *Entità principali nei modelli biologico e computazionale. Immagine tratta da [6]*

L'ambiente fornisce le **risorse**, che gli organismi possono usare, e gli **eventi**, che possono essere osservati dagli organismi e che possono cambiare all'interno del sistema. Gli organismi possono comunicare fra loro, percepire l'ambiente ed agire su di esso. Gli organismi, dunque, sono autonomi, proattivi e hanno una conoscenza parziale del mondo. L'ambiente è dinamico ed agisce sulle risorse e sugli organismi. La comunicazione fra gli organismi può essere diretta o indiretta. La comunicazione indiretta utilizza l'ambiente per depositare l'informazione in modo che gli altri organismi la possano percepire. Il modello biologico, quindi, può essere suddiviso in due livelli: organismi ed ambiente, come mostrato in figura 1.1(a).

Nel modello computazionale ispirato al modello biologico viene però inserito un nuovo livello, il livello dell'**infrastruttura**, che risulta essere necessario dato che un agente software deve essere inserito in un dispositivo dotato di potenza computazionale, la quale fornisce agli agenti la capacità di interagire con l'ambiente e comunicare con gli altri agenti, come mostrato in figura 1.1(b). Le entità proposte nel modello computazionale sono:

- gli **agenti**, entità software autonome e proattive;
- l'**infrastruttura**, che contiene degli **host** dotati di potenza computazionale, ovvero sensori ed attuatori;

- l'**ambiente**, lo spazio del mondo reale dove l'infrastruttura è situata.

Gli **eventi** sono fenomeni di interesse che si presentano nell'ambiente, essi possono essere percepiti dagli agenti utilizzando i dispositivi degli host. Ogni agente necessita di un host per: essere eseguito, comunicare con gli altri agenti, percepire gli eventi ed agire nell'ambiente. L'infrastruttura, difatti, fornisce degli agenti dotati degli strumenti necessari per simulare il comportamento degli organismi ed un luogo dove l'informazione può essere memorizzata ed eventualmente letta dagli altri agenti. Nella maggior parte dei sistemi biologici l'ambiente assume un ruolo chiave, dovuto alla propria capacità di agire sulle entità presenti nel sistema. Per affrontare questa capacità dell'ambiente, ogni host nell'infrastruttura ha un software embedded, chiamato **agente infrastrutturale** (IA). Si noti che sia i comportamenti degli IA che degli agenti devono essere progettati seguendo dei pattern di auto-organizzazione. Gli IA giocano un ruolo fondamentale quando gli agenti possono muoversi liberamente fra gli host. Per esempio, gli IA possono essere responsabili di gestire l'informazione depositata negli host dagli agenti oppure della diffusione (*spreading*) dell'informazione sugli altri host. In altri casi, l'IA rappresenta un software embedded, all'interno di un middleware, in grado di fornire funzionalità built-in (ad esempio l'evaporazione del feromone digitale).

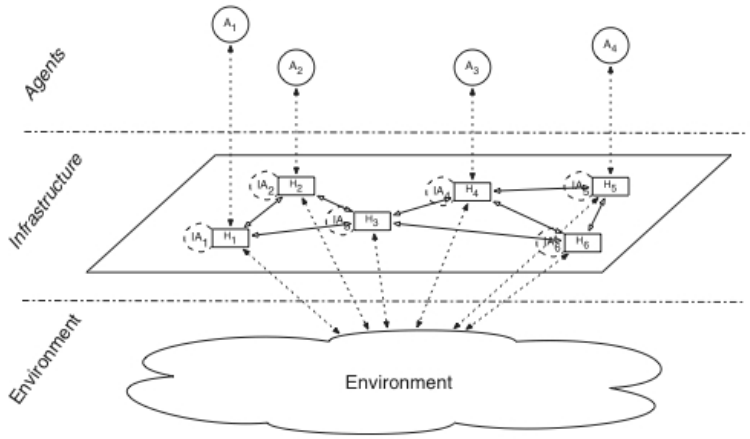


Figura 1.2: *Modello computazionale. Immagine tratta da [6]*

La figura 1.2 mostra i diversi livelli del modello computazionale e le interazioni corrispondenti. Il top level rappresenta gli agenti software del sistema. Gli agenti usano il livello dell'infrastruttura per ospitare loro stessi, comunicare con gli altri, percepire ed agire sull'ambiente e per depositare l'informazione che può essere letta dagli altri agenti. In realtà, esistono due varianti del modello: quando gli agenti possono muoversi liberamente fra gli host oppure quando essi vengono accoppiati con gli host. La separazione fra il livello degli agenti e dell'infrastruttura comporta la possibilità di un'ampia gamma

di scenari. In sostanza, l'infrastruttura può essere fissa o mobile e gli host mobili possono essere controllati da agenti (ad esempio da un robot) o no (ad esempio i movimenti dei PDA sono sotto il controllo dei loro proprietari), in questo ultimo caso si forma quella che viene solitamente indicata come un'infrastruttura opportunistica, ovvero dove i nodi si muovono in accordo alla direzione scelta dagli utenti che li trasportano e gli agenti possono liberamente spostarsi da un nodo ad un altro. Un sistema, dunque, è composto: dagli agenti, l'infrastruttura, gli agenti infrastrutturali, gli host e l'ambiente. Il comportamento degli agenti e degli agenti infrastrutturali è definito da un insieme di regole (da qui in poi chiamate regole di transizione), mentre gli host sono definiti dall'interfaccia che forniscono.

1.4 Perché introdurre i design pattern

Negli ultimi dieci anni, i meccanismi auto-organizzanti basati sulla biologia (*bio-inspired*) sono stati applicati in diversi domini raggiungendo risultati migliori rispetto agli approcci tradizionali. I ricercatori, però, di solito utilizzano questi meccanismi in una configurazione ad-hoc. Per questo motivo, la loro interpretazione, definizione ed implementazione varia nella letteratura esistente, impedendo di fatto che tali meccanismi possano essere applicati in modo chiaro e sistematico per la risoluzione di problemi ricorrenti. Per facilitare la progettazione dei sistemi artificiali *bio-inspired*, si è deciso di definire tali meccanismi in termini di design pattern modulari, riusabili ed organizzati in diversi livelli.

Nelle prossime sezioni verranno introdotti i design-pattern per i sistemi auto-organizzanti basandosi principalmente basato sul paper [6].

1.5 Design pattern come metodologia

Le metodologie correnti per i sistemi auto-organizzanti seguono le tipiche fasi della metodologia dell'ingegneria del software: requisiti, analisi, progettazione, implementazione e collaudo. In generale, le varie metodologie esistenti mettono in luce differenti aspetti, in particolare, ciascuna di esse introduce una specifica fase di progettazione dove i meccanismi d'interazione vengono: identificati, modellati, ridefiniti e, se possibile, simulati. Di conseguenza, i design pattern auto-organizzativi hanno la massima utilità nella fase di progettazione all'interno della metodologia scelta. I design pattern entrano in gioco, quindi, durante la fase di progettazione, la quale viene divisa in tre passi (figura 1.3):

- una prima fase di progettazione dove viene effettuata la scelta dei design pattern. I design pattern auto-organizzativi hanno lo scopo di identificare il problema da risolvere, in modo da determinare la soluzione appropriata per risolvere tale pro-

blema. In sintesi, aiutano a determinare i confini di ogni problema e la soluzione corrispondente fornita dai pattern;

- una fase di perfezionamento, durante la quale le entità attuali e le rispettive dinamiche vengono definite. Le dinamiche dei pattern servono a: raffinare il modello, identificare le entità e le loro precise interazioni, responsabilità individuali ed anticipare il comportamento emergente;
- nella fase di simulazione, infine, si specifica la descrizione dell'implementazione dei pattern utile a stabilire i dettagli in relazione al modello computazionale.

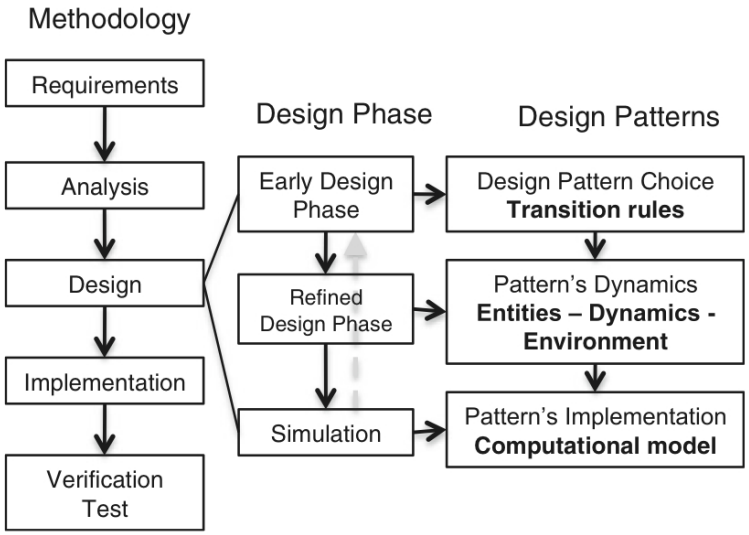


Figura 1.3: *I design pattern all'interno della fase di progettazione. Immagine tratta da [6]*

Questi tre passi possono essere iterati in un ciclo al fine di realizzare un perfezionamento incrementale o una rivisitazione del modello. Un tema importante dei meccanismi nei sistemi auto-organizzanti riguarda il tuning dei parametri. I pattern sono dotati di una descrizione dei principali parametri utilizzati e dei loro effetti sul comportamento risultante. La fase di simulazione è, dunque, cruciale per stabilire i valori di tali parametri. Al fine di determinare il catalogo dei pattern, si analizzano le inter-relazioni fra i meccanismi auto-organizzativi e la progettazione di sistemi auto-organizzanti presenti in letteratura, in modo da capire il funzionamento di tali sistemi e facilitare il loro adattamento o estensione per affrontare nuovi problemi. Il processo di classificazione dei pattern inizia con la selezione dei meccanismi ad alto livello noti in letteratura e l'individuazione dei casi in cui sono stati applicati con successo nei differenti sistemi auto-*.

Poi, attraverso l'analisi dei loro comportamenti, si identificano meccanismi comuni di basso livello, alcuni dei quali di base (atomici) ed altri composti da quelli di base. Il risultato è la classificazione dei pattern in tre livelli. I meccanismi di base possono essere utilizzati singolarmente o insieme ad altri per formare pattern più complessi, essi sono collocati nel livello più basso. Nel livello in mezzo, invece, sono presenti i meccanismi formati dalle combinazioni con quelli a più basso livello, mentre a più alto livello sono contenuti i pattern ad alto livello che mostrano diversi modi di utilizzare i meccanismi di base e composti.

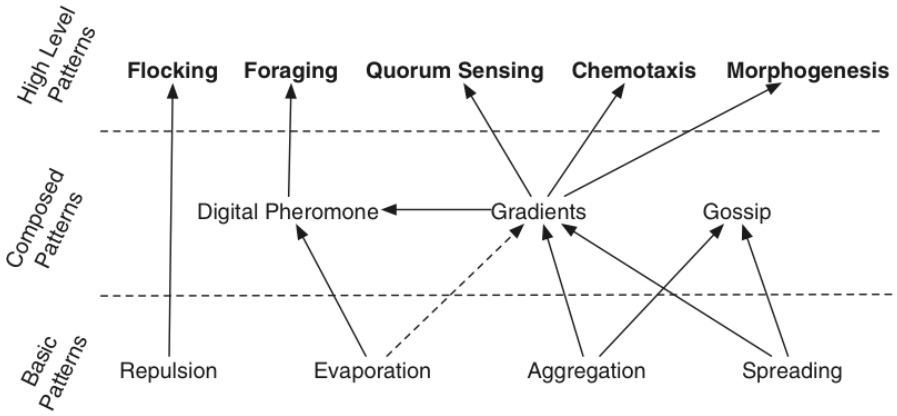


Figura 1.4: *I pattern e le loro relazioni. Immagine tratta da [6]*

La figura 1.4 mostra i diversi design pattern che verranno descritti nelle prossime sezioni. Si noti che i pattern di repulsione, gossip e *flocking* non sono stati utilizzati nella tesi e quindi non verranno descritti, per essi si faccia riferimento al paper [6]. Una freccia tratteggiata indica che la relazione è opzionale (ad esempio il pattern gradiente può usare l'evaporazione, ma l'evaporazione non è necessaria per implementare i gradienti). Lo scopo della classificazione è di elencare i meccanismi esistenti in letteratura, identificare i loro confini, le loro inter-relazioni e i problemi ricorrenti che risolvono. Per ogni pattern, oltre al suo nome ed ad altri appellativi (alias) conosciuti, si indica lo scenario a cui si rivolge il problema e la soluzione che fornisce viene identificata in modo chiaro. Altri campi precisano l'ispirazione biologica dei pattern, l'effetto dei parametri chiave coinvolti nel pattern, le entità coinvolte e le loro dinamiche, e i requisiti dell'ambiente. Il comportamento dei pattern viene descritto mediante le regole di transizione definite dalla seguente semplice notazione. Ogni informazione presente nel sistema viene modellata attraverso una tupla $\langle L, C, \rangle$, dove L indica la posizione in cui l'informazione viene memorizzata, mentre C è il suo contenuto corrente. Le regole di transizione sono simili a reazioni chimiche che lavorano sulla base delle tuple dei pattern, le quali sono

del tipo:

$$name :: \langle L_1, C_1\rangle, \ldots, \langle L_n, C_n\rangle \xrightarrow{r} \langle L'_1, C'_1\rangle, \ldots, \langle L'_m, C'_m\rangle$$

dove:

- il lato sinistro (i reagenti) specifica le tuple coinvolte nella regola di transizione, queste verranno rimosse come effetto dell'esecuzione della regola;
- il lato destro (i prodotti) specifica quali tuple verranno inserite come conseguenza nelle posizioni specificate, potrebbero essere nuove tuple, trasformazione di uno o più reagenti oppure i reagenti inalterati;
- il rate r è un tasso, il quale indica la velocità/frequenza con cui le regole dovrebbero scattare, ovvero la politica di scheduling.

Le regole possono essere fornite attraverso un insieme di regole di transizione che determinano le variabili del lato destro come funzioni di quelle sul lato sinistro. Tali funzioni possono essere soggetto di condizioni e di vincoli, che possono essere specificati insieme alla reazione. Si noti che le funzioni potrebbero avere i parametri:

1. fissati del sistema modellato;
2. estratti automaticamente dai reagenti;
3. specificati nella regola di transizione.

Il modello delle regole di transizione intenzionalmente si astrae da questi aspetti. Come notazione di convenienza si userà $\{x, y, z, ...\}$ per gli insiemi e $(x; y; z; ...)$ per le sequenze ordinate.

1.6 Pattern di base

I pattern di base sono pattern atomici e vengono utililzzati per comporre pattern più complessi posizionati nel middle e nel top layer. Tali pattern, introdotti in questa sezione, descrivono i meccanismi base frequentemente presenti in letteratura.

1.6.1 Spreading pattern

Lo spreading pattern è basato sulla comunicazione diretta fra agenti per l'invio progressivo di informazione sul sistema. La diffusione di informazione nei sistemi multi-agente permette agli agenti di incrementare la loro conoscenza globale del sistema.

Alias: lo spreading è anche conosciuto come diffusione dell'informazione, disseminazione di dati o informazioni, *flooding*, broadcast o diffusione epidemica.

Problema: nei sistemi, dove gli agenti compiono solo interazioni locali, il ragionamento degli agenti soffre di mancanza di conoscenza sul sistema globale.

Soluzione: una copia dell'informazione viene inviata ai vicini e propagata sulla rete da un nodo all'altro. L'informazione si diffonde progressivamente sul sistema riducendo la mancanza di conoscenza degli agenti e, al tempo stesso, mantenendo i vincoli dell'interazione locale.

Ispirazione: lo spreading è un pattern base esteso ed utilizzato dalla maggior parte degli altri pattern presentati in questo capitolo. Lo spreading appare nei processi importanti, così come la morfogenesi, la chemiotassi o il *quorum sensing*. In natura, lo spreading è un processo svolto dall'ambiente.

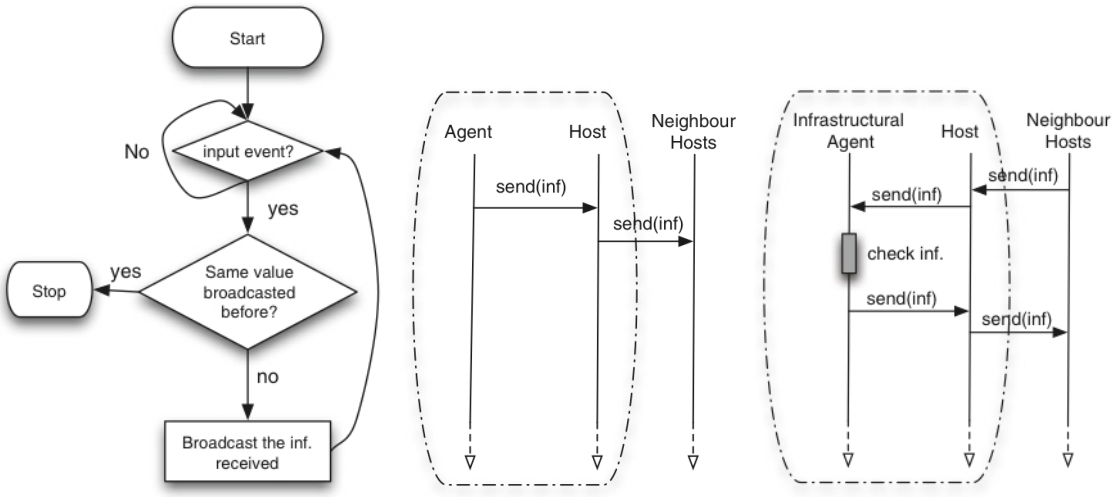
Vincoli: se lo *spreding* si verifica con alta frequenza, l'informazione si diffonde sulla rete velocemente ma il numero di messaggi cresce. Una diffusione veloce è desiderata quando l'ambiente cambia continuamente e gli agenti devono conoscere i nuovi valori ed adattarsi ad essi. Può accadere che l'informazione sia interessante solamente per gli agenti vicino alla sorgente. In quel caso, l'informazione viene diffusa su un determinato numero di passi, riducendo il numero di messaggi. Un altro modo di ridurre il numero di messaggi è determinare il numero di nodi del vicinato che riceve l'informazione. E' stato dimostrato che non è necessario inviare l'informazione a tutti i nodi del vicinato per assicurarsi che ogni nodo abbia ricevuto l'informazione.

Entità-dinamica-ambiente: le entità coinvolte nel processo di spreading sono host, agenti ed agenti infrastrutturali. Il processo di spreading viene avviato da un agente che prima diffonde l'informazione nell'host in cui risiede. Quando questa informazione arriva ai nodi del vicinato, l'agente infrastrutturale è in grado di ri-inviare l'informazione ai nodi vicini, producendo la diffusione dell'informazione sull'intero sistema. Ogni agente infrastrutturale inoltra l'informazione ricevuta ad uno specifico numero di vicini e fino ad un numero specificato di passi. La dinamica è di solito estesa per evitare cicli infiniti e lo spreco di consegne duplicate. La regola di transizione 1.1 descrive più formalmente lo spreading pattern.

$$\begin{aligned} spreading ::&\langle L, C\rangle \xrightarrow{r_{spr}} \langle L_1, C_1\rangle, ..., \langle L_n, C_n\rangle \\ &where\,(L_1; ...; L_n) = v\,(L)\,, (C_1; ...; C_n) = \sigma\,(C, L) \end{aligned} \tag{1.1}$$

Una funzione $v\,(L)$ è data per determinare la sequenza di posizioni, fra i vicini di L, nei quali l'informazione in input deve essere diffusa. L'insieme di tali posizioni non può essere vuoto, non può essere composto solo da L, ma può essere composto da tutto il vicinato di L, incluso L stesso. La funzione $\sigma\,(C, L)$ è data per elaborare il contenuto della nuova informazione, che può cambiare all'interno del processo di spreading.

Implementazione: l'algoritmo più comune utilizzato per diffondere l'informazione ai vicini è l'algoritmo di broadcast. E' noto che il broadcast causa il problema della tempesta di trasmissioni (*broadcast storm*). Il problema della tempesta di trasmissioni si presenta quando il raggio del segnale di più nodi si sovrappone. Una trasmissione diretta



(a) Comportamento agente infrastrutturale (b) Inizializzazione (c) Interazioni

Figura 1.5: *Spreading: comportamento agenti infrastrutturali 1.5(a), inizializzazione corrispondente 1.5(b) ed interazioni con i propri host e gli host del vicinato 1.5(c). Immagine tratta da [6]*

in base al *flooding*, infatti, provocherà problemi di ridondanza, contese e collisioni. Per risolvere il problema della tempesta di trasmissioni può essere implementato un broadcast ottimizzato, il quale può seguire uno schema probabilistico basato sul conteggio, sulla distanza, sulla posizione o sui cluster. Questa sezione presenta una implementazione di base per illustrare come lavora lo spreading e come è stato implementato in letteratura. La figura 1.5(a) mostra il diagramma di flusso dell'informazioni diffusa dopo la ricezione. La figura 1.5(b) mostra il diagramma di interazione dell'inizializzazione dello *spreading*. La figura 1.5(c) mostra il diagramma di interazione fra l'informazione che arriva al vicinato.

Usi conosciuti: il meccanismo di spreading è stato utilizzato in diverse applicazioni: *swarm motion coordination*, coordinazione nei giochi e problemi di ottimizzazione. Altri riferimenti di applicazioni possono essere trovati in pattern di alto livello che coinvolgono lo *spreading pattern*.

Conseguenze: lo *spreading* viene utilizzato in pattern di alto livello come il pattern del gradiente, della morfogenesi e della chemiotassi.

1.6.2 Aggregation pattern

L'*aggregation pattern* è un pattern di base usato per l'aggregazione dell'informazione. La diffusione dell'informazione nei sistemi in larga scala, sia essa depositata da parte

dagli agenti o generata dall'ambiente, può produrre un sovraccarico della rete e della memoria. L'*aggregation pattern*, difatti, è stato introdotto al fine di ridurre l'ammontare d'informazione presente, il suo compito è di sintetizzare l'informazione significativa.

Alias: l'aggregazione è anche conosciuta come fusione.

Problema: nei sistemi in larga scala, l'eccesso di informazione prodotta dagli agenti può determinare sovraccarichi nella rete e nella memoria. L'informazione deve essere processata in modo distribuito per ridurre l'ammontare di informazioni ed ottenere l'informazione significativa.

Soluzione: l'aggregazione consiste nell'applicazione locale dell'operatore di fusione per processare l'informazione e sintetizzare la macro informazione. L'operatore di fusione può assumere diverse forme, come il filtraggio, il merging, l'aggregazione o la trasformazione.

Ispirazione: in natura, l'aggregazione (somma) del feromone delle formiche permette alla colonia di trovare il percorso più breve per raggiungere il cibo, scartando, di conseguenza, quelli più lunghi. In natura l'aggregazione è un processo effettuato dall'ambiente. L'ambiente continua ad eseguire il processo di aggregazione anche nel caso in cui non vi siano agenti presenti nel sistema.

Vincoli: l'aggregazione può essere applicata a tutta l'informazione disponibile localmente o solo ad una parte di essa. La quantità di informazione che viene diffusa nel sistema è un parametro importante per determinare l'impiego della memoria del sistema stesso.

Entità-dinamiche-ambiente: l'aggregazione viene eseguita dagli agenti o dagli agenti infrastrutturali. In entrambi i casi gli agenti aggregano l'informazione a cui accedono localmente. L'informazione può provenire dall'ambiente o da altri agenti. Nel caso in cui l'informazione risulti proveniente dall'ambiente è, di solito, letta mediante i sensori. In accordo con il modello presentato nella sezione 1.3, l'aggregazione viene eseguita da un agente che riceve l'informazione dall'host in cui risiede. Tale host è un sensore che legge l'informazione indipendentemente dall'ambiente oppure un dispositivo di comunicazione che riceve l'informazione dagli host del vicinato. L'aggregazione può essere applicata da ogni agente che riceve l'informazione indipendentemente dall'infrastruttura sottostante. Il processo di aggregazione non risulta essere ciclico e termina quando un agente esegue la funzione di aggregazione. La regola di transizione per l'aggregazione 1.2 trasforma l'informazione in input (possibilmente un insieme di informazione) in un nuovo insieme di informazioni con cardinalità minore rispetto all'insieme in input attraverso la funzione di aggregazione α.

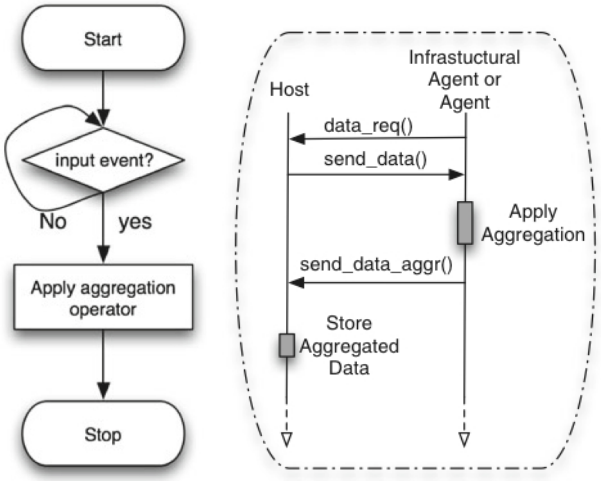
$$\begin{aligned} aggregation ::&\langle L, C_1\rangle, ..., \langle L, C_n\rangle \xrightarrow{r_{aggr}} \langle L, C'_1\rangle, ..., \langle L, C'_m\rangle \\ &where\ \{C'_1, ..., C'_m\} = \alpha\left(\{C_1, ..., C_n\}\right) \end{aligned} \tag{1.2}$$

Implementazione: l'informazione disponibile assume la forma di uno stream di eventi. L'aggregazione dell'informazione può assumere varie forme da un operatore sem-

plice come nell'*Ant Colony Optimization* (ACO) ad operatori più complessi. Gli operatori di aggregazione sono classificati in quattro differenti gruppi:

- filtro: questo operatore seleziona un sotto-insieme degli eventi ricevuti;
- trasformatore: questo operatore cambia il tipo dell'informazione ricevuta in input;
- fusione (merger): questo operatore unifica tutta l'informazione ricevuta e gli output di tutta l'informazione ricevuta come una singola parte dell'informazione;
- aggregatore: questo operatore applica una operazione specifica ad una o più informazioni in ingresso; i tipi in input ed in output possono essere tutti diversi.

Il diagramma di flusso 1.6(a) mostra che il processo di aggregazione inizia quando un agente riceve un'informazione (un evento). Dopo, esso applica l'operatore di fusione ed invia le informazioni aggregate in risposta all'host. La figura 1.6(b) mostra come un agente o un agente infrastrutturale usa l'interfaccia fornita dall'host per ottenere i dati, ovvero applica un operatore di fusione e deposita i dati storici aggregati nell'host.



(a) Flusso (b) Interazione

Figura 1.6: *Aggregation: comportamento di un agente. Immagine tratta da [6]*

Usi conosciuti: l'aggregazione è stata utilizzata nell'algoritmo ACO per aggregare feromone, emulare un'alta concentrazione quando due o più feromoni sono vicino ad un altro. L'aggregazione è, inoltre, utilizzata nel feromone digitale per la coordinazione autonoma di *swarming* UVAs e nel campo della fusione dell'informazione, il quale studia come aggregare le basi delle credenze individuali in una collettiva o per il *truth-tracking* nei MAS.

Conseguenze: l'aggregazione aumenta l'efficenza nelle reti riducendo il numero di messaggi. L'aggregazione, quindi, fornisce un meccanismo per estrarre macro-informazione nei sistemi in larga scala, in particolare come ricavare informazione significativa dai dati ottenuti dai sensori. La quantità di memoria usata dal sistema, di conseguenza, si riduce.

Pattern relativi: l'*aggregation pattern* può essere implementato insieme all'*evaporation* e al *gradient* pattern per formare i feromoni digitali. L'evaporazione può essere utilizzata con l'aggregazione in modo da aggregare l'informazione collezionata più recentemente dall'ambiente. Il gossip pattern è un pattern composto dall'*aggregation pattern*, questo pattern non è stato utilizzato all'interno della tesi e perciò non verrà descritto (per una sua descrzione si faccia riferimento al paper [6]) e lo *spreading pattern*.

1.6.3 Evaporation pattern

L'*evaporation pattern* è un pattern che permette di affrontare gli ambienti dinamici, ovvero dove l'informazione utilizzata dagli agenti può diventare obsoleta. Negli scenari del mondo reale, l'informazione appare e cambia nel tempo e la sua rivelazione, predizione o rimozione è di solito costosa o addirittura impossibile. Nel momento in cui gli agenti modificano il proprio comportamento tenendo conto dell'informazione proveniente dall'ambiente, l'informazione raccolta di recente deve essere più rilevante rispetto all'informazione raccolta nel passato. L'evaporazione è, dunque, un meccanismo che riduce progressivamente la pertinenza dell'informazione. L'informazione recente, infatti, diventa più rilevante rispetto all'informazione elaborata in precedenza. L'evaporazione è stata proposta come design pattern sia per i sistemi multi-agenti auto-organizzanti sia per l'ACO.

Alias: l'evaporazione è anche conosciuta come decadenza, funzione temporale di degradazione o freschezza.

Problema: l'informazione obsoleta deve essere rimossa altrimenti la sua rilevazione comporta un costo che potrebbe essere evitato. Le decisioni degli agenti si basano sulla freschezza dell'informazione presente nel sistema, questo permette loro di rispondere correttamente agli ambienti dinamici.

Soluzione: l'evaporazione è un meccanismo che periodicamente riduce l'importanza dell'informazione. L'informazione recente, infatti, diventa più rilevante di quella passata.

Ispirazione: l'evaporazione è presente in natura. Per esempio, nelle colonie delle formiche quando esse depositano il feromone nell'ambiente, questo attrae le altre formiche e guida i loro movimenti dal nido al cibo e viceversa. Tale meccanismo permette alle formiche di trovare il percorso più breve al cibo, anche quando avvengono perturbazioni dell'ambiente. Le formiche, dunque, sono sempre in grado di trovare il nuovo percorso più breve eliminando quelli vecchi.

Vincoli: l'evaporazione viene controllata dai parametri del fattore di evaporazione e la frequenza di evaporazione, quest'ultima viene usata per incrementare la rilevanza dell'informazione. Il fattore e la frequenza di evaporazione devono essere tarate sulla

dinamicità dell'ambiente; se l'evaporazione è troppo veloce si può perdere l'informazione, altrimenti se l'evaporazione è troppo lenta l'informazione potrebbe essere sorpassata e guidare in modo scorretto il comportamento dell'agente. Un fattore di evaporazione più alto richiede meno memoria, ma riduce anche l'informazione disponibile nel sistema per gli agenti. Quando l'evaporazione viene applicata per la ricerca collaborativa o l'ottimizzazione di algoritmi, il fattore di evaporazione controlla il bilanciamento fra l'esplorazione e lo sfruttamento: alti rate dell'evaporazione riducono la conoscenza dell'agente sull'ambiente, aumentando l'esplorazione e producendo un veloce adattamento ai cambiamenti dell'ambiente. Si noti che un fattore di evaporazione troppo elevato abbatte le performance nel caso in cui non occorrono così frequenti cambiamenti nell'ambiente.

Entità-Dinamiche-Ambiente: l'evaporazione può essere applicata ad ogni informazione presente nel sistema. Periodicamente la rilevanza dell'informazione diminuisce nel tempo. Il risultato è che l'informazione recente diventa più rilevante rispetto all'informazione elaborata in precedenza.

L'evaporazione viene eseguita dagli agenti o dagli agenti infrastrutturali periodicamente eseguendo la regola di transizione 1.3.

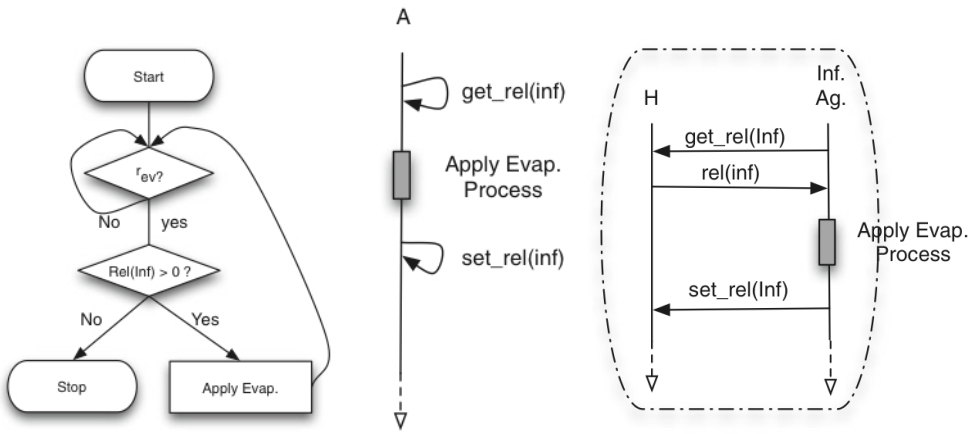
$$\begin{aligned} evaporation ::& \langle L, C \rangle \xrightarrow{r_{ev}} \langle L, C' \rangle \\ & where\ C' = \epsilon\,(C) \end{aligned} \tag{1.3}$$

La regola interessa il valore di rilevanza contenuto in C applicando la funzione ϵ che può, per esempio, imporre che $Rel_{C'} = Rel_C * Ev_{factor}$ con $Ev_{factor} \in [0, ..., 1]$ o che $Rel_{C'} = Rel_C - Ev_{factor}$. Il requisito per $\epsilon\,(C)$ è che il valore della rilevanza decresce con l'applicazione della regola.

Implementazione: l'*evaporation pattern* viene eseguito da un agente con lo scopo di aggiornare la rilevanza della propria informazione interna, o dagli agenti infrastrutturali che cambiano la rilevanza dell'informazione depositata in un ambiente. Si distinguono due approcci. Nel primo approccio un agente incapsula l'informazione e ne diminuisce la sua rilevanza. In questo caso, l'agente segue il diagramma di flusso in figura 1.7(a) e il diagramma delle interazioni 1.7(b). Nel secondo approccio l'informazione viene depositata da un agente in un host ed un agente infrastrutturale interagisce con l'host per il decadimento della rilevanza dell'informazione. L'host fornisce una interfaccia per leggere e modificare il valore della rilevanza. L'interazione fra l'agente infrastrutturale e l'host è mostrata in figura 1.7(c).

Usi conosciuti: l'evaporazione è stata utilizzata principalmente nell'ottimizzazione dinamica. Esempi di algoritmi che usano l'evaporazione sono ACO e il *Quantum Swarm Optimisation Evaporation* (QSOE). In altri lavori l'evaporazione è eseguita utilizzando un parametro chiamato freschezza associato all'informazione.

Conseguenze: l'evaporazione favorisce l'adattamento alle perturbazioni ambientali. L'uso dell'evaporazione negli scenari statici, però, può peggiorare le performance, a causa della perdita di informazione implicita in questo meccanismo. Il pattern di evaporazio-



(a) Flusso di un agente (b) Agente (c) Agente - Host

Figura 1.7: *Evaporation: comportamento di un agente 1.7(a), evaporazione effettuata dall'agente stesso 1.7(b) ed evaporazione effettuata dall'host 1.7(c). Immagine tratta da [6]*

ne fornisce l'abilità di auto-adattamento per i cambiamenti ambientali aumentando, di conseguenza, la tolleranza al rumore.

Pattern relativi: il pattern di evaporazione è usato dai pattern di alto livello come il pattern del feromone digitale e del gradiente.

1.7 Pattern composti

Questa sezione analizza le composizioni di pattern base, ampiamente utilizzati in letteratura. I pattern composti possono essere usati da soli o estesi a loro volta da pattern di più alto livello.

1.7.1 Gradient pattern

Il *gradient pattern* è un'estensione dello spreading pattern dove l'informazione viene propagata in modo tale da fornire una valutazione sulla distanza del mittente. In particolare, un attributo della distanza viene aggiunto all'informazione oppure il valore dell'informazione viene modificato in modo tale da riflettere la sua concentrazione - valori più alti di concentrazione indicano che il mittente è vicino, come per il feromone delle formiche. Inoltre, il *gradient pattern* usa l'*aggregation pattern* per unire i diversi gradienti creati dai diversi agenti o per unire i gradienti provenienti dallo stesso agente ma con differenti percorsi. Tale pattern può essere applicato nel caso in cui solo l'informazione con

la distanza più breve dal mittente debba essere mantenuta oppure per l'informazione sull'aumento della concentrazione dell'informazione.

Alias: il *gradient pattern* è un particolare tipo di campo computazionale.

Problema: gli agenti appartenenti a sistemi grandi soffrono di mancanza d'informazione globale per la valutazione delle conseguenze delle loro azioni o di quelle eseguite dagli agenti al di fuori del raggio di comunicazione.

Soluzione: l'informazione diffusa da una posizione viene inizialmente depositata ed aggregata quando incontra altra informazione. Durante la diffusione, informazione aggiuntiva sulla distanza dal mittente e la direzione viene fornita attraverso: o un valore della distanza o modificando l'informazione che rappresenta la loro concentrazione. Gli agenti che ricevono i gradienti, infatti, hanno un'informazione che proviene al di fuori del loro raggio di comunicazione, aumentando così la propria conoscenza globale del sistema non solo con informazioni sui gradienti ma anche con la direzione e la distanza della sorgente dell'informazione. Durante il processo di aggregazione, un operatore di filtro mantiene solo l'informazione con la distanza più grande (o più piccola), o modifica la concentrazione. I gradienti sono in grado di gestire i cambiamenti della topologia della rete. In particolare, in questo caso, l'informazione viene diffusa periodicamente ed è soggetta ad evaporazione, riducendo la propria rilevanza nel tempo e permettendo al gradiente di adattarsi ai cambiamenti della topologia delle reti. Tali gradienti sono chiamati gradienti attivi.

Ispirazione: i gradienti appaiono in diversi processi biologici. I processi più conosciuti sono: *ant foraging*, *quorum sensing*, morfogenesi e chemiotassi. In questi processi i gradienti supportano le comunicazioni a largo raggio fra entità mediante le interazioni locali.

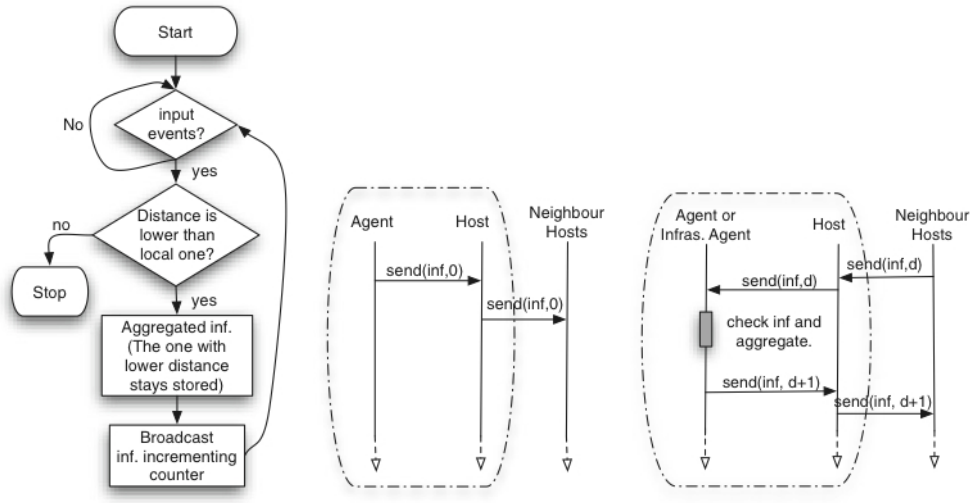
Vincoli: l'adattamento alle perturbazioni ambientali è più veloce quando le frequenze di aggiornamento sono alte, si aumenta così il sovraccarico della rete. Abbassando le frequenze di aggiornamento si riduce il sovraccarico della rete, ma può portare a valori obsoleti quando si verificano i cambiamenti ambientali. Nasce un trade-off fra il raggio di diffusione (numero di passi) e il carico nella rete. Un più alto raggio di diffusione porta l'informazione lontano dalla propria sorgente, ma fornisce l'orientamento anche agli agenti distanti. Esso, però, incrementa notevolmente il carico della e può perfino bloccarla.

Entità-Dinamiche-Ambiente: le entità che agiscono sulla base del *gradient pattern* sono agenti, host ed agenti infrastrutturali. In modo analogo allo spreading pattern, quando un gradiente viene creato, esso viene diffuso ai suoi vicini. Le regole di transizione per il *gradient pattern* sono specifiche istanze della regola di transizione 1.1 e 1.2. Un esempio è dato dalla regola di transizione 1.4. Si assume che ogni tupla contenga un

attributo D che rappresenta la distanza dall'host corrente alla sorgente del gradiente.

$$\begin{aligned} spreading ::& \langle L, [D, C] \rangle \xrightarrow{r_{spr}} \langle L_k, [D + \Delta D, C] \rangle \\ & where\ L_k = random\left(\{L_1, \ldots, L_n\}\right) \\ aggregation ::& \langle L, [D_1, C] \rangle, \ldots, \langle L, [D_n, C] \rangle \xrightarrow{r_{aggr}} \langle L, [D', C] \rangle \\ & where\ D' = min/max\left(\{D_1, \ldots, D_n\}\right) \end{aligned} \tag{1.4}$$

La prima regola di transizione modella l'informazione di diffusione modificando l'attributo distanza incrementando o decrementando il suo valore così da ottenere un gradiente a forma di cono con il vertice verso l'alto (o il basso). La regola, inoltre, definisce un'istanza specifica della funzione $v(L)$ introdotta nella regola di transizione 1.1 per determinare la sequenza di posizioni, fra i vicini di L, ai quali l'informazione in input deve essere diffusa. La funzione $random\left(\{L_1, \cdots, L_n\}\right)$ seleziona in modo casuale una posizione fra tutte le posizioni dei vicini di L. La seconda regola di transizione modella il corrispondente caso di aggregazione quando più tuple con lo stesso contenuto ma attributi di distanza diversi sono presenti in locale. Tale regola modella il caso di un'aggregazione dove solo l'informazione con la distanza più lunga(/breve) viene mantenuta. Risulta importante notare che D potrebbe essere utile anche rappresentare le concentrazioni invece delle distanze.



(a) (b) (c)

Figura 1.8: *Gradienti: comportamento di un agente 1.8(a), inizializzazione 1.8(b), agente ed agente infrastrutturale 1.8(c). Immagine tratta da [6]*

Implementazione: gli agenti iniziano il processo di trasmissione dell'informazione al loro vicinato, come mostrato in figura 1.8(b) nel caso del valore della distanza. Nel momento in cui un agente riceve l'informazione incrementa l'attributo distanza o riduce

il valore della concentrazione in accordo con il valore della concentrazione presente nell'informazione ed inoltra di nuovo il gradiente a tutti i suoi vicini (spreading pattern) come mostrato nel diagramma di flusso in figura 1.8(a) e il diagramma di sequenza in figura 1.8(b) per il caso relativo al valore della distanza. Nel momento in cui un host riceve un gradiente, gli agenti infrastrutturali lo diffondo ulteriormente. Da notare che questo pattern può anche essere eseguito dagli agenti. Quando un agente riceve più di un gradiente, esso esegue l'aggregazione (*aggregation pattern*) come mostrato nel diagramma di sequenza in figura 1.8(c). Per esempio, poi esso può filtrare solo il gradiente con il più basso valore della distanza.

Usi conosciuti: il gradient pattern aggiunge un'informazione extra (la distanza). La distanza può essere usata per limitare il numero di passi durante il processo di spreading.

Pattern relativi: il *gradient pattern* è una composizione dello *spreading* e dell'*aggregation* pattern, esteso con il valore della distanza o informazione sulla concentrazione. Esso è usato nei pattern della morfogenesi, della chemiotassi e del quorum sensing. Il *gradient pattern* può essere combinato con l'*evaporation pattern* per creare gradienti attivi per supportare l'adattamento quando gli agenti modificano le loro posizioni o la topologia della rete cambia.

1.7.2 Digital pheromone pattern

Il *digital pheromone pattern* è un meccanismo utilizzato nella *swarm coordination* basato sulla comunicazione indiretta. In questo pattern gli agenti depositano i feromoni digitali negli host. Un feromone digitale è un contrassegno (*mark*) che diffonde un gradiente sull'ambiente e persiste nell'ambiente per qualche tempo, dissolvendosi nel tempo. In questo modo anche gli agenti al di fuori del range di comunicazione possono riceve l'informazione trasmessa dal feromone digitale. I feromoni digitali sono registrati negli host e rimangono attivi anche quando gli agenti che depositano i feromoni digitali si allontanano. I feromoni digitali possono essere identici fra loro, come nell'algoritmo ACO o possono essere specializzati per uno specifico compito, come nel controllo dello *swarming vehicle*. I feromoni digitali sono un caso particolare di comunicazione stigmergica. Si nota che la stigmergia risulta essere più generale, dato che rappresenta qualsiasi comunicazione indiretta attraverso l'ambiente e, quindi, non necessariamente un mark.

Alias: nessuno a nostra conoscenza.

Problema: la coordinazione di agenti in ambienti in larga scala utilizzano la comunicazione indiretta.

Soluzione: il feromone digitale fornisce un modo per coordinare il comportamento degli agenti utilizzando la comunicazione indiretta in ambienti altamente dinamici. I feromoni digitali creano gradienti che vengono diffusi sull'ambiente e che trasportano informazioni sulla propria distanza e direzione. Gli agenti, infatti, possono percepire feromoni da lontano ed aumentare la loro conoscenza sul sistema. Il feromone digitale

evapora con il passare del tempo, contribuendo così l'adattamento del comportamento alle perturbazioni ambientali.

Ispirazione: il *digital pheromone pattern* prende ispirazione dalle colonie di formiche. Le colonie di formiche, infatti, sono capaci di trovare il percorso più breve dal nido alla sorgente di cibo usando le interazioni locali e la comunicazione indiretta basata sui feromoni. I feromoni sono depositati nell'ambiente dalle formiche per segnare il percorso che stanno seguendo dal nido alla sorgente di cibo e ritorno. I feromoni evaporano velocemente così questi devono essere rilasciati continuamente per mantenere le informazioni sul percorso. Le colonie sono in grado di adattarsi ai cambiamenti ambientali (come ad esempio: nuovi ostacoli, nuove sorgenti di cibo, sorgenti di cibo che diventano vuote, ecc...).

Vincoli: l'implementazione del *digital pheromone pattern* coinvolge l'implementazione del *gradient* e dell'*evaporation* pattern al fine di creare un **gradiente attivo**. La principale differenza fra i gradienti attivi ed i feromoni digitali è che il feromone implica la comunicazione indiretta, mentre un gradiente si diffonde da agente ad agente. I principali parametri da considerare sono:

- come per l'*evaporation pattern*, quanto e con quale frequenza l'evaporazione viene usata in ogni iterazione;

- come per il *gradient pattern*, il *digital pheromone pattern* è composto dall'*aggregation* e dallo *spreading* pattern, così più la diffusione del feromone è frequente, maggiore è la banda necessaria. Diffondere il feromone a grandi distanze permette a più agenti di ricevere l'informazione, ma consuma più memoria e più banda.

Entità-dinamiche-ambiente: gli agenti sono le sole entità che possono depositare feromone. I feromoni vengono depositati negli host, gli agenti infrastrutturali poi applicano i meccanismi di diffusione, aggregazione ed evaporazione. I feromoni sono diffusi attraverso la rete, aggregati negli host nel momento in cui arrivano due o più informazioni sul feromone ed evaporano nel corso del tempo fino a scomparire. Durante il ciclo di vita del feromone, esso può essere percepito anche al di fuori del raggio di comunicazione, quando il feromone è attualmente depositato il suo effetto risulta essere dovuto allo *spreading pattern*.

La regola di transizione per il *digital pheromone pattern* viene ottenuta componendo i tre pattern di base: *spreading*, *aggregation* ed *evaporation*, come mostrato nella regola

di transizione 1.5.

$$\begin{aligned} spreading ::& \langle L, [PhV, C] \rangle \xrightarrow{r_{spr}} \langle L_k, [PhV - \Delta PhV, C] \rangle \\ & where\ L_k = random\left(\{L_1, \ldots, L_n\}\right) \\ aggregation ::& \langle L, [PhV_1, C] \rangle, \ldots, \langle L, [PhV_n, C] \rangle \xrightarrow{r_{aggr}} \langle L, [PhV_i, C] \rangle \\ & where\ PhV_i = max\left(\{PhV_1, \ldots, PhV_n\}\right) \\ evaporation ::& \langle L, [PhV, C] \rangle \xrightarrow{r_{ev}} \langle L, [PhV', C] \rangle \\ & where\ PhV' = PhV * Ev_{factor} \end{aligned} \tag{1.5}$$

In modo simile al *gradient pattern*, la prima regola di transizione modella la diffusione dell'informazione modificando l'attributo di concentrazione PhV, in particolare diminuendo il suo di valore di un intervallo ΔPhV, il quale può rappresentare per esempio la distanza fra due posizioni. La selezione della posizione di destinazione è la stessa come per il *gradient pattern*. La seconda regola di transizione modella il caso corrisponde dell'aggregazione dove solo il feromone con il valore maggiore viene mantenuto. La terza regola di transizione modella l'evaporazione dei feromoni attraverso l'Ev_{factor} compreso nell'intervallo [0..1].

Implementazione: i feromoni digitali sono di solito implementati utilizzando l'evaporazione statica moltiplicativa. Indipendentemente dai pattern usati per implementare il *digital pheromone pattern*, i feromoni possono essere depositati negli host, simulati via software o implementati utilizzando sensori RFID. Nel *digital pheromone pattern*, gli agenti solamente rilasciano e percepiscono i feromoni. Gli agenti insfratturali sono in grado di di diffondere, aggregare e far evaporare i feromoni. Il modo in cui gli agenti sfruttano i feromoni digitali coinvolge i nuovi pattern che sanno spiegati nelle prossime sezioni.

Usi conosciuti: i feromoni digitali sono stati usati principalmente nell'*autonomous coordination of swarming*. Altre applicazioni di feromone digitale possono essere trovate nella descrizione dell'ant foraging pattern.

Conseguenze: l'implementazione dei feromoni digitali per la *swarm coordination* prevede le seguenti caratteristiche del sistema:

- semplicità, rispetto alla logica necessaria in un approccio centralizzato,
- scalabilità, i feromoni digitali lavorano in un modo totalmente decentralizzato,
- robustezza, dovuta alla decentralizzazione ed il continuo processo di auto-organizzazione che il feromone digitale prevede, alcuni agenti potrebbero guastarsi ma il sistema è abbastanza robusto da superare questi guasti.

Pattern relativi: il *digital pheromone pattern* è composto dall'*evaporation* e dal *gradient* pattern, quest'ultimo a sua volta è composto dall'*aggregation* e dallo *spreading*

pattern, così è possibile dire che il *digital pheromone pattern* implica i pattern di base della diffusione e dell'evaporazione. Il *digital pheromone pattern* viene sfruttato dall'*ant foraging pattern* fra i pattern di alto livello.

1.8 Pattern di alto livello

Questa sezione descrive i tre pattern di alto livello utilizzati in letteratura il cui contributo è stato dimostrato in diversi campi.

1.8.1 Ant foraging pattern

L'*ant foraging* è l'attività in cui un insieme di formiche collaborano per trovare il cibo. L'*ant foraging pattern* è un pattern di ricerca collaborativa decentralizzata. Principalmente, l'*ant foraging pattern* è stato applicato per problemi di ottimizzazione e usato nella *swarm robotics*.

Alias: ACO.

Problema: l'*ant foraging pattern* fornisce regole per esplorare l'ambiente in modo decentralizzato e per sfruttare le risorse.

Ispirazione: l'*ant foraging pattern* si ispira al comportamento della ricerca di cibo della colonia di formiche. Nelle colonie di formiche, le formiche coordinano il loro comportamento per trovare il percorso più breve dal nido al cibo attraverso una comunicazione stigmergica. Il feromone guida il comportamento delle altre formiche della colonia. In particolare, le contrazioni di feromone vengono usate per attirare le altre formiche. Seguendo la concentrazione maggiore di feromone, quindi, le formiche trovano il percorso più breve dal nido al cibo ed adattano questo percorso quando degli ostacoli si presentano o quando il cibo viene esaurito.

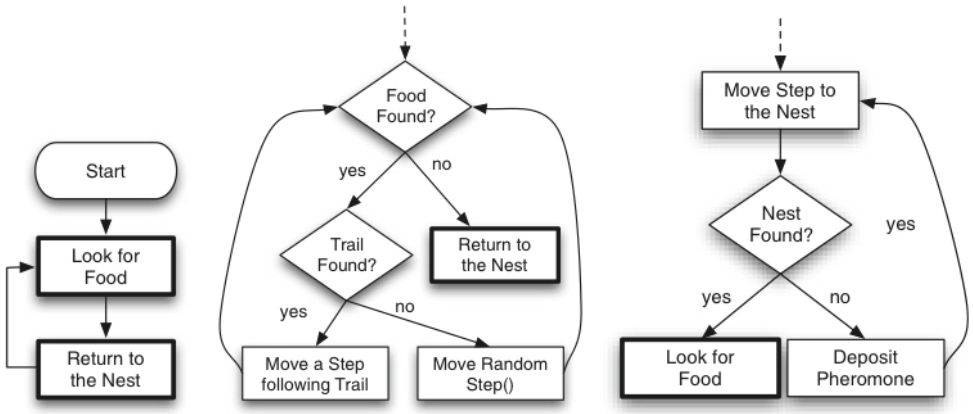
Vincoli: ogni formica ha una certa probabilità di seguire il gradiente prodotto dal feromone. Quando una formica non sta seguendo il gradiente, essa si muove in modo casuale nell'ambiente cercando nuove risorse (esplorazione). Se la probabilità di esplorazione è alta, le formiche si adattano rapidamente ai cambiamenti ambientali ma sono più lente nella ricerca delle risorse. Si noti che con una bassa esplorazione, le formiche sono veloci nello sfruttare le risorse poiché la maggior parte delle formiche segue il percorso diretto alla risorsa. A causa della mancanza di esplorazione, però, le formiche impiegano più tempo a trovare nuove risorse e l'adattamento risulta essere lento. L'*ant foraging pattern* richiede le medesime forze del digital pheromone pattern. Se il tasso di evaporazione del feromone è troppo basso, la traccia del feromone non evapora abbastanza velocemente e rimane dove è stato rilasciato. L'ambiente viene riempito di feromone e la sua utilizzazione non risulta efficiente. Un alto tasso di evaporazione causa l'evaporazione del feromone prima che le formiche riescano a costruire un percorso per mantenerlo, riducendo lo sfruttamento ed incrementando l'esplorazione.

Enità-dinamiche-ambiente: le entità coinvolte nell'*ant foraging pattern* sono le stesse del *digital pheromone pattern*. Nel momento in cui un agente percepisce la presenza di un feromone digitale, può decide di seguire il gradiente oppure di muoversi in modo casuale.

La regola di transizione 1.6 descrive il comportamento delle formiche in ricerca di cibo. Essa estende la regola di transizione 1.5 che crea il campo dei feromoni.

$$\begin{aligned} up\_move ::& \langle L, [PhV_1, C]\rangle, \ldots, \langle L, [PhV_n, C]\rangle \xrightarrow{r_{umove}} \langle L_i, [PhV_i, C]\rangle \\ & where\ PhV_i = max\left(\{PhV_1, \ldots, PhV_n\}\right) \\ random\_move ::& \langle L, C\rangle \xrightarrow{r_{rmove}} \langle L_i, C\rangle \\ & where\ L_i = random\left(\{L_1, \ldots, L_n\}\right) \end{aligned} \tag{1.6}$$

La prima regola modella un agente che percepisce i valori del campo del feromone nella sua posizione e nel vicinato e che decide di seguire la direzione del valore maggiore del gradiente in ricerca del cibo. La seconda regola modella un agente che si muove casualmente. Entrambe queste regole sono soggette a tassi che regolano il bilanciamento fra le attività di sfruttamento ed esplorazione.



(a) (b) (c)

Figura 1.9: *Ant foraging: flusso generale 1.9(a), ricerca del cibo 1.9(b) e ritorno al nido 1.9(c). Immagine tratta da [6]*

Implementazione: in base alla probabilità di esplorazione, gli agenti seguono gli esploratori (detti anche scout ovvero le formiche reclutate per la ricerca del cibo) oppure eseguono una ricerca casuale. Nel caso delle formiche, gli scout depositano i feromoni nel proprio ambiente, i quali saranno poi percepiti dalle altre formiche alla ricerca delle sorgenti di cibo. La figura 1.9(a) mostra il comportamento generale delle formiche, la figura 1.9(b) mostra il comportamento delle formiche che stanno ricercando il cibo, seguendo una traccia o prendendo un percorso a caso, infine la figura 1.9(c) mostra il

ritorno al nido e la diminuzione del feromone una volta che un pezzo di cibo è stato trovato.

Usi conosciuti: l'*ant foraging pattern* è stato utilizzato principalmente nell'ACO.

Pattern relativi: l'*ant foraging pattern* sfrutta il *digital pheromone pattern*, il quale a sua volta usa i pattern per l'evaporazione, la diffusione e l'aggregazione.

1.8.2 Chemotaxis pattern

Il *chemotaxis pattern* fornisce un meccanismo per eseguire un movimento coordinato nei sistemi in larga scala. Il *chemotaxis pattern* estende il *gradient pattern*, gli agenti identificano la direzione del gradiente e decidono la direzione dei loro successivi movimenti.

Alias: nessuno in nostra conoscenza.

Problema: il movimento coordinato decentralizzato è volto a rilevare sorgenti o confini di eventi.

Soluzione: gli agenti percepiscono localmente l'informazione del gradiente e seguono il gradiente in una direzione specifica.

Ispirazione: in biologia, la chemiotassi è il fenomeno in cui gli organismi mono o multi-cellulari dirigono il loro movimenti in accordo con alcune sostanze chimiche presenti nel loro ambiente. Si noti che in biologia, la chemiotassi è anche uno dei meccanismi di base della morfogenesi. Essa guida le cellule durante lo sviluppo in modo che esse siano posizionate nella giusta posizione finale. In questa sezione il termine chemiotassi viene usato per indicare un movimento coordinato sulla base del quale seguire i gradienti mentre il termine morfogenesi viene usato per definire comportamenti specifici basati sulle relative posizioni determinate attraverso i gradienti.

Vincoli: il *chemiotaxis pattern* sfrutta il *gradient pattern*. Nel *chemiotaxis pattern* l'intervallo di comunicazione gioca un ruolo fondamentale. Quando l'intervallo di comunicazione è ampio, gli agenti si muovono più velocemente seguendo i gradienti. Questo, però, causa problemi nell'individuazione precisa delle sorgenti. D'altra parte, un breve intervallo di comunicazione necessita di un elevato numero di passi per seguire il gradiente, ma permette di trovare le sorgenti con una maggiore precisione.

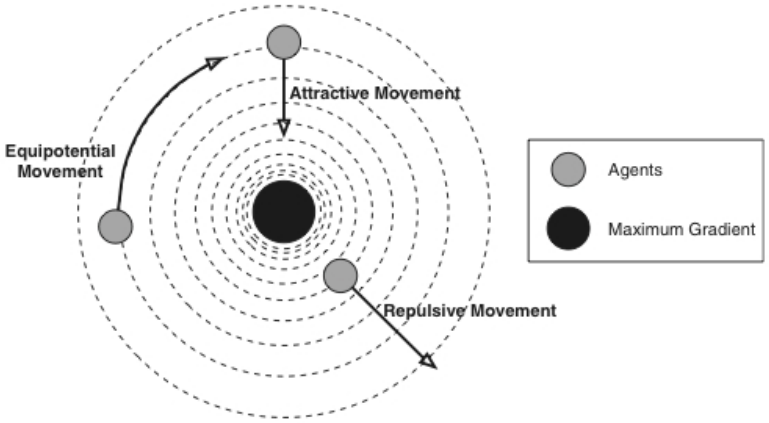


Figura 1.10: *Chemiotaxis pattern. Immagine tratta da [6]*

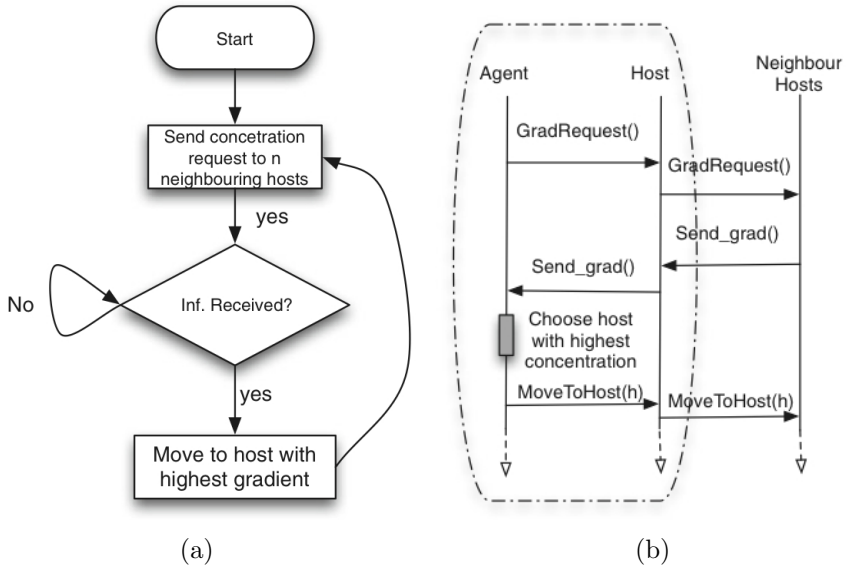
Entità-dinamiche-ambiente: la concentrazione del gradiente guida i movimenti degli agenti in tre diversi modi, come mostrato in figura 1.10:

- movimenti attrattivi, quando gli agenti modificano le loro posizioni seguendo valori maggiori dei gradienti,
- movimenti repulsivi, quando gli agenti seguono valori decrescenti dei gradienti, incrementando, di conseguenza, la distanza fra l'agente e la sorgente del gradiente,
- movimenti equipotenziali, quando gli agenti seguono i gradienti compresi fra delle soglie.

Data la regola di transizione 1.4 che crea il gradiente, la regola di transizione 1.7 determina il movimento dell'agente attraverso il valore del gradiente maggiore, minore o equipotenziale in base al caso considerato.

$$\begin{aligned} move ::&\langle L, [D_1, C]\rangle, \ldots, \langle L_n, [D_n, C]\rangle \xrightarrow{r_{move}} \langle L_i, [D_i, C]\rangle \\ &where\ D_i = min/max/equal\,(\{D_1, \ldots, D_n\}) \end{aligned} \tag{1.7}$$

Implementazione: la chemiotassi può essere implementata in due differenti modi. Il primo utilizzando gradienti esistenti nell'ambiente per coordinare le posizioni o le direzioni dell'agente oppure utilizzando movimenti attrattivi per determinare il contorno degli eventi diffusi dalle sorgenti degli eventi attraverso approccio su un'infrastruttura di una rete di sensori. Il secondo utilizzando i campi del gradiente generati dagli agenti. I diagrammi in figura 1.11 mostrano un caso particolare di implementazione, dove gli agenti prendono una decisione su dove dirigersi in futuro. Come mostrato nel diagramma



(a) (b)

Figura 1.11: *Chemotaxis: comportamento di un agente 1.11(a), interazione di un agente 1.11(b). Immagine tratta da [6]*

1.11(a) ogni agente sceglie casualmente n host nel suo vicinato ed invia loro una richiesta della concentrazione del gradiente. Poi l'agente sceglie gli host del vicinato che hanno il valore maggiore della concentrazione del gradiente e si trasferisce in tale posizione.

Usi conosciuti: la chemiotassi viene usata per coordinare la posizione di semplici robot mobili e per instradare i messaggi negli scenari della computazione pervasiva (*pervasive computing*).

Pattern relativi: il *chemotaxis pattern* estende il *gradient pattern.*

1.8.3 Morphogenesis pattern

L'obiettivo del *morphogenesis pattern* è di selezionare il comportamento dell'agente in base alla sua posizione nel sistema. Il *morphogenesis pattern* sfrutta il *gradient pattern*: l'informazione sulla posizione spaziale relativa viene valutata attraverso una o più sorgenti del gradiente generati dagli altri agenti. Il processo della morfogenesi in biologia è stato considerato come una sorgente di ispirazione per i campi del gradiente.

Alias: nessuno in nostra conoscenza.

Problema: nei sistemi decentralizzati in larga scala, gli agenti prendono le decisioni in base al proprio ruolo oppure pianificano le loro attività sulla base della loro posizione spaziale.

Soluzione: gli agenti specifici diffondono i gradienti della morfogenesi. Gli agenti valutano la propria posizione nel sistema elaborando la loro distanza relativa alle sorgenti

dei gradienti della morfogenesi.

Ispirazione: nel processo della morfogenesi biologica alcune cellule creano e modificano le molecole, le quali diffondono e creano dei gradienti delle molecole. L'organizzazione nello spazio di tali gradienti è il gradiente della morfogenesi, il quale viene usato dalle cellule per differenziare il ruolo che essi hanno dentro il corpo.

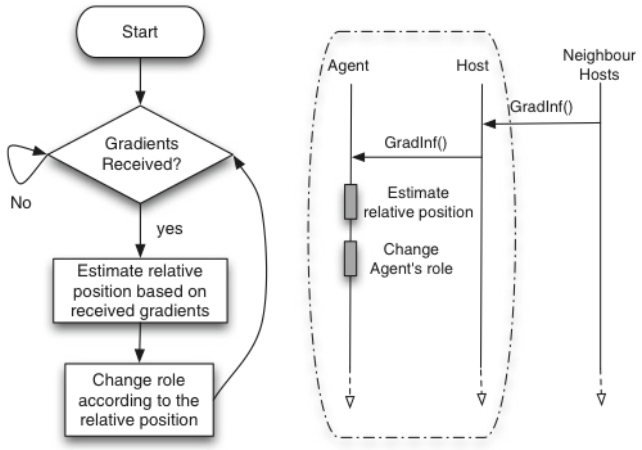
Vincoli: i vincoli presenti in questo pattern sono i medesimi del *gradient pattern*.

Entità-dinamiche-ambiente: le entità coinvolte nel processo della morfogenesi sono gli agenti, gli host e gli agenti infrastrutturali. All'inizio, alcuni agenti diffondono uno o più gradienti della morfogenesi, implementati attraverso il *gradient pattern*. Altri agenti, invece, percepiscono il gradiente della morfogenesi in modo da calcolare le loro posizioni relative. Sulla base delle loro posizioni, gli agenti adottano differenti ruoli e coordinano le loro attività per raggiungere gli obiettivi di collaborazione.

Data la regola di transizione 1.4 che crea il gradiente, la regola di transizione 1.8 modella un agente che percepisce il proprio valore del gradiente locale ed adatta il suo comportamento in base alla sua posizione relativa rispetto alla sorgente del gradiente.

$$\begin{aligned} state\_evolution ::&\langle L, [D, State, C]\rangle \xrightarrow{r_{move}} \langle L, [D, State', C]\rangle \\ &where\ State' = \pi\,(D) \end{aligned} \tag{1.8}$$

La funzione $\pi\,(D)$ cambia le variabili di stato dell'agente, evolvendo il proprio stato in base alle informazioni che percepisce nell'ambiente a livello locale.



(a) (b)

Figura 1.12: *Morphogenesis: comportamento 1.12(a) ed interazione 1.12(b) di un agente.* Immagine tratta da [6]

Implementazione: i diagrammi in figura 1.12 mostrano come gli agenti valutano la loro posizione in risposta all'informazione del gradiente propagato dagli host del vicinato.

Usi conosciuti: il *morphogenesis pattern* viene usato per implementare le tecniche di controllo per modellare l'auto-configurazione dei robot.

Conseguenze: il *morphogenesis pattern* dota gli agenti di un meccanismo per coordinare le loro attività in base alle loro posizione relative. Come gli altri meccanismi precedentemente presentati, la robustezza e la scalabilità risultano essere proprietà assicurate da questo pattern.

Pattern relativi: il *morphogenesis pattern* estende il *gradient pattern*. Il *morphogenesis pattern* può essere combinato con il *digital pheromone pattern* dove il ruolo e il comportamento degli agenti dipende dalle distanze dalle sorgenti del feromone.

1.8.4 Quorum sensing pattern

La rilevazione del quorum è un processo di decisione per il comportamento coordinato ed utile per prendere decisioni collettive in modo decentralizzato. L'obiettivo del *quorum sensing pattern* è di fornire una stima del numero di agenti (o della densità degli agenti) nel sistema utilizzando solo le interazioni locali. Il numero degli agenti nel sistema è cruciale in queste applicazioni, in particolare un numero minimo di agenti sono necessari per collaborare su determinati compiti.

Alias: nessuno in nostra conoscenza.

Problema: le decisioni collettive in sistemi decentralizzati su larga scala richiedono una soglia sul numero di agenti (o stima della densità degli agenti) nel sistema tramite l'uso delle sole interazioni locali.

Soluzione: il *quorum sensing pattern* permette di prendere decisioni collettive attraverso una stima effettuata dai singoli agenti della densità degli agente, ovvero valutare il numero degli altri agenti con cui essi interagiscono, e la determinazione della soglia del numero di agenti necessari per prendere la decisione.

Ispirazione: il *quorum sensing pattern* è ispirato dal processo di rilevazione del quorum (*Quorum Sensing*), il quale è un tipo di segnale intercellulare usato dai batteri per monitorare la densità cellulare per diversi scopi. Un esempio è il batterio bioluminiscente trovato in alcune specie di calamari. Tali batteri auto-organizzano il loro comportamento per produrre la luce solo quando la loro densità risulta essere sufficientemente alta. I batteri costantemente producono e secernono alcune molecole di segnalazione chiamate auto-induttori. In presenza di un alto numero di auto-indutturi, il livello di auto-induttori aumenta esponenzialmente, in altre parole maggiore è il livello di auto-induttori che un batterio rileva e più auto-induttori produce. Un altro interessante esempio è dato dalle colonie di formiche, in particolare quando la colonia deve trovare un nuovo nido. Una piccola porzione delle formiche ricerca dei potenziali nuovi siti di nidificazione e ne valuta la qualità. Quando ritornano nel vecchio nido, loro aspettando un certo periodo di tempo prima di reclutare altre formiche (valutazioni più alte producono tempi di attesa inferiori). Le formiche reclutate visitano il potenziale nuovo sito di nidificazione e danno la loro valutazione sulla qualità del nido ritornando al vecchio nido e ripetono il processo

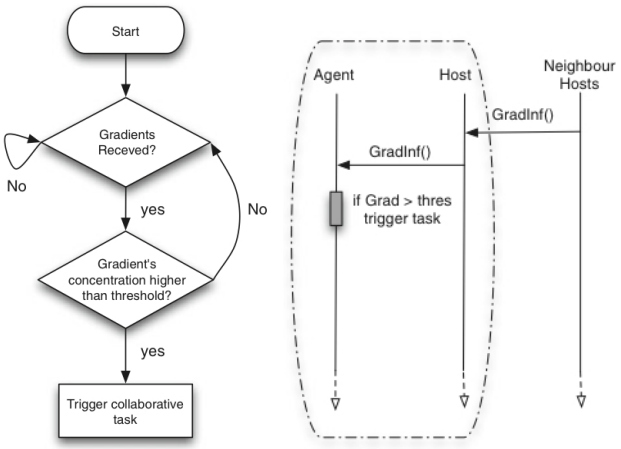
di reclutamento. A causa dei periodi di attesa, il numero di formiche presenti nel nido migliore tenderà ad aumentare. Sebbene le formiche in questo nido percepiscono che il tasso con cui incontrano le altre formiche supera una determinata soglia, il numero del quorum è raggiunto. Gli sciami come le api o le vespe usano le stesse tecniche per cercare il nido.

Vincoli: il *quorum sensing pattern* usa i gradienti che presentano gli stessi parametri del *gradient pattern*. La soglia, che indica il numero del quorum che è stato raggiunto, fa scattare il comportamento collaborativo. Il *Quorum Sensing* fornisce una stima della densità degli agenti presenti nel sistema. Questo pattern, comunque, non offre una soluzione per calcolare il numero degli agenti necessari a svolgere un compito collaborativo.

Entità-dinamiche-ambiente: le entità coinvolte nel *quorum sensing pattern* sono le medesime del *gradient pattern*, ossia gli agenti, gli host e gli agenti infrastrutturali. La concentrazione viene calcolata sull'aggregazione dei gradienti.

La regola di transizione per il il quorum pattern può essere modellata attraverso la regola di transizione 1.8, nella quale la funzione $\pi(D)$ è data dall'equazione 1.9.

$$\pi(D) = \begin{cases} State & \text{if } D \leq threshold \\ State' & \text{if } D > \ threshold \end{cases} \tag{1.9}$$



(a) (b)

Figura 1.13: *Quorum sensing: comportamento 1.13(a) ed interazione 1.13(b) di un agente. Immagine tratta da [6]*

Implementazione: non esiste una specifica implementazione per il *quorum sensing pattern*. I sistemi biologici presentati in precedenza, però, forniscono alcune idee su

come implementare tale pattern. Si propongono due diversi approcci per implementare il *quorum sensing pattern*:

- usare il *gradient pattern* per simulare gli auto-induttori come per i batteri luminescenti. In questo caso la concentrazione del gradiente fornisce degli agenti che effettuano sua stima della densità degli agenti;

- come nei sistemi delle formiche, la densità degli agenti può essere stimata attraverso la frequenza nella quale gli agenti sono nel raggio di comunicazione. L'uso dei gradienti fornisce la migliore stima rispetto all'uso delle frequenze. Tale soluzione è più costosa computazionalmente e richiede più comunicazioni attraverso la rete. I diagrammi in figura 1.13 mostrano gli agenti che identificano se la concentrazione del gradiente ha raggiunto la soglia, in seguito alle informazioni del gradiente propagato dagli host vicini.

Usi conosciuti: il *quorum sensing pattern* viene usato per aumentare il risparmio energetico nelle reti di sensori wireless. La rilevazione del quorum permette di creare dei cluster basati sulla struttura dei parametri di interesse osservati, e quando solo un nodo per ogni cluster invia l'informazione per conto del quorum. Un altro uso conosciuto è per la coordinazione degli *autonomous swarm robots*.

Conseguenze: ogni agente può stimare la densità di nodi degli altri agenti nel sistema usando solo l'informazione locale ricevuta dai vicini, anche quando il sistema è molto grande e gli agenti sono anonimi.

Pattern relativi: il *quorum sensing pattern* usa , in modo diverso a seconda dell'implementazione, il *gradient pattern*.

2
Ecosistemi di servizi pervasivi

In questo capitolo si introduce il concetto di sistema pervasivo (come riportato in [18]), si presenta un framework di coordinazione (SAPERE) e si discute sull'utilità dell'auto-organizzazione nei sistemi pervasivi. A questo punto, si introduce il concetto di *spatial computing* ed i pattern che garantiscono avere strutture spaziali. In particolare, si presenta la struttura dello *spatial gradient* prima attraverso la sua definizione e poi in un scenario di *crowd steering*.

2.1 Definizione e requisiti di un sistema pervasivo

Un sistema pervasivo è un *computing system* presente nel nostro ambiente quotidiano, fatto di attori e componenti di vario genere (i quali possono essere raggruppati con il temine **individui**): persone, i loro smarphone, servizi software, schermi pervasivi, sensori e dispositivi diffusi in tutto l'ambiente, sorgenti di conoscenza, dati ed eventi. Tutti interagiscono in modo opportunistico per raggiungere i loro obiettivi individuali, ma sono anche guidati e governati globalmente da alcune "leggi" presenti nell'infrastruttura.

Le infrastrutture a livello globale possono essere rese più utili ed efficaci sia per gli utenti che per i fornitori di informazioni/servizi se diventano generali, **aperti** e sistemi **auto-adattivi**. Si evince, dunque come da [18], che i sistemi di servizi pervasivi devono soddisfare i requisiti di: *situatedness*, adattività e tolleranza alla diversità.

Il primo requisito da affrontare è la situatedness: i sistemi pervasivi gestiscono le attività degli utenti *spatially-* e *socially-situated* e dovrebbero, pertanto, essere in grado di interagire con il mondo circostante ed adattare il loro comportamento di conseguenza. Generalmente tale requisito viene realizzato dalle infrastrutture attraverso la reificazione di dati/conoscenza/eventi dal punto preciso (o regione precisa) dello spazio a cui essi appartengono e la promozione delle interazioni basate sulla vicinanza.

Un altro requisito complementare è l'adattività: i sistemi pervasivi e le loro infrastrutture dovrebbero intrinsecamente presentare proprietà di adattamento e gestione autonoma per sopravvivere agli **imprevisti** senza l'intervento umano, supervisione globale o entrambi. L'adattività è spesso raggiunta con la progettazione di regole di coordina-

mento, che agendo a livello locale (ovvero in una determinata zona della rete), fanno emergere in modo **dinamico** proprietà globali.

Infine l'infrastruttura dovrebbe intrinsecamente tollerare la diversità, vale a dire sostenere **modelli aperti** di produzione e servizio ed il loro utilizzo. Il numero e le classi di servizi offerti non dovrebbe essere limitato, piuttosto l'iniezione di nuovi servizi dovrebbe essere considerata, quando possibile, un vantaggio per sfruttare, integrare ed aggiungere ulteriore valore ai servizi esistenti. La tolleranza della diversità può essere ottenuta rendendo standard il modo in cui gli individui (indipendentemente dalla loro natura) manifestano la loro esistenza nel sistema ed il modo con il quale tale manifestazione viene utilizzata per attivare le interazioni.

2.2 Framework degli ecosistemi pervasivi

Gli ecosistemi di servizi pervasivi richiedono un paradigma alternativo per progettare sistemi *context-aware* dotati di capacità di adattamento e self-awareness.

Il progetto `SAPERE` (acronimo di *"Self-adaptive Pervasive Service Ecosystems"*, `www.sapere-project.eu`) risponde a tali requisiti attraverso l'introduzione di un substrato spaziale, rappresentato come un ecosistema di individui autonomi di varie tipologie, fra i quali servizi pervasivi, dispositivi e persone. Tali individui si incontrano, interagiscono, competono e si combinano fra loro (in una parola si coordinano) sulla base di semplici leggi (dette *eco-laws*). L'applicazione di tali leggi provoca l'evoluzione della popolazione degli individui nel sistema.

2.2.1 Architettura ed il linguaggio delle eco-laws

L'approccio nel progetto `SAPERE` trae ispirazione dal modello chimico (come da [18]). L'idea di base all'interno di tale framework è di modellare tutti i componenti dell'ecosistema, chiamati individui, in modo uniforme, sia che questi siano persone che agiscono sul sistema o lo percepiscono oppure che siano PDA, dispositivi pervasivi o servizi software.

Dal punto di vista architetturale (in figura 2.1) tale progetto prevede, come anticipato, la realizzazione di un substrato spaziale, il quale viene mappato sopra l'infrastruttura di rete pervasiva ed è formato da nodi, ciascuno di essi rappresentante una specifica area locale. In particolare, ogni nodo è composto da uno spazio atto alla condivisione delle informazioni fra gli individui che in un dato momento si troveranno nella suddetta area. I servizi a disposizione dell'infrastruttura e degli individui si manifesteranno al sistema inserendo e gestendo negli spazi dei nodi delle rappresentazioni apposite chiamate LSA (acronimo di *Live Semantic Annotation*). Queste rappresentazioni sono state definite Live per la loro capacità di rappresentare la dinamicità associata allo scenario ed ai singoli componenti cui sono riferiti, i quali nella maggior parte dei casi risultano essere entità mutevoli ed il relativo stato o contesto può cambiare anche molto frequentemente. La

coordinazione fra gli individui ed il sistema sarà, invece, garantita, come già detto, da leggi di ispirazione chimica dette eco-laws, che utilizzano gli LSA in veste di reagenti, quindi li modificheranno permettendo così l'evoluzione di questi e l'emergenza di pattern di coordinazione complessi. Ad esempio, non appena un componente entra nell'ecosistema, il suo LSA viene automaticamente creato ed iniettato nel substrato, di modo che, essendo questo condiviso, tutti gli individui interessati ne possano venire a conoscenza.

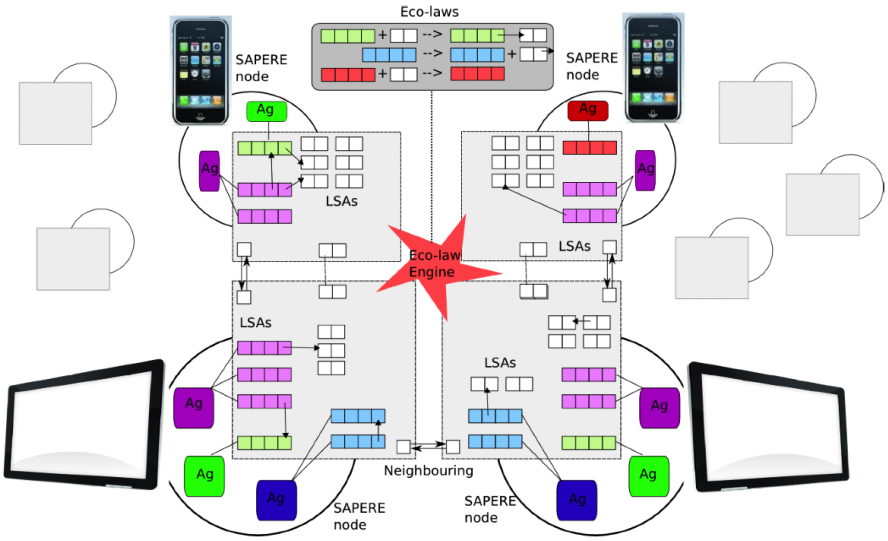


Figura 2.1: *Un vista architetturale di un ecosistema pervasivo. Immagine tratta da [18]*

In una visione topologica, lo spazio condiviso è strutturato come una rete di **LSA-spaces**. Un LSA-space è un componente passivo, simile ad uno spazio di tuple, il quale fornisce un'interfaccia che permette l'inserimento, la rimozione, modifica e lettura degli LSA contenuti al suo interno. Ogni LSA-Space viene ospitato in un nodo dell'infrastruttura e ha il compito di memorizzare i relativi LSA locali. Inoltre, ognuno di essi viene regolato dalle **eco-laws**, le quali definiscono le politiche per regolare le **reazioni** fra LSA, con un particolare riguardo al coordinamento dei dati e dei servizi. Le LSA sono viste come reagenti chimici in una ecologia in cui le interazioni e composizioni si verificano attraverso reazioni chimiche che si attivano sulla base del *pattern-matching* fra LSA. Gli effetti di tali reazioni possono essere:

- modifica dello stato delle LSA dipendenti dal contesto,
- produzione di nuove LSA,
- diffusione di LSA ai nodi vicini. In particolare, la vicinanza di due *LSA-space* abilita una comunicazione diretta.

Coordinazione, adattività e resilienza nel framework SAPERE non sono state inserite fra le capacità dei componenti individuali, bensì emergono dalle dinamiche globali dell'ecosistema. Perturbazioni nel sistema, risultanti come effetti delle **eco-laws**, possono portare alla creazione/rimozione/modifica di altre LSA. L'architettura SAPERE promuove, quindi, l'adattività e la coordinazione non adottando la resilienza a livello dei componenti, piuttosto promuovendo una sorta di "resilienza del sistema".

Di seguito si presenta il linguaggio delle *eco-laws* in modo informale, in accordo con [11]. Nel framework SAPERE le LSA sono annotazioni semantiche, poiché esprimono l'informazione con la stessa espressività dei framework standard come RDF. In questo contesto, però, si è scelto di considerare un notazione semplificata, ovvero un LSA viene semplicemente modellato come una tupla $\langle v_1, \ldots, v_n \rangle$ (sequenza ordinata) di valori tipati.

Una *eco-law* in modo simile ad una reazione chimica lavora sui pattern delle LSA. Un LSA pattern P è un LSA in cui si possono avere delle variabili al posto di uno o più argomenti di una tupla, ed un LSA L si dice che fa *match* con il pattern P se esiste una sostituzione di variabili attraverso la quale è possibile applicare P dato L. In particolare, una *eco-law* è del tipo $P_1, \ldots, P_n \xrightarrow{r} P'_1, \ldots, P'_m$ dove:

- il lato sinistro (reagenti) specifica i pattern che dovrebbero fare match con le LSA $L_1, \ldots, L_n$ estratte dal *LSA-space*;
- il lato destro (prodotti) specifica i pattern delle LSA, le quali dovrebbero essere inserite come risultato nel *LSA-space*;
- il rate r è un valore numerico positivo che indica la frequenza media con la quale l' *eco-law* viene attivata.

Tale semplice linguaggio viene esteso con alcuni ingredienti chiave per permettere il raggiungimento degli obiettivi dei vari framework. Per prima cosa per permettere l'interazione fra *LSA-space* diversi si introduce il concetto di **pattern remoto**, indicato con $+P$, il quale è un pattern che può fare match con un LSA presente in un *LSA-space* vicino (chiamato LSA remoto).

Allo scopo di permettere l'applicazione delle *eco-laws* su una più vasta gamma di LSA, l'argomento di un pattern può contenere anche un'espressione matematica o una variabile di sistema. In particolare, alcune variabili di sistema possono essere usate sia nei reagenti che nei prodotti per inferire l'informazione sul contesto fornita dall'infrastruttura; esse sono precedute da # , includono $\#T$ che è legato al tempo rispetto al quale l'*eco-law* viene attivata, $\#D$ esplicita la distanza topologica fra lo spazio locale e remoto, e $\#O$ indica l'orientamento dello spazio remoto rispetto allo spazio locale. Ad esempio, le reazioni impiegate per creare una struttura di tuple a gradiente (figura 2.2) potrebbe

essere:

$$\langle gradient, Source, Dist\rangle \rightarrow \langle gradient, Source, Dist\rangle, +\langle gradient, Source, Dist + \#D\rangle$$
$$\langle gradient, Source, Dist\rangle, \langle gradient, Source, Dist2\rangle \rightarrow \langle gradient, Source, min(Dist, Dist2)\rangle$$

La prima reazione propaga un clone della tupla gradiente su ogni vicino sostituendo adeguatamente il valore della distanza; l'ultima prende due tuple gradiente della stessa sorgente e mantiene quella con la distanza più breve dalla sorgente. Per un esempio più formale si faccia riferimento alla sezione 2.8.1.

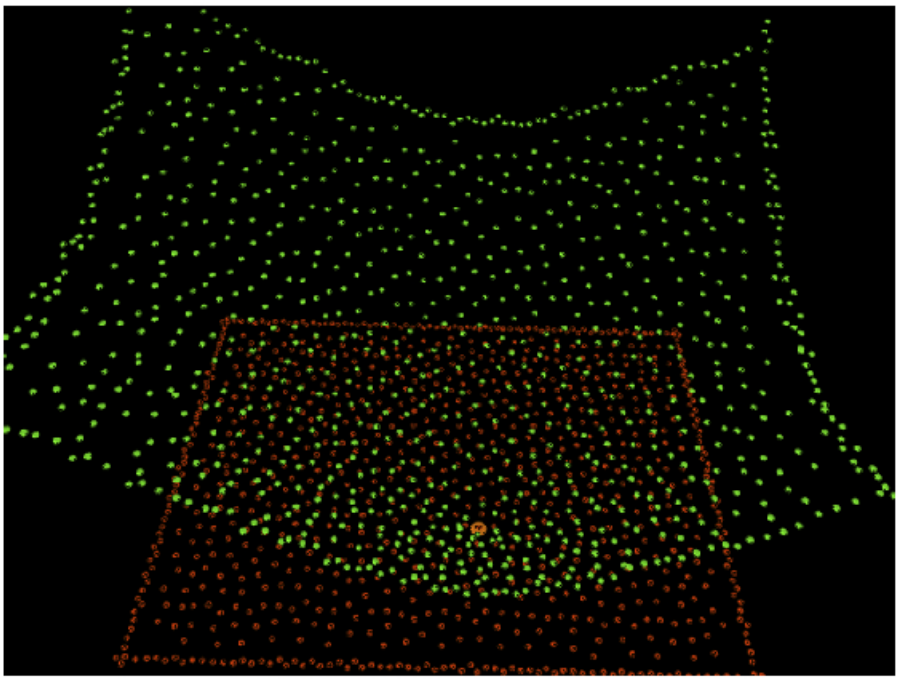


Figura 2.2: *Una struttura a gradiente. Immagine tratta da [1]*

2.3 Auto-organizzazione nei servizi pervasivi

Nei sistemi naturali (a livello fisico, chimico, biologico o sociale) tutte le attività dei componenti del sistema sono intrinsecamente situate nello spazio e guidate da interazioni locali. Tali interazioni non sono governate da pattern di orchestrazione predefiniti. Le interazioni locali sono, infatti, semplicemente oggetto di una serie limitata di leggi naturali, attraverso le quali anche complessi modelli di interazione emergono attraverso

l'auto-organizzazione. In questo modo, l'adattività diventa una caratteristica intrinseca derivante dalla presenza di auto-organizzazione nei modelli delle interazioni, la cui struttura può essere rimodellata in modo flessibile e robusto in risposta agli imprevisti. Tipici meccanismi di auto-organizzazione sono quelli che utilizzano la stigmergia, come il comportamento coordinato delle formiche in ricerca di cibo, addestramento e flocking per la coordinazione dei movimenti o sistemi basati sui gradienti.

I sistemi di servizi pervasivi hanno delle strette relazioni con gli ecosistemi naturali ed in particolare condividono con essi la necessità di una intrinseca auto-organizzazione attraverso la quale **il comportamento inteso a livello globale emerge dalle interazioni locali degli individui**, guidato dalle manipolazioni locali emanate dalle leggi ovvero, nella terminologia dell'approccio introdotto, il trigger delle *eco-laws*. Esempi di contesti applicativi comprendo la fornitura di servizi di visualizzazione su schemi adattativi, servizi per città *smart*, come il controllo intelligente del traffico e la realtà e società aumentata. Per rendere l'auto-organizzazione un meccanismo applicabile in modo sistematico, i suoi meccanismi sono stati descritti nella forma di design-pattern, questi sono stati presentati nella sezione 1.5.

2.4 Spatial computing

I modelli computazionali tradizionali sono luoghi fisici astratti nello spazio (ad esempio Internet ed i processori superscalari) e l'implementazione è, prevalentemente, sequenziale. La maggior parte delle strutture di dati comuni esistenti risultano essere spazialmente indipendenti, ma:

- su scala continua, ovvero le computazioni (sia come elementi primitivi atomici, sia come numero di elementi composti) devono essere distribuite nello spazio e la posizione nello spazio influisce sulle prestazioni e la fattibilità della computazione;
- con la computazione accoppiata ed embedded nel mondo fisico (ad esempio costruzioni smart, superfici reattive, materia programmabile, robotica distribuita), la posizione e la forma sono primarie, poiché servono sia come input per il calcolo, sia come parte fondamentale del risultato atteso dalla computazione;
- come si capisce dai sistemi computazionali naturali (ad esempio cellule, colonie di formiche, biologia del sistema) la localizzazione e la topologia definiscono la computazione.

Di conseguenza è importante rendere lo spazio un'astrazione di primo ordine, che è necessario ottimizzare. La caratteristica distintiva della *spatial computing* risulta essere, in accordo con [2], che la computazione è embeded nello spazio, in particolare, la posizione assume un ruolo chiave sia per il problema che per il risultato della computazione.

La figura 2.3 riassume alcune applicazioni di *spatial computing* suddividendole in tre categorie:

- **intensive computing**, dove lo spazio è usato come mezzo e come risorsa;
- **computation embedded**, nello spazio dove la posizione è importante per il problema;
- **space computation**, dove lo spazio è fondamentale per il problema e è un risultato della computazione.

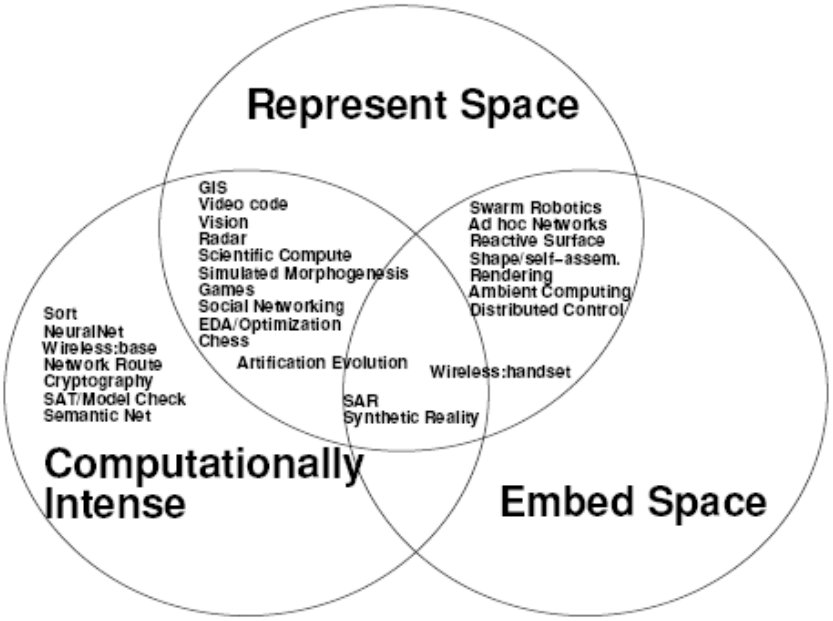


Figura 2.3: *Esempio di applicazioni spatial computing e le loro relazioni rispetto alle tre categorie: computationally intense, embedded in space, represent space. . Immagine tratta da [2]*

Ognuna di queste tre categorie della *spatial computing* ha le proprie motivazioni e fondamenti logici. Esse, comunque, sono molto intrecciate. Mentre vari problemi e questioni assumono priorità diverse in ogni dominio, un insieme comune di teoria e tecniche possono essere applicate a tutti i domini. Le computazioni distribuite, auto-organizzanti e potenzialmente tolleranti ai guasti sono essenziali per le computazioni embedded nello spazio. Tali algoritmi possono anche alleggerire le computazioni *computationally intense*.

2.4.1 Spatial computer

Creare un collegamento fra i sistemi biologici e l'approccio ingegneristico è spesso difficile. La complessità del mondo biologico ed il pensiero architetturale della metodologia ingegneristica corrente risultano spesso in contrasto. In molti casi, tuttavia, si può trovare qualche utile punto in comune tra questi due mondi nella nozione di spatial computer:

> *Qualsiasi collezione potenzialmente elevata di dispositivi computazionali, in cui la capacità di comunicare dei dispositivi risulta essere fortemente correlata alla distanza fisica.*

Esistono molti esempi di tali sistemi in diversi campi dell'ingegneria, come ad esempio le reti di sensori, reti mobili ad-hoc, sistemi pervasivi, *robotic swarms* ed architetture di calcolo riconfigurabili. Ci sono anche molti esempi nel mondo naturale, come ad esempio la biopellicola, gli organismi pluricellulari, gli stormi di uccelli e le colonie di insetti. Sebbene i dettagli di tali sistemi siano spesso radicalmente diversi, **i vincoli imposti dalla località spaziale frequentemente dettano soluzioni simil**. Ad esempio, i gradienti sono un pattern comune nei sistemi biologici, questi guidano fenomeni come lo sviluppo embrionale ed il movimento delle formiche alla ricerca del cibo. I gradienti sono anche usati nel campo dell'ingegneria in diverse aree come il *distance vector routing*, la costruzione collettiva e la gestione delle folle (*crowd management*). Esistono una serie di pattern di base, come la rottura della simmetria ed il consenso approssimato, che sono emersi in modo indipendente in entrambi i mondi o dove i modelli di fenomeni naturali sono stati adattati per l'impiego in ingegneria e molti di questi sono di natura spaziale.

Un esempio è la modellazione ed il controllo di sistemi composti da più dispositivi computazionali, che risulta essere un problema ben noto. Questo problema è cresciuto sempre di più nel corso degli anni dato che il numero e la densità dei dispositivi computazionali continua ad aumentare rapidamente. In termini di macro-scala il numero di computer per persona continua a salire rapidamente e reti più pervasive si legano in sistemi sempre più ampi (figura 2.4). Questa tendenza si estende anche nel mondo naturale, come la computazione complessa effettuata dagli aggregati degli organismi viventi, si pensi alle cellule che compongono gli organismi.

Negli ultimi anni la spatial computing (computazione spaziale) è emersa come promettente approccio per la modellazione ed il controllo di questo tipo di sistemi pervasivi. In questo contesto la visione di base della *spatial computing* è semplice: quando la densità dei dispositivi computazionale è elevata, vi è una stretta relazione tra la struttura della rete dei dispositivi e la geometria dello spazio in cui sono distribuiti. In termini più formali, per i sistemi pervasivi la spatial computing è:

> *Una aggregazione di dispositivi nella quale la difficoltà di trasferimento dell'informazione fra due dispositivi è fortemente dipendente dalla distanza fra essi e gli obiettivi funzionali del sistema vengono generalmente definiti in termini della struttura spaziale del sistema.*

(a)

(b)

(c)

Figura 2.4: *Esempio di sistema pervasivo situato nell'ambiente. Immagine tratta da [5]*

Si noti che le definizioni sono state tratte da [4].

2.5 Framework per l'analisi della spatial computing

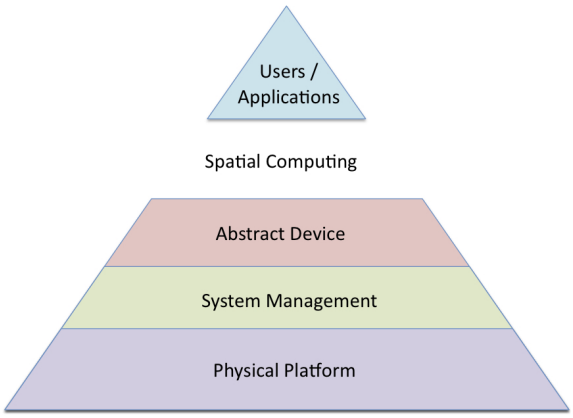


Figura 2.5: *La spatial computing colma un gap architetturale fra la computazione dei singoli dispositivi e gli utenti che desiderano di controllare il comportamento aggregato. Immagine tratta da [4]*

La forte crescita del numero dei dispositivi computazionali presenti nei sistemi ingegnerizzati ha creato un ampio gap tra le esigenze dell'utente per controllare l'aggregazione dei dispositivi e le complesse tecnologie dei singoli dispositivi. La *spatial computing* tenta di colmare questo gap per i sistemi di comunicazione locali sfruttando la connessione fra la **posizione fisica** e la **connettività del dispositivo**. Geometria e topologia includono diversi concetti di aggregazione intermedia, come le regioni, i vicinati ed i flussi che possono essere utilizzati come elementi di base per costruire applicazioni software ed associare automaticamente il comportamento ai singoli dispositivi.

Allo scopo di avere più chiare le caratteristiche del gap che si intende colmare, in [4] si definisce un'architettura generica per la programmazione aggregata (figura 2.5) composta da cinque livelli. I tre livelli inferiori gestiscono solamente i singoli dispositivi e le interazioni locali. Dal livello più basso a quello più alto si presentano:

- il livello della **piattaforma fisica**, il mezzo su cui avviene la computazione, potrebbe essere qualsiasi cosa da uno smartphone ad un drone fino ad una cellula biologica o un dispositivo virtuale utilizzano nelle simulazioni;
- il livello della **gestione del sistema**, dove il sistema operativo e qualsiasi servizio viene integrato nella piattaforma "viva". Ad esempio, su un dispositivo embedded

questo livello potrebbe includere la gestione dei processi in tempo reale, dei driver dei sensori ed attuatori e la rete di basso livello;

- il livello dell' **astrazione del dispositivo**, nasconde i dettagli del dispositivo, presentando ai dispositivi con cui interagisce un'interfaccia semplificata.

Sopra il livello dell'astrazione del dispositivo si trova il gap, in particolare risulta assente un collegamento tra l'interfaccia per la programmazione dei singoli dispositivi ed il livello dei dispositivi aggregati desiderato dagli utenti. Le astrazioni ed i modelli dello *spatial computing* permettono di colmare tale gap fra gli utenti e gli aggregati. Nella parte superiore della piramide, invece, sono collocati gli utenti e le applicazioni che vogliono utilizzare non i singoli dispositivi, bensì i comportamenti dei dispositivi aggregati.

2.5.1 Tipi di operazioni spazio-temporali

La figura 2.6 mostra i due elementi di base della *spatial computing* e le relazioni tra essi, in accordo con [4]. La *spatial computing* è una entità duplice: da un lato, si ha la dimensione dello spazio-tempo dove si collocano fisicamente i dispositivi e, dell'altro, l'informazione che è associata alle posizioni della dimensione spazio-tempo.

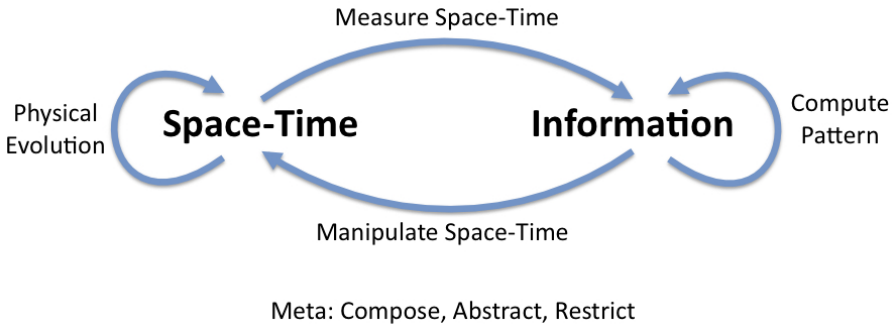


Figura 2.6: *La dualità di base dello spazio-tempo e l'informazione nella spatial computing implica quattro classi generali di operazioni, più le meta-operazioni che manipolano le computazioni. Immagine tratta da [4]*

Da tale dualità si possono ricavare quattro classi di operazioni:

- **misurare lo spazio-tempo**: si utilizzano operatori che ricavano le proprietà geometriche dalla dimensione spazio-tempo e le traducono in informazioni. Esempi sono le misure di: distanza, angolo, durata, area, densità e curvatura.

- **manipolare lo spazio-tempo**: questi operatori lavorano in modo contrario rispetto a quelli di misurazione, essendo essi attuatori, infatti, prendono le informazioni e modificano le proprietà della dimensione spazio-tempo. Esempi sono i dispositivi mobili, i quali possono cambiare la curvatura (ad esempio, flettendo una superficie), espandere o contrarre lo spazio locale (ad esempio, modificando la dimensione delle cellule in un tessuto), oppure perturbare le proprietà fisiche locali, tra cui la rigidità che influenza direttamente l'evoluzione della fisica sistema.

- **computare un pattern**: operazioni presenti esclusivamente nel mondo informativo, possono essere viste ad un livello elevato di astrazione come modelli di calcolo sullo spazio-tempo. Per esempio, le strisce sono un modello nello spazio, un timer è un modello nel tempo mentre un'onda sinusoidale di moltiplicazione è un modello nello spazio-tempo. Questa categoria comprende non solo la computazione, ma anche la comunicazione e qualsiasi sensore o attuatore "puntuale che non interagisce direttamente con la geometria, come un sensore di luce o suono oppure un attuatore con il LED.

- **evoluzione fisica**: molti sistemi fisicamente esemplificati presentano una dinamica tale da permettere che la forma dello spazio cambi nel tempo anche senza l'impiego di attuatori in grado di manipolare lo spazio-tempo. Si pensi al movimento inerziale dei robot o delle forze adesive che permettono la formazione di una colonia di cellule. Per loro natura le operazioni citate non possono essere direttamente parte dei programmi, ma in linguaggi che potrebbero riuscire a gestire le dinamiche attraverso funzioni specifiche.

Qualsiasi *spatial computation* può essere descritta in termini di queste quattro classi di operazioni. Le meta-operazioni possono essere utilizzate per per combinare e modulare le computazioni spaziali. Si identificano due classi di meta-operazioni:

- **astrazione e composizione**: le operazioni nascondono i dettagli implementativi delle computazioni spaziali, permettono di essere combinate insieme ed avere istanze multiple della computazione eseguita;

- **restrizione**: le operazioni modulano una computazione spaziale attraverso la selezione di un particolare sotto-spazio nel quale la computazione dovrebbe essere eseguita.

2.6 Spatial DSLs

Dallo studio eseguito in [4] è emerso che esistono relativamente pochi esempi di Domain Specific Language (DSL) per il *pervasive computing* con **operatori spaziali**: la maggior parte, come il linguaggio di coordinazione LINDA o altri linguaggi che usano gli

spazi delle tuple, cercano di astrarre dalla programmazione della rete, questo porta alla nascita di diversi linguaggi e modelli. **Tuples on the Air** (TOTA) è un middleware per la condivisione dei dati in forma di tuple in modo efficiente attraverso la rete. Il modello basato sulle reazioni chimiche si ispira alle reazioni chimiche per la diffusione dei dati attraverso la rete. Infine, *Zone di influenza* (ZoI) dei sistemi *Peer-It* modellano i dispositivi pervasivi che possono influenza od interagire fra loro. Tutti i DSL per il pervasive computing si servono di un programma uniforme mirato per i dispositivi discreti, sebbene i dispositivi possano essere spostati da agenti esterni, quali persone che li trasportano o li usano.

TOTA, come il linguaggio di coordinazione LINDA ed altri linguaggi basati sullo spazio di tuple, cerca principalmente proprietà spaziali astratte della rete. Pertanto TOTA ha pochissime proprietà spaziali ad eccezione della propagazione sul vicinato. Il modello basato sulle reazioni chimiche, invece, usa un spatial gradient (gradiente spaziale) per diffondere le informazioni ai vicini attraverso la rete (sezione 2.7.1). Le tuple *spatially-scoped*, quindi, offrono espliciti pattern per la comunicazione a livello di vicinato.

La differenza fra i due framework è nella modalità di comunicazione. TOTA utilizza un meccanismo distribuito pubblica/sottoscrivi per la comunicazione globale e multicast, inoltre dispone di un raggio di comunicazione a livello del vicinato. Il modello basato sulle reazioni chimiche e le tuple *spatially-scoped* utilizzano un raggio di comunicazione a livello di vicinato per i singoli utenti (cioè con granuralità unicast).

2.7 Campo computazionale negli ecosistemi pervasivi

Le *eco-laws* (discusse nella sezione 2.2.1) possono essere utilizzate per generare computational fields (campi computazionali), noto meccanismo di auto-organizzazione. Un *computational field* è una struttura dati distribuita basata su astrazioni spaziali (come ad esempio distanza e regione). Esempi di campi computazionali includono:

- gradienti: mappano ogni nodo con la distanza minima da una sorgente;
- **percorsi**: mappano ogni nodo lungo il percorso ottimale da una sorgente e una distanza di un valore non nullo;
- **partizioni**: mappano ogni nodo con quello più vicino in un insieme finito di nodi.

Tra i pattern per i sistemi auto-organizzanti (capitolo 1), quelli che garantiscono avere strutture dati spaziali sono il *chemotaxis* (sezione 1.8.2) e il *gradient* (sezione 1.7.1). La **chemotaxis** è la capacità di un agente di spostarsi in una struttura dati distribuita

per raggiungere il nodo sorgente del gradiente anche nel caso in cui l'ambiente sia estremamente mobile e dinamico. Nelle applicazioni *situated* tale pattern viene utilizzato per il recupero fisico di elementi. Un ruolo importante è, quindi, svolto dalla struttura dello **spatial gradient** (gradiente spaziale), un mattone fondamentale dei meccanismi di auto-organizzazione che viene in genere progettato per fornire percorsi ottimali di cammino in un sistema distribuito, anche nel contesto di ambienti molto articolati (un edificio con stanze e corridoi o un scenario di traffico) e di adattamento dinamico (*self-healing*) a situazioni impreviste come una brusca interruzione di una strada. Lo *spatial gradient* risulta, infatti, molto studiato ed utilizzato, perciò si è deciso di fornirne una descrizione più precisa nella sottosezione seguente.

2.7.1 Spatial gradient

Il gradiente è un meccanismo fondamentale nei sistemi di computazione pervasiva ed il *gradient pattern* (sezione 1.7.1) può essere visto come un *computational field*.

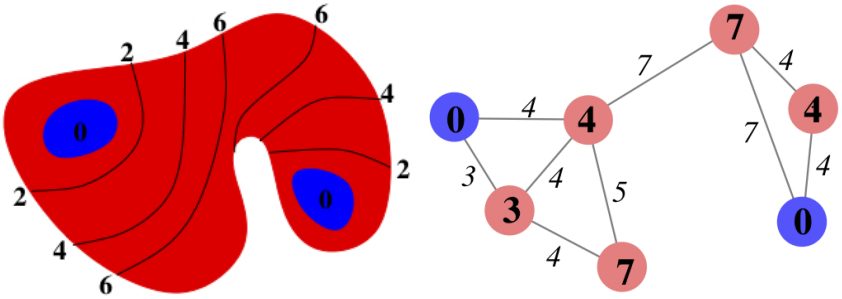
Si è visto che il gradiente può essere descritto come un'operazione *bio-inspired*, in particolare è un campo scalare che indica l'insieme dei valori della distanza minima di ogni punto dalla sorgente (figura 2.7(a)). Nel dettaglio, tale valore viene calcolato basandosi sulle tuple che trasportano la distanza stimata (0 nella sorgente). Tali tuple vengono diffuse in un vicino con la distanza aumentata, l'entità di tale incremento potrebbe essere semplicemente 1 per ogni passo oppure potrebbe stimare la distanza fisica # D, e aggregate in ogni nodo così da mantenere il valore più recente e con la distanza minima. Il *decay* (decadimento) viene usato come meccanismo di invecchiamento, è utile, ad esempio, per eliminare un gradiente quando la sua sorgente viene rimossa. È interessante notare che il *gradient pattern* può essere progettato anche per effettuare un adattamento dinamico della rete rispetto a cambiamenti nella topologia ed al movimento della sorgente.

Per mappare un gradiente in una rete di sensori, si effettua un'approssimazione che assegna ad ogni dispositivo della rete il rispettivo valore (figura 2.7(b)).

Per lo scopo di questa tesi il modello di riferimento sarà rappresentato da un ecosistema pervaso da sensori. Le caratteristiche di tale modello discreto della rete sono:

- il numero di sensori è finito e può variare da decine a centinaia;
- i sensori sono rappresentati da nodi immobili e sono distribuiti in modo arbitrario nello spazio;
- i nodi comunicano attraverso una trasmissione broadcast non affidabile sul vicinato;
- la memoria e la potenza non sono una risorsa limitata[1];

[1] L'eccessivo consumo di potenza risulta dannoso in tali dispositivi poiché nella realtà sono dotati di energia limitata.



(a) Gradiente continuo (b) Gradiente discreto

Figura 2.7: *Un gradiente è un campo scalare dove il valore di ogni punto è la distanza più breve dalla regione della sorgente (blu). Il valore di un gradiente in un rete approssima il valore del gradiente nello spazio continuo contente la rete. Immagine tratta da [3]*

- l'esecuzione avviene in modo asincrono;
- i sensori sono in grado di di stimare la propria distanza dai vicini, senza però avere accesso ad altre informazioni globali. In particolare, il *naming*, il *routing* e le coordinate non sono fornite[2];
- cambiamenti nelle connessioni di rete possono verificarsi in qualsiasi istante, ovvero i sensori possono guastarsi, collegarsi alla rete o abbandonarla.

Il punto critico nel collegamento fra il modello discreto e continuo è stato gestito attraverso il concetto di vicinato (*neighborhood*). Ogni sensore trasmette periodicamente al suo vicinato, aggiornando la loro visione del suo stato. Il vicinato di un nodo è, quindi, definito dall'insieme di nodi che ricevono periodicamente il suo stato. Nel momento in cui un nodo non riceve più alcun messaggio da un nodo precedentemente considerato come vicino, esso assume che il sensore rappresentato da quel nodo sia guasto.

Le trasmissioni contengono il valore del campo del nodo trasmittente, informazione utile ai vicini. Tutti i sensori sono programmati allo stesso modo, quindi un nodo per avere una rappresentazione del vicinato completa dovrà ricevere i valori da tutti i sensori considerati vicini. Le trasmissioni possono essere usate, inoltre, per valutare in termini di spazio-tempo le posizioni di ogni nodo rispetto agli altri, inserendo informazioni opportune nei messaggi.

2.8 Gradiente computazionale in SAPERE

All'interno del progetto SAPERE (sezione 2.2) tutte le informazioni scambiate sono in forma di LSA e le regole sono espresse in forma di eco-law. Seguendo tale approccio, il

[2]Queste caratteristiche possono essere realizzate come valori dei nodi.

gradiente base introduce due forme di LSA:

$$\langle \texttt{pump}, tstamp \rangle$$
$$\langle \texttt{field}, value, tstamp \rangle$$

Una pump LSA viene usata come sorgente, il cui scopo è di generare un campo: si utilizza l'etichetta *tstamp* per indicare il momento della creazione della LSA. Un field LSA, invece, viene usato per i singoli valori del gradiente e l'etichetta *tstamp* è utile per discriminare il valore più recente del campo.

$$\langle \texttt{pump}, t \rangle \xmapsto{r_{pump}} \langle \texttt{pump}, t+1 \rangle, \langle \texttt{field}, 0, t \rangle$$
$$\langle \texttt{field}, v, t \rangle \xmapsto{r_{spread}} \langle \texttt{field}, v, t \rangle, *\langle \texttt{field}, v + \#D, t \rangle$$
$$\langle \texttt{field}, v, t \rangle, \langle \texttt{field}, v', t' \rangle \mapsto \langle \texttt{field}, v', t' \rangle \; if \; t' > t$$
$$\langle \texttt{field}, v, t \rangle, \langle \texttt{field}, v', t \rangle \mapsto \langle \texttt{field}, min(v, v'), t \rangle$$

Figura 2.8: *Eco-law che descrivono il gradiente*

Il gradiente base viene creato attraverso le quattro regole in figura 2.8. La prima *eco-law*, data una sorgente, inizializza un campo con il valore 0 e re-inserendo la LSA pump aumentando di 1 il *tstamp* permette successive generazioni del gradiente, questo fa sì che il gradiente possa rispondere a future modifiche dell'ambiente. La seconda *eco-law*, quando un nodo contiene un field LSA, diffonde sul vicinato un field LSA in cui aumenta il valore in ingresso di #D, ovvero la distanza fra il nodo trasmittente ed il ricevente. Come anticipato, l'etichetta *tstamp* si è introdotta per affrontare la dinamicità dello scenario. In particolare, la prima *eco-law* aggiorna e scrive *tstamp* nel field LSA mentre la terza e la quarta leggono tale valore per discriminare l'informazione. Nel dettaglio, la terza *eco-law* scarta il field LSA meno recente, difatti viene chiamata *youngest*; mentre la quarta fra due field LSA con lo stesso *tstamp* scarta quello con il valore superiore, difatti viene detta *shortest*.

La sezione 2.8.1 descrive l'applicazione del gradiente su uno scenario di *crowd steering* (guida della folla) ed i risultati della simulazione effettuata con ALCHEMIST (framework di simulazione presentato nel dettaglio in sezione 3.2).

2.8.1 Uno scenario di crowd steering

Si propone uno scenario di crowd steering come caso di studio per dimostrare la possibilità di sfruttare le *eco-law* (sezione 2.2.1) per condurre in breve tempo le persone

nella posizione desiderata all'interno di un **ambiente complesso**, evitando ostacoli come le **regioni affollate** e **senza la supervisione a livello globale**. Per la descrizione completa di tale caso di studio si faccia riferimento al paper [15].

Ambiente

Si consideri un museo con una serie di stanze, il cui pavimento è ricoperto da una serie di dispositivi computazionali (nodi infrastrutturali). Questi dispositivi possono:

- scambiarsi informazioni tra loro in base alla vicinanza,
- sentire la presenza dei visitatori,
- mantenere informazioni sulle esposizioni attualmente presenti nel museo.

Ogni stanza dispone di quattro uscite collegate da corridoi esterni e ciascun visitatore del museo è dotato di un dispositivo portatile, il quale contiene le sue preferenze. Un visitatore tramite l'interazione con i nodi infrastrutturali può essere guidato verso le camere in cui è presente un obiettivo (o target) che corrisponde ai suoi interessi. In particolare, il visitatore viene guidato attraverso dei segnali che appaiono in modo dinamico sul proprio smartphone. Per realizzare ciò si utilizzano delle tecniche proposte in ambito della spatial computing (sezione 2.4), in questo caso di specie i gradienti computazionali (sezione 2.7.1) si iniettano in una sorgente e si diffondono attorno ad essa in modo che ciascun nodo contenga la distanza minima da essa.

Modello

L'ambiente è costituito da nodi infrastrutturali, mentre gli smartphone sono agenti collegati dinamicamente con i sensori vicini all'interno di un certo raggio r. Tale parametro r del modello definisce il raggio da cui poter recuperare i dati al fine di suggerire ai visitatori dove andare. I visitatori sono anch'essi agenti e tendono a seguire i consigli del proprio dispositivo portatile. Essi possono muoversi per passi discreti all'interno dell'ambiente. Si è definita, inoltre, una distanza minima fra i visitatori, per modellare il limite fisico ed il fatto che due visitatori non possono essere situati nello stessa posizione contemporaneamente.

Nell'approccio SAPERE (sezione 2.2), tutte le informazioni scambiate sono in forma di LSA e le regole sono espresse in forma di eco-law (sezioni 2.2 e 2.2.1). Seguendo tale approccio, in questo scenario sono state introdotte tre forme di LSA:

$$\langle \texttt{source}, id, type, N_{max}, \pi, \mu, type' \rangle$$
$$\langle \texttt{field}, id, type, value, \pi, \mu, type', tstamp \rangle$$
$$\langle \texttt{pre\_field}, id, type, value, \pi, \mu, type', tstamp \rangle$$

Una source LSA viene usata come una sorgente, il cui scopo è di generare un campo: si utilizzano delle etichette *id* per identificare la sorgente in modo da distinguere sorgenti dello stesso tipo; *type* indica il tipo di campo (`target` viene utilizzato per pubblicizzare le esposizioni e `crowd` per diffondere informazioni sull'affollamento); N_{max} è il valore massimo del campo, π e μ sono due funzioni rispettivamente usate per calcolare il nuovo valore del campo dopo che è stato propagato o trasformato in base al valore di campo di un altro tipo (*type'*).

Un field LSA viene usato per i singoli valori in un gradiente: il valore indica il valore individuale; *tstamp* indica il momento della creazione della LSA, gli altri parametri hanno lo stesso significato descritto nella source LSA.

Un pre_field LSA viene usato per diffondere il campo prima che venga influenzato dalla regola di trasformazione.

$$\langle \texttt{source}, id, type, N_{max}, \pi, \mu, type' \rangle \xmapsto{r_{init}} \langle \texttt{source}, id, type, N_{max}, \pi, \mu, type' \rangle, \langle \texttt{field}, id, type, N_{max}, \pi, \mu, type', \#T \rangle$$

$$\langle \texttt{field}, id, type, N, \pi, \mu, type', t \rangle \xmapsto{r_{diff}} \langle \texttt{field}, id, type, N, \pi, \mu, type', t \rangle, +\langle \texttt{pre\_field}, id, type, \pi(N, \#D), \pi, \mu, type', t \rangle$$

$$\langle \texttt{pre\_field}, id, type, N, \pi, \mu, type', t \rangle, \langle \texttt{field}, id', type', M, \pi', \mu', Type, t' \rangle \mapsto \langle \texttt{field}, id, type, \mu(N, M), \pi, \mu, type', t \rangle, \langle \texttt{field}, id', type', M, \pi', \mu', Type, t' \rangle$$

$$\langle \texttt{field}, id, type, N, \pi, \mu, type', t \rangle, \langle \texttt{field}, id, type, M, \pi, \mu, type', t' + t \rangle \mapsto \langle \texttt{field}, id, type, M, \pi, \mu, type', t' + t \rangle$$

$$\langle \texttt{field}, id, type, N, \pi, \mu, type', t \rangle, \langle \texttt{field}, id, type, M, \pi, \mu, type', t \rangle \mapsto \langle \texttt{field}, id, type, max(M, N), \pi, \mu, type', t \rangle$$

$$\langle \texttt{field}, id, type, N, \pi, \mu, type', t \rangle, \langle \texttt{field}, id', type, N + M, \pi, \mu, type', t' \rangle \mapsto \langle \texttt{field}, id', type, N + M, \pi, \mu, type', t' \rangle$$

Figura 2.9: *Eco-law che descrivono l'applicazione del museo.*

Si modella l'esecuzione di una *eco-law* come una transizione CTMC con tasso Markoviano (frequenza media) r. Se non viene specificato un tasso di trigger della reazione viene considerato "as soon as possible" (ASAP), che significa che il tasso associato alla reazione tende all'infinito. Per consentire l'interazione tra i diversi LSA-space è stato utilizzato il concetto di *remote pattern*, descritto nella sezione 2.2.1.

Come le source LSA vengono iniettate nei nodi, i gradienti vengono costruiti dalle prime tre regole in figura 2.9. La prima *eco-law*, data una sorgente, inizializza un campo con il suo valore massimo. La seconda *eco-law*, quando un nodo contiene un field LSA, diffonde un pre_field LSA ad un nodo adiacente scelto casualmente con un nuovo valore calcolato secondo la funzione di propagazione, π, che elabora la distanza tra i sensori, indicata dalla variabile # D, e l'attuale valore del field LSA. La terza *eco-law*, quando un nodo contiene un LSA pre_field di tipo type e un LSA field di tipo type' da cui il pre_field dipende, rimuove il pre_field LSA e crea un nuovo field LSA con un valore calcolato in

accordo con la funzione di trasformazione, μ, che elabora i valori N e M dei due reagenti. Lo scopo di questa legge è quello di modellare le interazioni tra i campi. Per esempio possiamo supporre che se c'è una folla che blocca una regione del museo, il percorso che attraversa tale regione per raggiungere il target dovrebbe avere meno probabilità di essere scelto, questo viene effettuato, in pratica, riducendo il valore del campo target. Come conseguenza di tali leggi, ogni nodo porterà un field LSA che indica la distanza topologica dalla sorgente. Più vicino è il valore field a N_{max}, più vicina è la sorgente del campo. Quando i valori diffusi raggiungono il valore minimo di 0, il gradiente deve diventare un *plateau*. Per affrontare la dinamicità dello scenario in cui le persone si muovono, i target possono essere eventualmente spostati e la folla si forma e si dissolve, si è introdotto il meccanismo seguente. Ci si aspetta che se un sorgente del gradiente si muova e il valore diffuso cambi in base alla nuova posizione. Questo è lo scopo del parametro *tstamp* che viene utilizzato nella quarta *eco-law*, che aggiorna continuamente i valori vecchi con altri più recenti (youngest eco-law). In questo modo si garantisce che il sistema sia in grado di adattarsi ai cambiamenti di posizione della sorgente. Infine, la spreading eco-law descritta precedentemente può produrre nelle varie posizioni valori duplicati, a causa di sorgenti multiple dello stesso tipo (indicate con identificativi diversi), percorsi multipli per una sorgente, o addirittura la diffusione di LSA multipli nel tempo. Per questo motivo si sono introdotte le ultime due *eco-law*. Esse mantengono solo il valore massimo, cioè la distanza minima. La prima quando ci sono due LSA identici a parte un valore, la seconda quando l'identificativo è diverso (shortest eco-law). Le *eco-law* in figura 2.9 descrivono il comportamento dell'ecosistema museo in una visione orientata alla chimica e sono modellati in un simulatore, di conseguenza, come reazioni.

Si noti che le persone sono modellate come agenti. Si muovono in base al valore del campo calcolato in modo probabilistico per il loro target scegliendo il vicino con il valore del campo maggiore. Il comportamento dei visitatori è, anch'esso, modellato come una reazione, in particolare con un'azione speciale per esprimere il comportamento dei visitatori. L'architettura proposta è intrinsecamente in grado di adattarsi dinamicamente ad eventi imprevisti (come nodi guasti, isolamento della rete, *crowd formation* e così via) pur mantenendo le proprie funzionalità.

Risultati

Il comportamento di ogni nodo è stato programmato secondo il modello di coordinazione delle *eco-law* mostrato in figura 2.9. Ogni nodo presente nell'ambiente contiene di default per ogni tipo (`target` e `crowd`) nel sistema un LSA della forma:

$$\langle \texttt{field}, id, target, type, 0, \mu, type', 0 \rangle$$

Le sorgenti dei gradienti sono iniettate dai sensori quando un target o un numero di persone viene percepito, con i seguenti valori:

$$\langle \texttt{source}, id, \texttt{target}, \texttt{T}_{\texttt{max}}, \pi_t, \mu_t, \texttt{crowd} \rangle$$
$$\langle \texttt{source}, id, \texttt{target}, \texttt{C}_{\texttt{max}}, \pi_c, \mu_c, \texttt{crowd} \rangle$$

Per il primo tipo di sorgente si assume che si possano avere target diversi in base alle differenti preferenze degli utenti. Per la *crowding source* (sorgente di affollamento), invece, si può supporre che i sensori siano calibrati in modo da iniettare localmente un LSA che indica il livello di affollamento, cioè il numero di persone e viene periodicamente aggiornato. Le funzioni di propagazione e trasformazione presentano la seguente forma:

$$\begin{array}{rcccl} \pi_t & = & \pi_c & = & N_{t,c} - \#D \\ & & \mu_t & = & N_t - k * N_c \\ & & \mu_c & = & N_c \end{array}$$

dove $N_{t,c}$ sono i valori correnti dei due tipi di campi e k è un parametro di modello usato per modulare l'effetto che l'affollamento può avere sul campo target. Altri parametri sono: $T_{max} = 1000$, $C_{max} = 1$. Quando $\mu_t, \pi_{t,c} < 0$ si considera che il valore di tali parametri sia pari a zero.

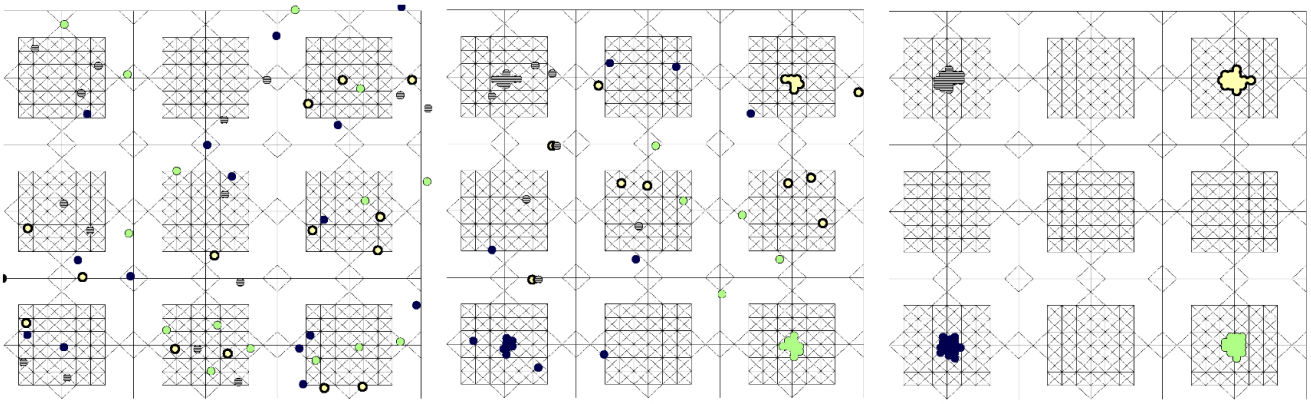


Figura 2.10: *Una esecuzione della simulazione dell'esposizione di riferimento. Immagine tratta da [15]*

I rate delle reazioni sono stati individuati a mano eseguendo diverse simulazioni con parametri diversi. I risultati riportati di seguito sono stati ottenuti con $r_{init} = 1$ e

$r_{diff} = 50$. Le altre leggi non indicano alcun rate perché è stato assunto essere infinito. Si sono presentate delle simulazioni condotte su una esposizione, con nove stanze connesse da corridoi. Quattro snapshot di un'esecuzione della prima simulazione sono riportate in figura 2.10. Si sono presi in considerazione quattro diversi target posizionati in quattro stanze del museo vicine agli angoli dell'ambiente. Le persone sono inizialmente diffuse nel museo, sono mostrate nel primo snapshot ed alla fine raggiungono la stanza in cui è collocato il target desiderato, come mostrato nell'ultimo snapshot.

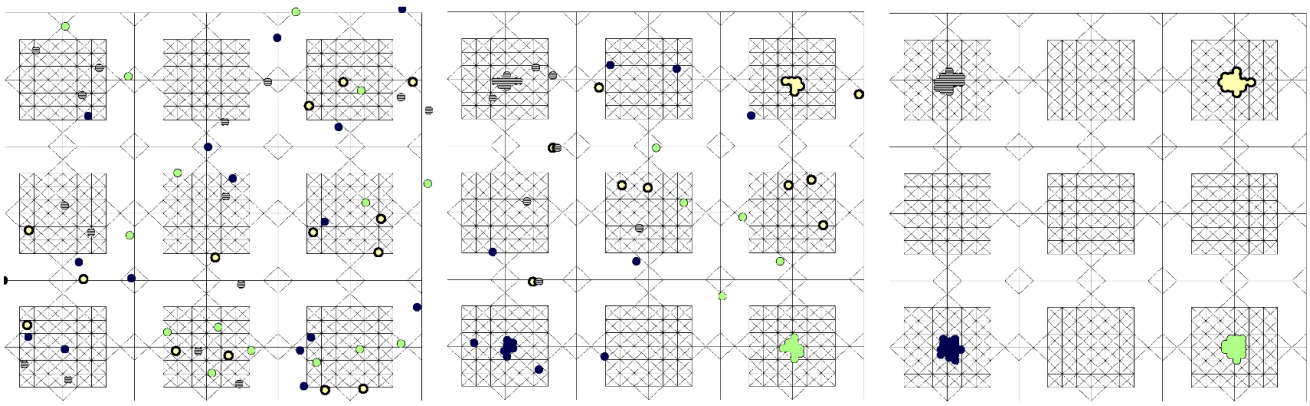


Figura 2.11: *Una esecuzione che mostra l'effetto del crowding (affollamento): i visitatori scuri occupano una stanza centrale, provocando lo spostamento a sinistra degli altri visitatori da destra così da effettuare un percorso più lungo ma meno affollato. Immagine tratta da [15]*

La figura 2.11 mostra una simulazione eseguita con l'effetto di affollamento nel movimento delle persone. Due gruppi di persone, indicate con cerchi vuoti e pieni, aventi interessi comuni sono state inizialmente situate in due stanze diverse, come mostrato nel primo snapshot. L'obiettivo per i visitatori scuri si trova nella stanza centrale della seconda riga, mentre per gli altri è nella stanza a destra della seconda riga. Nella simulazione, i visitatori scuri raggiungono presto il loro target essendo molto vicini, e formano una zona affollata che interseca il percorso più breve verso il tagert degli altri visitatori. A causa di questo ingorgo gli altri visitatori scelgono un percorso diverso che è più lungo ma meno affollato.

3

Un framework di simulazione per gli ecosistemi di servizi pervasivi

Il primo scopo di questo capitolo è di mettere in luce un elemento fondamentale dell'ingegneria dei sistemi pervasivi la simulazione. In passato, gli approcci coinvolgevano la simulazione solamente dopo che lo sviluppo del software era già terminato: in sostanza le simulazioni erano impiegate per fornire una caratteristica statica delle prestazioni. Negli ultimi anni, invece, si è promosso il suo utilizzo fin dai primi stadi dello sviluppo del software, vista la forte potenzialità di permette di osservare l'andamento qualitativo delle dinamiche del sistema. La simulazione, quindi, è uno strumento fondamentale per analizzare il comportamento del sistema prima del *deployment*, ed è un approccio obbligatorio nei sistemi che richiedono di sviluppare un comportamento emergente.

Per sottolineare l'importanza della simulazione nello sviluppo del software, in questo capitolo si introduce il framework di simulazione utilizzato in questa tesi come supporto per la realizzazione dei nuovi algoritmi presentati nel capitolo precedente. Nel dettaglio, nella prima sezione si descrivono le motivazioni della scelta di Alchemist come framework di simulazione, mentre nelle successive si entrerà nel dettaglio delle caratteristiche di tale framework, fino ad arrivare alle implementazioni nel DSL per `SAPERE` degli algoritmi.

3.1 Motivazioni

Oggi diversi approcci sono stati proposti in ambito dei modelli di coordinazione e di middleware per gli scenari di *pervasive computing*. Questi cercano di raccogliere caratteristiche legate alla località, coordinamento spontaneo ed opportunistico, auto-adattamento ed auto-organizzazione, tuttavia solitamente si propongono soluzioni ad-hoc per problemi specifici, mancando di generalità.

Nel contesto del progetto `SAPERE` (sezione 2.2), la simulazione può essere utile per supportare la progettazione delle *eco-law*, del comportamento dell'agente e dell'intero ecosistema di servizi pervasivi. Si dà la possibilità di effettuare esperimenti utilizzando

meccanismi ecologici, tra i quali quelli ispirati alla biologia, e di mostrare attraverso la simulazione il comportamento generale di un sistema progettato sulla base delle *eco-law*, oltre agli scenari *what-if*. Un framework ben progettato, inoltre, permetterà ai ricercatori di analizzare in modo formale le proprietà di tali sistemi pervasivi attraverso lo *stochastic model checking*. In questo modo si riesce a catturare tutta la complessità dell'approccio `SAPERE`. In particolare, il modello deve risultare coerente con le seguenti astrazioni:

- ambienti altamente dinamici composti da nodi diversi, mobili e comunicanti;
- comportamenti reattivi espressi da reazioni *chemical-like* oltre a LSA;
- comportamenti di agenti autonomi.

Da un lato l'adozione dello standard *Agent-Based Models* (ABM [10]) sembra essere del tutto naturale, in quanto il sistema pervasivo è stato progettato adottando il paradigma ad agenti ed affidandosi ad una forma di interazione mediata. Esistono difatti diversi lavori che applicano questo approccio in contesti diversi, dai sistemi sociali ai sistemi biologici. In generale, un ABM verte sugli agenti autonomi e possibilmente eterogenei che possono essere situati in un ambiente. Essi percorrono la linea d'azione più opportuna, eventualmente interagendo con altri agenti o con l'ambiente. Il comportamento dell'agente è modellato tramite un insieme di regole che descrivono come deve rispondere alle condizioni dell'ambiente. Tali regole possono essere di diversi tipi, a seconda del modello specifico: a partire da semplici regole reattive, che specificano come l'agente deve reagire a stimoli o percezioni ambientali, e proattive, che specificano come l'agente si deve comportare in base ai propri obiettivi e compiti. L'ambiente nell'ABM è una astrazione di prima classe, la cui struttura, topologia e dinamica possono essere modellate in modo esplicito.

Da un altro lato, l'ABM non fornisce strumenti per la progettazione sofisticata di regole del comportamento nell'ambiente. Tuttavia, in questo campo pochi simulatori permettono di definire in modo flessibile le topologie di rete. In paticolare, la maggior parte di essi gestiscono una singola sostanza chimica con pochi compartimenti, e non si forniscono funzioni per la mobilità della rete e di controllo sul comportamento diverso dei diversi nodi, in quanto questo contesto sfugge ai sistemi biologici. Inoltre, le *eco-law* non si adattano esattamente alle reazioni chimiche, perché gestiscono molecole strutturate e algoritmi di matching avanzati in un modo che i simulatori chimici esistenti difficilmente possono affrontare.

Per prendere il meglio di entrambi gli approcci è stato sviluppato un nuovo framework di simulazione, chiamato Alchemist, il quale intende affrontare in modo nativo i requisiti sopra esposti. In particolare, si implementa una versione ottimizzata dello SSA di Gillespie, chiamato Next Reaction Method [8], esteso con la possibilità di eseguire reazioni dinamiche, ovvero reazioni che possono essere aggiunte o rimosse in fase di simulazione grazie alla mobilità della rete e adottando la semantica del linguaggio delle *eco-law*.

3.2 Alchemist

Il framework, chiamato ALCHEMIST (`http://alchemist-maven.apice.unibo.it/`), è stato sviluppato dall'Ing. Pianini, all'interno del gruppo di ricerca della Seconda Facoltà di Ingegneria nell'ambito del progetto SAPERE. Scopo del framework è di affrontare in modo nativo le astrazioni della metafora bio-chimica. Il modello ed il framework di simulazione che si presentano risultano essere abbastanza generici e, come tali, possono avere una vasta gamma di applicazioni: *pervasive computing*, *computational biology* ed interazioni sociali.

Questa sezione, dunque, ha lo scopo di mostrare le caratteristiche di ALCHEMIST: nella prima sottosezione si introduce il modello computazionale, poi si descrive l'architettura del framework, mostrando nel dettaglio il motore di simulazione ed il DSL per SAPERE.

3.2.1 Modello computazionale

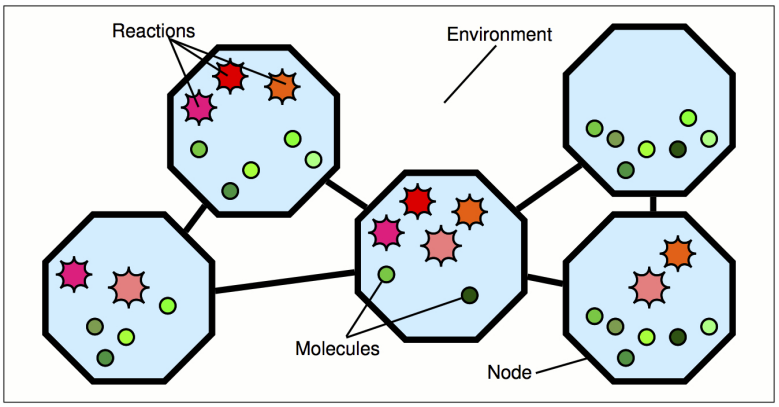


Figura 3.1: *Modello computazionale di* ALCHEMIST*: dotato di uno spazio possibilmente continuo con regole di collegamento e nodi che le contengono. Ogni nodo è programmato con una serie di reazioni e contiene un insieme di molecole strutturate. Immagine tratta da [13]*

La figura 3.1 rappresenta il modello computazionale di base. In questa semplice visione del mondo, un ambiente è uno spazio multidimensionale, continuo o discreto, in grado di contenere i nodi e responsabile di collegarli in base ad una regola. L'ambiente può o non può permettere ai nodi di muoversi. I nodi sono entità che possono essere programmate attraverso una serie di reazioni possibilmente che cambiano nel tempo. Essi contengono anche molecole, ognuna dotata di un valore di concentrazione.

Il concetto di reazione usato nel simulatore è lo stesso descritto (figura 3.2). Questo consente, ad esempio, di modellare le reazioni che sono più veloci, se un nodo ha molti vicini, o anche reazioni analoghe a complessi fenomeni biologici. Consente, inoltre, di definire quale tipo di distribuzione temporale viene usata per innescare le reazioni: questo permette di modellare e simulare sistemi basati sulle catene di Markov a tempo continuo (CTMCs), per aggiungere *trigger* o anche per utilizzare la visione classica di tempo discreto tramite i "*tick*".

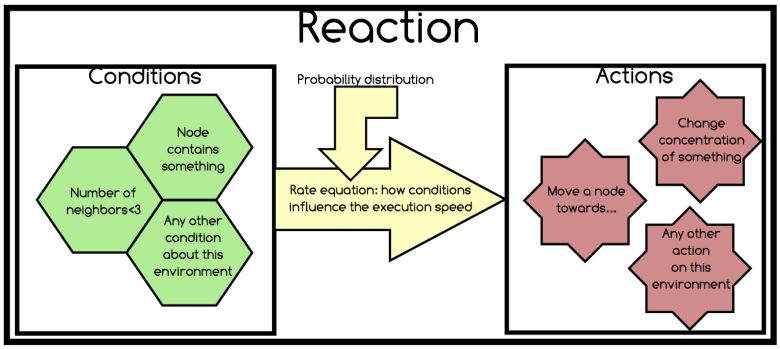


Figura 3.2: ALCHEMIST *modello della reazione: un insieme di condizioni sull'ambiente che determina se la reazione è eseguibile, un tasso (rate equation) che descrive quanto velocemente la reazione cambia in risposta alle perturbazioni ambientali, una distribuzione di probabilità per l'evento e un insieme di azioni, le quali saranno l'effetto dell'esecuzione della reazione. Immagine tratta da [13]*

Prendendo come riferimento il modello presentato in 2.2, si mappano nel dettaglio i concetti di `SAPERE` in ALCHEMIST di nodo e ambiente.

Nodo

In accordo con la metafora bio-chimica, lo stato del nodo nel modello si presenta come un insieme di molecole, mentre il comportamento autonomo interno attraverso un insieme di reazioni di ispirazione chimica. Dato uno stato di un nodo solo un sottoinsieme delle reazioni chimiche è applicabile. Il risultato dell'esecuzione della reazione è un'azione (o insieme di reazioni) che può modificare lo stato del sistema, ovvero la configurazione delle molecole. Le molecole sono una sorta di modelli di dati, ciascuno descritto da un valore di concentrazione e da una serie di proprietà dato che lo stato del sistema può essere complesso ed articolato. Tali modelli vengono mappati nell'approccio `SAPERE` nelle **annotazioni**. Il concetto di reazioni è più elaborato di quello utilizzato in chimica: nei modelli chimici classici, una reazione elenca un numero di molecole reagenti le quali, combinate, producono un insieme di molecole prodotti. Questo tipo di descrizione è troppo

ristretta per lo scopo che si vuole raggiungere. Un concetto più generale è considerare una **reazione** come un insieme di condizioni sullo stato del sistema, le quali eseguono un *trigger* per l'esecuzione di un insieme di azioni. Una condizione è una funzione che associa un valore booleano allo stato corrente del sistema (o ad una sottoparte di esso), una azione è, invece, una procedura che modifica le annotazioni che la rappresentano. Le condizioni possono spaziare dal caso classico di una lista di annotazioni che devono essere disponibili per la reazione in modo che possa essere eseguita, al numero di vicini. Le azioni possono trasformare, rimuovere o produrre annotazioni, movimenti o duplicazioni di agenti e così via. Infine, la velocità di una reazione classica viene descritta dalla funzione di **propensity**, la quale è fissata e dipende solo dalla concentrazione dei reagenti. La *propensity* consente una maggiore flessibilità: essa è in funzione del tasso della reazione, le condizioni e lo stato dell'ambiente. La comunicazione, che avviene tramite lo scambio di annotazioni, tra nodi è permessa solo attraverso i vicini.

Ambiente

L'ambiente, possibilmente continuo, è responsabile dei collegamenti fra i nodi e di definire il vicinato; il concetto di **vicinato** è di natura fisica ovvero di nodi all'interno di un raggio. Un ruolo chiave dell'ambiente è di supportare la diffusione delle annotazioni, la quale dovrebbero essere svolta da regole *chemical-like* adeguate.

3.2.2 Architettura

L'intero framework è stato progettato per essere pienamente modulare ed estendibile (figura 3.3). L'intero motore o parti di esso possono essere re-implementate senza modificare niente nel modello, in altre parole il modello può essere esteso e modificato senza toccare il motore.

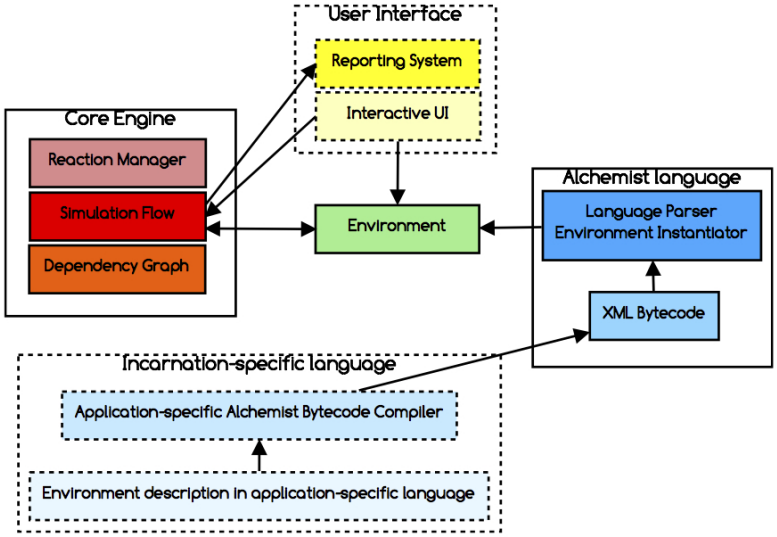


Figura 3.3: *Architettura di* ALCHEMIST. *Gli elementi disegnati con le linee continue indicano componenti comuni ad ogni scenario o già sviluppati, quelli con le linee tratteggiate sono componenti di specifiche estensioni le quali devono essere sviluppate con in mente una specifica incarnazione. Immagine tratta da [13]*

È importante notare che non c'è nessuna restrizione sul genere della struttura dati che rappresenta la concentrazione, essa può, infatti, essere usata per modellare un'informazione strutturata definendo un nuovo tipo di struttura per la concentrazione, questo è possibile per incarnare il simulatore per usi specifici differenti. Per esempio, se si definisce la concentrazione come un numero intero, rappresentante il numero di molecole correntemente presenti in un nodo, ALCHEMIST diventerebbe un simulatore stocastico chimico. Un esempio più complesso potrebbe essere la definizione di concentrazione come un *tuple set*, e la definizione di molecole come *tuple template*. Se si adottasse questa visione ALCHEMIST potrebbe essere un simulatore per una rete di spazi di tuple. Ogni volta che una nuova definizione di concentrazione e molecola viene determinata, una nuova "**incarnazione**" di ALCHEMIST viene automaticamente stabilita. Per ogni incarnazione, un insieme di specifiche azioni, condizioni, reazioni e nodi possono essere definiti e tutte le entità già definite per un tipo di concentrazione più generica possono essere ri-usate.

3.2.3 Motore di simulazione

In ALCHEMIST una simulazione viene modellata come un'entità autonoma dotata di proprio flusso di controllo, la quale può essere avviata e fermata ed, inoltre, consente di accedere all'ambiente a runtime.

La struttura `IReaction Manager` sarà quella responsabile della selezione della prossima reazione da eseguire. La sua implementazione corrente si basa sull'algoritmo `SSA Next Reaction`, una versione ottimizzata di *First Reaction* che prevede l'uso di due strutture dati, il Dependency Graph (DG) e l'Indexed Priority Queue (IPQ).

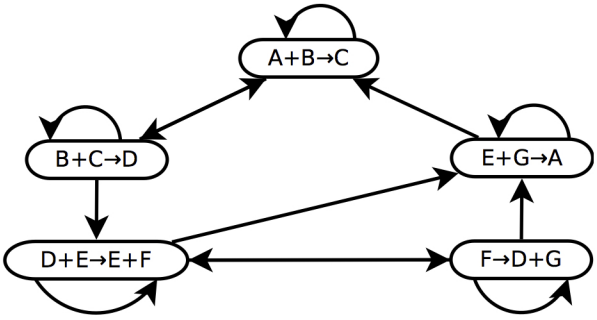
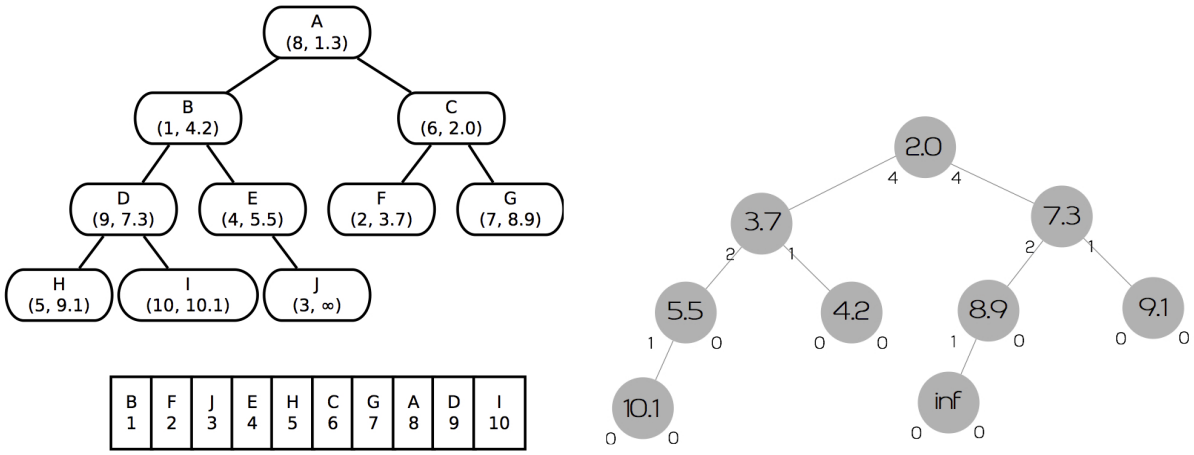


Figura 3.4: *Esempio di Dependency Graph (DG)*

Il DG (figura 3.4) avrà il compito di selezionare quali reazioni saranno effettivamente aggiornate ad ogni step della simulazioni e si appoggerà all'`IReactionHandler`, una struttura che lega all'entità di reazione utili dettagli implementativi. Il grafo si basa su relazioni esistenti fra le condizioni e le azioni delle reazioni nel sistema; per ogni nuova reazione verranno ricavate quali reazioni influenza e da quali risultati viene influenzata e le varie dipendenze verranno memorizzate nel DG stesso, evitando così aggiornamenti inutili di reazioni indipendenti. L'IPQ (figura 3.5) consiste in un albero i cui nodi contengono una coppia (i,τ_i) in cui i è il riferimento alla reazione e τ_i è il suo tempo putativo. La proprietà principale dell'albero è dovuta al fatto che ciascun nodo ha tempo inferiore a quello di tutti i suoi figli. Inoltre, a causa della dinamicità dei sistemi sarà necessario tenere traccia anche del numero di figli per ogni singolo nodo, al fine di mantenere un bilanciamento dell'albero. Un IPQ prevede operazioni per la costruzione, per lo scambio di nodi (*swap*) e per l'*update* di nodi il cui valore è stato modificato. Il motore di simulazione è stato sviluppato come modulo quasi del tutto indipendente dal resto del modello (deve solo conoscere la struttura dell'ambiente, il tempo e le reazioni), questo offre la possibilità di estendere il simulatore senza preoccuparsi del motore o, in alternativa, implementare diversi motori, sfruttando altri algoritmi, da intercambiare in funzione delle caratteristiche richieste nello specifico scenario da simulare.

3.2.4 DSL per `SAPERE`

Un Domain Specific Language per `SAPERE` è stato sviluppato. Esso è stato realizzato utilizzando il framework Xtext e come conseguenza un prodotto Eclipse dotato di: *syntax*



(a) IPQ. Immagine tratta da [7]

(b) IPQ esteso con il conteggio del numero di figli per ramo. Immagine tratta da [13]

Figura 3.5: *Esempi di Indexed Priority Queue (IPQ)*

highlighting, *code suggestion* ed *automatic code generation*. Per attivare queste funzionalità, occorre lanciare l'ambiente Eclipse generato dall'Xtext e creare un nuovo file con estensione "alsap".

Ogni file definisce un ambiente, ovvero uno scenario di simulazioni Gli elementi che possono essere definiti sono:

- ambiente: `environment` *`params`*
- LSA: `lsa` *`molecule`*
- nodi: `place node in` *`geometry`*
- reazioni: `[`*`conditions`*`] -`*`rate`*`-> [`*`actions`*`]`

La parola chiave `environment` segnala l'inizio della specifica. Essa può essere seguita da vari parametri tra cui:

- *`name`*: che indica il nome dell'ambiente ed è un parametro opzionale;
- `of type AnEnvironment params ''`*`some,parameters,2`*`''`: per inizializzare un IEnvironment la cui classe Java è AnEnvironment. Se questa linea non viene specificata il sistema attribuisce come tipo di default Continuous2DEuclideanDistanceAutolink, ambiente in cui lo spazio è continuo ed i nodi sono collegati fra loro se la loro distanza è all'interno di un determinato intervallo. Se, invece, si specifica, ad esempio, `environment of type InfiniteHalls params ''8''`, si costruisce un ambiente caricando la classe InfiniteHalls e passando come parametro 8 (quest'ultimo valore rappresenta la dimensione della hall);

- `with random seed` ***number***: è opzionale e di default è un numero casuale. Può essere utile se si vuole forzare la riproducibilità del proprio esperimento.

Le LSA in ALCHEMIST sono delle tuple piane, come in Linda. In una riga con `lsa` si effettua una dichiarazione, in particolare si assegna un nome alla LSA che segue. Ad esempio, con `lsa source` $\langle source, gradient0, 0\rangle$ si dichiara una LSA con nome *source* e che contiene la molecola $\langle source, gradient0, 0\rangle$.

Il linguaggio dispone di funzionalità per creare un singolo nodo in un punto specifico dello spazio o più nodi simili insieme in una zona rettangolare o circolare. Ad esempio, si analizzano i token presenti in `place node at point` $(0, 0)$ `containing anLSA with reaction` $[\,] - - > [\,]$:

- `place node`: crea un singolo nodo;
- `at point` $(0, 0)$: è il punto nello spazio bidimensionale dove il nodo verrà situato;
- `containing anLSA`: è possibile specifica quali LSA dovranno essere contenuti nel nodo. É possibile riferirsi sia ad un LSA pattern definito in precedenza o ad un nuovo LSA;
- `with reaction`: dopo questo token inizia la definizione delle reazioni.

Per creare più nodi è possibile specificare, ad esempio, `place 10 nodes in rect` $(0, 0, 2, 3)$ `interval 1`, questa specifica pone 10 nodi in una regione rettangolare il cui punto più in basso a sinistra è $(0, 0)$ e il più alto a destra è $(2, 3)$. Il token `interval` è opzionale se non specificato i nodi vengono posizionati in modo casuale nella regione, viceversa se specificato i nodi sono posizionati con la distanza fissata.

Le *eco-law* possono essere definite come `eco-law lawName` $[] - - > []$:

- `eco-law lawName`: parola chiave opzionale che permette di specificare il nome della *eco-law*;
- $[\,] - - > [\,]$: questo è il cuore della reazione. Dentro le parentesi di sinistra vengono specificate le condizioni, mentre dentro quelle di destra le azioni. Il rate può essere specificato all'interno della freccia. Se nessun rate è specificato $(- - >)$ significa che la reazione è ASAP.

Le condizioni e le azioni sono, rispettivamente, cosa abilita la reazione e cosa produce quando viene eseguita. Le reazioni operano in modo molto simile alle reazioni chimiche, rimuovendo dal LSA space quello che è definito nella parentesi a sinistra ed aggiungendo ciò che è definito a destra. Ad esempio:

- $[definedLSA] - - > [\langle L, 5, a\rangle]$, `definedLSA` verrà rimossa, mentre $\langle L, 5, a\rangle$ verrà aggiunta;

- $[\langle fiel, Val, Orientation\rangle] - - > [* \langle field, Val + \#D, \#O\rangle]$, il carattere * davanti ad una azione significa the viene eseguita su tutto il vicinato. Questo è il metodo standard per implementare le diffusioni. # D e # O sono dei valori speciali validi solo nelle reazioni che coinvolgono il vicinato. Il primo misura la distanza fra il nodo e il vicino corrente, il secondo è un vettore (nella forma "$[X, Y]$") che punta al nodo corrente dal vicino;

- $[\ ] - - > [agentAgentClassparams"ENV, NODE, REACTION, RANDOM]$ specifica l'azione di caricare una classe Java. Questo è particolarmente utile per definire agenti mobili. Implementare un agente si riduce ad implementare una classe Java che estende LsaAbstractAction. Quando si specifica una classe, i valori `ENV, NODE, REACTION` e `RANDOM` possono essere usati per riferirsi rispettivamente a: l'ambiente corrente, il nodo corrente, la reazione corrente e il corrente RandomEngine.

La descrizione eseguita ha avuto l'obiettivo di analizzare i token principali, per una definizione completa del linguaggio si faccia riferimento al manuale.

4
Gradiente NBR

Scopo di questa tesi è di studiare e sviluppare algoritmi di auto-organizzazione basati su meccanismi ispirati ai sistemi ecologici. Per realizzare ciò si è utilizzato un approccio iterativo ed evolutivo, le cui fasi sono state:

- **modellazione**: proporre un modello per il sistema sulla base dei design pattern esistenti;
- **simulazione**: analisi delle dinamiche in diversi scenari;
- **tuning**: regolare il comportamento del sistema ed elaborare un insieme di parametri per il sistema.

Tale lavoro ha portato all'introduzione dei seguenti algoritmi:

- gradiente NBR (capitolo 4):
  - di base (sezione 5.3.2);
  - con risparmio energetico (sezione 4.5);
- algoritmo di *join* (capitolo 5), calcolo baricentro e ricongiungimento.

4.1 Motivazioni

L'approccio con cui si genera il gradiente mostrato nella sezione 2.8 (il DSL relativo è stato inserito nell'appendice A) crea dei casi di glitch, ovvero delle fluttuazioni nei valori del campo che rappresentano il gradiente.

Un caso di glitch si presenta quando la prima catena di nodi che collega in modo ordinato la sorgente ed un nodo non è la più corta. In modo più formale, per la figura 4.1, si ha che, dato per scontato lo scatto iniziale del nodo S0, il tempo in cui scatta S1 è maggiore di quello in cui scattano S2 e S3:

$$p\left(e_{S1} > \sum_{i=S2}^{S3} e_i\right)$$

Questo è causato dal fatto che ogni nodo ha la stessa probabilità di scattare in ogni istante di tempo, indipendentemente dagli scatti passati (sistema *memoryless*), di conseguenza si ha che non è possibile stabilire a priori quali sarà la sequenza di scatti dei nodi. In realtà, il percorso più corto risulta essere anche il più probabile, ma in percentuale si ha che maggiore è la distanza fra i due nodi, più la probabilità di percorre per prima la strada più breve diminuisce.

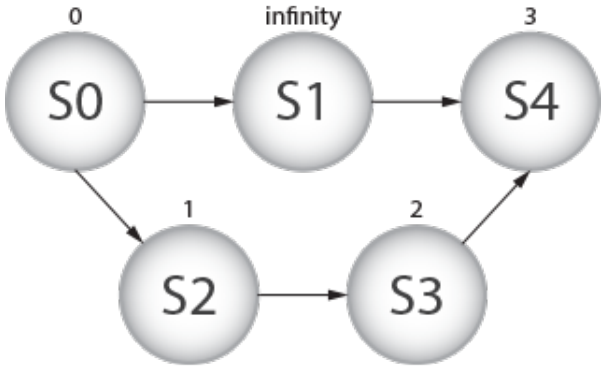
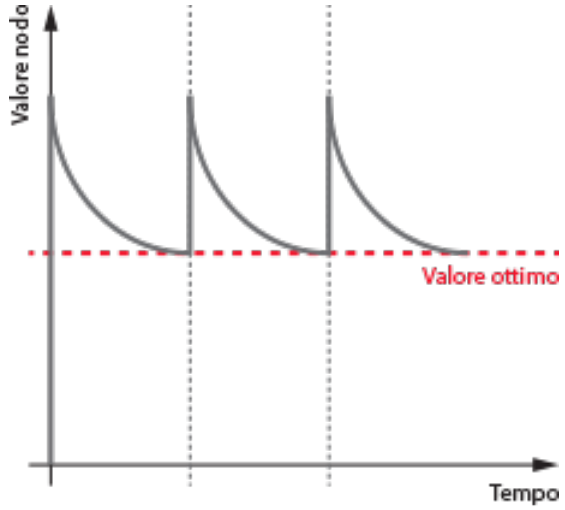
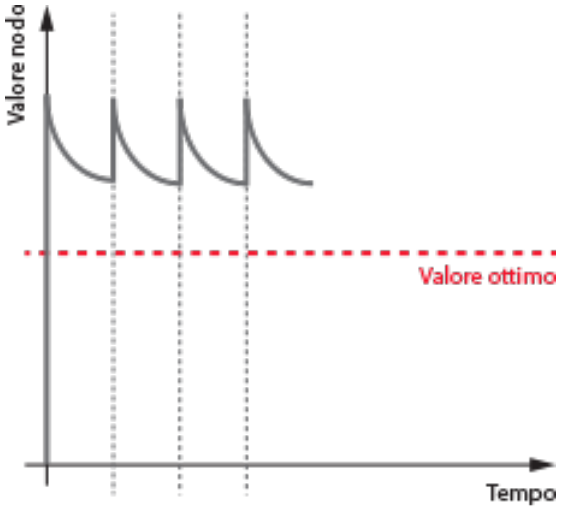


Figura 4.1: *S0 rappresenta la sorgente, mentre S4 rappresenta il nodo di cui si vuole campionare il valore.*

Un altro caso di glitch riguarda l'assestamento del valore del nodo. Una caratteristica del gradiente `SAPERE` è che per essere reattivo a perturbazioni ambientali vengono eseguite più *pump* dalla sorgente. Ogni *pump* della sorgente produce un'ondata di nuovi valori che sovrascrivono i valori dei campi correnti. Se venisse eseguita solo una *pump* si avrebbe solo il primo glitch di stabilizzazione. L'esecuzione di più *pump*, invece, provoca che quando si verificano delle perturbazioni, il nodo richiede un certo tempo per assestarsi e delle volte nell'assestarsi assume dei valori non corretti. Può verificarsi, ad esempio, che il nodo inizialmente ha un tempo di reazione, poi assume prima un valore non ottimo e dopo quello ottimo, poi gli arrivano i valori dell'ondata successiva che provocano la sovrascrittura dei valori facendo sì che il valore fluttui nel tempo. Inoltre, si ha che se le *pump* vengono eseguite ad una distanza troppo breve nel tempo, il gradiente non riesce mai a stabilizzarsi. Nel dettaglio si ha che il nodo non riceve il valore ottimo della *pump* precedente che già gli arrivano i valori di quella successiva, così che il valore cavalca sempre il glitch. Riassumendo, se il rate della *pump* è sufficientemente basso il gradiente riesce a stabilizzarsi prima della nuova ondata, viceversa se è troppo alto il gradiente non riesce mai a stabilizzarsi (figura 4.2). Tenere un rate basso però significa avere una rete meno reattiva ai cambiamenti, quindi è necessario definire nel sistema un giusto *trade-off* fra stabilità e reattività.

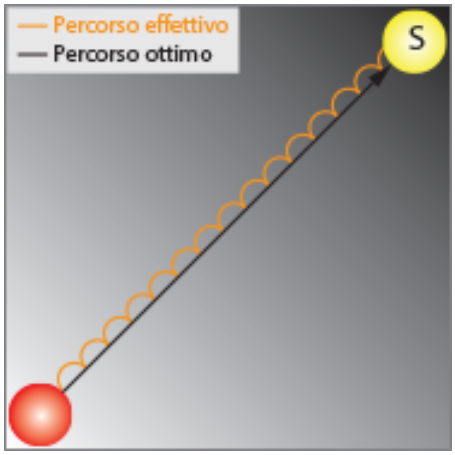


(a) Rate dell'eco-law di pump che permette al nodo di raggiungere il valore ottimo

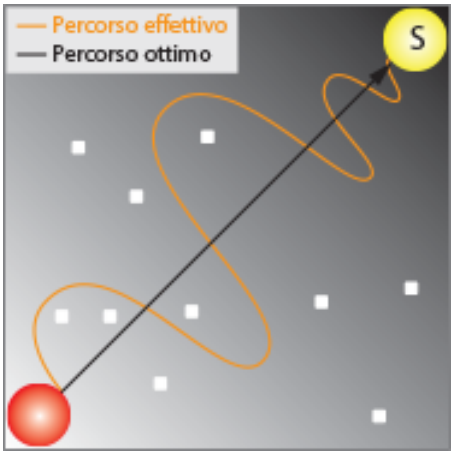


(b) Rate dell'eco-law di pump che non permette al nodo di raggiungere il valore ottimo

Figura 4.2: *Il rate dell'eco-law di pump risulta essere un parametro fondamentale per raggiungere la stabilità del sistema*



(a) La stabilità del gradiente fa sì che un nodo riesce a risalirlo seguendo il percorso ottimo



(b) La non stabilità del gradiente computazionale, dovuta ad un rate dell'eco-law di pump troppo elavato, provoca che il nodo per raggiungere la sorgente non segue il percorso ottimo

Figura 4.3: *La sorgente, nodo giallo, genera il gradiente computazionale i cui valori sono mostrati sulla scala di grigio. Un nodo, disegnato di rosso, deve raggiungere la sorgente risalendo il suo gradiente. Nella figura 4.3(b) i quadratini bianchi indicano che in quel nodo il valore del gradiente assume il valore infinity, ovvero non si è ancora verificata una catena di scatti di nodi che parte dalla sorgente che lo raggiungono*

Il gradiente NBR risulta essere molto più stabile rispetto ai casi di glitch dato che la sorgente esegue la *pump* solo una volta, ma nel contempo riesce a rispondere alle perturbazioni ambientali.

4.2 Ambiente di simulazione

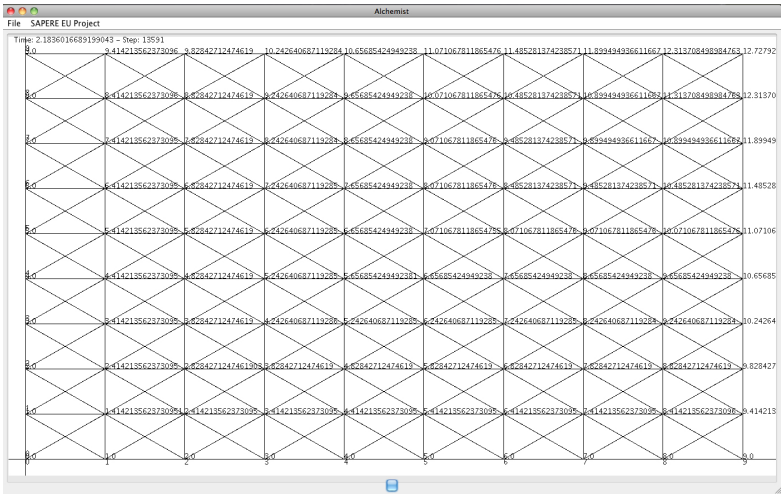


Figura 4.4: *Ambiente virtuale per le simulazioni*

L'ambiente utilizzato sarà quello in figura 4.4, cioè un ambiente costituito da 100 sensori posizionati a distanza omogenea ed unitaria fra loro. Sopra ad ogni nodo si indica il corrispondente valore del gradiente in esame. Al fine di avere un'alta precisione nel valore del nodo è stato definito come un numero con quindici cifre decimali. Per la rappresentazione grafica del gradiente il valore ∞ , equivalente all'assenza di gradiente, è stato semplificato con 100. Di conseguenza i valori non potranno crescere sopra tale soglia.

Una nota riguarda l'assegnamento dell'identificativo di ogni nodo. Nella disposizione dei nodi scelta, si ha che gli *id* dei nodi aumentano di 10 ad ogni riga, nella prima riga ai sensori saranno attribuiti, dunque, gli *id* da 0 a 9, nella seconda da 10 a 19 e così via. Di particolare interesse nella successiva analisi saranno i nodi in posizione (0,0) il cui *id* è lo 0, ed il nodo in posizione (9,9) il cui *id* è il 99.

Nell'ambiente si noti che i nodi vicini ad un certo nodo in esame possono essere al massimo 8, dato che la rappresentazione del vicinato è nella forma cosiddetta "ad asterisco". Ad esempio, nel caso in cui il nodo in esame sia in posizione (2,1) (*id* 12), il suo vicinato sarà composto dai nodi: (1,0) con id 1, (2,0) con id 2, (3,0) con id 3, (1,1) con id 11, (3,1) con id 13, (1,2) con id 21, (2,2) con id 22 e (2,3) con id 23.

4.3 Simulazioni gradiente SAPERE

In questa sezione si propone un'analisi del comportamento del gradiente SAPERE tramite l'esecuzione di simulazioni effettuate su ALCHEMIST, nell'ambiente di simulazione mostrato in figura 4.4. L'analisi è riferita a due aspetti:

- i glitch;
- le performance.

4.3.1 Analisi glitch

Per prima cosa è stata effettuata l'analisi dei glitch, in particolare si fornisce il valore dei seguenti parametri:

- numero di glitch (variazione rispetto al valore corretto);
- ampiezza dei glitch;
- periodo del glitch (quanto tempo rimane sul valore non corretto);
- percentuale del tempo di "valore non corretto" sulla durata della simulazione.

Sui valori ottenuti per questi parametri si è calcolata la media e la deviazione standard.

L'analisi seguente è stata effettuata sul modello inserito nell'appendice B. Si noti che il modello proposto in queste simulazioni, rispetto a quello inserito in appendice A, gestisce la *pump* in modo diverso. In particolare, la *pump* è composta da due *eco-law*. La prima ha rate ASAP e viene effettuata solo la prima volta, tutte le volte successive viene effettuata la seconda che ha il rate scelto. Questa scelta è motivata dal fatto che bisognava rendere confrontabili le diverse simulazioni. ALCHEMIST , descritto nel capitolo 3, si basa sul SSA di Gillespie e questo fa sì che la prima reazione di *pump* possa avere tempi di scatto molto diversi. In particolare, si ha che minore è il rate, maggiore è tale variabilità. Tale varianza diventa meno significativa se vengono effettuati dei *long run*. Nella seguente analisi, invece, è inaccettabile, poiché influisce sul tempo in cui la simulazione "inizia realmente", ovvero in cui vi è il *firing* della prima reazione, rispetto al tempo totale della simulazione, alterando, di conseguenza, il numero di glitch che si verificheranno.

L'analisi dei parametri indicati si svolge attraverso lo studio rapporto fra la frequenza di *pump* e di *spread*. In particolare, nelle simulazioni si è mantenuto fisso il rate della *eco-law spread* a 10 e si è variato il rate di quella di *pump*. Per ogni diverso rate della *eco-law* di *pump* sono state effettuate 20 simulazioni, in modo da avere un campione significativo. Nella tabella 4.1 si mostrano i risultati con sei diversi rapporti fra il rate

(a) 0.001

Parametro	Media	Deviazione standard
Numero glitch	2.85	1.31
Ampiezza	13.1	3.03
Periodo	1.26	0.83
% tempo valore non corretto	0.44	0.28

(b) 0.01

Parametro	Media	Deviazione standard
Numero glitch	27.8	6.82
Ampiezza	13.72	0.1
Periodo	12.86	3.25
% tempo valore non corretto	4.3	1.08

(c) 0.1

Parametro	Media	Deviazione standard
Numero glitch	180.2	9.53
Ampiezza	13.82	0.04
Periodo	99.3	1.78
% tempo valore non corretto	33.16	1.78

(d) 1

Parametro	Media	Deviazione standard
Numero glitch	205.6	9.62
Ampiezza	14.04	0.04
Periodo	249.6	3.09
% tempo valore non corretto	83.19	1.03

(e) 10

Parametro	Media	Deviazione standard
Numero glitch	161.9	10.55
Ampiezza	14.15	0.03
% tempo valore non corretto	87.88	1.18

(f) 100

Parametro	Media	Deviazione standard
Numero glitch	167.7	10.84
Ampiezza	14.16	0.04
% tempo valore non corretto	87.44	0.99

Tabella 4.1: *Analisi dei parametri di glitch del sistema al nodo in posizione (9,9) sulla base del rapporto fra il rate della eco-law di pump e di spread. Ogni sottotabella indica i parametri per uno specifico rapporto*

(a) 0.001

Parametro	Media	Deviazione standard
Numero glitch	1.95	1.12
Ampiezza	4.75	1.65
Periodo	0.28	0.26
% tempo valore non corretto	0.09	0.09

(b) 0.01

Parametro	Media	Deviazione standard
Numero glitch	18.45	5.05
Ampiezza	5.12	0.16
Periodo	2.67	0.95
% tempo valore non corretto	0.89	0.32

(c) 0.1

Parametro	Media	Deviazione standard
Numero glitch	159.4	10.54
Ampiezza	5.14	0.03
Periodo	23.98	2.17
% tempo valore non corretto	7.99	0.77

(d) 1

Parametro	Media	Deviazione standard
Numero glitch	583.6	14.48
Ampiezza	5.17	0.02
Periodo	91.85	3.75
% tempo valore non corretto	30.62	1.25

Tabella 4.2: *Analisi dei parametri di glitch del sistema al nodo con in posizione (3,3) sulla base del rapporto fra il rate della eco-law di pump e di spread. Ogni sottotabella indica i parametri per uno specifico rapporto*

della *eco-law* di *spread* e quella di *pump* separata di un ordine di grandezza, da 0.001 a 100.

Si noti che il numero di glitch diventa un dato più significativo se confrontato con il numero di *pump*. In linea di principio, per ogni nuova *pump* si crea un glitch. Date le frequenze scelte il numero di *pump* atteso è 3 per 0.01, 30 per 0.1 e 300 per 1 e così via. In tutti e sei i casi però si ha che il numero di *glitch* è inferiore. Tale risultato è giustificato dal fatto che si siano generati dei *glitch* complessi. Il concetto di *glitch* complesso indica dei *glitch* che non sono *spike*, ovvero non ritornano subito al valore ottimo, ma formano un "gradino", assumo cioè un valore intermedio prima di quello ottimo. Questo si verifica perché mentre si ha un *glitch* non ancora risolto, arriva il *glitch* successivo e i due si uniscono in un *glitch* unico. Nel caso in cui venissero solamente calcolate il numero di *pump*, si perderebbero i *glitch* complessi. La tabella 4.1 deve essere analizzata proprio sulla base di tale considerazione.

Il parametri del numero di glitch se studiato senza la percentuale di tempo in cui il sistema ha un valore scorretto può essere fuorviante. Si noti, infatti, che aumentando il rapporto fra i rate delle *eco-law* di *pump* e di *spread* il numero di glitch ad un certo punto diminuisce. Si guardi il numero di glitch nella sottotabella associata al rapporto 10, che risulta essere inferiore rispetto a associata al rapporto 1. Questo è giustificato dal fatto che nel caso del rapporto 10 si formano in totale un numero inferiore di glitch, ma essi hanno un periodo più lungo in quanto risultano essere glitch complessi. Tale riflessione trova conferma sia nella percentuale di tempo in cui il sistema risulta essere scorretto sia nell'ampiezza dei glitch, entrambi i valori aumentano in quanto si formano i cosiddetti glitch complessi poiché al sistema arriva la nuova ondata di valori dalla *eco-law* di *pump* prima che sia riuscito a processare tutti i valori della *pump* precedente.

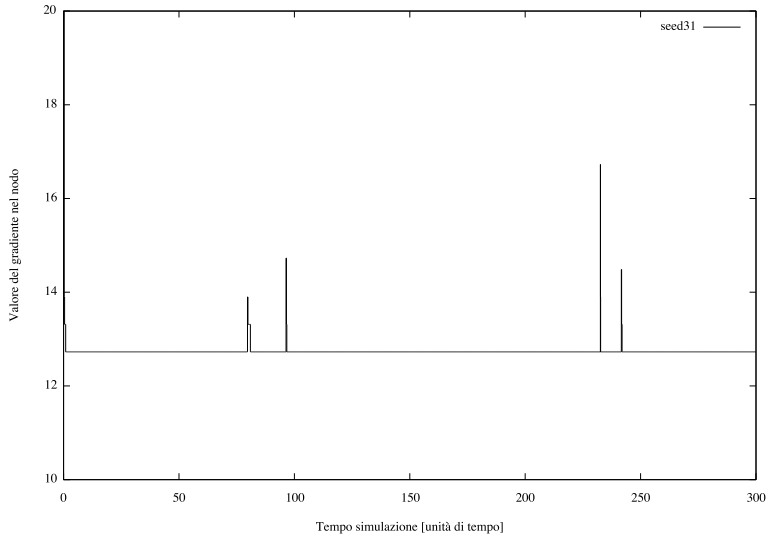


Figura 4.5: *Glitch presenti nel nodo in posizione (9,9) con rapporto fra i rate pari a 0.001. Si noti che anche con un rapporto così basso si creano dei glitch complessi. In particolare, si verifica che il primo glitch risulta essere, ben visibile anche dall'immagine, di tipo complesso*

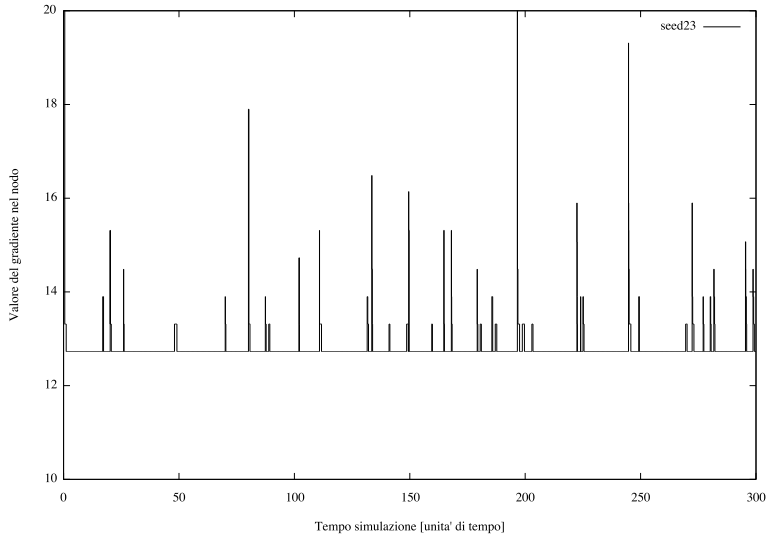


Figura 4.6: *Glitch presenti nel nodo in posizione (9,9) con rapporto fra i rate pari a 0.01*

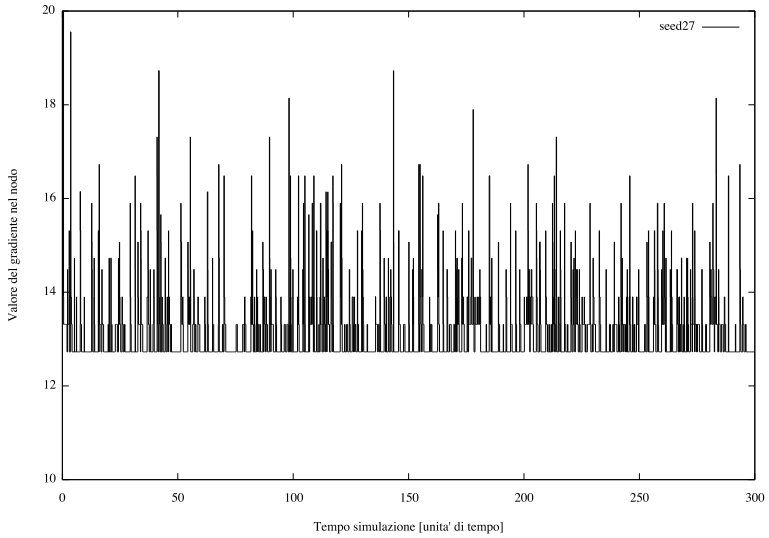


Figura 4.7: *Glitch presenti nel nodo in posizione (9,9) con rapporto fra i pump rate pari a 0.1*

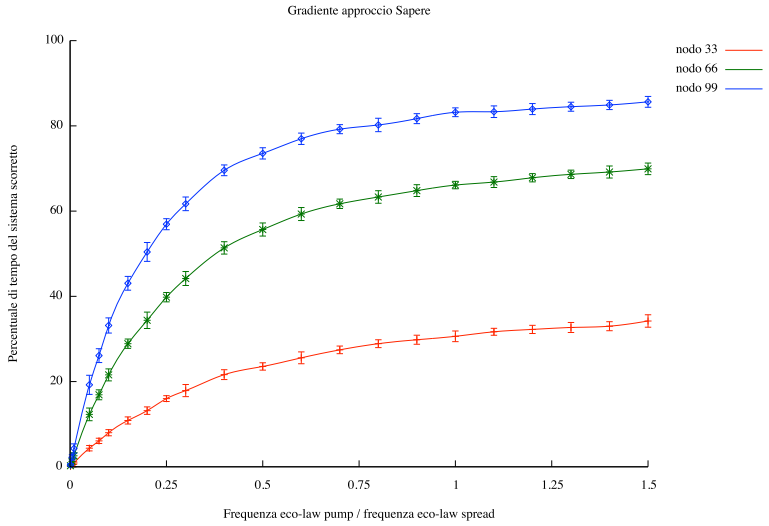


Figura 4.8: *Variare della % di tempo in cui il sistema è scorretto alla modifica del rapporto fra i rate dell'eco-law di pump e di spread*

Nel grafico 4.8 si mostra che nell'approccio SAPERE all'aumentare del rapporto fra la frequenza di *firing* dell'*eco-law* di *pump* e di *spread*, fa crescere, di conseguenza, la

percentuale di tempo, nei nodi, in cui il sistema è scorretto. Le barrette indicano per ogni campione di simulazioni il valore medio e la deviazione standard ottenuti. Si noti che più il nodo è lontano dalla sorgente del gradiente, più è soggetto a glitch in quanto il percorrere la strada ottima per raggiungerlo diventa sempre meno probabile.

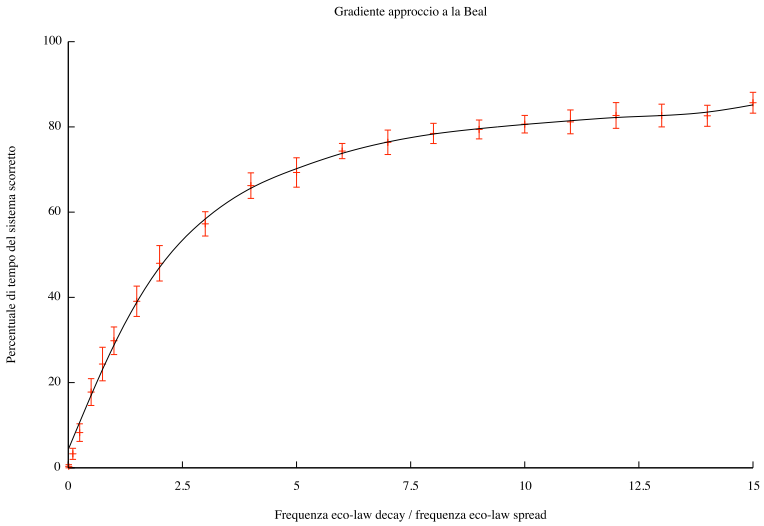


Figura 4.9: *Variare della % di tempo in cui il sistema è scorretto alla modifica del rapporto fra i rate dell'eco-law di decay e di spread*

Nel grafico 4.9 si mostra che nell'approccio `NBR` all'aumentare del rapporto fra la frequenza di *firing* dell'*eco-law* di *decay* e di *spread*, fa crescere, di conseguenza, la percentuale di tempo, nel nodo in posizione (9,9), in cui il sistema è scorretto. Le barrette rosse indicano per ogni campione di simulazioni il valore medio e la deviazione standard ottenuti.

4.3.2 Analisi performance

Per l'analisi delle performance, invece, si fornisce il valore dei seguenti parametri:

- tempo impiega il gradiente per costituirsi;
- numero di reazioni che vengono eseguite nell'unità di tempo.

Dai risultati emersi presentati in tabella 4.3 si conclude che il numero di reazioni scattate in un nodo risulta essere poco influenzato dal *rate* della *eco-law* di *pump*.

Per studiare la stabilità, si assume che il tempo in cui arrivi un valore al nodo in posizione (9,9) implica che l'intero gradiente computazionale si sia generato correttamente con l'approccio `SAPERE` sono stati ottenuti i risultati mostrati in figura .

Rapporto Rate	Media	Deviazione standard
0.001	2,348,039.75	4,070.46
0.01	2,349,895.5	3,536.61
0.1	2,349,816.3	4,205.71
1	2,354,314.45	4,909.86

Tabella 4.3: *Numero reazioni scattate nel nodo in posizione (3,3) con una durata della simulazione pari a 300 unità di tempo al variare del rapporto fra il rate della eco-law di pump e di spread*

(a) 0.001

Parametro	Media	Deviazione standard
Tempo	0.33	0.25
Numero reazioni	450.55	917.95

(b) 0.01

Parametro	Media	Deviazione standard
Tempo	0.33	0.25
Numero reazioni	450.55	917.95

(c) 0.1

Parametro	Media	Deviazione standard
Tempo	0.36	0.25
Numero reazioni	741	1261.32

(d) 1

Parametro	Media	Deviazione standard
Tempo	0.36	0.19
Numero reazioni	540.3	656.05

Tabella 4.4: *Analisi della stabilità del sistema al nodo con in posizione (3,3) sulla base del rapporto fra il rate della eco-law di pump e di spread. Ogni sottotabella indica i parametri per uno specifico rapporto*

Parametro	Media	Deviazione standard
Tempo	0.94	0.42
Numero reazioni	4,786.05	2,345.06

Tabella 4.5: *Stabilità gradiente SAPERE con rate dell'eco-law di spread pari a 10*

4.4 Gradiente NBR

In questa sezione si introduce il gradiente computazionale di base progettato da Jacob Beal e Jonathan Bachrach, le cui caratteristiche sono:

- ogni nodo non deve considerare il proprio valore corrente per il calcolo di quello successivo;
- la sorgente diffonde solamente una volta il gradiente, saranno poi i nodi che a loro volta diffonderanno il proprio valore al vicinato;
- presenza di un'alta asincronia negli scatti dei nodi. Non si può assumere che quando un nodo diffonde il proprio valore abbia ricevuto il messaggio da tutti i vicini perché ognuno di essi computa alla propria velocità (*agent-speed*).

La prima caratteristica risulta essere fondamentale, in quanto permette di tenere aggiornato il valore del nodo. Ogni volta che arriva un nuovo valore in un nodo, lo si sostituisce sempre a quello esistente, perché risulta essere l'informazione più recente che ho ricevuto.

Per realizzare ciò si è pensato a due possibili strategie:

1. la prima necessita di quattro tipi di LSA;
2. la seconda necessita di tre tipi di LSA e di un contatore, interno ad ogni nodo, dei messaggi inviati.

Prima Strategia

La prima strategia, come anticipato, necessita di quattro tipi di LSA, i quali sono indicati in figura 4.10.

$\langle \texttt{msg}, type, node, distance \rangle$
$\langle \texttt{msgread}, type, node, distance \rangle$
$\langle \texttt{field}, type, value \rangle$
$\langle \texttt{tempfield}, type, list, value \rangle$

Figura 4.10: *LSA necessari per modellare la prima strategia*

Un msg LSA rappresenta un messaggio spedito o ricevuto da un nodo, l'etichetta *type* indica il gradiente a cui si riferisce il messaggio (ogni sorgente produce un gradiente di tipo diverso), *node* indica il nodo che ha inviato o ricevuto il messaggio, mentre *distance* indica la distanza del nodo dalla sorgente del gradiente. Un msgread LSA rappresenta un messaggio letto, le etichette hanno lo stesso significato come nel msg LSA. Un field

LSA rappresenta il campo del gradiente, l'etichetta *value* indica il valore del campo in quel nodo, le altre etichette hanno lo stesso significato come nel msg LSA. Un tempfield LSA rappresenta il campo che si sta computando, l'etichetta *list* contiene la lista dei nodi i cui valori sono stati sommati, le altre etichette hanno lo stesso significa delle LSA precedenti. La coppia delle LSA field e tempfield fa sì che il field sia visto come il risultato di un'operazione che unisce i valori dei tempfield.

Le figure 4.11 e 4.12 presentano gli LSA presenti inizialmente rispettivamente in ogni nodo non sorgente o sorgente.

$\langle \texttt{field}, type, 0, infinity \rangle$
$\langle \texttt{tempfield}, type, [], infinity \rangle$
$\langle \texttt{id}, name \rangle$

Figura 4.11: *LSA contenuti inizialmente in ogni nodo non sorgente del gradiente NBR*

$\langle \texttt{source}, type \rangle$

Figura 4.12: *LSA contenuti inizialmente in ogni nodo sorgente del gradiente NBR*

Si è assunto per semplicità che il valore *infinity*, equivalente all'assenza di gradiente, sia pari a 100. Un id LSA rappresenta l'identificativo del nodo, l'etichetta *name* indica il suo nome, il quale deve risultare univoco in modo tale ogni nodo sia distinguibile dagli altri. In particolare, si ha che ogni nodo spedisce ai suoi vicini il messaggio con il proprio valore, in questo modo ogni nodo ha la rappresentazione del proprio vicinato. Un source LSA rappresenta l'identificativo di una sorgente, l'etichetta *type* indica il suo tipo. Ogni sorgente possiede il proprio tipo, in modo da poter distinguere i gradienti formati da sorgenti diverse.

La soluzione mostrata in figura 4.13 necessita di un concetto non presente nell'approccio SAPERE, ovvero la non presenza di un LSA in un nodo (rappresentata con !), per questo motivo si è scelto di realizzare la seconda strategia.

Seconda Strategia

La seconda strategia, come anticipato, necessita di tre tipi di LSA, i quali sono indicati in figura 4.15.

In questa soluzione rispetto alla precedente si introduce l'etichetta *tstamp* che indica il numero del messaggio inviato dal nodo, in modo che possa essere sempre determinato l'ultimo messaggio inviato, e non vi è più la msgread LSA.

Nelle figure 4.16 e 4.17 sono state inserite le leggi per la seconda strategia.

$$\langle \texttt{msg}, type, node, dist\rangle, !\langle \texttt{msgread}, type, node, dist'\rangle \mapsto \langle \texttt{msgread}, type, node, dist\rangle$$

$$\langle \texttt{msg}, type, node, dist\rangle, \langle \texttt{msgread}, type, node, dist'\rangle \mapsto \langle \texttt{msgread}, type, node, dist\rangle$$

$$\langle \texttt{msgread}, type, node, dist\rangle, \langle \texttt{tempfield}, type, list, dist'\rangle \mapsto \langle \texttt{msgread}, type, node, dist\rangle, \langle \texttt{tempfield}, type, [node|list], min(dist, dist')\rangle \; if \; node \notin list$$

$$\langle \texttt{msgread}, type, node, dist\rangle \xmapsto{r_{decay}} \langle\,\rangle \; if \; node \neq source$$

$$\langle \texttt{tempfield}, type, list, dist\rangle, \langle \texttt{field}, type, dist'\rangle, \langle \texttt{id}, name\rangle \xmapsto{r_{field}} \langle \texttt{tempfield}, type, [], 100\rangle, \langle \texttt{field}, type, name, dist\rangle, *\langle \texttt{msg}, type, name, dist + \#D\rangle, \langle \texttt{id}, name\rangle \; if \; list \neq empty$$

Figura 4.13: *Eco-law che descrivono la prima strategia in un nodo non sorgente*

$$\langle \texttt{source}, type\rangle \xmapsto{r_{pump}} *\langle \texttt{msg}, type, source, \#D\rangle$$

$$\langle \texttt{msgread}, type, node, list, dist\rangle \xmapsto{r_{discard}} \langle\,\rangle$$

Figura 4.14: *Eco-law che descrivono la prima strategia in un nodo sorgente*

$$\langle \texttt{msg}, type, tstamp, node, distance\rangle$$
$$\langle \texttt{field}, type, tstamp, value\rangle$$
$$\langle \texttt{tempfield}, type, list, value\rangle$$

Figura 4.15: *LSA necessari per modellare la seconda strategia*

Come le annotazioni della sorgente sono iniettate dal nodo sorgente nei nodi del suo vicinato (prima legge in figura 4.16), i gradienti sono costruiti dalle quattro leggi in figura 4.17. La prima dato un messaggio di un nodo di cui ancora non è stato registrato un valore, lo si registra e si assegna al valore del campo temporaneo il valore minore fra quello corrente ed il nodo inserito. Si noti che il messaggio non viene cancellato così da ottenere una cache dei messaggi ricevuti. La seconda dati due messaggi provenienti dallo stesso nodo ma con tempi diversi, tiene solo quello più recente, ovvero con *timestamp* superiore (legge di *youngest*). La terza elimina un messaggio, in modo che un nodo possa accorgersi della caduta di un link. In particolare, solamente attraverso una esplicita eliminazione del messaggio il nodo si può accorgere del link caduto, mantenendo la cache dei messaggi, difatti, potrebbe continuare a conservare un vecchio messaggio. Un'altra nota riguarda il messaggio ricevuto dalla sorgente. Dato che la sorgente esegue solo una volta la *pump* non è possibile eliminare il suo messaggio ed, inoltre, si noti che si sta computando

$$\langle \texttt{msg}, type, tstamp, node, dist\rangle, \langle \texttt{tempfield}, type, list, dist'\rangle \mapsto \langle \texttt{tempfield}, type, [node|list], min(dist, dist')\rangle, \langle \texttt{msg}, type, tstamp, node, dist\rangle \; if \; node \notin list$$

$$\langle \texttt{msg}, type, tstamp, node, dist\rangle, \langle \texttt{msg}, type, tstamp', node, dist'\rangle \mapsto \langle \texttt{msg}, type, tstamp', node, dist'\rangle \; if \; tstamp' > tstamp$$

$$\langle \texttt{msg}, type, tstamp, node, dist\rangle \xmapsto{r_{decay}} \langle\,\rangle \; if \; node \neq source$$

$$\langle \texttt{tempfield}, type, list, dist\rangle, \langle \texttt{field}, type, tstamp, dist'\rangle, \langle \texttt{id}, name\rangle \xmapsto{r_{field}} \langle \texttt{tempfield}, type, [], 100\rangle, \langle \texttt{id}, name\rangle, \langle \texttt{field}, type, tstamp + 1, name, dist\rangle, *\langle \texttt{msg}, type, tstamp, dist + \#D\rangle \; if \; list \neq empty$$

Figura 4.16: *Eco-law che descrivono la seconda strategia in un nodo non sorgente*

$$\langle \texttt{source}, type\rangle \mapsto *\langle \texttt{msg}, type, source, \#D\rangle$$

$$\langle \texttt{msg}, type, tstamp, node, dist\rangle \xmapsto{r_{discard}} \langle\,\rangle$$

Figura 4.17: *Eco-law che descrivono la seconda strategia in un nodo sorgente*

il gradiente di una sorgente e se questa viene meno, calcolare il suo gradiente sarebbe impossibile. La quinta legge assegna al nodo il valore del campo temporaneo al campo, svuota il campo temporaneo e diffonde il valore del campo ai nodi vicini, il parametro # D misura la distanza fra i due nodi. La seconda legge in figura 4.17 semplicemente cancella tutti i messaggi ricevuti, nel nodo sorgente, infatti, non interessa calcolare il valore del campo, dato che si assume che sia sempre zero.

La strategia proposta è intrinsecamente capace di adattarsi dinamicamente ad eventi inaspettati (come il fallimento di un nodo) mantenendo le proprie funzionalità.

In appendice C è stata inserita l'implementazione nel DSL per SAPERE.

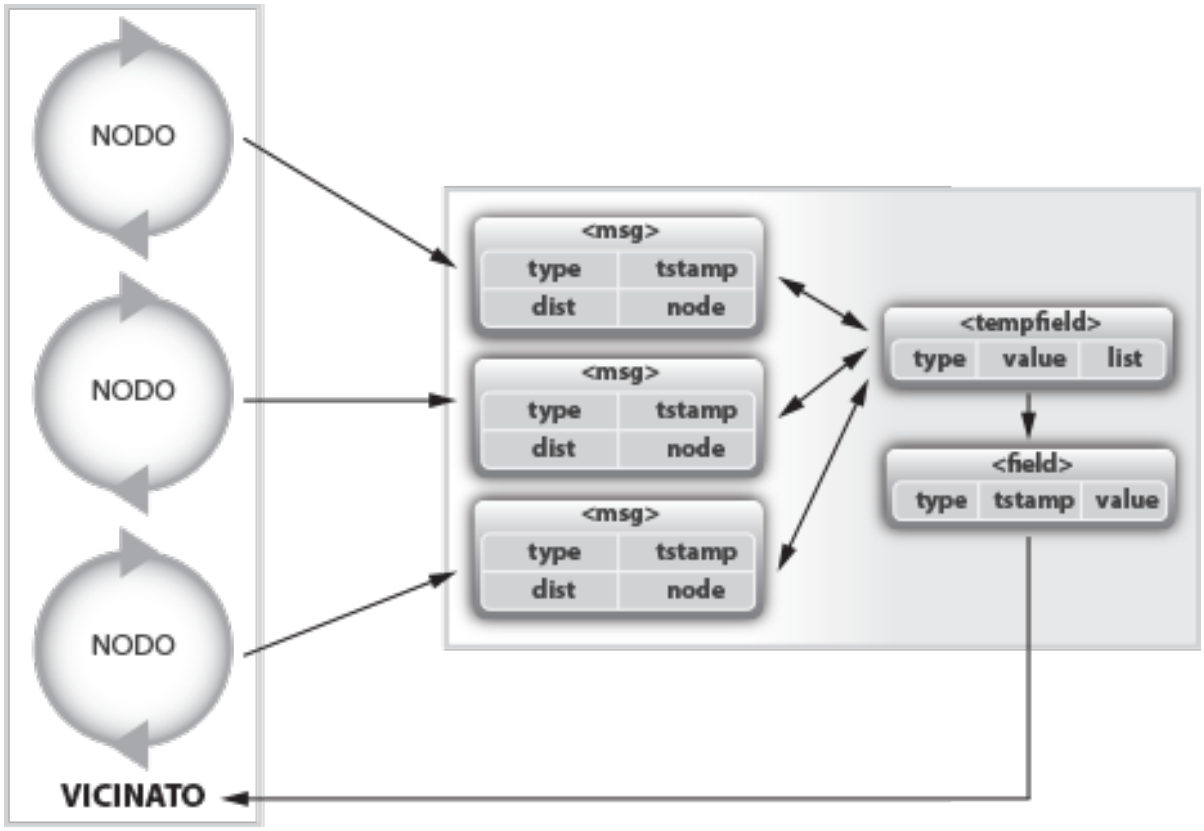


Figura 4.18: *Strategia 2*

4.5 Gradiente NBR con risparmio di energia

Si è visto che l'implementazione della comunicazione nel vicinato è la chiave nel bilanciamento fra reattività e stabilità. In particolare, quando la situazione del sistema risulta essere stabile, si può pensare di ridurre le trasmissioni in modo da ottenere un risparmio energetico nei singoli nodi. Le trasmissioni sono inviate durante le computazioni per ridurre il disallineamento nel tasso con cui l'informazione viene propagata. Quando non ci sono cambiamenti, il nodo, di conseguenza, rallenta dimezzando ogni volta il proprio rate di trasmissione fino ad un limite minimo fissato. Quando l'informazione del vicinato cambia di nuovo, il nodo ritorna alla frequenza di trasmissione di pieno regime.

Questo approccio necessita di tre nuovi tipi di LSA:

Un rate LSA viene usato come indicatore del rate corrente di un nodo: si utilizzano le etichette *name* per identificare il nodo e *value* per il valore del rate. Le LSA rateMin e rateMax stabiliscono rispettivamente il rate minimo e massimo che può raggiungere un nodo. Si noti che, di conseguenza, il valore del rate nella LSA rate sarà sempre un valore

$\langle \texttt{rate}, name, value \rangle$
$\langle \texttt{rateMin}, value \rangle$
$\langle \texttt{rateMax}, value \rangle$

compreso, estremi inclusi, fra quelli indicati nelle altre due LSA.

$$\langle \texttt{tempfield}, type, list, dist \rangle, \langle \texttt{field}, type, tstamp, dist \rangle, \langle \texttt{id}, name \rangle, \langle \texttt{rateMin}, rMin \rangle, \langle \texttt{rate}, name, r \rangle \xmapsto{r} \langle \texttt{tempfield}, type, [\,], 100 \rangle, \langle \texttt{field}, type, tstamp + 1, dist \rangle, *\langle \texttt{msg}, type, tstamp, name, dist + \#D \rangle, \langle \texttt{id}, name \rangle, \langle \texttt{rate}, name, r/2 \rangle, \langle \texttt{rateMin}, rMin \rangle \; if\ list \neq empty\ and\ r\ > rMin$$

$$\langle \texttt{tempfield}, type, list, dist \rangle, \langle \texttt{field}, type, tstamp, dist \rangle, \langle \texttt{id}, name \rangle, \langle \texttt{rateMin}, rMin \rangle, \langle \texttt{rate}, name, r \rangle \xmapsto{r} \langle \texttt{tempfield}, type, [\,], 100 \rangle, \langle \texttt{field}, type, tstamp + 1, dist \rangle, *\langle \texttt{msg}, type, tstamp, name, dist + \#D \rangle, \langle \texttt{id}, name \rangle, \langle \texttt{rate}, name, rMin \rangle, \langle \texttt{rateMin}, rMin \rangle \; if\ list \neq empty\ and\ r\ <= rMin$$

$$\langle \texttt{tempfield}, type, list, dist \rangle, \langle \texttt{field}, type, tstamp, dist' \rangle, \langle \texttt{id}, name \rangle, \langle \texttt{rateMax}, rMax \rangle, \langle \texttt{rate}, name, r \rangle \xmapsto{r} \langle \texttt{tempfield}, type, [\,], 100 \rangle, \langle \texttt{field}, type, tstamp + 1, dist \rangle, *\langle \texttt{msg}, type, tstamp, name, dist + \#D \rangle, \langle \texttt{id}, name \rangle, \langle \texttt{rate}, name, rMax \rangle, \langle \texttt{rateMax}, rMax \rangle \; if\ list \neq empty\ and\ r\ <= rMax\ and\ dist \neq dist'$$

Figura 4.19: *Eco-law che descrivono il risparmio energetico*

In figura 4.19 sono state inserite le leggi per il risparmio energetico.

La prima legge dimezza il rate del nodo e si applica nel caso in cui il valore del campo temporaneo e del campo sono uguali e il rate corrente del nodo è superiore al valore minimo di scatto. La seconda mantiene il rate al valore minimo e si applica nel caso in cui il valore del campo temporaneo e del campo sono uguali e il rate corrente del nodo è uguale (o inferiore) al valore minimo di scatto. La terza riporta o mantiene il rate del nodo al valore massimo e si applica nel caso in cui il valore del campo temporaneo e del campo siano diversi.

In appendice D è stata inserita l'implementazione nel DSL per `SAPERE`.

4.6 Sistema di reportistica

Nel procedere con l'esecuzione dell'analisi è nata la necessità di creare un sistema di reportistica atto a fornire documentazione sulle simulazioni e capace di rispondere alle esigenze dello scenario applicativo. L'obiettivo è di analizzare il numero di glitch e le performance di un nodo posto all'interno di una griglia di sensori (figura 4.4).

Tale sistema si sviluppa in due parti:

- la prima, prende in ingresso il file *xml* (generato in modo automatico in base al file *alsap*, come descritto nella sottosezione 3.2.4) e produce in uscita un file *dat* contenente il risultato dettagliato della simulazione;
- la seconda, prende in ingresso il file *dat* creato nel passo precedente e produce i file di report a vari livelli di aggregazione.

Si noti che tutti i risultati delle simulazioni introdotte in questo capitolo sono stati realizzati usufruendo di tale sistema. Il sistema è stato progettato in modo che possa essere facilmente espandibile e semplice da utilizzare. Per ulteriori dettagli, si faccia riferimento all'appendice F.

5
Baricentro e ricongiungimento

Dopo aver introdotto, nel capitolo 4, i concetti chiave del gradiente computazionale NBR ed analizzate le sue caratteristiche, ora si entra nel dettaglio di un problema specifico che utilizza un algoritmo di coordinamento di un gruppo di persone, dotate di PDA, distribuite in un ambiente. In particolare, si presentano alcuni scenari in cui si mostra l'utilità dell'applicazione di tale algoritmo *gradient-based* nell'ambito della *pervasive computing* rispetto ad altri approcci. Tale algoritmo è stato validato attraverso l'esecuzione di simulazioni, eseguite sul framework Alchemist (capitolo 3), descritte nel dettaglio di seguito.

5.1 Scenario reale

L'obiettivo è riunire delle persone distribuite nello spazio nel punto baricentrico rispetto alle loro posizioni iniziali. Ad esempio, si pensi ai visitatori di in un museo, che nell'attendere la guida iniziano a visitarlo in modo sparso. Nel momento in cui la guida arriva si colloca in un punto stabilito del museo ed inizia ad emettere un fischio così da permettere ai visitatori di riconosce la direzione in cui muoversi per raggiungerla. Una volta che tutte le persone sono radunate, la guida potrà iniziare l'itinerario di visita del museo. Dal punto di vista dell'Intelligenza Artificiale, si noti che determinare il baricentro di un sistema è un *pattern formation* molto importante, in quanto identifica un punto di riferimento ragionevole da cui iniziare le successive attività per la realizzazione della forma desiderata.

Per raggiungere tale obiettivo l'ambiente in cui sono collocate le persone è coperto da sensori, disposti a griglia, capaci di rilevare la loro presenza, interagire sia con gli altri sensori posti nelle vicinanze, sia con i PDA che le persone portano con sé. In particolare, la realizzazione di tale obiettivo presuppone l'esecuzione di una serie di passi:

- individuare la posizione di ogni utente;
- determinare il punto baricentrico rispetto alla posizioni degli utenti;
- far ricongiungere gli utenti di raggiungere in quel determinato punto.

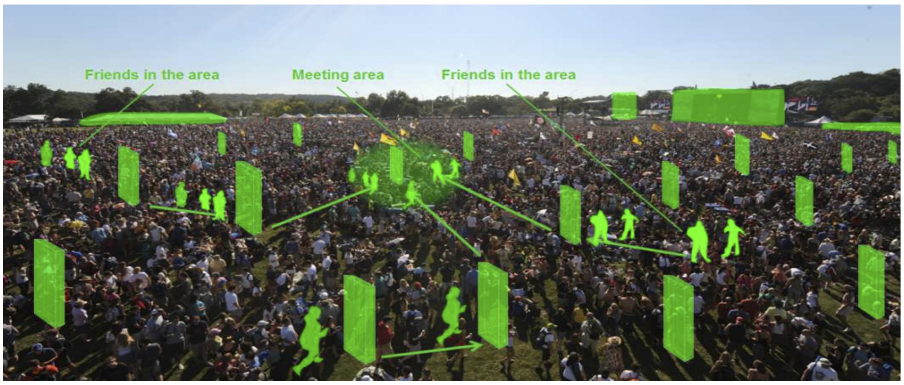


Figura 5.1: *Guida di folle (crowd steering). Immagine tratta da [14]*

Nello svolgimento di tali passi occorre tenere in considerazione la struttura dell'ambiente e le zone di affollamento.

5.2 Modello

Si presenta il modello per lo scenario del baricentro e ricongiungimento. L'obiettivo di questo scenario è di ricongiungere un gruppo di persone nel punto dell'ambiente eletto come baricentro rispetto alle loro posizioni, seguendo, possibilmente, il percorso più breve e più veloce il quale viene computato dinamicamente in base alla forma dell'ambiente.

5.2.1 Requisiti

Per prima cosa è stato eseguito uno studio dei requisiti del problema, il quale ha portato alla produzione delle informazioni seguenti:

- non si conosce a priori il numero di persone;
- il numero di persone può variare nel tempo;
- i dati devono essere computati in maniera semplice. In particolare, l'operazione scelta deve essere indipendente dall'ordine con cui i valori vengono elaborati;
- le leggi devono essere indipendenti dalla rappresentazione del vicinato.

In fase di analisi, si è deciso di sfruttare il concetto di baricentro come il punto che minimizza la distanza fra le posizioni di tutti i punti. Nel ricongiungimento di persone sarà, dunque, il punto che minimizza lo spostamento collettivo.

La politica di coordinazione sfrutta i gradienti computazionali `NBR` preservando la separazione fra quelli individuali e gli aggregati. Si mostrano ora i passi della politica di coordinazione scelta:

1. ogni utente ed il *token* creano un gradiente computazionale (figura 5.2);
2. i gradienti degli utenti vengono combinati allo scopo di trovare il baricentro;
3. il *token* si sposta nel baricentro;
4. gli utenti risalgono il gradiente computazionale del *token*.

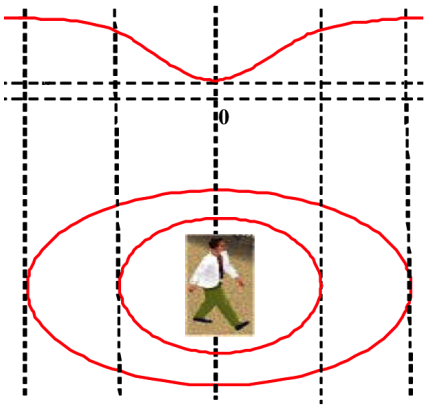
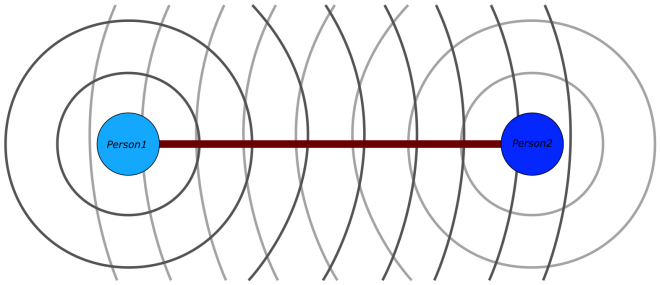


Figura 5.2: *Gradiente computazionale di presenza individuale. Immagine tratta da [5]*

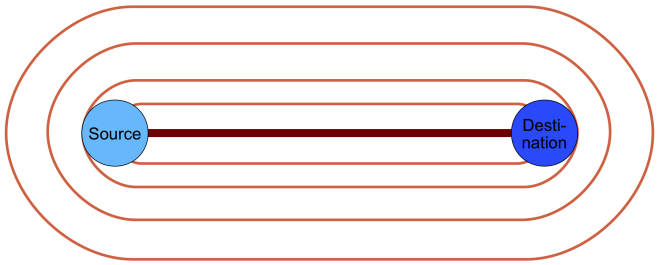
Il *token* è un'astrazione utile per effettuare il servizio di *meeting* (figura 5.1). In particolare, il *token* può essere posizionato inizialmente in un qualunque nodo della griglia e la sua posizione non partecipa al calcolo del baricentro. La scelta dell'introduzione di tale astrazione è dettata dal fatto che il punto baricentrico potrebbe essere in un punto dell'ambiente non accessibile dagli utenti, ad esempio a causa di un muro o di zone affollate. Questo potrebbe creare il generarsi di più punti baricentrici con la conseguenza che le persone si dividano fra essi non creando un'unica *meeting area*. Il *token* evita proprio questa situazione, in quanto è solamente esso la sorgente del gradiente computazionale che le persone risalgono, al limite se si presentano più punti eleggibili come baricentro sarà esso a sceglierne uno.

In particolare, per determinare il baricentro si calcolano le distanze fra le sorgenti dei gradienti ed ogni nodo della griglia. Nel caso specifico, l'informazione utilizzata è il quadrato della distanza. L'introduzione di una operazione non lineare, il quadrato, fa sì che si trovi sempre il baricentro fra i punti. Ad esempio, se si pensa alla griglia come un piano cartesiano, e dunque una visione discreta del gradiente computazionale, se si

hanno due persone lungo una stessa coordinata, tutti i nodi fra essi risultano avere lo stesso valore della somma dei due gradienti (figura 5.3).



(a) Ogni persona viene vista come una sorgente di un campo computazionale



(b) I gradienti computazionali generati dalle persone vengono sommati

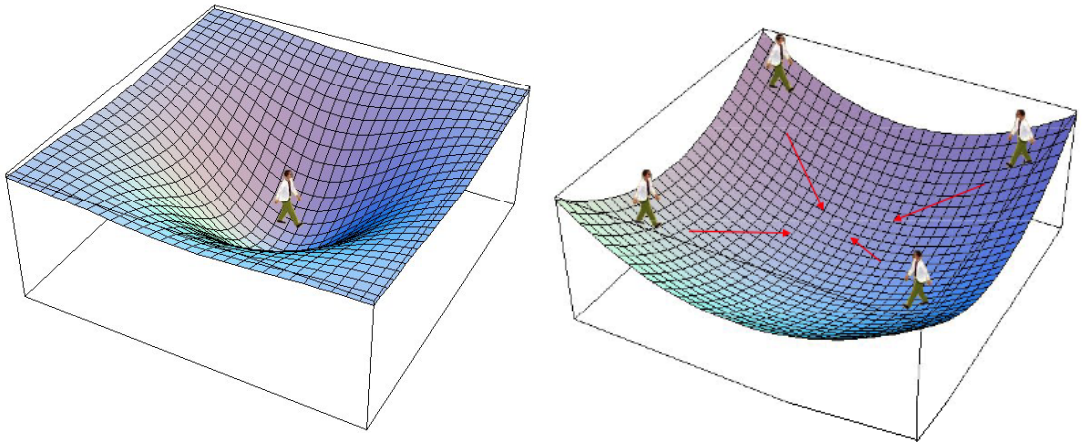
Figura 5.3: *Si mostra come la semplice somma di due gradienti computazionali produca dei valori che risultano essere uguali (indicati con una linea continua rossa) per diverse zone dello spazio. Immagine tratta da [5]*

Riassumendo, le fasi principali dell'algoritmo risultano essere:

1. generazione dei gradienti delle persone del token (figura 5.4(a));
2. calcolo baricentro rispetto ai gradienti delle persone;
3. l'agente eletto risale il gradiente della somma dei quadrati delle distanze e crea il gradiente del baricentro;
4. gli altri agenti seguono il gradiente creato dall'agente eletto, servizio di *join* (figura 5.4(b)).

5.3 Modello SAPERE

L'ambiente è costituito da una griglia di nodi. Ogni persona possiede un PDA, il quale invia e riceve dati dai nodi della griglia. Ogni PDA è collegato dinamicamente ai sensori



(a) Gradiente computazionale di una persona

(b) Gradiente computazionale del token risultante dalla combinazione dei gradienti delle persone

Figura 5.4: *Due fasi dell'algoritmo di join. Immagine tratta da [5]*

vicini - il vicinato sono i sensori all'interno di un certo raggio r, parametro del modello - dai quali ricevono dati dai PDA delle persone. Le persone seguono le indicazioni suggerite dal PDA posseduto.

Nell'approccio SAPERE (sezione 2.2), tutte le informazioni scambiate assumono la forma di LSA e le regole sono espresse in forma di *eco-law*.

5.3.1 Tipi di LSA nel sistema

Questo scenario richiede, oltre agli LSA introdotti nell'approccio di creazione del gradiente NBR (4.4), altri tipi di LSA mostrati in figura 5.5.

$\langle \texttt{sum}, value \rangle$
$\langle \texttt{tempsum}, list, value \rangle$
$\langle \texttt{field}, barycenter, tstamp, value \rangle$
$\langle \texttt{tempfield}, barycenter, list, value \rangle$
$\langle \texttt{namebarycenter}, barycenter \rangle$
$\langle \texttt{listSource}, list \rangle$
$\langle \texttt{token} \rangle$
$\langle \texttt{id}, name \rangle$

Figura 5.5: *LSA necessari per modellare l'algoritmo di join*

La LSA sum rappresenta il campo del gradiente riferito alla somma delle distanze al quadrato fra le persone, l'etichetta *value* indica il valore del campo in quel nodo. La LSA tempsum rappresenta il campo somma che si sta computando, l'etichetta *list* contiene la lista dei nodi i cui valori sono stati considerati, *value* indica il valore corrente della somma. Le LSA field e tempfield hanno lo stesso significato delle LSA precedenti, la peculiarità è l'etichetta *type* a cui è stato attribuito il valore barycenter in quando indica il campo del gradiente computazionale del baricentro. La LSA namebarycenter indica il nodo sorgente del gradiente computazionale, il cui identificativo è stato inserito nell'etichetta *barycenter*. La LSA listsources rappresenta la lista dei nodi sorgenti, i quali sono inseriti nell'etichetta *list*. Gli ultimi due tipi di LSA sono state creati poiché è necessario non cancellare alcuna sorgente dei gradiente computazionali. La LSA token rappresenta il *token*, ovvero il nodo dove le persone si devono incontrare. La LSA id rappresenta l'identificativo del nodo, l'etichetta *name* è riferita al nome univoco associato al nodo.

La figura 5.6 presenta gli LSA contenuti in ogni nodo, sia esso sorgente o non sorgente. Nel dettaglio, si avrà una coppia delle LSA field e tempfield in cui l'etichetta *type* conterrà un identificativo per ogni persona e per il baricentro. Nel sistema, infatti, avere una coppia di tali LSA indica il gestire il gradiente computazionale del tipo indicato nell'etichetta *type*.

$\langle \texttt{field}, type, 0, 100 \rangle$
$\langle \texttt{tempfield}, type, [], 100 \rangle$
$\langle \texttt{sum}, 100 \rangle$
$\langle \texttt{tempsum}, [], 0 \rangle$
$\langle \texttt{nameBarycenter}, barycenter \rangle$
$\langle \texttt{listsources}, list \rangle$
$\langle \texttt{id}, name \rangle$

Figura 5.6: *LSA contenuti inizialmente in ogni nodo*

Nelle figure 5.7, 5.8 e si mostrano gli LSA contenuti nella fase iniziale dell'algoritmo di *join* rispettivamente in ogni nodo non sorgente, sorgente di un gradiente computazionale riferito ad una persona e sorgente del gradiente computazionale riferito al baricentro.

$\langle \texttt{sensor}, id \rangle$

Figura 5.7: *LSA contenuti inizialmente in ogni nodo non sorgente*

Situazione iniziale in ogni nodo sorgente:

$$\langle \texttt{sensor}, id \rangle$$

Figura 5.8: *LSA contenuti inizialmente in ogni nodo sorgente del gradiente computazionale riferito alla presenza di una persona*

5.3.2 Eco-law

$$\langle \texttt{msg}, type, tstamp, node, dist \rangle, \langle \texttt{tempfield}, type, list, dist' \rangle \mapsto \langle \texttt{tempfield}, type, [node|list], min(dist, dist') \rangle, \langle \texttt{msg}, type, tstamp, node, dist \rangle \; if \; node \notin list$$

$$\langle \texttt{msg}, type, tstamp, node, dist \rangle, \langle \texttt{msg}, type, tstamp', node, dist' \rangle \mapsto \langle \texttt{msg}, type, tstamp', node, dist' \rangle \; if \; tstamp' > tstamp$$

$$\langle \texttt{msg}, type, tstamp, node, dist \rangle, \langle \texttt{listsources}, list \rangle \xmapsto{r_{decay}} \langle \texttt{listsources}, list \rangle \; if \; node \notin list$$

$$\langle \texttt{tempfield}, type, list, dist \rangle, \langle \texttt{field}, type, tstamp, dist' \rangle, \langle \texttt{id}, name \rangle \xmapsto{r_{field}} \langle \texttt{tempfield}, type, [\,], 100 \rangle, \langle \texttt{field}, type, tstamp + 1, dist \rangle, *\langle \texttt{msg}, type, tstamp, id, dist + \#D \rangle, \langle \texttt{id}, name \rangle \; if \; list \neq empty$$

$$\langle \texttt{tempfield}, barycenter, list, dist \rangle, \langle \texttt{field}, barycenter, tstamp, dist' \rangle, \langle \texttt{id}, name \rangle \xmapsto{r_{barycenter}} \langle \texttt{tempfield}, barycenter, [\,], 100 \rangle, \langle \texttt{id}, name \rangle, \langle \texttt{field}, barycenter, tstamp + 1, dist \rangle, *\langle \texttt{msg}, barycenter, tstamp, id, dist + \#D \rangle \; if \; list \neq empty$$

$$\langle \texttt{namebarycenter}, barycenter \rangle, \langle \texttt{field}, type, tstamp, dist \rangle, \langle \texttt{tempsum}, list, dist \rangle \mapsto \langle \texttt{tempsum}, [type|list], value + dist * dist \rangle, \langle \texttt{field}, type, tstamp, dist \rangle, \langle \texttt{namebarycenter}, barycenter \rangle \; if \; type \neq barycenter \; and \; type \notin list \; and \; dist \neq 100$$

$$\langle \texttt{sum}, value \rangle, \langle \texttt{tempsum}, list, value \rangle \xmapsto{r_{sum}} \langle \texttt{sum}, value \rangle, \langle \texttt{tempsum}, [\,], 0 \rangle \; if \; list \neq empty$$

$$\langle \texttt{token} \rangle, \langle \texttt{msg}, barycenter, tstamp, node, 0 \rangle \xmapsto{r_{token}} \$\langle \texttt{token} \rangle, \$\langle \texttt{msg}, barycenter, tstamp, node', 0 \rangle$$

Figura 5.9: *Eco-law che descrivono il calcolo del baricentro in ogni nodo della griglia*

$$\langle \texttt{token} \rangle, \langle \texttt{msg}, barycenter, tstamp, node, 0 \rangle \xmapsto{r_{token}} \$\langle \texttt{token} \rangle, \$\langle \texttt{msg}, barycenter, tstamp, node', 0 \rangle$$

Figura 5.10: *Eco-law posta nel punto dove parte il token e che fa partire il suo movimento verso il punto baricentro*

Nelle figure 5.6, 5.10 e 5.11 sono state inserite le leggi per l'algoritmo di *join*.

$$\langle \texttt{source}, type \rangle \quad \mapsto \quad *\langle \texttt{msg}, type, 0, source, \#D \rangle$$

Figura 5.11: *Eco-law in ogni nodo sorgente di un gradiente relativo alla presenza di una persona*

Le prime quattro leggi in figura 5.6 hanno il compito di gestire il gradiente computazionale nell'approccio NBR, pertanto per una spiegazione dettagliata si faccia riferimento alla sezione ( ). L'unica nota è per la *eco-law* di *decay* che nel caso specifico è riferita a tutti i nodi sorgente dei gradienti computazionali, i quali sono racchiusi in una lista nella LSA listsources. La quinta *eco-law* permette di gestire in modo opportuno il gradiente di tipo barycenter, in particolare la differenza rispetto alla quarta è nel rate. Nel dettaglio, il rate della quinta deve essere maggiore di quello della quarta, in quando la sorgente del gradiente del baricentro è mobile. La sesta *eco-law* gestisce il gradiente computazionale della somma dei quadrati delle distanze fra le persone, per questo motivo, infatti, si controlla che il field da inserire nel conteggio non sia di tipo barycenter. Gli altri controlli verificano quali tipi di gradiente sono già stati inseriti e se hanno un valore significativo, ovvero diverso da 100 che corrisponde all'astrazione di infinito. La settima gestisce il gradiente aggregato riferito alla somma, in particolare associa alla LSA sum il valore di tempsum e poi svuota quest'ultimo, in modo che possa ricominciare un'altra elaborazione con i dati più aggiornati. L'ultima *eco-law* ha lo scopo di spostare la sorgente del gradiente barycenter. Il simbolo $ viene utilizzato in modo ad hoc in questo scenario, per indicare il movimento delle LSA nel vicino migliore rispetto al gradiente della somma.

In figura 5.10 non vi è altro che l'ottava *eco-law* presentata in figura 5.6 e, dunque, ha lo stesso significato, la trattazione singola sta ad indicare che è possibile inserire la LSA token in qualsiasi posizione che essa riuscirà a risalire il gradiente computazionale della somma.

In figura 5.11 è stata inserita l'*eco-law* con lo scopo di fare la prima iniezione del gradiente computazionale riferito alla presenza di una persona dalla sorgente, ovvero dalla posizione in cui è posta, a tutti i nodi del vicinato.

In appendice E è stata inserita l'implementazione nel DSL per SAPERE.

5.4 Gestione *rate*

Prima di eseguire le simulazioni, risulta necessario definire due elementi fondamentali dell'algoritmo:

- il *set* di *rate* adatti per il modello ed il rapporto in termini di ordini di grandezza fra essi,

- la scelta di istanze di *test* che risultano essere istanti chiave durante l'evoluzione del comportamento del sistema.

Nell'ambito dei sistemi auto-organizzanti per la definizione dei *rate* si fa affidamento sulla simulazione stocastica, al fine di catturare sia i tempi sia la distribuzione di probabilità. I parametri di questo tipo sono tipicamente espressi in termini di *rate* d'azione definiti sulla base di opportune distribuzioni stocastiche. La distribuzione esponenziale è tipicamente utilizzata per i sistemi *memoryless*, di cui i sistemi auto-organizzanti fanno parte. Inoltre, l'uso della distribuzione esponenziale consente la mappatura nelle Catene di Markov Tempo Continuo (CTMC), le quali vengono comunemente utilizzate nelle simulazioni e nell'analisi delle prestazioni. I *rate* dovrebbero, inoltre, rispecchiare le condizioni nello scenario sviluppato, altrimenti i risultati delle simulazioni non risulteranno essere significative.

Nell'algoritmo del baricentro (figura 5.9) i rate da gestire sono riferiti al: (i) decadimento (r_{decay}); (ii) assegnamento del valore aggiornato relativo al tipo di campo computazionale considerato (r_{field}), (iii) assegnamento del valore aggiornato al campo computazione del baricentro, ovvero quello che gestisce la somma dei quadrati delle distanze delle persone rispetto ai nodi della griglia ($r_{barycenter}$), (iv) assegnamento del valore aggiornato al campo computazione della somma, ovvero quello che gestisce la somma di tutti i gradienti computazionali che indicano la presenza di una persona (r_{sum}), (v) movimento del *token* verso il punto eletto come baricentro (r_{token}), (vi) movimento delle persone come risalita del gradiente computazionale del *token* (r_{person}). Il rapporto fra i rate è:

$$r_{field} > r_{sum} > r_{token} > r_{person} >> r_{decay}$$

Tale rapporto è giustificato dal fatto che il token si muove solamente dopo che il campo computazione della somma dei gradienti che gestiscono la presenza delle persone è stabile. Si vuole, inoltre, che le persone risalgano il gradiente del *token* quando risulta essere arrivato nel punto eletto come baricentro, altrimenti si otterrebbe un comportamento delle persone che inseguono il *token*. Per ottenere un sistema reattivo alle perturbazioni ambientali è stato necessario inserire una *eco-law* di decadimento dei messaggi presenti nella cache dei nodi. Il *rate* di tale legge deve essere scelto in modo da ottenere un buon compromesso fra stabilità e reattività, in quanto la prima lo vorrebbe basso mentre la seconda alto.

Le istanze di tempo che risultano essere significative per il sistema auto-organizzante risultano essere:

- primo movimento del token verso la sorgente ed ultimo verso il punto eletto come baricentro,

- arrivo di tutte le persone nel punto in cui è posto il token.

5.5 Simulazioni

Il comportamento di ogni nodo della griglia è programmato in accordo con il modello di coordinazione delle *eco-law* in figura 5.9, quello degli agenti che rappresentano le persone con le *eco-law* in figura 5.11, mentre quello del nodo che rappresenta il *token* con le *eco-law* in figura 5.10.

Si noti che il movimento delle persone è composto da passi discreti nell'ambiente. In particolare, è possibile definire la distanza massima fra i passi compiuti dalle persone in modo da modellare il limite fisico di spazio percorso nel tempo.

Nelle prossime sezioni sono stati inseriti vari scenari, ognuno dei quali ha lo scopo di illustrare una componente diversa del modello del baricentro e ricongiungimento delle persone oggetto di studio nel capitolo corrente. Per ogni scenario: si descrive il *setting* dell'ambiente, si mostrano i risultati ottenuti dalla simulazione in termini di analisi quantitativa e qualitativa.

5.5.1 Scenario 1

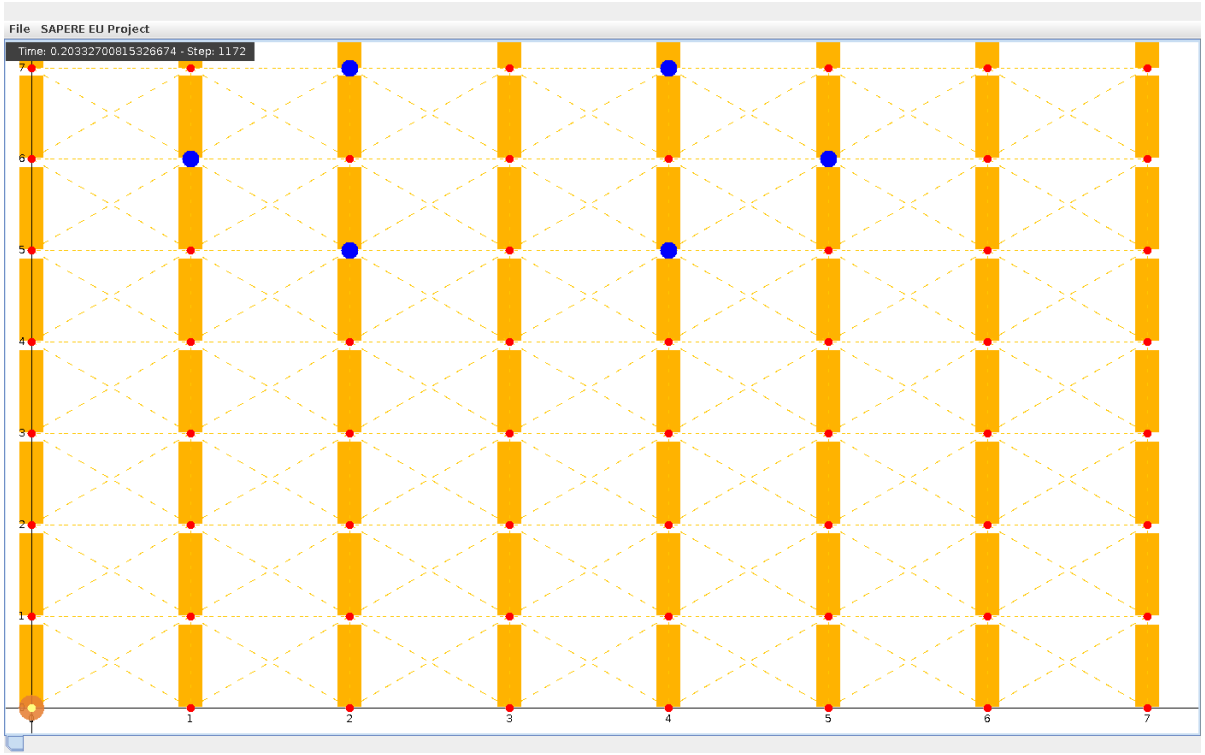
Nelle tabelle 5.1 e 5.2 si mostrano rispettivamente per questo scenario il *setting* dell'ambiente e l'analisi quantitativa. Nelle figure 5.12, 5.13 e 5.14 sono stati inseriti degli *snapshot* che illustrano i momenti chiave nell'esecuzione della simulazione dello scenario in esame.

Dimensione griglia	8x7
N° nodi	64
N° ostacoli	0
N° persone	6
Posizioni persone	(2,7); (4,7); (1,6); (5,6); (2,5); (4,5)
Posizione di partenza del token	(0,0)
Punto baricentrico	(3,6)
Accessibile	sì

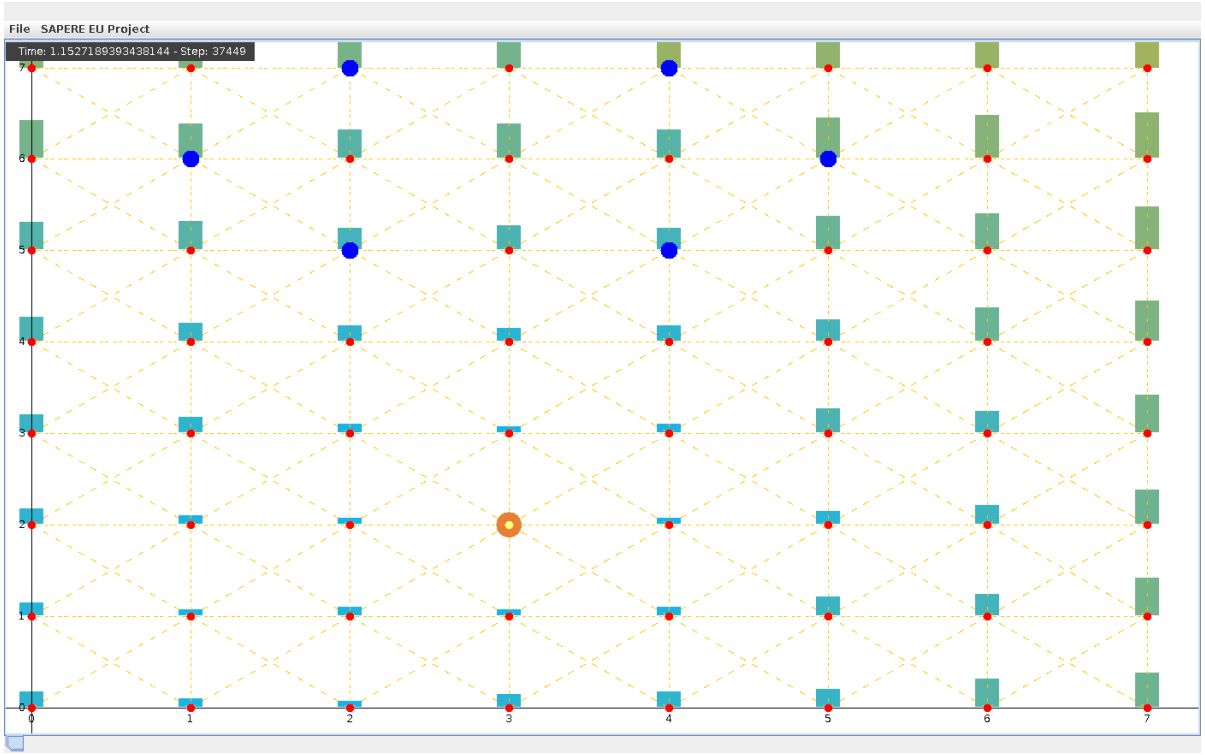
Tabella 5.1: *Setting ambiente scenario 1*

Analisi qualitativa

In questo scenario si è mostrato un ambiente di base senza ostacoli. La posizione del *token* coincide con il punto baricentrico. Si noti che tutte le persone effettuano il cammino minimo per raggiungere il *token*.

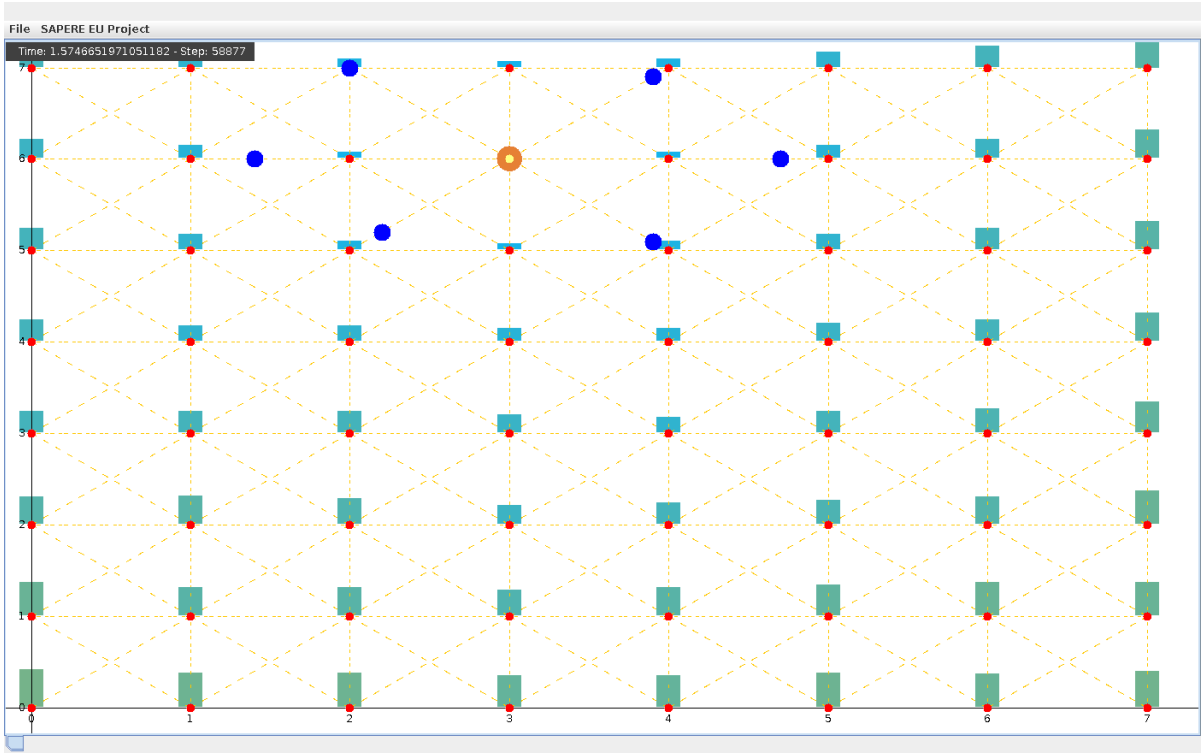


(a) Situazione iniziale

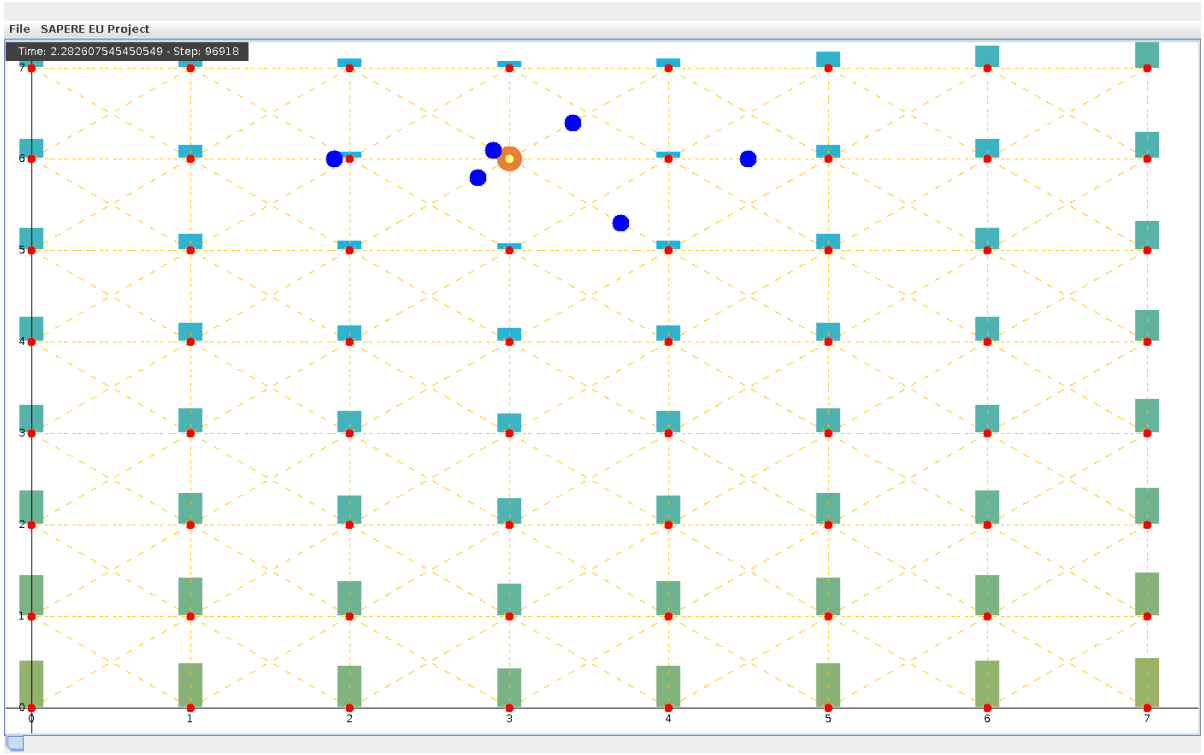


(b) Il token si sta muovendo verso il baricentro

Figura 5.12: *Ambiente di simulazione scenario 1*

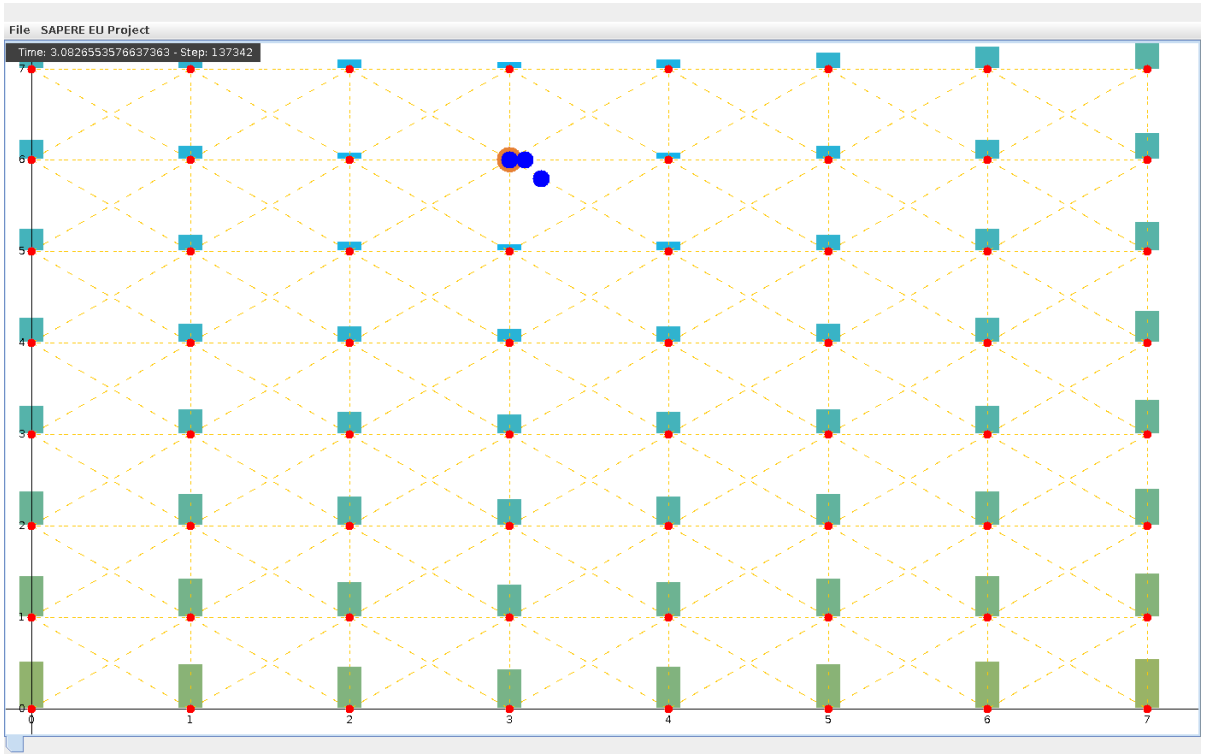


(a) Token arrivato nel punto eletto come baricentro



(b) Le persone si iniziano a muovere verso il token

Figura 5.13: *Ambiente di simulazione scenario 1*



(a) Le persone stanno raggiungendo il token

Figura 5.14: *Ambiente di simulazione scenario 1*

Punto raggiunto dal token	(3,6)
Tempo simulato necessario	1.2
N° reazioni necessarie	77,614
Numero persone raggiungono il token	6
Tempo simulato necessario	4.02
N° reazioni necessarie	29,315

Tabella 5.2: *Analisi quantitativa scenario 1*

5.5.2 Scenario 2

Nelle tabelle 5.3 e 5.4 si mostrano rispettivamente per questo scenario il *setting* dell'ambiente e l'analisi quantitativa. Nelle figure 5.15, 5.16 e 5.17 sono stati inseriti degli *snapshot* che illustrano i momenti chiave nell'esecuzione della simulazione dello scenario in esame.

Dimensione griglia	8x8
N° nodi	62
N° ostacoli	1
Posizione ostacolo	(2,2)-(2,3)
N° persone	6
Posizioni persone	(2,7); (4,7); (1,6); (5,6); (2,5); (4,5)
Posizione di partenza del token	(0,0)
Punto baricentrico	(3,6)
Accessibile	sì

Tabella 5.3: *Setting ambiente scenario 2*

Punto raggiunto dal token	(3,6)
Tempo simulato necessario	1.4
N° reazioni necessarie	51,863
Numero persone raggiungono il token	6
Tempo simulato necessario	3.89
N° reazioni necessarie	169,675

Tabella 5.4: *Analisi quantitativa scenario 2*

Analisi qualitativa

In questo scenario si è mostrato un ambiente con un ostacolo. Si ha che la posizione dell'ostacolo non influenza la posizione del *token*, in quanto il punto baricentrico rimane in una zona dell'ambiente accessibile. Di conseguenza, il nodo calcolato come baricentro e dove si posiziona il *token* è lo stesso.

L'ostacolo, però, giustifica l'approccio con cui è stato progettato il calcolo del baricentro, ovvero attraverso la somma dei quadrati delle distanze delle persone da ogni punto dell'ambiente. Ad esempio si pensi di aver modellato il calcolo attraverso le coordinate cartesiane, questo approccio non risulta essere adeguato in quanto non riesce a rilevare la presenza di ostacoli. I percorsi eseguiti dalle persone, infatti, devono essere computati dinamicamente in base alla struttura dell'ambiente.

Per questo scenario si è fatto partire il *token* in punti diversi dell'ambiente il cui percorso minimo di raggiunta del baricentro è influenzato dalla presenza dell'ostacolo. Il risultato è che il *token* riesce in ogni configurazione a posizionarsi nel punto baricentrico aggirando l'ostacolo.

5.5.3 Scenario 3

Nelle tabelle 5.5 e 5.6 si mostrano rispettivamente per questo scenario il *setting* dell'ambiente e l'analisi quantitativa. Nelle figure 5.18, 5.19 e 5.20 sono stati inseriti degli *snapshot* che illustrano i momenti chiave nell'esecuzione della simulazione dello scenario in esame.

Dimensione griglia	10x10
N° nodi	92
N° ostacoli	2
Posizione ostacolo	(2,3)-(2,6); (6,4)-(7,5)
N° persone	10
Posizioni persone	(1,9); (2,1); (9,5); (3,5); (4,8); (6,2) (6,7); (7,7); (8,1); (8,7)
Posizione di partenza del token	(0,0)
Punto baricentrico	(6,5)
Accessibile	no

Tabella 5.5: *Setting ambiente scenario 3*

Punto raggiunto dal token	(5,5)
Tempo simulato necessario	1.43
N° reazioni necessarie	100,033
Numero persone raggiungono il token	10
Tempo simulato necessario	6.84
N° reazioni necessarie	620,115

Tabella 5.6: *Analisi quantitativa scenario 3*

Analisi qualitativa

In questo scenario si è mostrato un ambiente di dimensioni maggiori, con un numero di persone maggiore e con due ostacoli. Si noti la scalabilità del sistema che riesce a calcolare il baricentro nello stesso tempo simulato sia in questo scenario che lo scenario 2, mentre si ha, naturalmente, la richiesta di un tempo maggiore per far ricongiungere le persone nel punto eletto come baricentro dato che il loro numero e maggiore e la loro distanza rispetto al baricentro risulta essere superiore. Si noti che il parametro relativo alla velocità di movimento delle persone è stato realizzato considerando una camminata di un adulto a velocità media.

5.5.4 Scenario 4

Nella tabella 5.5 si mostra per questo scenario il *setting* dell'ambiente. Nelle tabelle 5.8, 5.9, 5.10. 5.24 e 5.11 si mostra l'analisi quantitativa dello scenario in esame nel caso in cui il *token* sia posizionato inizialmente in vari punti dell'ambiente, in particolare si considerano i quattro angoli della griglia. Nelle figure 5.22, 5.23, 5.24 e 5.25 è stata inserita una coppia di *snapshot* che illustrano la situazione iniziale, in particolare la posizione di partenza del *token*, e la situazione finale, ovvero il ricongiungimento delle persone.

Analisi qualitativa

In questo scenario si è mostrato un ambiente in cui il punto baricentrico non è accessibile e questo comporta la possibilità di eleggere più punti come baricentro. In tale scenario si mostra l'utilità dell'introduzione dell'astrazione del *token*. La situazione in esame, infatti, potrebbe far sì che risalendo semplicemente il campo computazionale della somma dei quadrati delle distanze, le persone raggiungano punti diversi dell'ambiente. Il *token*, invece, assicura che tutte le persone si riuniscano in un solo punto poiché risalgono il suo gradiente computazionale. Si noti, quindi, che il campo computazionale della somma dei quadrati potrebbe non avere un unico punto con il valore minimo, situazione che invece è necessaria nel caso in cui si generi un gradiente computazionale, a tale punto,

Dimensione griglia	10x10
N° nodi	91
N° ostacoli	1
Posizione ostacolo	(4,4)-(6,6);
N° persone	4
Posizioni persone	(2,5); (8,5); (5,2); (5,8);
Posizione di partenza del token	(0,0); (0,9) (9,0); (9,9);
Punto baricentrico	(5,5)
Accessibile	no

Tabella 5.7: *Setting ambiente scenario 4*

Punto raggiunto dal token	(4,3)
Tempo simulato necessario	0.87
N° reazioni necessarie	23,307
Numero persone raggiungono il token	4
Tempo simulato necessario	6.8
N° reazioni necessarie	366,720

Tabella 5.8: *Analisi quantitativa scenario 4 con token in posizione iniziale (0,0)*

Punto raggiunto dal token	(4,3)
Tempo simulato necessario	1.43
N° reazioni necessarie	46,200
Numero persone raggiungono il token	4
Tempo simulato necessario	7.43
N° reazioni necessarie	386,744

Tabella 5.9: *Analisi quantitativa scenario 4 con token in posizione iniziale (9,0)*

come introdotto in precedenza nel percorso della tesi, viene associato il concetto di nodo sorgente.

5.5.5 Risultati

Dall'esecuzione attraverso il framework Alchemist delle simulazioni negli scenari proposti si ha che il risultato ottenuto risulta essere sempre quello atteso, ovvero il *token* rag-

Punto raggiunto dal token	(6, 7)
Tempo simulato necessario	1.36
N° reazioni necessarie	45,831
Numero persone raggiungono il token	4
Tempo simulato necessario	8.69
N° reazioni necessarie	467,780

Tabella 5.10: *Analisi quantitativa scenario 4 con token in posizione iniziale (0,9)*

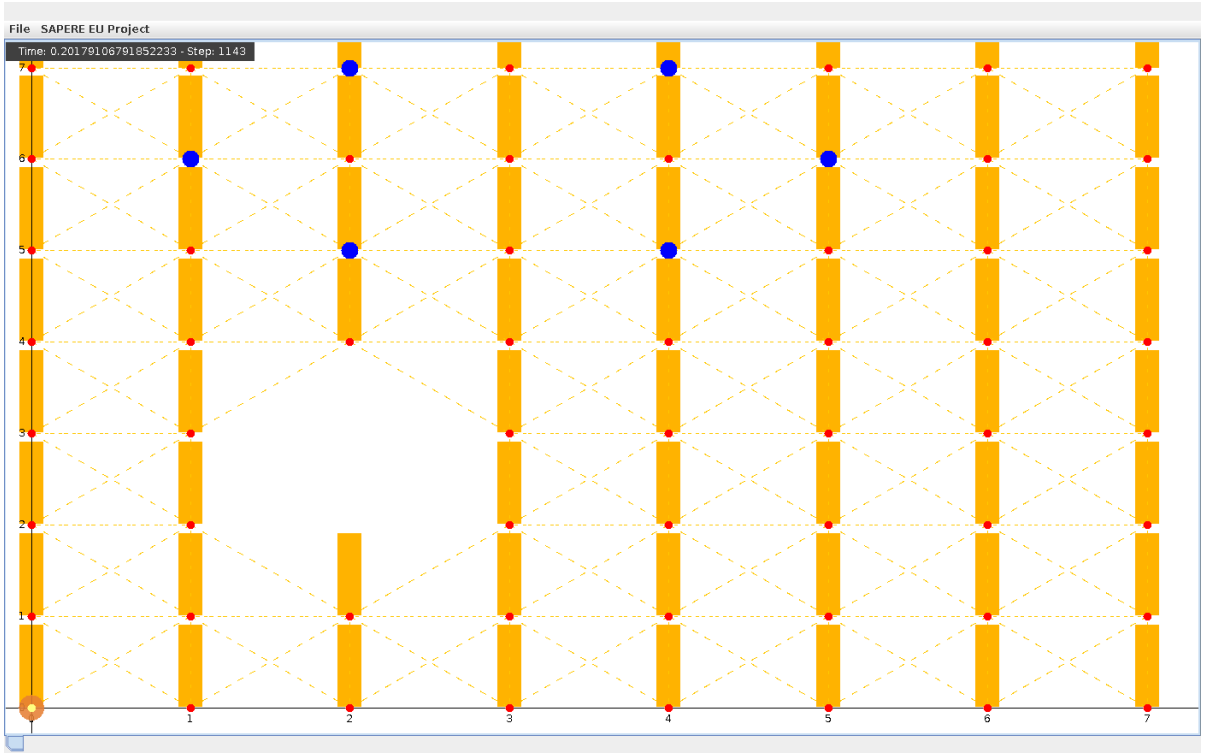
Punto raggiunto dal token	(6, 7)
Tempo simulato necessario	2.1
N° reazioni necessarie	89,558
Numero persone raggiungono il token	4
Tempo simulato necessario	7.25
N° reazioni necessarie	385,540

Tabella 5.11: *Analisi quantitativa scenario 4 con token in posizione iniziale (9,9)*

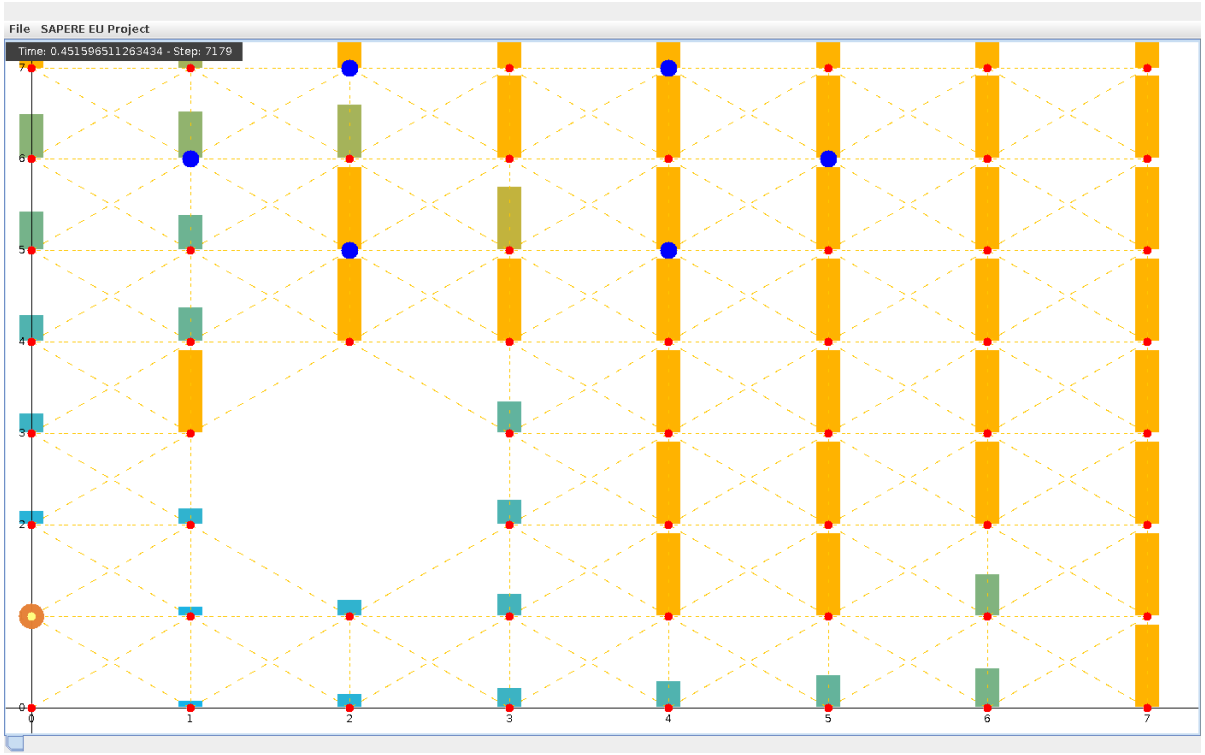
giunge sempre il punto che elegge essere il baricentro e tutte le persone si ricongiungono in tale punto.

L'algoritmo risulta essere robusto alla presenza di ostacoli sia "fissi" che "dinamici", ovvero già inseriti all'inizio della simulazione o che si creano in qualsiasi momento durante la stessa. Nell'astrazione in cui l'ambiente sia una stanza si ha che gli ostacoli fissi possono essere associati ad esempio a muri, mentre quelli dinamici ad una zona affollata, ovvero uno spazio in cui sono riunite delle altre persone, e che si vuole evitare.

L'algoritmo, inoltre, risulta scalabile sia in termini della dimensione della griglia che sul numero di persone. Tale algoritmo richiede al framework Alchemist, però, un notevole lavoro computazionale in quanto aumentando la dimensione della griglia crescono i numeri di nodi da gestire e poiché ognuno di essi ha una *cache* interna il numero di reazioni nel sistema aumentano, aumentando il numero di persone cresce la dimensione della cache in ogni nodo e quindi la sua gestione risulta essere più costosa. Questo lavoro comporta l'avere una notevole differenza fra il tempo simulato ed il tempo reale.

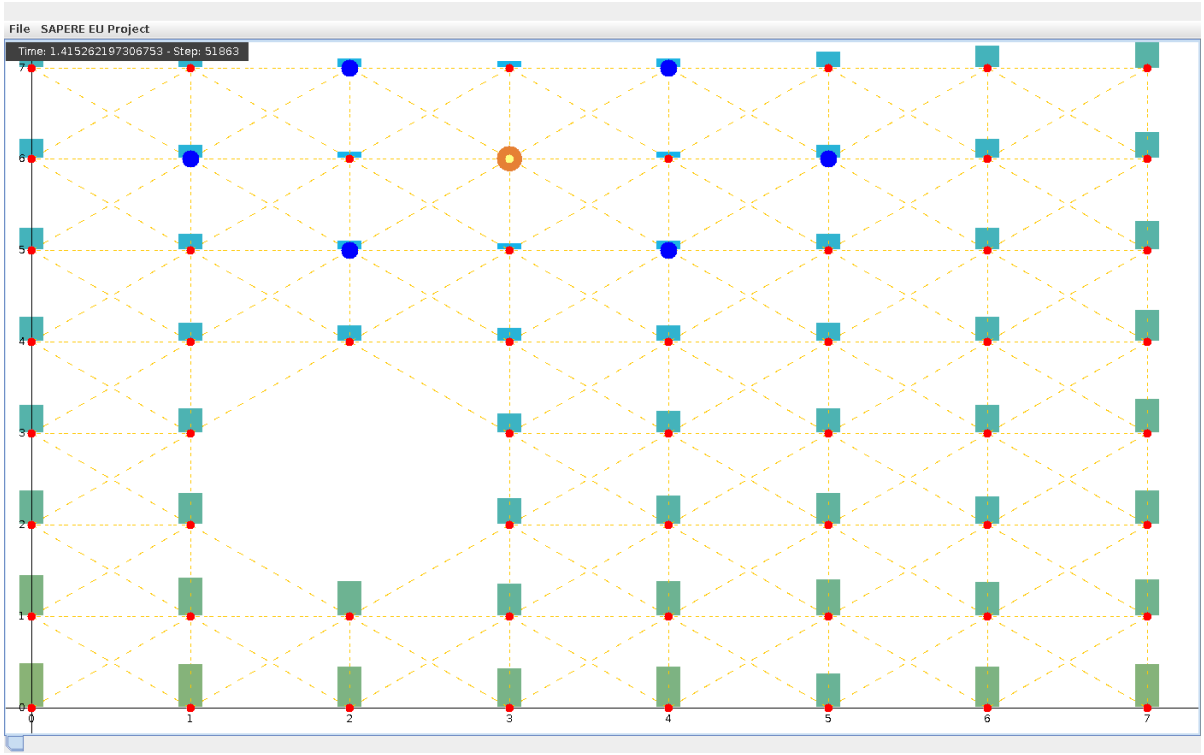


(a) Situazione iniziale

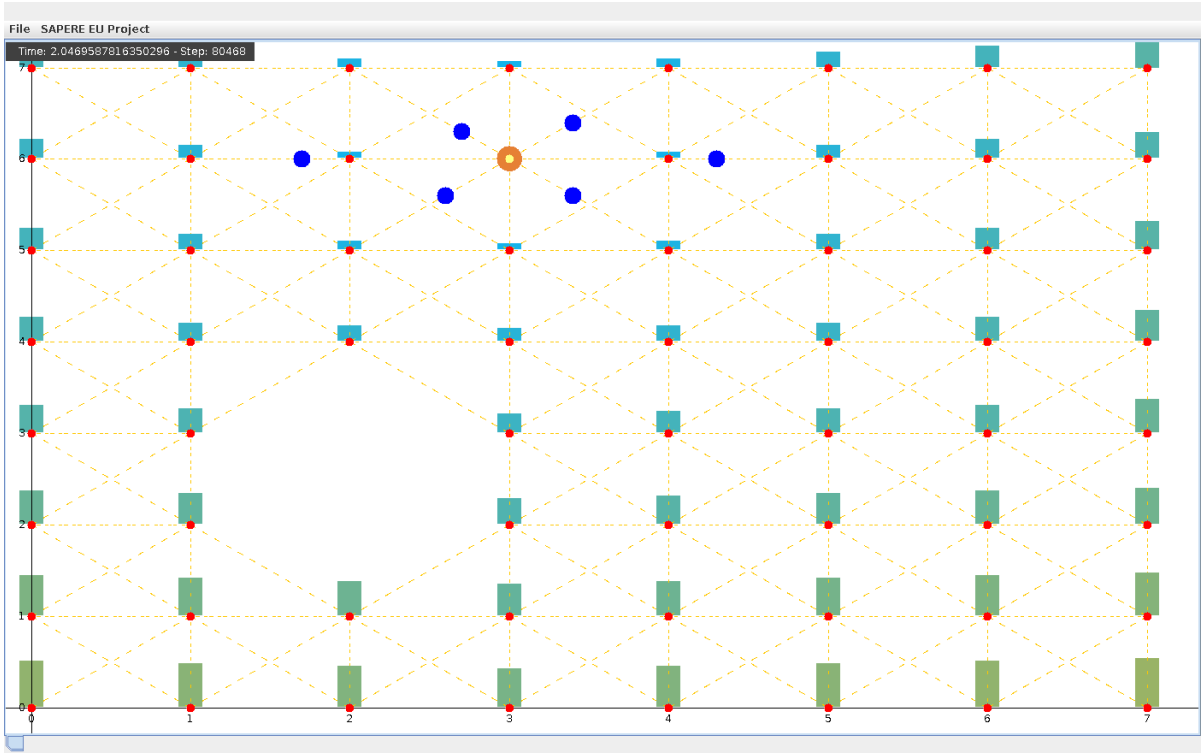


(b) Il token si sta muovendo verso il baricentro

Figura 5.15: *Ambiente di simulazione scenario 2*

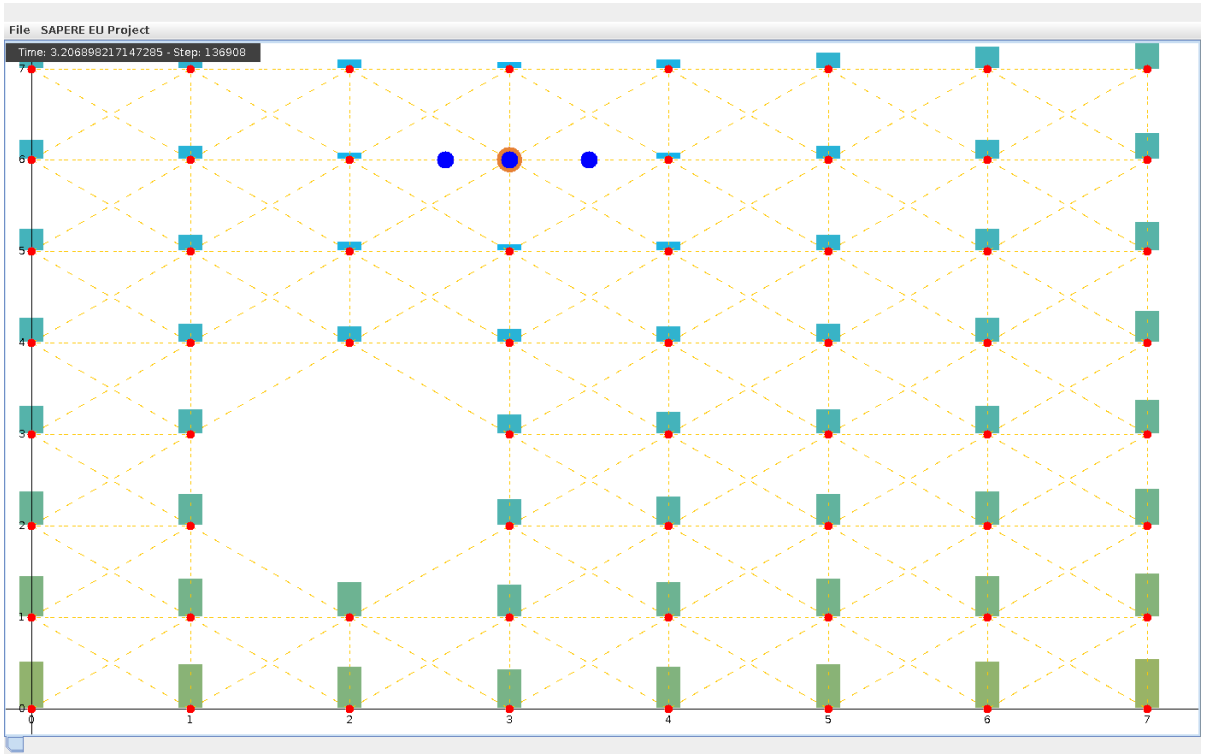


(a) Token arrivato nel punto eletto come baricentro



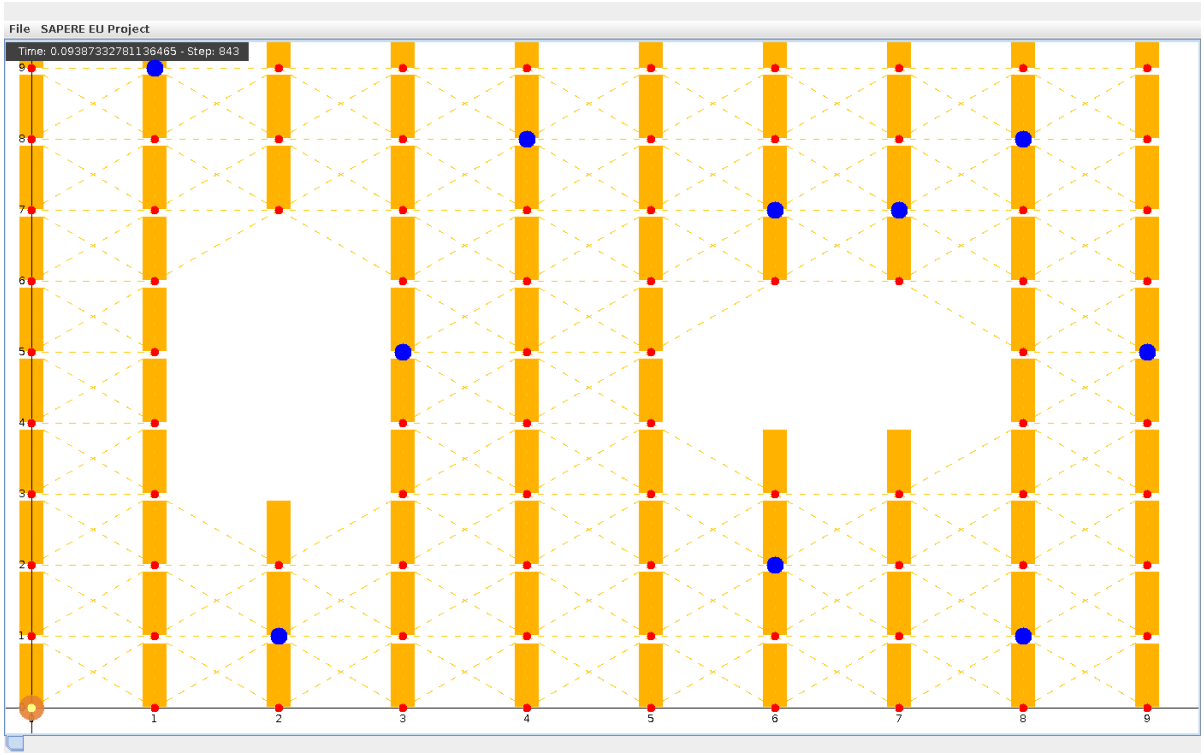
(b) Le persone si iniziano a muovere verso il token

Figura 5.16: *Ambiente di simulazione scenario 2*

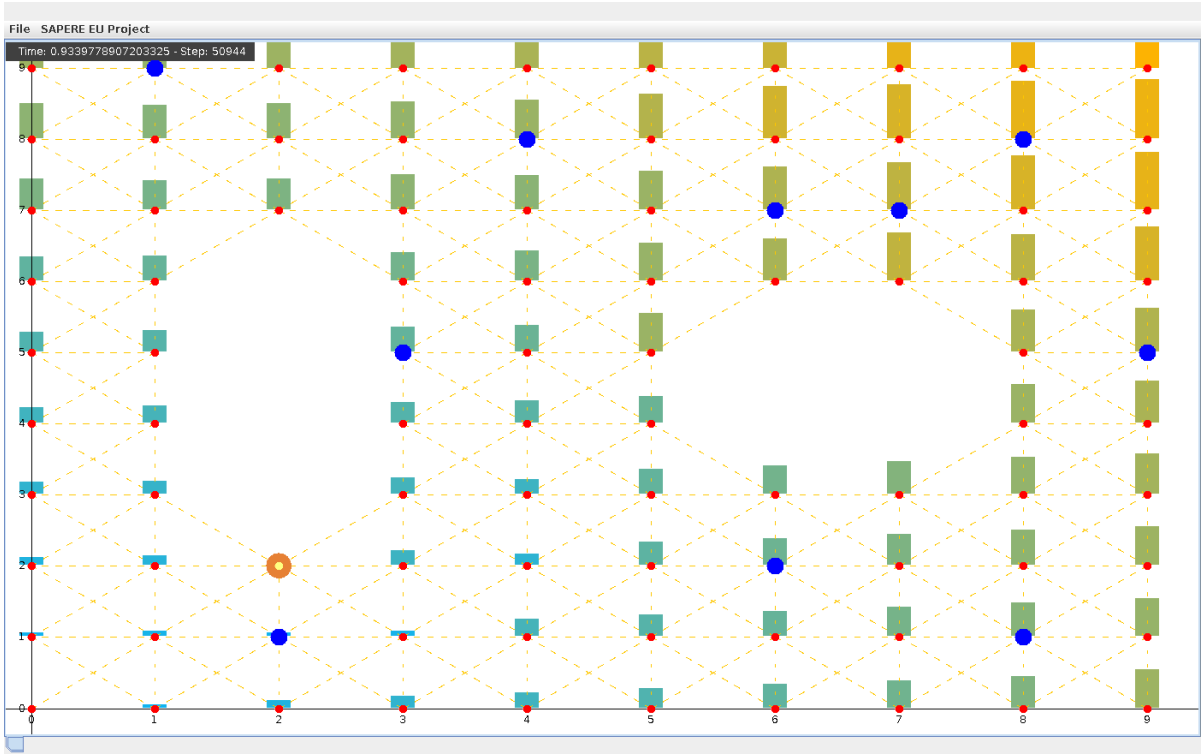


(a) Le persone stanno raggiungendo il token

Figura 5.17: *Ambiente di simulazione scenario 2*

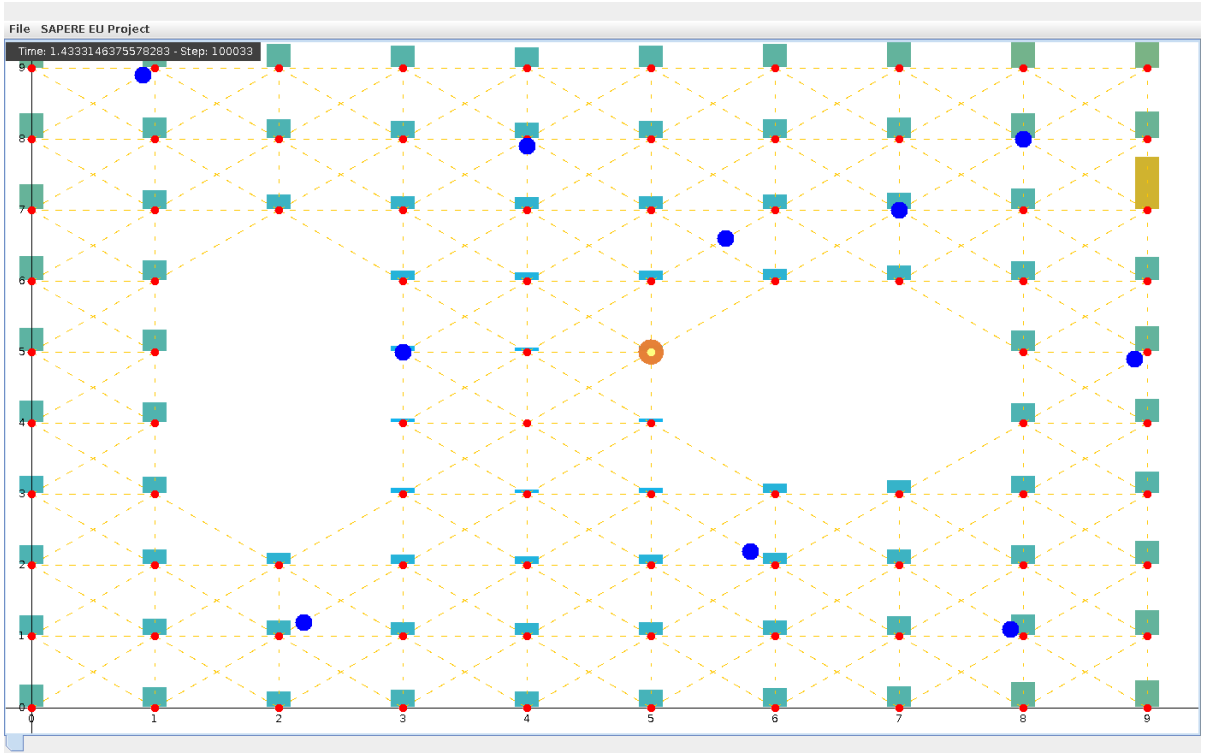


(a) Situazione iniziale

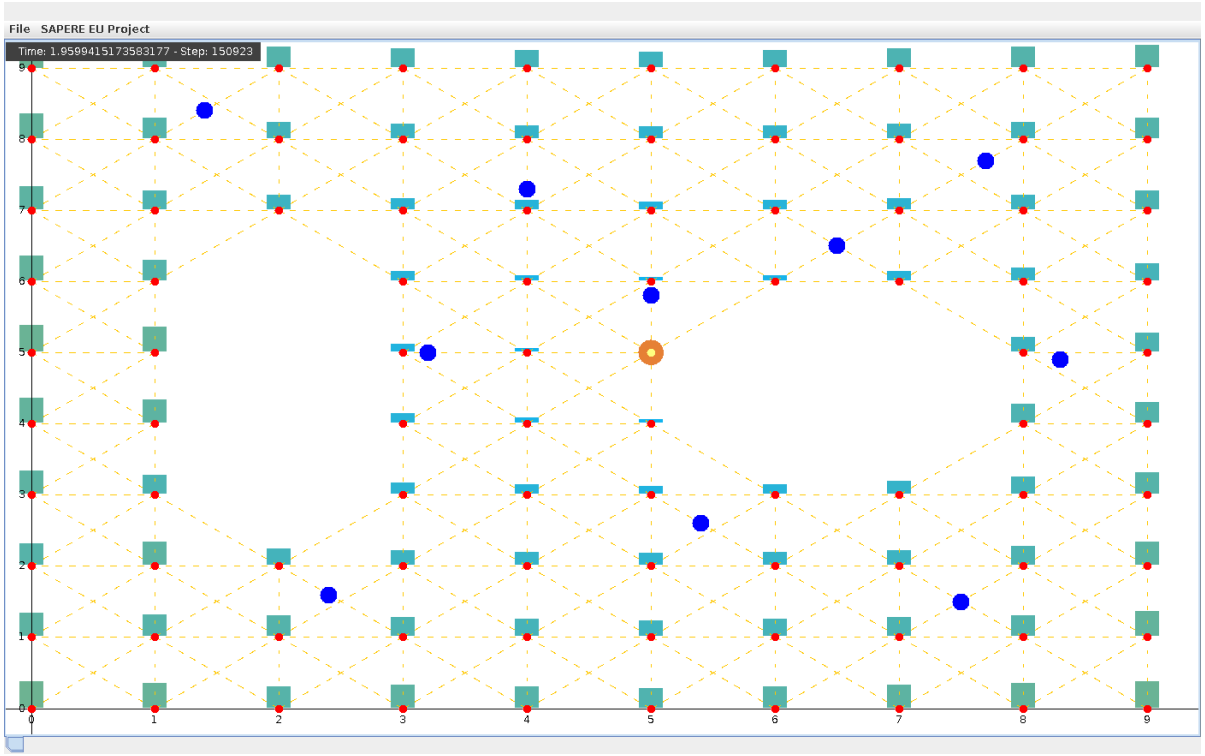


(b) Il token si sta muovendo verso il baricentro

Figura 5.18: *Ambiente di simulazione scenario 3*

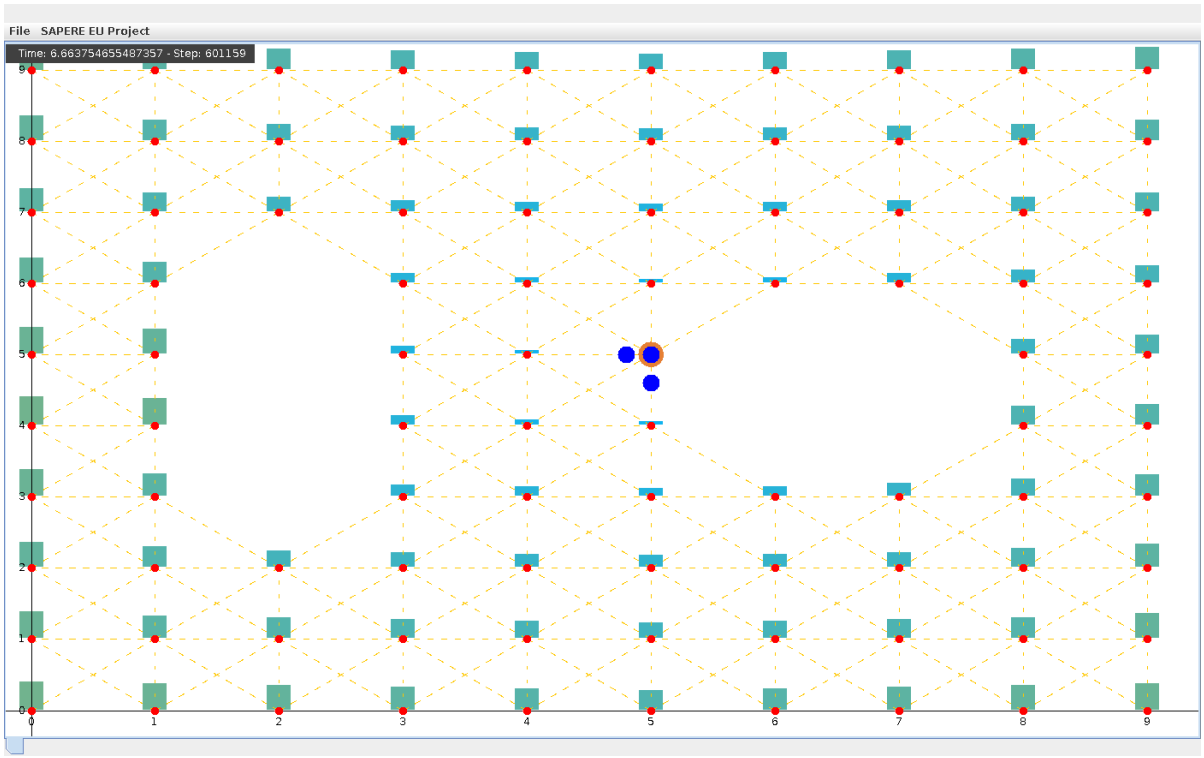


(a) Token arrivato nel punto eletto come baricentro



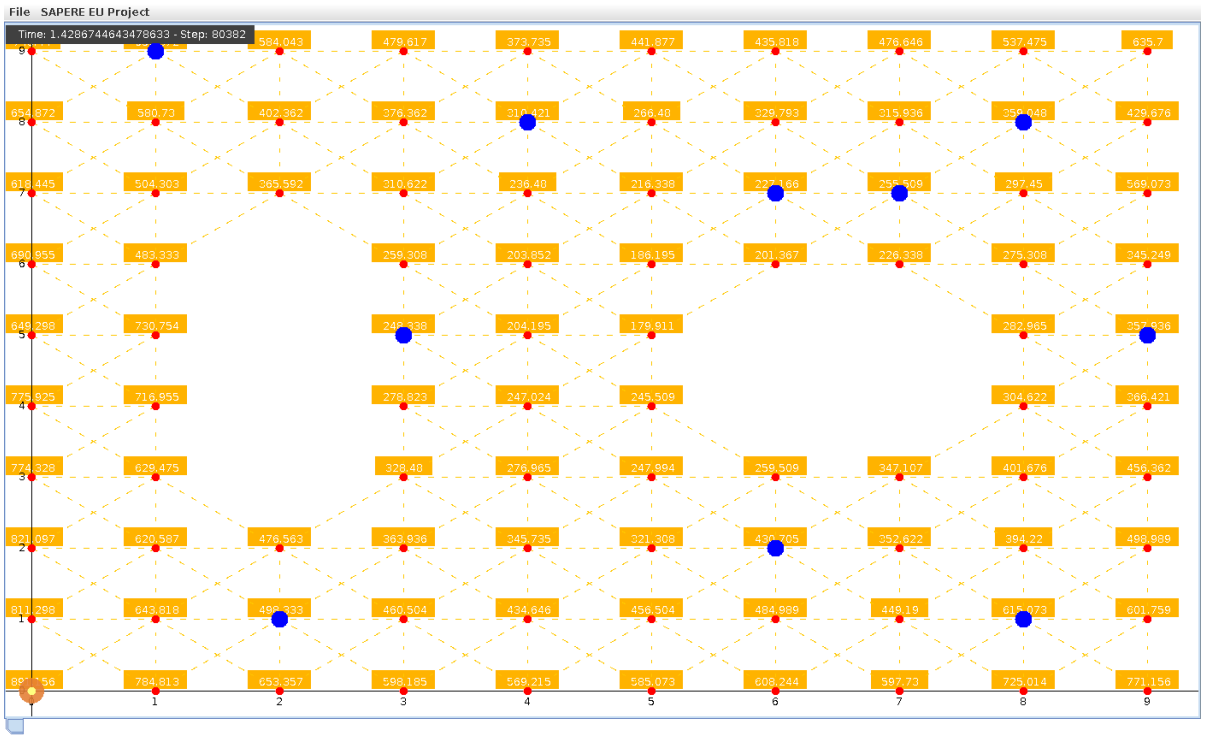
(b) Le persone si iniziano a muovere verso il token

Figura 5.19: *Ambiente di simulazione scenario 3*



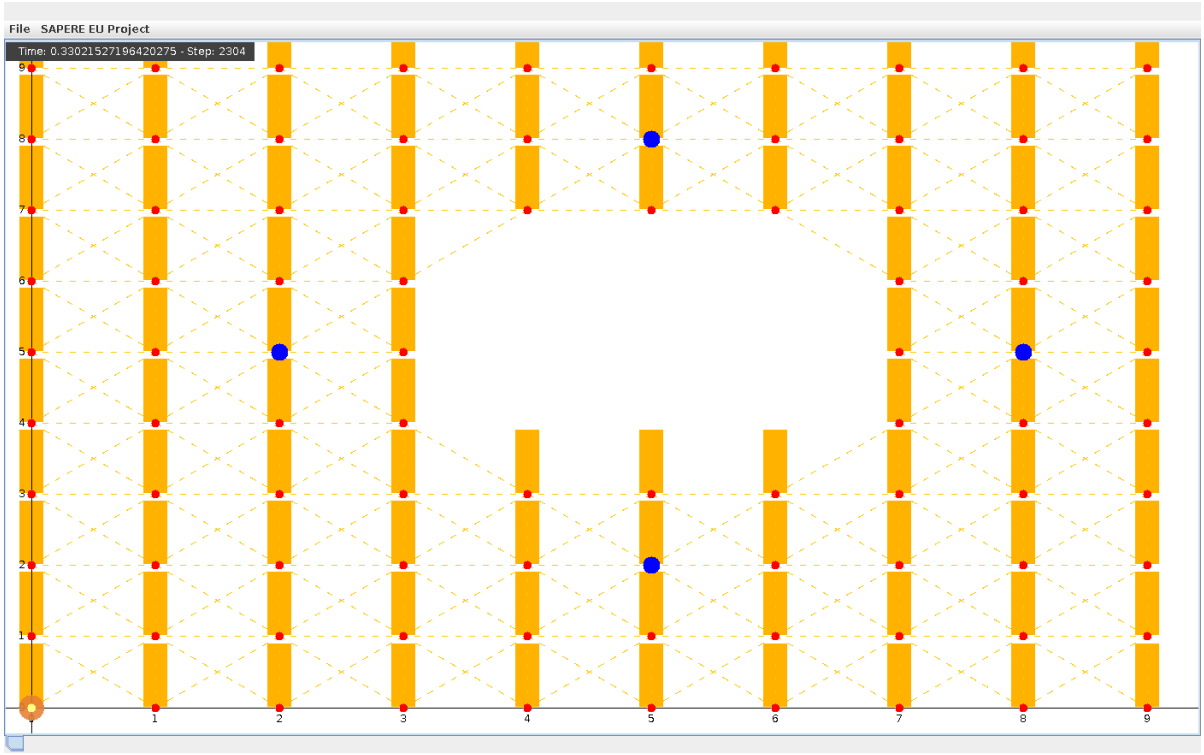
(a) Le persone stanno raggiungendo il token

Figura 5.20: *Ambiente di simulazione scenario 3*

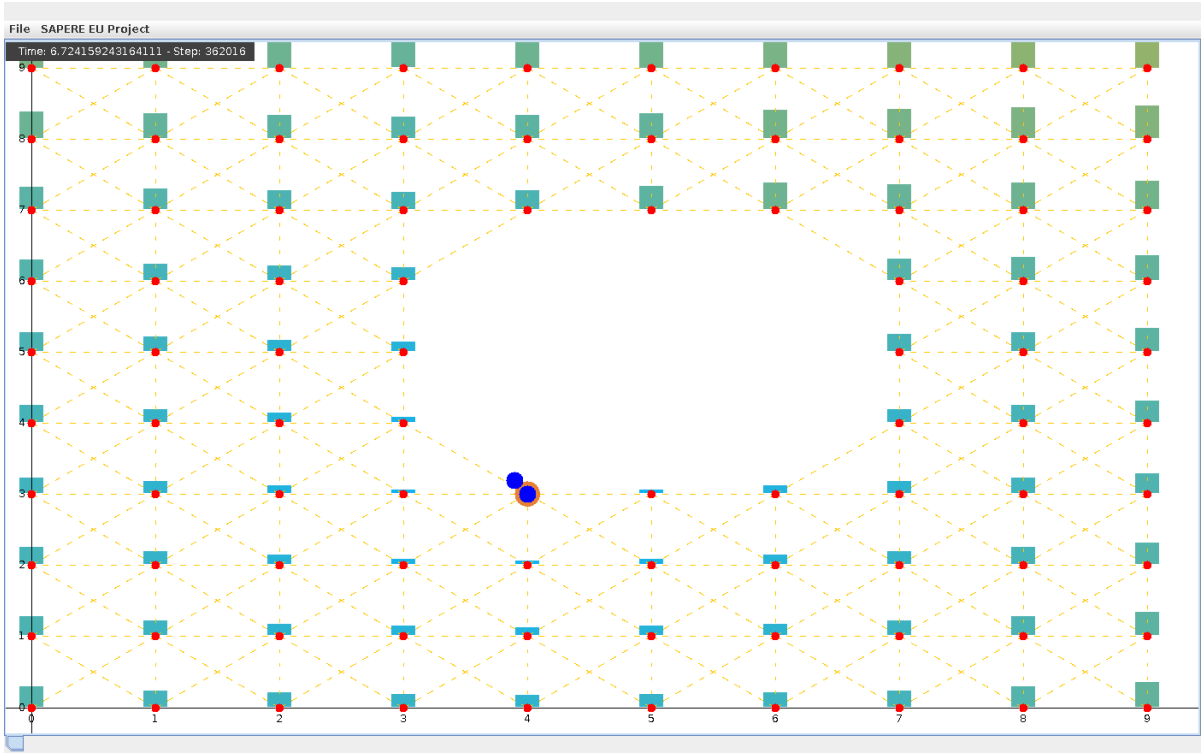


(a) Campo computazionale della somma dei quadrati delle distanze fra le persone

Figura 5.21: *Ambiente di simulazione scenario 3*

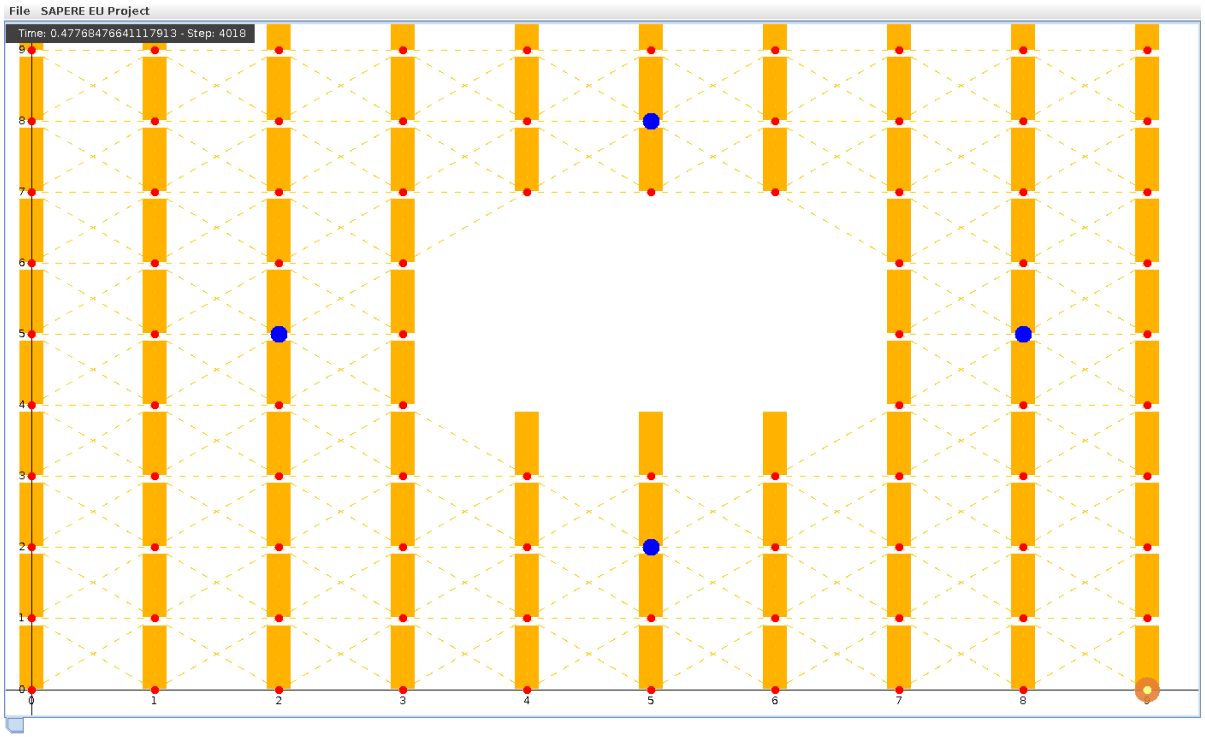


(a) Situazione iniziale

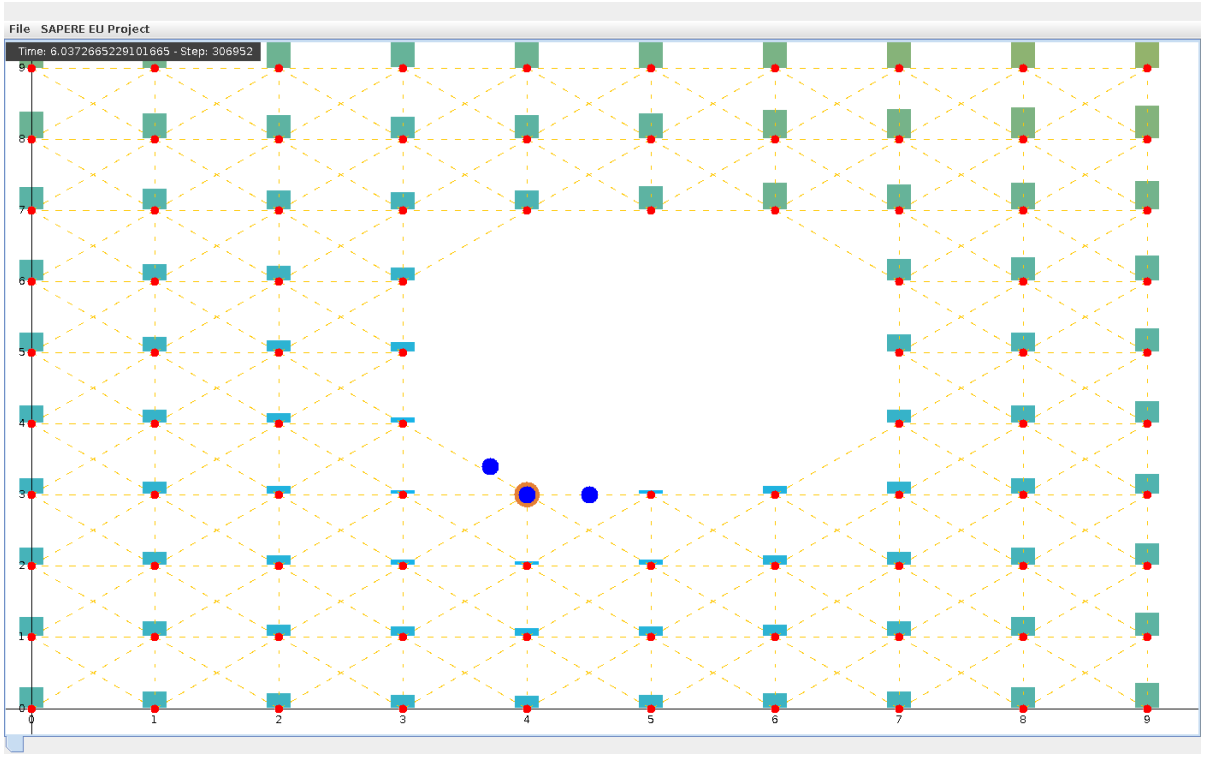


(b) Situazione finale, le persone si stanno dirigendo verso il punto eletto essere il baricentro

Figura 5.22: *Ambiente di simulazione scenario 4 con token in posizione iniziale (0,0)*

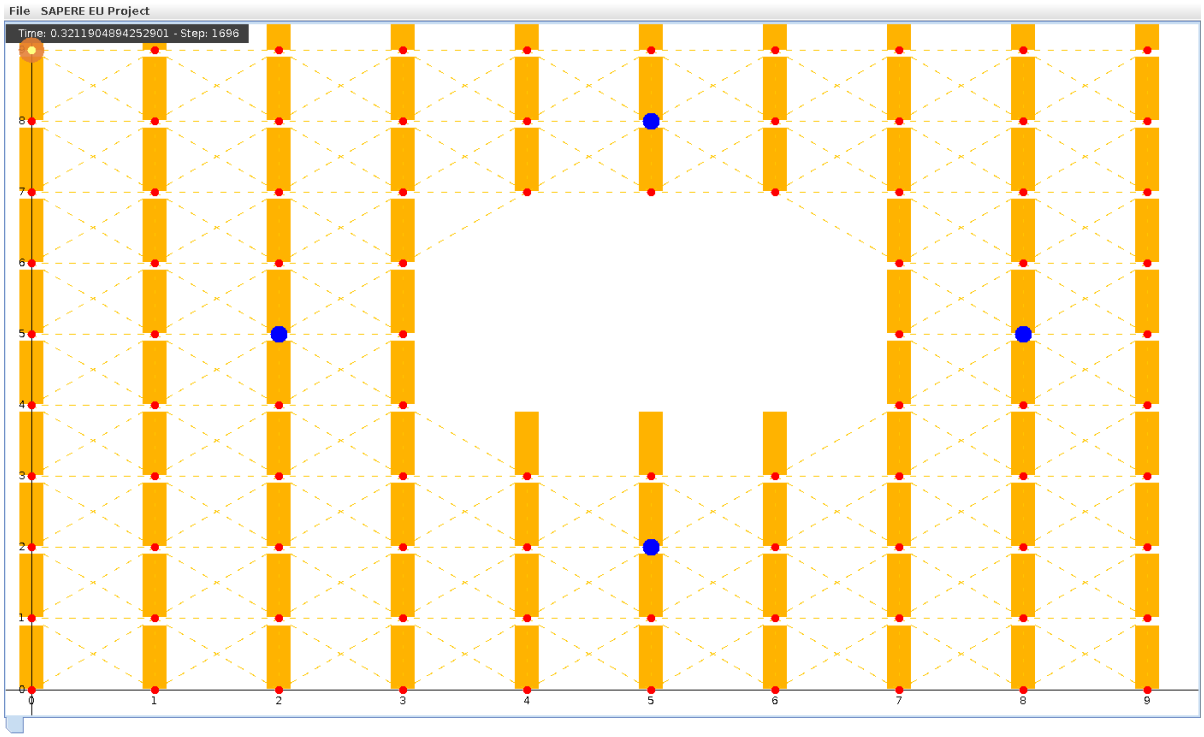


(a) Situazione iniziale

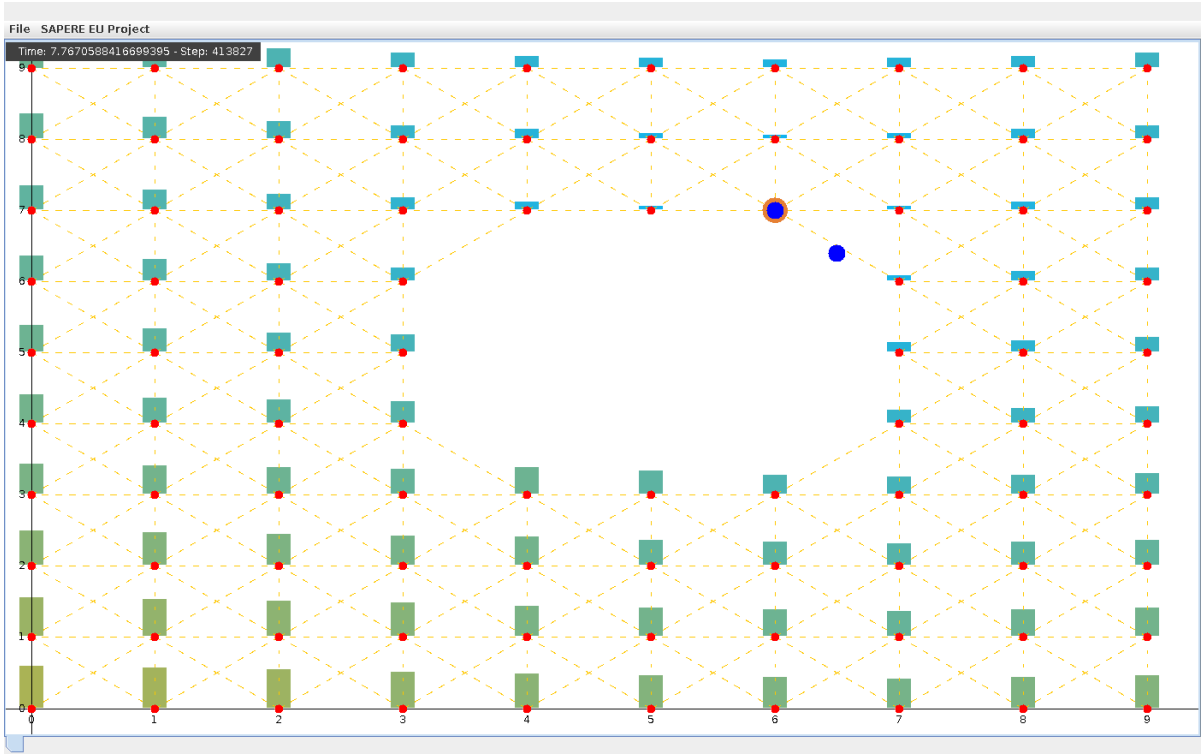


(b) Situazione finale, le persone si stanno dirigendo verso il punto eletto essere il baricentro

Figura 5.23: *Ambiente di simulazione scenario 4 con token in posizione iniziale (9,0)*

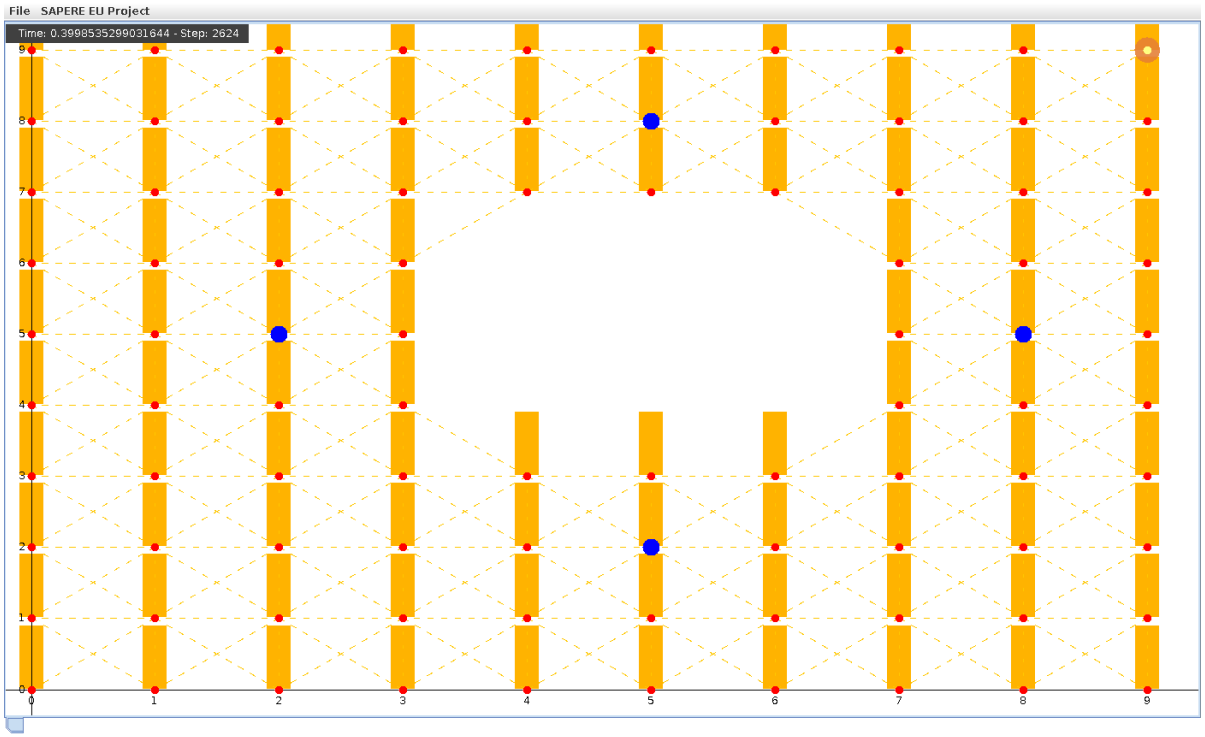


(a) Situazione iniziale

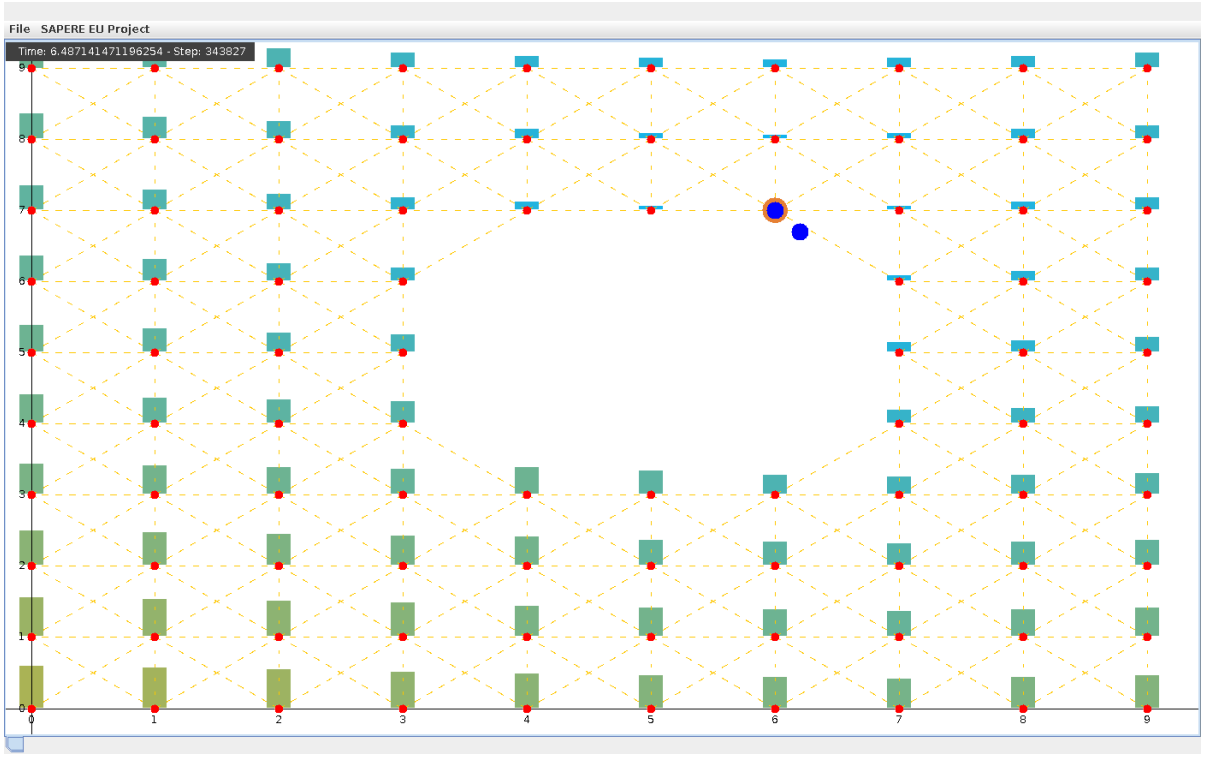


(b) Situazione finale, le persone si stanno dirigendo verso il punto eletto essere il baricentro

Figura 5.24: *Ambiente di simulazione scenario 4 con token in posizione iniziale (0,9)*



(a) Situazione iniziale



(b) Situazione finale, le persone si stanno dirigendo verso il punto eletto essere il baricentro

Figura 5.25: *Ambiente di simulazione scenario 4 con token in posizione iniziale (9,9)*

6
Conclusioni e sviluppi futuri

In questa tesi l'obiettivo è stata la progettazione di un nuovo algoritmo di auto-organizzazione basato sul gradiente computazionale ed è possibile concludere che osservando i risultati presentati nel capitolo 5 tale obiettivo è stato raggiunto. In questo capitolo sono riassunti i contributi sperimentali della presente tesi e sono proposti alcuni sviluppi futuri.

6.1 Riassunto e contributi

Per supportare l'adattività e la *context-awareness* nelle applicazioni dell'ambito della computazione pervasiva, in questa tesi sono stati sfruttati i meccanismi di auto-organizzazione presenti nei sistemi naturali, i quali attraverso interazioni esclusivamente locali riescono a garantire proprietà emergenti di auto-organizzazione dell'intero sistema. Lavori recenti hanno identificato e, successivamente, classificato una serie di *pattern* di auto-organizzazione che possono essere iniettati nel framework *framework* di ecosistemi pervasivi. Tale *framework* è stato sviluppato all'interno del progetto europeo `SAPERE` basato sull'idea che ogni agente (servizio, dispositivo, persona) manifesta la propria esistenza nell'ecosistema attraverso *Live Semantic Annotation* (LSA) e che le attività degli agenti vengano coordinate attraverso un *set* limitato e fissato di *eco-law*, le quali risultano essere reazioni di tipo *chemical-like* che permettono l'evoluzione delle LSA distribuita nell'ecosistema.

Il contributo principale è stato quello di fornire un nuovo algoritmo di auto-organizzazione basato sui gradienti computazionali nel contesto del *pervasive computing*. L'obiettivo dell'algoritmo è quello di consentire il ricongiungimento di un gruppo di persone distribuite nello spazio in un unico punto. Il punto di *meeting* scelto è il baricentro spaziale rispetto alle posizioni iniziali delle persone. In particolare, il baricentro spaziale viene visto come il punto nel quale la somma al quadrato fra le distanze fisiche delle persone risulta essere minore.

L'approccio per la progettazione di tale algoritmo è stato iterativo e si è basato su quattro fasi:

- **modellazione**: proporre un modello per il sistema sulla base dei design pattern esistenti;
- **simulazione**: analisi delle dinamiche in diversi scenari;
- **validazione**: verifica che le proprietà di interesse siano valide in ogni scenario dell'ambiente;
- **tuning**: regolare il comportamento del sistema ed elaborare un insieme di parametri per il sistema.

L'idea alla base delle modello è che l'algoritmo identifica il punto baricentrico sulla base della creazione di un gradiente computazionale per ogni persona. Dato il ruolo chiave del gradiente computazionale in questo algoritmo, sono state studiate varie modalità di creazione di gradienti computazionali disponibili in letteratura. In particolare, si è studiato in modo approfondito il gradiente computazionale costituito dall'approccio `SAPERE` [18] e dall'approccio `NBR` [3]. In letteratura del gradiente `NBR` era disponibile il modello matematico e su tale base si è progettato un *set* di *eco-law* e LSA per modellarlo secondo l'approccio degli ecosistemi pervasivi. Si è poi effettuata un'approfondita analisi sia sulla qualità che sulle *performance* dei due approcci, con il risultato che il gradiente `NBR` è significativamente più stabile ma nel contempo richiede maggiore lavoro computazionale al sistema.

L'algoritmo è stato validato attraverso l'esecuzione di simulazioni di vari scenari eseguite sul framework Alchemist, sviluppato nel gruppo di ricerca della Seconda Facoltà di Ingegneria appositamente per gestire le astrazioni proprie degli ecosistemi pervasivi. Il gradiente `NBR` , ulteriore contributo di questa tesi, ha stressato notevolmente il framework Alchemist, su cui sono state eseguire le simulazioni, a causa della richiesta di gestire sia *eco-law* contenenti strutture dati, sia di un'elevata quantità di *LSA* in tutti i nodi dell'ambiente. In particolare si ha che entrambe queste caratteristiche aumentano di impegno computazionale con l'aumentare del numero di nodi e il numero di persone inseriti nell'ambiente.

Si è effettuata, inoltre, una massiccia operazione di *tuning* per ottenere l'emergere del comportamento globale del sistema desiderato.

Il risultato delle simulazioni dimostra che l'algoritmo realizzato appare robusto ad ostacoli nell'ambiente sia "dinamici" che statici, ovvero presenti nella configurazione iniziale del sistema (ad esempio muri) o che si presentano successivamente (ad esempio zone affollate da evitare). Le persone riescono a riunirsi a partire da qualsiasi loro disposizione iniziale, dimensione della griglia e numero di ostacoli.

Si ritiene a questo punto utile effettuare delle brevi considerazioni sullo stato delle tecnologie utilizzate. Il framework Alchemist essendo uno strumento di ricerca risulta essere in uno stato preliminare. Di qui alcune difficoltà nello sviluppo del modello fra cui documentazione non esaustiva e nella mancanza di un sistema di *debug*.

6.2 Sviluppi futuri

Prospettive per il futuro includono:

- una approfondita analisi delle prestazioni del framework ALCHEMIST, confrontando le soluzioni con e senza l'adozione del gradiente `NBR`. Ad esempio si propone di modellare il gradiente `NBR` attraverso un approccio ibrido, ovvero senza la cache che contiene tutti i valori ricevuti,
- i *design pattern*, per loro stessa natura, tendono a sottolineare i problemi della fase di analisi e progettazione del sistema; al contrario i sistemi auto-organizzanti enfatizzando le dinamiche di sistema, di conseguenza può essere considerato utile elaborare nuovi approcci per la modellazione delle strategie, come ad esempio in questa tesi è stato il gradiente `NBR`,
- studiare altri modelli per l'analisi nell'ambito degli ecosistemi pervasivi e *context-aware computing*.

A livello tecnologico si propone l'introduzione di una piattaforma concorrente sulla quale effettuare simulazioni al fine di ridurne in modo massiccio i tempi di attesa dei risultati.

Bibliografia

[1] Mit proto. software available at `http://proto.bbn.com/`, November 2008.

[2] Spatial computing. `http://www.spatial-computing.org/start`, 2012.

[3] Jacob Beal, Jonathan Bachrach, Dan Vickery, and Mark Tobenkin. Fast self-healing gradients. In *Proceedings of the 2008 ACM symposium on Applied computing*, SAC '08, pages 1969–1975, New York, NY, USA, 2008. ACM.

[4] Jacob Beal, Stefan Dulman, Kyle Usbeck, Mirko Viroli, and Nikolaus Correll. Organizing the aggregate: Languages for spatial computing. *CoRR*, abs/1202.5509, 2012.

[5] Jacob Beal, Marco Mamei, and Christian Borcea. Spatial approaches to pervasive computing. `http://openmap.bbn.com/~jbeal/Tutorials/Spatial-Pervasive-SASO2008.pdf`, 2008.

[6] Jose Luis Fernandez-Marquez, Giovanna Di Marzo Serugendo, Sara Montagna, Mirko Viroli, and Josep Lluis Arcos. Description and composition of bio-inspired design patterns: a complete overview. *Natural Computing*, May 2012. Online First.

[7] M. A. Gibson and J. Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *The Journal of Physical Chemistry A*, 104(9):1876–1889, March 2000.

[8] Michael A. Gibson and Jehoushua Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *The Journal of Physical Chemistry A*, 104(9):1876–1889, March 2000.

[9] H. Haken. *Information and Self-organization: A Macroscopic Approach to Complex Systems*. Springer, Berlin, 2 edition, 2000.

[10] Charles M. Macal and Michael J. North. Tutorial on agent-based modelling and simulation. *Journal of Simulation*, 4(3):151–162, 2010.

[11] Sara Montagna, Mirko Viroli, Matteo Risoldi, Danilo Pianini, and Giovanna Di Marzo Serugendo. Self-organising pervasive ecosystems: A crowd evacuation example. In *3rd International Workshop on Software Engineering for Resilient Systems*, volume 6968 of *Lecture Notes in Computer Science*, pages 115–129. Springer, Geneva, Switzerland, 29–30 September 2011.

[12] Danilo Pianini. Un framework di simulazione per ecosistemi di servizi pervasivi. Master's thesis, Seconda facoltà di Ingegneria, Cesena, 2011.

[13] Danilo Pianini. The alchemist simulator full manual. Technical report, Università di Bologna, May 2012.

[14] Danilo Pianini. Pervasive service ecosystems. `http://aixia.deis-ce.unibo.it/poster/poster.pdf`, 2012.

[15] Danilo Pianini, Sara Montagna, and Mirko Viroli. A chemical inspired simulation framework for pervasive services ecosystems. In Maria Ganzha, Leszek Maciaszek, and Marcin Paprzycki, editors, *Proceedings of the Federated Conference on Computer Science and Information Systems (FedCSIS 2011)*, pages 667–674, Szczecin, Poland, 18-21 September 2011. IEEE Computer Society Press.

[16] Giacomo Pronti. Simulazione di ecosistemi di servizi pervasivi con supporto ad annotazioni tuple based. Master's thesis, Seconda facoltà di Ingegneria, Cesena, 2012.

[17] Mirko Viroli, Elena Nardini, Gabriella Castelli, Marco Mamei, and Franco Zambonelli. A coordination approach to adaptive pervasive service ecosystems. In *1st Awareness Workshop "Challenges in achieving self-awareness in autonomous systems" (AWARE 2011)*. SASO 2011, Ann Arbor, MI, USA, 7 October 2011.

[18] Mirko Viroli, Danilo Pianini, Sara Montagna, and Graeme Stevenson. Pervasive ecosystems: a coordination model based on semantic chemistry. In Sascha Ossowski, Paola Lecca, Chih-Cheng Hung, and Jiman Hong, editors, *27th Annual ACM Symposium on Applied Computing (SAC 2012)*, Riva del Garda, TN, Italy, 26-30 March 2012. ACM.

A

DSL gradiente SAPERE

```
environment
type InfiniteHalls params "8"
with random seed 21

place 100 nodes in rect (0,0,9,9) interval 1

containing
in point (0,0) <pump,0>
in all <sensor>

with reactions

eco-law pump
[<pump,Tstamp>] --> [<field,0,Tstamp>]

eco-law spread
[<field,V,T>] -10-> [<field,V,T>, *<field,V+#D,T>]

eco-law youngest
[<field,V,T>, <field,V1, def: T1 > T>]
-->
[<field,V1,T1>]

eco-law shortest
[<field,V,T>, <field, def: V1 <= V,T>]
-->
[<field,V1,T>]
```

B

DSL gradiente SAPERE con prima pump forzata

```
environment
type InfiniteHalls params "8"
with random seed 26

place 100 nodes in rect (0,0,9,9) interval 1

containing
in point (0,0) <pump,0>
in all <sensor>

with reactions

eco-law pump
[<pump,0>] --> [<pump,1>, <field,0,0>]

[<pump, def: Tstamp>0>] -0.1-> [<pump,Tstamp+1>, <field,0,Tstamp>]

eco-law spread
[<field,Value,Tstamp>]
-10->
[<field,Value,Tstamp>, *<field,Value+#D,Tstamp>]

eco-law youngest
[<field,Value,Tstamp>, <field,Value1, def: Tstamp1 > Tstamp>]
-->
[<field,Value1,Tstamp1>]

eco-law shortest
```

```
[<field,Value,Tstamp>, <field, def: Value1 <= Value,Tstamp>]
-->
[<field,Value1,Tstamp>]
```

C

DSL gradiente NBR

```
environment
type InfiniteHalls params "8"
with random seed 15

place 100 nodes in rect (0,0,9,9) interval 1

containing
in all
<field,barycenter,0,100>
<tempfield,barycenter,[],100>
<nameSource, barycenter>


in point (0,0) <id,sensore0>   <barycenter,barycenter>
in point (1,0) <id,sensore1>   in point (2,0) <id,sensore2>
in point (3,0) <id,sensore3>   in point (4,0) <id,sensore4>
in point (5,0) <id,sensore5>      in point (6,0) <id,sensore6>
in point (7,0) <id,sensore7>  in point (8,0) <id,sensore8>
in point (9,0) <id,sensore9>

in point (0,1) <id,sensore10>   in point (1,1) <id,sensore11>
in point (2,1) <id,sensore12> in point (3,1) <id,sensore13>
in point (4,1) <id,sensore14>   in point (5,1) <id,sensore15>
  in point (6,1) <id,sensore16>   in point (7,1) <id,sensore17>
in point (8,1) <id,sensore18> in point (9,1) <id,sensore19>

  in point (0,2) <id,sensore20>  in point (1,2) <id,sensore21>
in point (2,2) <id,sensore22>in point (3,2) <id,sensore23>
in point (4,2) <id,sensore24>  in point (5,2) <id,sensore25>
  in point (6,2) <id,sensore26>in point (7,2) <id,sensore27>
```

```
in point (8,2) <id,sensore28>in point (9,2) <id,sensore29>

  in point (0,3) <id,sensore30>  in point (1,3) <id,sensore31>
in point (2,3) <id,sensore32>in point (3,3) <id,sensore33>
in point (4,3) <id,sensore34>  in point (5,3) <id,sensore35>
  in point (6,3) <id,sensore36>in point (7,3) <id,sensore37>
in point (8,3) <id,sensore38>in point (9,3) <id,sensore39>

  in point (0,4) <id,sensore40>  in point (1,4) <id,sensore41>
in point (2,4) <id,sensore42>in point (3,4) <id,sensore43>
in point (4,4) <id,sensore44>  in point (5,4) <id,sensore45>
  in point (6,4) <id,sensore46>  in point (7,4) <id,sensore47>
in point (8,4) <id,sensore48>in point (9,4) <id,sensore49>

  in point (0,5) <id,sensore50>  in point (1,5) <id,sensore51>
in point (2,5) <id,sensore52>in point (3,5) <id,sensore53>
in point (4,5) <id,sensore54>  in point (5,5) <id,sensore55>
  in point (6,5) <id,sensore56>in point (7,5) <id,sensore57>
in point (8,5) <id,sensore58>in point (9,5) <id,sensore59>

  in point (0,6) <id,sensore60>  in point (1,6) <id,sensore61>
in point (2,6) <id,sensore62>in point (3,6) <id,sensore63>
in point (4,6) <id,sensore64>  in point (5,6) <id,sensore65>
  in point (6,6) <id,sensore66>in point (7,6) <id,sensore67>
in point (8,6) <id,sensore68>in point (9,6) <id,sensore69>

  in point (0,7) <id,sensore70>  in point (1,7) <id,sensore71>
in point (2,7) <id,sensore72>in point (3,7) <id,sensore73>
in point (4,7) <id,sensore74>  in point (5,7) <id,sensore75>
  in point (6,7) <id,sensore76>in point (7,7) <id,sensore77>
in point (8,7) <id,sensore78>in point (9,7) <id,sensore79>

  in point (0,8) <id,sensore80>  in point (1,8) <id,sensore81>
in point (2,8) <id,sensore82>in point (3,8) <id,sensore83>
in point (4,8) <id,sensore84>  in point (5,8) <id,sensore85>
  in point (6,8) <id,sensore86>in point (7,8) <id,sensore87>
in point (8,8) <id,sensore88>in point (9,8) <id,sensore89>

  in point (0,9) <id,sensore90>  in point (1,9) <id,sensore91>
in point (2,9) <id,sensore92>in point (3,9) <id,sensore93>
```

```
in point (4,9) <id,sensore94>  in point (5,9) <id,sensore95>
  in point (6,9) <id,sensore96>in point (7,9) <id,sensore97>
in point (8,9) <id,sensore98>in point (9,9) <id,sensore99>

with reactions

  eco-law worseMsgValueThanCurrent
[<msg, Type ,Tstamp,Node,Dist>,
<tempfield, Type, def: L hasnot [Node;], def: Dist1<= Dist>]
-->
[<tempfield, Type, Node add L, Dist1>, <msg,Type,Tstamp,Node,Dist>]

eco-law betterMsgValueThanCurrent
[<msg,Type,Tstamp,Node,Dist>,
<tempfield, Type, def: L hasnot [Node;], def: Dist1> Dist>]
-->
[<tempfield, Type, Node add L, Dist>, <msg,Type,Tstamp,Node,Dist>]

  eco-law youngestMsg
[<msg, Type, Tstamp,Node,Dist>,
<msg,Type,def: Tstamp1>Tstamp,Node,Dist1>]
-->
[<msg,Type,Tstamp1,Node,Dist1>]

[<nameSource, B>, <msg,Type,Tstamp,def: Node!=B,Dist>]
-0.01->
[<nameSource, B>]

eco-law assignField
[<tempfield, Type, def: L notempty, Dist>, <field, Type, Tstamp, Dist1>,
<id, Name>]
-10->
[<tempfield,Type,[],100>, <field,Type,Tstamp+1,Dist>,
*<msg,Type,Tstamp,Name,Dist+#D>, <id, Name>]

eco-law pump
[<barycenter,Type>] --> [*<msg,Type,0,Type,#D>]
```

D
DSL risparmio energetico

```
environment
type InfiniteHalls params "8"
with random seed 21

/* Types of LSA:
 * <msg,Type,Tstamp,Node,Dist> => the message
 * <field, Type, Tstamp, inf> => the tuple field (inf = 100)
 * <tempfield, Type, ListNode, inf> => the field that we are computing
 * <rate, Node, R> => the current rate of reaction (Rmin <= R <= Rmax)
 * <rateMin, Rmin> => the minimum rate
 * <rateMax, Rmax> => the maximum rate
 */

/* ######################## sensor nodes ######################## */

place 6 nodes in rect (0,0,2,2) interval 1

containing
in all <field,Type,0,100> <tempfield,Type,[],100> <rateMin,1> <rateMax,4>
in point (0,0) <id,sensore0> <rate,sensore0,4>
in point (1,0) <id,sensore1> <rate,sensore1,4>
in point (2,0) <id,sensore2> <rate,sensore2,4>
in point (0,1) <id,sensore3> <rate,sensore3,4>
in point (1,1) <id,sensore4> <rate,sensore4,4>
in point (2,1) <id,sensore5> <rate,sensore5,4>

with reactions

/* new msg from a node not in list */
```

```
eco-law worseMsgValueThanCurrent
[<msg,Type,Tstamp,Node,Dist>,
<tempfield, Type, def: L hasnot [Node;], def: Dist1<= Dist>]
-->
[<tempfield, Type, Node add L, Dist1>, <msg,Type,Tstamp,Node,Dist>]

eco-law betterMsgValueThanCurrent
[<msg,Type,Tstamp,Node,Dist>,
<tempfield, Type, def: L hasnot [Node;], def: Dist1> Dist>]
-->
[<tempfield, Type, Node add L, Dist>, <msg,Type,Tstamp,Node,Dist>]

/* more msg from same node, choose the youngest */

eco-law youngestMsg
[<msg,Type,Tstamp,Node,Dist>,
<msg,Type,def: Tstamp1>Tstamp,Node,Dist1>]
-->
[<msg,Type,Tstamp1,Node,Dist1>]

/* decay link */

eco-law decay
[<msg,Type,Tstamp1,source,Dist1>, <msg,Type,Tstamp,Node,Dist>]
-0.01->
[<decay,Type,Tstamp,Node,Dist>, <msg,Type,Tstamp1,source,Dist1>]

/* assign the value at field */

/* eco-law assignField
[<tempfield, Type, def: L notempty, Dist>, <field, Type, Tstamp, Dist1>,
<id, Name>]
-5->
[<tempfield,Type,[],100>, <field,Type,Tstamp+1,Dist>,
*<msg,Type,Tstamp,Name,Dist+#D>, <id, Name>] */


/* both have Dist and current rate higher than min rate */
[<tempfield, Type, def: L notempty, Dist>, <field, Type, Tstamp, Dist>,
<id, Name>, <rateMin,Rmin>, <rate, Name, def: R > Rmin>]
```

```
-R->
[<tempfield,Type,[],100>, <field,Type,Tstamp+1,Dist>,
*<msg,Type,Tstamp,Name,Dist+#D>, <id, Name>, <rate, Name, R/2>,
<rateMin,Rmin>]

/* both have Dist and current rate lower or equal than min rate */
[<tempfield, Type, def: L notempty, Dist>, <field, Type, Tstamp, Dist>,
<id, Name>, <rateMin,Rmin>, <rate, Name, def: R <= Rmin>]
-R->
[<tempfield,Type,[],100>, <field,Type,Tstamp+1,Dist>,
*<msg,Type,Tstamp,Name,Dist+#D>, <id, Name>, <rate, Name, Rmin>,
<rateMin,Rmin>]

/* different values of distance */
[<tempfield, Type, def: L notempty, Dist>,
<field, Type, Tstamp, def: Dist1 != Dist>,
<id, Name>, <rateMax,Rmax>, <rate, Name, def: R <= Rmax>]
-R->
[<tempfield,Type,[],100>, <field,Type,Tstamp+1,Dist>,
*<msg,Type,Tstamp,Name,Dist+#D>, <id, Name>, <rate, Name, Rmax>,
<rateMax,Rmax>]


/* ######################## source node ######################## */

place node at point (0.5,0.5)
containing <person> <source,target0>
with reactions
eco-law init [<source,Type>] --> [*<msg,Type,0,source,#D>]
```

E

DSL algoritmo di join

```
environment
type InfiniteHalls params "8"
with random seed 21

lsa k <token>

place 64 nodes in rect (0,0,7,7) interval 1

containing
in all  <f,t0,0,100> <f,t1,0,100>
<f,t2,0,100> <f,t3,0,100>
<f,t4,0,100> <f,t5,0,100>
<f,b,0,100>
<t,t0,[],100> <t,t1,[],100>
<t,t2,[],100> <t,t3,[],100>
<t,t4,[],100> <t,t5,[],100>
<t,b,[],100>
<s,100> <ts,[],0>
<nb, b>
<l, [p1;p2;p3;p4;p5;p6;]>

in point (0,0) <n,s0>  in point (1,0) <n,s1>  in point (2,0) <n,s2>
in point (3,0) <n,s3>  in point (4,0) <n,s4>  in point (5,0) <n,s5>
in point (6,0) <n,s6>  in point (7,0) <n,s7>

in point (0,1) <n,s8>  in point (1,1) <n,s9>  in point (2,1) <n,s10>
in point (3,1) <n,s11>  in point (4,1) <n,s12>  in point (5,1) <n,s13>
  in point (6,1) <n,s14>  in point (7,1) <n,s15>

  in point (0,2) <n,s16>  in point (1,2) <n,s17>  in point (2,2) <n,s18>
```

```
in point (3,2) <n,s19>  in point (4,2) <n,s20>  in point (5,2) <n,s21>
  in point (6,2) <n,s22>in point (7,2) <n,s23>

  in point (0,3) <n,s24>  in point (1,3) <n,s25>  in point (2,3) <n,s26>
in point (3,3) <n,s27>  in point (4,3) <n,s28>  in point (5,3) <n,s29>
  in point (6,3) <n,s30>in point (7,3) <n,s31>

  in point (0,4) <n,s32>  in point (1,4) <n,s33>  in point (2,4) <n,s34>
in point (3,4) <n,s35>  in point (4,4) <n,s36>  in point (5,4) <n,s37>
  in point (6,4) <n,s38>  in point (7,4) <n,s39>

  in point (0,5) <n,s40>  in point (1,5) <n,s41>  in point (2,5) <n,s42>
in point (3,5) <n,s43>  in point (4,5) <n,s44>  in point (5,5) <n,s45>
  in point (6,5) <n,s46>in point (7,5) <n,s47>

  in point (0,6) <n,s48>  in point (1,6) <n,s49>  in point (2,6) <n,s50>
in point (3,6) <n,s51>  in point (4,6) <n,s52>  in point (5,6) <n,s53>
  in point (6,6) <n,s54>in point (7,6) <n,s55>

  in point (0,7) <n,s56>  in point (1,7) <n,s57>  in point (2,7) <n,s58>
in point (3,7) <n,s59>  in point (4,7) <n,s60>  in point (5,7) <n,s61>
  in point (6,7) <n,s62>in point (7,7) <n,s63>

with reactions

  eco-law worseMsgValueThanCurrent
[<m, T ,Ts,N,D>, <t, T, def: L hasnot [N;], def: D1<= D>]
-->
[<t, T, N add L, D1>, <m,T,Ts,N,D>]

eco-law betterMsgValueThanCurrent
[<m,T,Ts,N,D>, <t, T, def: L hasnot [N;], def: D1> D>]
-->
[<t, T, N add L, D>, <m,T,Ts,N,D>]

  eco-law youngestMsg
[<m, T, Ts,N,D>, <m,T,def: Ts1>Ts,N,D1>]
-->
[<m,T,Ts1,N,D1>]
```

```
eco-law decay
[<m,T,Ts,N,D>, <l, def: L hasnot [N;]>] -1-> [<l,L>]

eco-law assignfield
[<t, T, def: L notempty, D>, <f, T, Ts, D1>, <n, Na>]
-1->
[<t,T,[],100>, <f,T,Ts+1,D>, *<m,T,Ts,Na,D+#D>, <n, Na>]

eco-law assignbarycenter
[<t, b, def: L notempty, D>, <f, b, Ts, D1>, <n, Na>]
-20->
[<t,b,[],100>, <f,b,Ts+1,D>, *<m,b,Ts,Na,D+#D>, <n, Na>]

eco-law sum
[<nb, B>, <f, def: T!=B, Ts, def: D != 100>, <ts, def: L hasnot [T;], S>]
-->
[<ts, T add L, S+(D*D)>, <f, T, Ts,D>, <nb, B>]

[<s, S> <ts, def: L notempty, V>] -1000-> [<s,V>,<ts, [],0>]

eco-law token
[k, <m,b,Ts,Na,0>] -10-> [action LsaMoveToken params "ENV,NODE"]

place node at point (4.25,0.25)
containing <p> <poi> k
with reactions
[k] -10-> [action LsaMoveToken params "ENV,NODE"]


place node at point (2,7)
containing <p> <p1,t0>
with reactions
[<S,T>] --> [*<m,T,0,S,#D>]
[]
-3->
[agent LsaBarycenterAgent params "REACTION,ENV,NODE,RANDOM,1"]


place node at point (4,7)
containing <p> <p2,t1>
```

```
with reactions
[<S,T>] --> [*<m,T,0,S,#D>]
[]
-3->
[agent  LsaBarycenterAgent params "REACTION,ENV,NODE,RANDOM,1"]


place node at point (1,6)
containing <p> <p3,t2>
with reactions
[<S,T>] --> [*<m,T,0,S,#D>]
[]
-3->
[agent  LsaBarycenterAgent params "REACTION,ENV,NODE,RANDOM,1"]


place node at point (5,6)
containing <p> <p4,t3>
with reactions
[<S,T>] --> [*<m,T,0,S,#D>]
[]
-3->
[agent  LsaBarycenterAgent params "REACTION,ENV,NODE,RANDOM,1"]

place node at point (2,5)
containing <p> <p5,t4>
with reactions
[<S,T>] --> [*<m,T,0,S,#D>]
[]
-3->
[agent  LsaBarycenterAgent params "REACTION,ENV,NODE,RANDOM,1"]

place node at point (4,5)
containing <p> <p6,t5>
with reactions
[<S,T>] --> [*<m,T,0,S,#D>]
[]
-3->
[agent  LsaBarycenterAgent params "REACTION,ENV,NODE,RANDOM,1"]
```

F

Manuale sistema di reportistica

L'idea alla base del creare un sistema di reportistica è stata quella di rendere più agevole l'esecuzione delle simulazioni e l'elaborazione dei dati derivanti da esse. Le simulazioni, difatti, ricoprono un ruolo fondamentale in questo contesto di ricerca, ma al contempo risultano essere molto onerose sia in termini di tempo di esecuzione sia come risorse di calcolo richieste.

F.1 Organizzazione file e cartelle

Per l'elaborazione dei dati è necessario organizzare le cartelle contenenti i file nei seguenti livelli:

1. la prima cartella di nome *simulations*;
2. la quale contiene tante cartelle quanti sono i blocchi di simulazioni per parametro da eseguire (ad esempio nel gradiente `SAPERE` il parametro da variare è stato quello del rate della *eco-law* di *pump* e quindi sono state create ad esempio le cartelle 0.01, 0.1, 1);
3. ogni cartella contiene tanti file *xml* quante sono le simulazioni da eseguire.

F.1.1 Comprensione file generati

All'interno di ogni cartella di ogni blocco di simulazioni (livello 3) si generano: una cartella di nome *temp* nella quale vengono inseriti i file *dat* che contengono l'elenco di tutte le reazioni che scattano nella simulazione nel nodo in esame ed un file di testo chiamato *report*. Nel dettaglio, un file *dat* contiene tre colonne: il tempo di simulazione, il numero di reazione ed il valore del gradiente nel nodo. La scelta di generare un file così dettagliato è motivata dal fatto che questo potrebbe essere l'input di più processi di analisi. D'altra parte si noti che tali file per simulazioni con una lunga durata, o comunque in caso scattino un numero elevato di simulazioni in un nodo, risultano diventare velocemente di grandi dimensioni. Le simulazioni durante il periodo della tesi sono state

svolte su un server con una disponibilità di spazio su disco limitata, per questo motivo nel codice corrente si è scelto di memorizzare tali file in una cartella *temp* che verrà poi cancellata una volta che i file in essa contenuti non risultano essere più utili. Il file di testo *report* è stato suddiviso in due sezioni:

- nella prima sezione vi è un'analisi dei *glitch* e delle *performance* per ogni singolo file;
- nella seconda sezione vi sono le statiche globali di tutti i file contenuti nella cartella corrente.

La prima sezione contiene:

- il nome del file *dat* esaminato;
- una tabella in cui per ogni *glitch* si ha l'ampiezza massima e media ed il periodo;
- l'analisi dei *glitch*, che include:
  - il numero di *glitch* che si verificano durante la simulazione;
  - l'ampiezza massima e media dei *glitch*;
  - la percentuale di tempo rispetto alla durata della simulazione in cui il nodo in esame assume un valore scorretto (ovvero diverso dall'ottimo);
  - il periodo massimo, medio e la deviazione standard;
  - l'errore, ovvero la differenza fra il valore assunto nel nodo e il suo ottimo, medio e massimo ed il calcolo delle rispettive deviazioni standard;
- l'analisi delle *performance*, che include:
  - il numero totale di reazioni che scattate nel nodo in esame nell'intera durata della simulazione;
  - la frequenza con cui nel nodo sono scattate le reazioni;
  - il tempo trascorso ed il numero di reazioni scattate per far raggiungere la stabilità nel nodo, ovvero la prima volta che il nodo assume il valore ottimo;
- una eventuale lista di *glitch* pendenti, ovvero i *glitch* che non terminato prima della fine della simulazione. Tali *glitch* nella presente analisi sono stati esclusi poiché non terminando si sarebbe dovuta forzare la loro terminazione con la fine della simulazione, per completezza però sono stati tratti a parte.

La seconda sezione contiene:

- l'analisi dei *glitch*:

  - i valori di: numero di *glitch*, ampiezza massima e media, periodo e percentuale di tempo in cui il nodo assume un valore scorretto rispetto alla durata della simulazione. Per ogni valore è stata calcolata la media e la deviazione standard;
  - gli stessi valori presentati sopra anche per i *glitch* pendenti;

- l'analisi delle performance:
  - la media, la deviazione standard e la frequenza delle reazioni scattate in tutte le simulazioni. Rispetto alla frequenza si è calcolata la media e la deviazione standard;
  - il tempo in cui si verifica la prima stabilità nel nodo in termini di tempo e numero di reazioni, per entrambi si è calcolato sia la media che la deviazione standard.

All'interno della cartella *simulations* (livello 2), invece, si genera un file di testo chiamato *globalReport* il quale contiene sette colonne:

- il parametro considerato che viene associato al nome della cartella nel livello 2;
- la percentuale di tempo in cui il nodo assume un valore scorretto rispetto alla durata della simulazione;
- la deviazione standard della percentuale indicata sopra;
- l'errore massimo medio del blocco di simulazioni con il parametro scelto;
- la deviazione standard dell'errore medio indicato sopra;
- l'errore medio del blocco di simulazioni con il parametro scelto;
- la deviazione standard dell'errore medio indicato sopra.

Tale file è stato pensato per essere posto in ingresso ad un programma per produrre grafici, ad esempio nella presente tesi è stato utilizzato gnuplot.

F.2 Utilizzo sistema

Il sistema può essere facilmente utilizzato creando un *runnable jar* della classe *SamplingSimulationLoader* inserita all'interno del *plugin-sapereexperiments* del progetto ALCHEMIST. Tale *jar* può essere lanciato dalla console attraverso il semplice comando *java -jar nomeFile.jar*. L'esecuzione del comando comporta la richiesta di inserimento dei parametri di:

- tipo di gradiente da generare (`SAPERE` o `NBR`);
- durata della simulazione;
- nodo da campionare;
- valore ottimo del nodo da campionare.

L'introduzione di tali parametri fa sì che uno stesso *jar* possa essere utilizzato per l'esecuzioni di tipologie diverse di simulazioni.