ALMA MATER STUDIORUM
UNIVERSITÀ DEGLI STUDI DI BOLOGNA

Scuola di Ingegneria e Architettura
Corso di Laurea in Ingegneria Biomedica

SVILUPPO DEL SIMULATORE ALCHEMIST PER LA MODELLAZIONE DI MOVIMENTO CELLULARE

Elaborata nel corso di: Fondamenti di Informatica

Tesi di Laurea di:
FRANCO PRADELLI

Relatore:
Prof. MIRKO VIROLI

Correlatore:
Dott. SARA MONTAGNA

ANNO ACCADEMICO 2015–2016
SESSIONE II

PAROLE CHIAVE

Computational Systems Biology
Multicellular Systems Biology
Alchemist
Movimento cellulare
Modellazione

Indice

Introduzione

Obbiettivo del mio progetto di tesi è stata l'estensione dell'incarnazione biochimica del simulatore Alchemist [19], cioè di quella parte del simulatore dedicata alla modellazione degli ambienti multicellulari. Questa estensione è stata operata per fornire il supporto al movimento cellulare, un fenomeno che, come vedremo, riveste una grande importanza in molti eventi di rilevanza biologica e fisiologica. Il progetto rappresenta, però, solo una piccola parte di un lavoro che, speriamo, assumerà dimensioni sempre più importanti in futuro. Riteniamo, infatti, che il lavoro legato ai simulatori e al loro costante miglioramento sarà una grossa risorsa per la ricerca e per la tecnologia, specialmente in campo biologico. Questo perché essere in grado di simulare in maniera adeguata un sistema multicellulare significa poter avere una libertà di indagine molto superiore a quello della semplice sperimentazione. La classica sperimentazione, infatti, presenta tutti i problemi derivati dal fatto che si sta interagendo con un sistema reale, come la necessità di tecnologie adeguate, di campioni adatti e molto tempo; inoltre, qualsiasi studio sperimentale che si definisca tale presenta, per definizione, un'interazione con il sistema biologico, che inevitabilmente perturba il fenomeno che è oggetto di studio. Infine, la sperimentazione deve spesso limitarsi all'osservazione delle singole entità. L'utilizzo di simulatori computazionali, invece, non solo solleva da molti di questi problemi, ma permette allo scienziato di studiare in maniera estremamente approfondita il sistema simulato, controllandone ogni più piccolo aspetto e osservando contemporaneamente i diversi fenomeni che vi prendono parte. Può studiare come cambia la simulazione introducendo delle perturbazioni, cambiando le costanti cinetiche delle reazioni in gioco, oppure semplicemente facendone variare lo stato iniziale; il tutto, con la sola necessità di un computer

sufficientemente potente da realizzare le simulazioni. Tutto questo, però, a patto di avere un simulatore che simuli in maniera verosimile il fenomeno, come si accennava all'inizio.

Realizzare simulatori che garantiscano una sempre maggiore flessibilità e una sempre più fedele ricostruzione della realtà biologica è l'obbiettivo in cui una fetta sempre crescente della comunità scientifica si sta impegnando. Anche in questo lavoro abbiamo cercato di impegnarci in questa direzione, cercando di creare un'implementazione del movimento cellulare semplice e flessibile, ma al contempo vicina alla reale fenomenologia di ultimo. A tal fine i passi svolti durante la tesi sono riportati nel capitoli di questo elaborato nel modo seguente:

- nel primo capitolo si spiega il contesto scientifico in cui questo lavoro si colloca, la *Computational Systems Biology*, spiegando nel dettaglio il motivo della sua importanza e gli approcci che utilizza per studiare i fenomeni;

- nel secondo si cerca di dare una spiegazione riassuntiva ma chiara di cosa sia il simulatore Alchemist e quale sia lo scopo della sua creazione;

- nel terzo si spiega come sono state realizzate le basi dell'incarnazione biochimica, focalizzandosi sul modello adottato per adattare le entità in gioco nei sistemi multicellulari alle astrazioni di Alchemist;

- nel quarto si comincia a entrare nel merito del movimento cellulare, spiegando nel dettaglio alcuni aspetti che è stato necessario modellare prima ancora del movimento cellulare, affinché questo potesse essere poi realizzato con sufficiente realismo. In particolare, si discute di come sono stati modellati, e conseguentemente implementati l'ambiente extracellulare e le giunzioni;

- nel quinto si spiega, infine, il modello e le scelte implementative compiute per rendere possibile nell'incarnazione biochimica il movimento, e in particolare quello dovuto alla chemiotassi e all'interazione meccanica fra le cellule;

- nel sesto si sono esposti dei test eseguiti per validare il progetto, mostrandone la capacità di riprodurre sistemi ispirati alla realtà biologica.

Alla termine del percorso possiamo dirci pienamente soddisfatti dei risultati ottenuti: la modellazione e l'implementazione svolta si è rivelata efficace, rivelandosi pienamente all'altezza delle nostre aspettative. Alchemist è stato arricchito di un ambiente extracellulare in grado di interagire efficacemente con le cellule e di essere caratterizzato a desiderio dell'utente; inoltre, è stato reso possibile modellare il movimento cellulare, anche considerando l'interazione meccanica delle cellule. In conseguenza all'aggiunta dell'ambiente è ora possibile:

- definire reazioni intracellulari che permettano alle molecole di diffondere dalla cellula all'ambiente o viceversa;
- modellare la diffusione di molecole nell'ambiente dando luogo a gradienti, variando la velocità e forma;
- più in generale, caratterizzare separatamente zone diverse dell'ambiente extracellulare, ad esempio con un certo set di reazioni, o con una certa popolazione di molecole.

In conseguenza, poi, dell'aggiunta del movimento cellulare, ora in Alchemist è in grado di supportare:

- la modellazione del movimento cellulare nei casi particolari della chemiotassi e del movimento random;
- la modellazione dell'interazione meccanica fra le cellule;
- la definizione di reazioni che attivino risposte intracellulari quando la cellula viene sottoposta a una tensione da parte di altre cellule;
- la regolazione della velocità di movimento della cellula;
- la regolazione dell'influenza di ogni stimolo sul movimento complessivo della cellula.

Ognuno di questi aspetti è stato testato attraverso alcune simulazioni di prova, le più significative delle quali sono riportate nel sesto capitolo di questo elaborato; in tutti i casi, i risultati si sono dimostrati soddifacenti.

Capitolo 1

La Computational Biology e l'importanza dei simulatori

Come è risaputo, lo scopo ultimo della biologia dello sviluppo e della biologia in genere è quello di comprendere in maniera approfondita i meccanismi che regolano ogni aspetto della vita, dalla cellula agli organismi complessi, in modo da poter formulare predizioni sull'evoluzione di questi ultimi in determinate situazioni. Ovviamente, conoscere i meccanismi che regolano il comportamento di un sistema biologico ha immensa importanza in molti ambiti della scienza, in primis per la medicina e la farmacologia. Tuttavia il maggiore ostacolo nello studio di questi sistemi è proprio la loro estrema complessità; infatti, al fine di comprenderne il funzionamento, dalla metà del XX secolo la biologia sperimentale si è concentrata sui meccanismi biochimici che ne regolano il comportamento. Questo approccio riduzionista si diffuse soprattutto in seguito alla scoperta del DNA, che dimostrò come ogni aspetto della vita cellulare potesse essere originato da un'interazione di diverse molecole, spesso a partire dall'espressione di un particolare enzima da parte del DNA. Per buona parte del secolo scorso, quindi, gli studiosi si sono impegnati nell'individuazione dei singoli meccanismi molecolari che regolano la vita della cellula. Da tempo però esistono evidenze del fatto che questi meccanismi, presi singolarmente, non riescono a dare una descrizione completa degli eventi cellulari. Ogni singolo evento all'interno della cellula, infatti, è condizionato in maniera non trascurabile da una miriade di fattori interni o esterni che,

agendo in contemporanea, portano il sistema ad uno stato differente da quello che si sarebbe potuto prevedere prendendo in considerazione l'evento singolo. In sostanza, *il risultato è più che la somma delle parti* [16]. Questa idea sarà la base di una nuovo approccio nello studio della biologia, che spiegherò nella sezione successiva.

1.1 La Systems Biology e la Multicellular Systems Biology

Questo nuovo approccio nasce nell'ambito della *Systems Biology*, la branca della biologia che si occupa non più di studiare i meccanismi cellulari, ma come questi interagiscono fra loro. Questa filosofia, che nella Systems Biology è perlopiù applicata all'ambito uni-cellulare o sub-cellulare [6], trova grandi sviluppi anche nello studio di sistemi biologici di livello più elevato, quelli multicellulari. Ci sono infatti fenomeni di estrema rilevanza biologica, come lo sviluppo embrionale, la formazione dei tessuti e la progressione di malattie degenerative, che non possono essere spiegati prendendo in considerazione le singole cellule che ne fanno parte [6]. Anche qui vi è un'interazione fra le parti che compongono il sistema (le cellule, appunto), che condizionandosi a vicenda danno luogo ad una sua evoluzione complessiva. Lo studio di questo tipo di fenomeni è ambito di una sotto-branca della Systems Biology, la *Multicellular Systems Biology*.

Possiamo quindi notare che la ricerca si è evoluta in direzione di una crescente complessità: dai meccanismi sub-cellulari all'intera cellula e dalla cellula fino ai sistemi multicellulari. Questo ha provocato una sempre maggiore difficoltà nell'analisi sperimentale della biologia cellulare, e a una conseguente necessità di trovare nuovi strumenti di indagine.

1.2 La Computational Systems Biology

Entrando più nel particolare, la complessità dei sistemi multicellulari forza gli studiosi ad utilizzare un approccio più teorico e meno sperimentale alla risoluzione dei problemi e, sopratutto, a fare uso di

strumenti informatici per l'indagine scientifica. Questo porta la Multicellular Systems Biology a intrecciarsi sempre più saldamente con la *Computational System Biology*, cioè l'ambito della biologia dei sistemi che utilizza strumenti di simulazione computerizzata [13].

A partire dal fenomeno che si vuole studiare, l'approccio della Computational System Biology consiste nella formulazione di un *modello*, cioè una sua espressione teorica semplificata, che sarà poi inserito all'interno di un software di simulazione. Questo avrà il compito di calcolare, dato il modello e lo stato iniziale del sistema, come il fenomeno evolve nel tempo. Se il risultato riproduce in maniera sufficientemente fedele il sistema che ci interessa, studiando i parametri del modello potremo provare a formulare delle ipotesi su come questo si comporterà al variare di questi, o delle condizioni iniziali; se invece la riproduzione non dovesse essere soddisfacente, vorrà dire che il modello scelto è troppo semplice e bisognerà quindi modificarlo.

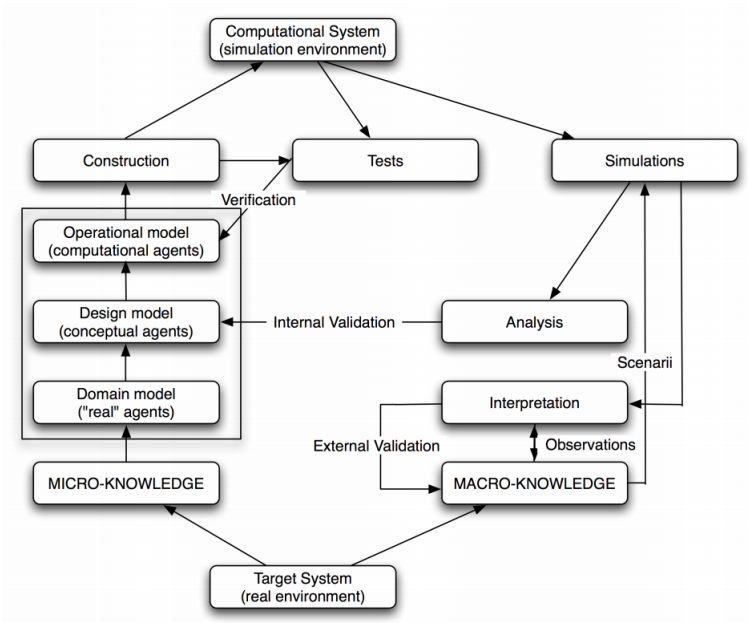

Figura 1.1: Una rappresentazione grafica della metodologia di studio adottata nella Computational System Biology e, in generale, in tutti gli studi basati su simulatori computerizzati.

1.3 L'importanza del processo di modellazione

L'aspetto centrale di questa metodologia di studio, come in ogni ambito dove non si possono ottenere sufficienti risultati dalla sola sperimentazione, è quindi la stesura del modello, che non deve essere una riproduzione dettagliata del fenomeno, spesso impossibile da costruire; lo scopo del modellista deve essere quello di comprendere gli aspetti principali che danno origine al fenomeno e di formulare un'ipotesi realistica su come essi interagiscono per dare luogo a quest'ultimo, trascurando tutti gli aspetti ritenuti secondari o estranei allo stesso.

Per capire meglio questo importante concetto, pensiamo al modellista come a un artigiano intento a costruire un manichino da disegno, cioè un modellino spesso utilizzato da illustratori e disegnatori per aiutarsi nel raffigurare il corpo umano con naturalezza (Figura 1.2).

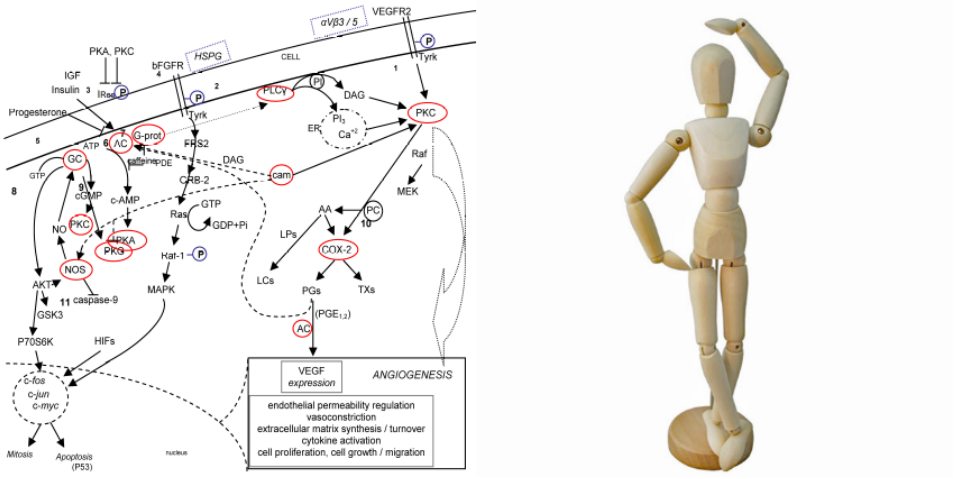

Figura 1.2: A sinistra è possiamo vedere la rappresentazione grafica di un modello utilizzato da B. Veleirinho et al. per lo studio della vasculogenesi [23]; a destra un tipico manichino da disegno

Lo scopo di questi manichini non deve essere l'esatta riproduzione della figura umana, bensì la precisa raffigurazione delle proporzioni di quest'ultima e delle articolazioni che ne permettono il movimento. La comprensione dei punti fondamentali di un fenomeno può essere para-

gonata allo studio che l'artigiano dovrà effettuare per capire quali siano le giunzioni che devono essere inserite nel manichino; l'inserimento di queste ultime nelle giuste posizioni e nel rispetto delle proporzioni è invece riconducibile alle comprensione, da parte del modellista, delle interazioni che devono esserci fra i meccanismi individuati precedentemente. Infine, possiamo vedere in figura che questi manichini non presentano i tratti del viso, della la muscolatura e della pelle; questi, sebbene siano aspetti non trascurabili della figura umana, non hanno un ruolo determinante nelle pose e quindi una loro aggiunta non causerebbe altro che un aumento nel prezzo del manichino. Allo stesso modo, la scelta di un modello troppo dettagliato comporta un'aumento del suo costo, in termini di lavoro speso per crearlo e in termini computazionali, cioè in termini di tempo che il simulatore utilizza per calcolare l'evoluzione del sistema. Questo nei simulatori comunemente utilizzati è in realtà un aspetto fondamentale, poichè riprodurre fenomeni complessi, che devono tener conto di più entità che interagiscono variando il proprio stato interno (come nel caso di un sistema multicellulare, appunto) può essere davvero difficile da gestire velocemente; come vederemo, infatti, il simulatore da noi utilizzato nasce proprio con l'intento di abbassare i tempi necessari ad effettuare i calcoli, in genere molto elevati.

1.4 I tipi di modelli

Abbiamo quindi compreso l'utilità dei modelli e come questi debbano essere costruiti affinché possano dare dei risultati soddisfacenti all'interno del simulatore. Tuttavia, bisogna anche pensare che per un fenomeno si possono scegliere vari modelli, a seconda del tipo di formalismo che si decide di utilizzare. Esistono infatti diversi approcci di modellazione, che possono essere suddivisi in due categorie di modelli: *matematici* e *computazionali* [16]. Non si pensi però che solo i secondi necessitino di un software di simulazione; spesso infatti tali modelli, sebbene facciano uso degli strumenti della matematica classica, sarebbero talmente complessi da calcolare su carta da rendere comunque necessario l'utilizzo di un algoritmo implementato su un calcolatore (si veda, a questo proposito, la sezione 1.5).

1.4.1 I modelli matematici

Come già accennato poco sopra, si dicono *matematici* quei modelli che fanno uso degli strumenti tipici della matematica classica, in particolare delle equazioni differenziali, per descrivere il fenomeno di interesse. I modelli più importanti di questa categoria infatti sono i modelli ad *equazioni differenziali ordinarie* (in inglese ODE, da *ordinary differential equation*) , i modelli ad *equazioni differenziali parziali* (PDE, *partially differential equation*) e la *chemical master equation*.

Equazioni differenziali ordinarie

I modelli di questo tipo utilizzano un sistema di equazioni differenziali per descrivere l'evoluzione nel tempo del sistema. Dato lo stato $\mathbf{S}$ del sistema, questo viene descritto da un inseme di variabili $\{s_1,\ s_2,\ ...,\ s_i\}$ (che spesso sono le concentrazioni delle specie chimiche presenti) e da un insieme di parametri $\{p_1,\ p_2,\ ...,\ p_i\}$, dove la variazione nel tempo di ogni variabile è definito da un'equazione del tipo:

$$\frac{s_i}{dt} = f_i(s_1, s_2, ..., s_i; p_1, p_2, ..., p_i) \tag{1.1}$$

dove f_i è una funzione generica, che sarà scopo del processo di modellazione determinare. Naturalmente, la soluzione di un sistema come descritto necessita non solo dei parametri, ma anche della conoscenza dello stato iniziale $\mathbf{S}_0$ del sistema.

Possiamo notare che nella descrizione dell'evoluzione del modello non è previsto lo spazio, ma solo il tempo; infatti in biologia questo tipo di modelli si utilizzano per studiare fenomeni che avvengono omogeneamente nello spazio, o nel caso la distribuzione spaziale del fenomeno no influisca in maniera determinante su di esso.

Equazioni differenziali parziali

Un approccio di modellazione che invece tiene conto dello spazio è quello ad equazioni differenziali parziali, che aggiunge questa dipendenza a quella del tempo. Se vi sono situazioni dove si può ipotizzare che lo spazio non conti, infatti, ve ne sono altre dove la distribuzione spaziale delle variabili di stato (cioè della concentrazione delle molecole coinvolte) è decisamente non trascurabile. Il sistema è quindi,

ne caso si abbia un sistema con una sola variabile spaziale $l \in [0, L]$, composto da equazioni in questa forma:

$$\frac{\partial s_i}{\partial t} = f(S) + D_i \frac{\partial^2 s_i}{\partial^2 l} \tag{1.2}$$

dove D_i è la costante di diffusione della sostanza considerata. Il termine spaziale è quindi un termine che tiene conto della diffusibilità della sostanza.

Chemical master equation

Fino ad ora abbiamo quindi visto solamente modelli puramente deterministici, dove lo stesso evento, a parità di stato del sistema, si sviluppa sempre nello stesso risultato. Tuttavia in natura non sempre le cose avvengono in maniera deterministica. Anzi, quasi mai si rileva nei fenomeni biochimici che lo stesso evento si traduca esattamente nelle stesse conseguenze, sia perché è molto raro che si conoscano esattamente tutte le cause che concorrono allo svolgimento del fenomeno, sia per l'intrinseca stocasticità che esiste nelle interazioni molecolari.

Esiste quindi un ultimo modello matematico, molto popolare in biologia, dove però quello che si studia non è la variazione nel tempo della concentrazione delle sostanze facenti parte del sistema, ma la variazione nel tempo della probabilità che il sistema passi dal suo stato attuale $\mathbf{S}_p$ ad un qualsiasi altro stato che possa assumere. Definite dunque le seguenti grandezze:

- $\mathbf{N}_r$ = numero delle reazioni che possono avvenire nel sistema;
- $\mathbf{N}_s$ = numero delle specie molecolari presenti;
- $\mathbf{X} = [x_1, x_2, ..., x_{N_s}]$ = vettore contenente il *numero di molecole* per ogni specie presente in un determinato stato del sistema;
- $p(\mathbf{X}, t)$ probabilità che il sistema si trovi nella configurazione $\mathbf{X}$ nell'istante di tempo t;
- c_j = costante cinetica stocastica della reazione j (con j = $[1, 2, ..., \mathbf{N}_r]$)
- R_j reazione j-esima;

- $\alpha_j \delta t$ = probabilità che la j-esima reazione avvenga nel tempo δt, dato che il sistema si trova nello stato $\mathbf{X}$;

- $\beta_j \delta t$ = probabilità che la j-esima reazione non venga eseguita nel lasso di tempo δt;

la probabilità che il sistema si trovi nello stato $\mathbf{X}$ in un intervallo di tempo $t + \Delta t$ viene descritta dall'equazione:

$$p(X, t + \Delta t) = p(X, t)(1 - \sum_{j=1}^{N_r} \alpha_j \Delta t) + \sum_{j=1}^{N_r} \beta_j \Delta t \tag{1.3}$$

Infatti possiamo vedere che l'equazione a destra è composta da due termini:

- $p(X, t)(1 - \sum_{j=1}^{N_r} \alpha_j \Delta t)$, cioè la probabilità che il sistema già si trovi nello stato $\mathbf{X}$ e che in $t + \Delta t$ non accada nulla che faccia variare il suo stato;

- $\sum_{j=1}^{N_r} \beta_j \Delta t$, cioè la probabilità che il sistema da un altro stato passi allo stato $\mathbf{X}$.

Se poi consideriamo una situazione continua, in cui $\Delta t \to 0$, si ottiene l'equazione differenziale:

$$\frac{\partial p(X, t)}{\partial t} = \sum_{j=1}^{N_r} (\beta_j - \alpha_j p(X, t)) \tag{1.4}$$

che descrive appunto la variazione nel tempo della probabilità che il sistema di trovi nello stato X. Lo scopo di questo approccio di modellazione è quindi quello di scrivere un sistema di equazioni di questo tipo, una per ogni stato possibile che il sistema può assumere, in modo da ottenere la completa distribuzione di probabilità per ogni possibile transizione che il sistema può attraversare, quindi avere una descrizione probabilistica di come evolve il sistema.

1.4.2 I modelli computazionali

Come anticipato, la matematica non è l'unico modo per descrivere l'evoluzione di un sistema. La matematica è solo la maniera di modellazione più classica, ma sono molti i modi in cui si può descrivere un fenomeno biochimico, in particolare attraverso algoritmi e processi computerizzati. I modelli computazionali sono esattamente questo: programmi che hanno la funzione di descrivere l'evoluzione di un sistema. Gli approcci più importanti in questo tipo di modellazione sono le *reti booleane*, i *metodi formali*, *l'algoritmo di Gillespie* e i modelli a *Cellular-automata*.

Reti booleane

Uno dei modi più semplici per descrivere sistemi biologici, anche se solo in termini qualitativi, è utilizzare una rete booleana. Lo scopo di questo approccio è la costruzione di vere e proprie *reti logiche* che descrivano lo stato di ogni gene di interesse all'interno di una cellula, naturalmente associando ad esso due possibili stati: attivato (true, 1) o disattivato (false, 0). L'attivazione o la disattivazione di ogni gene dipende dagli altri geni, la cui totalità definisce lo stato del sistema. Si spiega quindi perché in questa maniera si possano ottenere solo

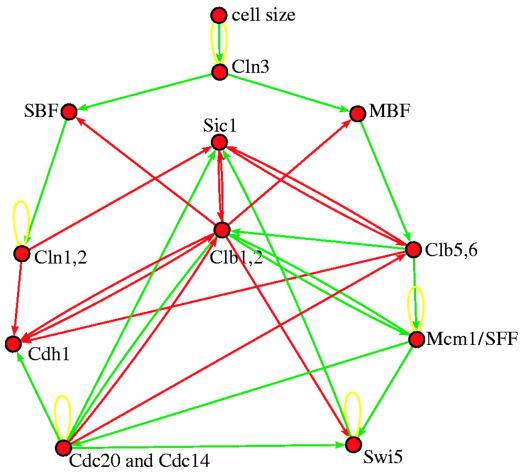

Figura 1.3: Un esempio di rete booleana esposta in un articolo di S. Bornholdt [3].

risultati qualitativi: nel modello non vi sono quantità di molecole che cambiano, dipendenze dal tempo o dallo spazio; solo transizioni da uno stato all'altro, definite sull'attivazione o meno dei geni.

I metodi formali

Anche i metodi formali, sebbene siano strumenti nati per lo sviluppo di sistemi software e hardware, hanno avuto applicazioni nella biologia computazionale. (Ma quelli che si stanno per esporre sono metodi formali? O sono approcci che utilizzano i metodi formali?)

Algebre di processi Per algebre di processi si intendono una serie di formalismi, come $\pi-$calcolus, $\kappa-$calcolus, Bio-PEPA e Beta-Binders che hanno lo scopo di semplificare la modellazione di fenomeni che sono dati l'interazione di processi concorrenti. la modellazione di questi sistemi avviene attraverso un insieme di *azioni* e *operatori*, che specificano l'evoluzione del sistema in maniera chiara e priva di ambiguità, in modo da interfacciarsi facilmente con un calcolatore. É quindi evidente come questi abbiamo trovato terreno fertile nell'ambito della system biology, dove si deve tener conto di molti eventi contemporanei che si influenzano a vicenda.

Reti di Petri A Carl Adam Petri sono dovuti i primissimi studi, durante gli anni '60, riguardo alla comunicazione e alla concorrenza, e sulla scia di questi studi inventò questo formalismo per la modellazione di sistemi distribuiti discreti. Le reti di Petri si basano su tre elementi: nodi di posizione, di transizione e archi. Gli archi possono collegare solo nodi di transizione e nodi di posizione, da cui consegue che ogni nodo di transizione presenta un contesto di input (l'insieme dei nodi di posizione che presentano archi *verso* quest'ultimo) e un contesto di output (analogamente, l'insieme dei nodi di posizione che presentano archi *provenienti* da quest'ultimo). Ogni nodo di posizione può contenere vari *token*, che ovviamente possono rappresentare qualsiasi cosa; una transizione agisce su questi token solo nei suoi contesti di input e di output e, in particolare, rimuovendo tokens dal suo contesto di input e aggiungendoli nel suo contesto di output. Il modo in cui questo viene fatto non è fissato, bensì definito all'interno della transizione

stessa attraverso dei *task* (letteralmente, "compito"). Infine, queste transizioni avvengono sulla base di una determinata *regola di scatto* (o di *firing*) che decreta se la transizione possa avvenire o meno in un certo momento. Ciò che però rende le reti di Petri particolarmente adatte a modellare sistemi di processi concorrenti è il fatto che lo scatto delle transizioni non avvenga in modo deterministico; non avviene, ad esempio, appena le condizioni della regola di scatto sono soddisfatte. Ogni transizione attiva in determinato momento può scattare o meno con la stessa probabilità, il che comporta che una quando una regola di scatto viene soddisfatta la transizione può scattare immediatamente, dopo un certo tempo o non scattare affatto. Questo comportamento si adatta molto alla simulazione delle reazioni biochimiche, delle quali accadimento è certamente dipendente dalle molecole presenti, ma anche in buona parte casuale.

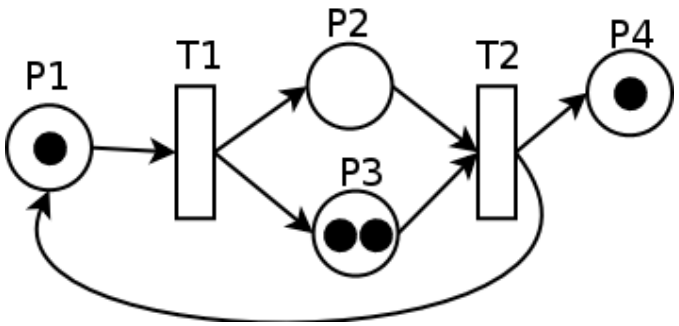

Figura 1.4: Un esempio di rete di Petri tratto da Wikipedia. T1 e T2 sono i nodi di transizione, mentre P1, P2, P3 e P4 sono i nodi di posizione.

I modelli ad agenti e a Cellular-Automata

La debolezza comune di tutti i modelli computazionali mostrati precedentemente è il non tenere in conto, almeno in maniera esplicita, dell'estensione spaziale del sistema. Se questo può essere accettato in alcuni casi, in altri, come nella modellazione sistemi dove più cellule interagiscono in un ambiente, non è un'assunzione realistica. I modelli ad agenti (ABM, *Agent Based Models*) e a Cellular-Automata sono uno

degli approcci più comuni alla modellazione di fenomeni multi-scala, nonché tra i più utilizzati per modellare gli ambienti multicellulari [17]. Questo tipo di approccio infatti modella un fenomeno emergente su scala macroscopica come risultato dell'interazione di più entità, gli agenti appunto, che vengono caratterizzate individualmente dal modellista sulla base del fenomeno che vuole ottenere. Questo tipo di modelli fornisce tre astrazioni fondamentali:

- quella di *agente*, corrispondente alla più piccola entità che contribuisce allo sviluppo dello stato del sistema;
- quella di *società*, corrispondente alla scala in cui si manifesta il fenomeno che si vuole indagare;
- quella di *ambiente*, che rappresenta lo spazio in cui gli agenti interagiscono.

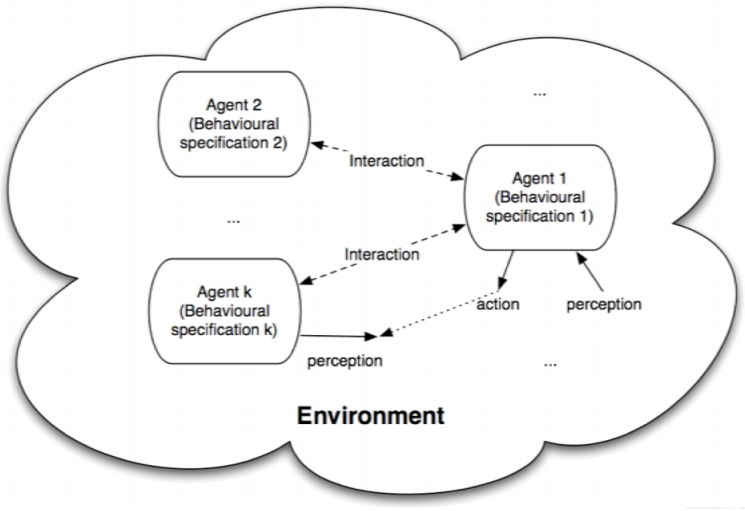

Figura 1.5: Rappresentazione grafica di un modello basato su agenti che mette in evidenza le varie astrazioni.

Ogni agente è dotato di uno stato interno e può condizionare gli altri agenti, o esserne condizionato. In questo modo, ogni passo nello sviluppo del modello corrisponde in realtà un'evoluzione dello stato di un agente, che influenza gli altri agenti, che insieme spingono il

sistema a un evoluzione globale. Il modo in cui, con un modello ad agenti, si riesce a riprodurre un fenomeno di alto livello è quindi simile alla messa in scena di un'opera teatrale: durante le prove, ogni attore viene istruito individualmente sui gesti che dovrà compiere e sulle parole che dovrà pronunciare, in relazione al momento e alle azioni degli altri attori. Durante la messa in scena poi, queste azioni individuali compiute da ogni membro della compagnia contribuirà alla nascita di qualcosa di superiore alla persona, ma anche alla somma di tutti gli interpreti presi singolarmente: il dramma stesso.

Inutile dire che questo approccio ha trovato molto successo in molti ambiti della scienza e che sono molti i simulatori che lo adottano. Repast [18] e NetLogo [24] sono alcuni esempi di simulatori ad agenti *general-purpose*, cioè programmabili per sistemi diversi, ma ne sono stati creati anche molti specificatamente orientati alla Systems Biology, come EPISIM [22]. Anche Alchemist [20] si può considerare parte si questa famiglia ma, come vedremo nel prossimo capitolo, nell'approccio si differenzia molto dai classici simulatori basati su agenti (ABS, *Agent Based Simulators*)).

1.5 La simulazione del modello

Una volta definito il modello è necessario simularlo per conoscerne gli sviluppi e avere così una rappresentazione teorica del fenomeno. Per i modelli matematici questa corrisponde alla soluzione delle equazioni differenziali, che può avvenire sia per via analitica, sia per via numerica. Sebbene la risoluzione analitica di questi modelli sia l'unico modo per giungere a un risultato esatto, spesso questa risulta impraticabile a causa della complessità troppo elevata, quindi vengono utilizzati appositi algoritmi che sono in grado di fornire delle soluzioni approssimate, come il metodo di Eulero, le serie di Taylor, il metodo Runge-Kutta e molti altri [17]. Per i modelli computazionali invece esistono altri algoritmi di simulazione, tra i quali uno dei più famosi e, in assoluto, quello di riferimento per la simulazione stocastica è sicuramente l'algoritmo di Gillespie [8], del quale Alchemist implementa una versione estesa, come spiegherò nel prossimo capitolo. Il motivo principale per cui pensiamo che questo algoritmo sia adatto per simulare ambienti

multicellulari è che crediamo nell'importanza che il caso ha nei fenomeni di scala molecolare e cellulare; nei sistemi viventi infatti quasi nulla è puramente deterministico, sia a causa dell'intrinseca aleatorietà di alcuni fenomeni (come la reazione di due molecole), sia a causa delle continue, minime variazioni che questi continuamente subiscono.

1.5.1 L'algoritmo di Gillespie

Nel 1976 Daniel T. Gillespie ideò un algoritmo di simulazione stocastico che automatizzava la risoluzione della Chemical Master Equation nell'ipotesi di sistemi omogenei e mono-compartimentali, in particolare nel caso di quantità di reagenti molto basse (decine di molecole) [8]. L'algoritmo basa la sua evoluzione sulla scelta di due numeri casuali, estratti a ogni passo: uno dei due determina la prossima reazione che sarà eseguita, mentre l'altro determina la sua durata. Dato quindi un sistema chimico popolato da n specie chimiche, ognuna distribuita in maniera omogenea nello spazio (ipotesi di sistema *ben mescolato*), si definiscono:

- Lo stato $\mathbf{X} = (x_1, x_2, ..., x_n)$ del sistema, dove $x_1, x_2, ..., x_n$ sono le quantità delle molecole per ogni specie chimica presente. Si noti che lo stato può essere definito in questo modo solo perché il sistema è ben mescolato; se così non fosse andrebbe introdotta anche una dipendenza dallo spazio.

- Un set di m reazioni $\mathbf{R} = (r_1, r_2, ..., r_m)$, tutte al massimo bimolecolari e non reversibili, che vengono assunte *indipendenti* fra di loro. Si noti che questo non significa che i fenomeni che coinvolgono reazioni di altro tipo non siamo simulabili con Gillespie, ma semplicemente che reazioni reversibili e con più di due reagenti devono essere spezzate in più reazioni.

- Sia c_j, con $j \in \{1, 2, .., m\}$, *rate statico* della reazione j-esima. Questo è definito in modo che $c_j dt$ rappresenti la probabilità che i reagenti della reazione j-esima interagiscano fra loro in un intervallo di tempo dt.

- Sia $a_j(\mathbf{X})$ la *propensità* della reazione, definita:

- per reazioni uni-molecolari, del tipo $S_1 \xrightarrow{c_j} prodotti$, come $a_j(\mathbf{X}) \equiv c_j[S_1]$;
- per reazioni bi-molecolari con due reagenti della stessa specie, del tipo $2S_1 \xrightarrow{c_j} prodotti$, come $a_j(\mathbf{X}) \equiv \frac{1}{2}c_j[S_1]([S_1] - 1)$;
- per reazioni biomolecolari fra due molecole diverse, del tipo $S_1 + S_2 \xrightarrow{c_j} prodotti$, come $a_j(\mathbf{X}) \equiv c_j[S_1][S_2]$.

La propensità definita in modo che $a_j(\mathbf{X})dt$ rappresenti la probabilità che la reazione j-esima *avvenga* in un intervallo di tempo dt.

Definiti questi parametri, possiamo passare alla apiegazione dei passaggi dell'algoritmo di Gillespie:

1. Dato il sistema allo stato X_0 al tempo t_0, si calcoli la propensità $a_j(X_0)$ di ogni reazione e se ne valuti la somma $a_0 = \sum_{j=1}^{m} a_j(X_0)$.

2. Si estraggano due numeri random z_1 e z_2, compresi fra 0 e 1;

3. Nell'insieme $r_j, j \in \{1, 2, .., m\}$, delle reazioni possibili, si sceglie la prossima reazione come quella con indice:

$$j = \text{il più piccolo intero che soddisfa } \sum_{j}^{j'=1} a_j(X_0) > z_2 a_0 \quad (1.5)$$

   Vediamo quindi che la scelta della reazione avviene in maniera casuale, ma sensata: il modo spiegato infatti garantisce che la si scelga con maggiore probabilità una reazione con un a_j elevato. Per questo a_j è detta propensità della reazione.

4. Si aggiorna lo stato del sistema a seconda della reazione eseguita;

5. Si valuta il tempo di durata della reazione come:

$$\tau = \frac{1}{a_0} ln(\frac{1}{z_1}) \quad (1.6)$$

Essendo $ln(\frac{1}{z_1})$ mediamente uguale a 1, significa che il tempo di schedulazione della prossima reazione è inversamente proporzionale alla somma delle propensità di tutte le reazioni. Questo è sensato perché, se a_0 è molto alto, vuol dire che ci sono molte reazioni che hanno alta probabilità di avvenire, quindi devono essere schedulate molte reazioni in breve tempo.

6. Si aggiorna il tempo della simulazione a $t_0 + \tau$;

7. Infine, si valuta se esiste una reazione che può essere eseguita (con $a_j > 0$); se non c'è la simulazione è finita, altrimenti si torna al punto 2.

Capitolo 2

Il simulatore Alchemist

Alchemist è un simulatore, sviluppato all'interno dell'università di Bologna, che nasce con l'obbiettivo primario di proporre un simulatore *general porpouse* con performance superiori rispetto agli altri simulatori di questo tipo [20]. La maggior parte dei simulatori generici, cioè in grado di simulare sistemi di diverso tipo, presenta infatti un grossa caduta di performance all'aumentare delle entità simulate, proprio perché ognuna di queste deve supportare diverse caratterizzazioni e comportamenti a seconda del sistema di interesse. Alchemist invece prende una strada inversa rispetto a questi simulatori, che partono dal generale per scendere al caratteristico: l'idea infatti è quella di partire da un metodo di simulazione estremamente veloce, ma non generico, ed estenderlo in modo da supportare delle astrazioni sufficientemente generiche da permetterci di simulare anche fenomeni diversi [20].

L'algoritmo di cui si parla è quello ideato da Gillespie nel 1976 [8], descritto nel capitolo precedente, che costituisce il cuore del motore di simulazione di questo software. Ciò implica che Alchemist sia un simulatore intrinsecamente stocastico, cioè nel quale la successione degli eventi non è deterministica, ma in parte casuale. Questo, come abbiamo detto nel precedente capitolo, si adatta bene al mondo della biologia, ma anche a moltissimi altri della scienza, dove gli effetti stocastici non sono trascurabili. Ma come si può rendere un algoritmo pensato per sistemi chimici adatto a simulare, possibilmente, qualsiasi cosa? Sarà compito di questo capitolo cercare di chiarire questo punto, oltre che spiegare come funziona questo innovativo simulatore.

2.1 L'architettura

L'architettura di Alchemist poggia su due grandi blocchi, uno che si occupa dell'evoluzione nel tempo della simulazione e uno che si occupa della sua definizione, a seconda del sistema che si intende simulare. Possiamo quindi dire che si divide in una parte *general-porpouse*, cioè il motore della simulazione, che non dipende dal tipo di sistema che si sta simulando, e una parte *domain-specific*, che riesce ad adattarsi al sistema che si sta simulando, rendendo quindi possibile la gestione da parte di questo simulatore di realtà estremamente differenti. Infine, a complemento di tutto ciò, vi è una parte dedicata all'interfacciamento con l'utente, che serve per impostare la simulazione in modo da rispecchiare il fenomeno desiderato.

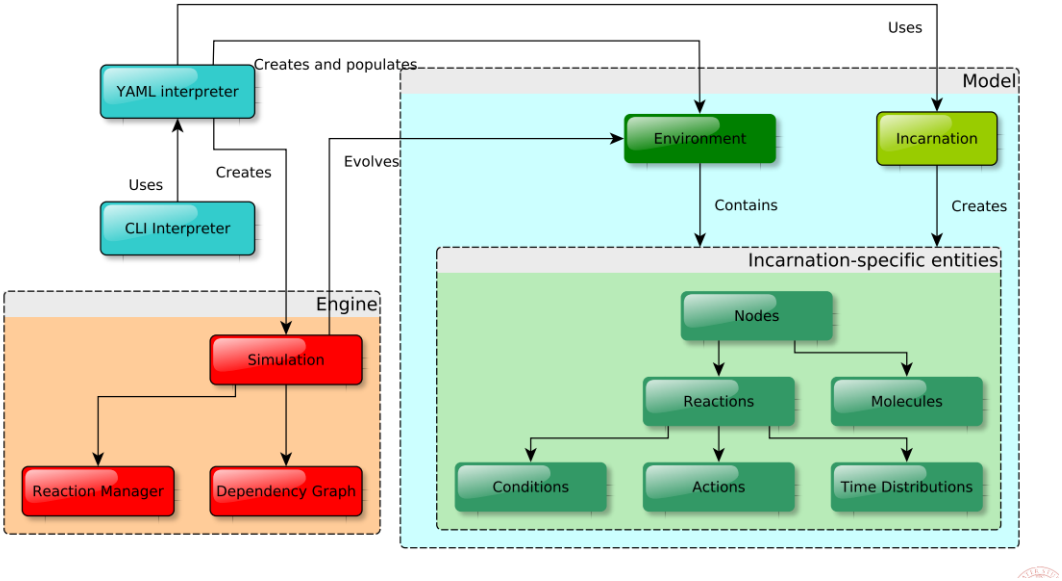

Figura 2.1: Una rappresentazione grafica dell'architettura del simulatore [20]. La parte racchiusa in Engine è la parte *general-porpouse*, che non dipende dal tipo di sistema che si sta simulando, mentre la parte racchiusa in Model è quella che si adatta al fenomeno simulato. Infine una piccola parte del simulatore si occupa dell'interfacciamento con l'utente, in modo che possa scrivere.

2.2 Le astrazioni di Alchemist

La grande genericità di Alchemist è dovuta all'adozione di poche, semplici astrazioni, che definiscono un generico modello di un sistema da simulare. Queste sono derivate, oltre che dal mondo della chimica dal quale l'algoritmo di Gillespie è stato preso, anche dai simulatori ad agenti classici (ABS). Ad una simulazione Alchemist infatti predono parte:

- dei *nodi*, analoghi agli agenti negli ABS, che corrispondono ai compartimenti che prendono parte alla simulazione;
- una *linking-rule*, che definisce il concetto di *vicinanza* dei nodi. Infatti ogni nodo può essere collegato ad uno o più nodi, che rappresentano il suo *vicinato*, con i quali può interagire in modo particolare;
- delle *molecole*, che sono contenute nei nodi e possono prendere parte alle reazioni. In un nodo, ogni molecola in un certo istante ha una determinata *concentrazione*;
- le *reazioni*, che sono ciò che fa variare lo stato della sistema. Come in Gillespie, ogni reazione presenta un rate statico c_j e una propensità a_j, che dipende dallo stato del sistema. Inoltre, queste possono cambiare lo stato di un solo nodo, dei suoi vicini o di tutti i nodi, a seconda del tipo di reazione;
- *l'ambiente*, che contiene i nodi e ne regola il posizionamento, la rimozione e lo spostamento.

Queste astrazioni non hanno una definizione fissa: assumono significato sia concettuale, che sostanziale soltanto all'interno di una specifica *incarnazione*, che rappresenta il contesto della simulazione che si sta eseguendo. Ogni incarnazione viene programmata specificatamente per un determinato contesto; prendendo ad esempio le tre incarnazioni esistenti, cioè *sapere*, *biochemistry* e *protelis*, queste sono dedicate rispettivamente al pervasive computing, alla system biology e alla simulazione di reti di dispositivi che utilizzano programmi Protelis (https://protelis.github.io/). Ognuna si occupa di dare una

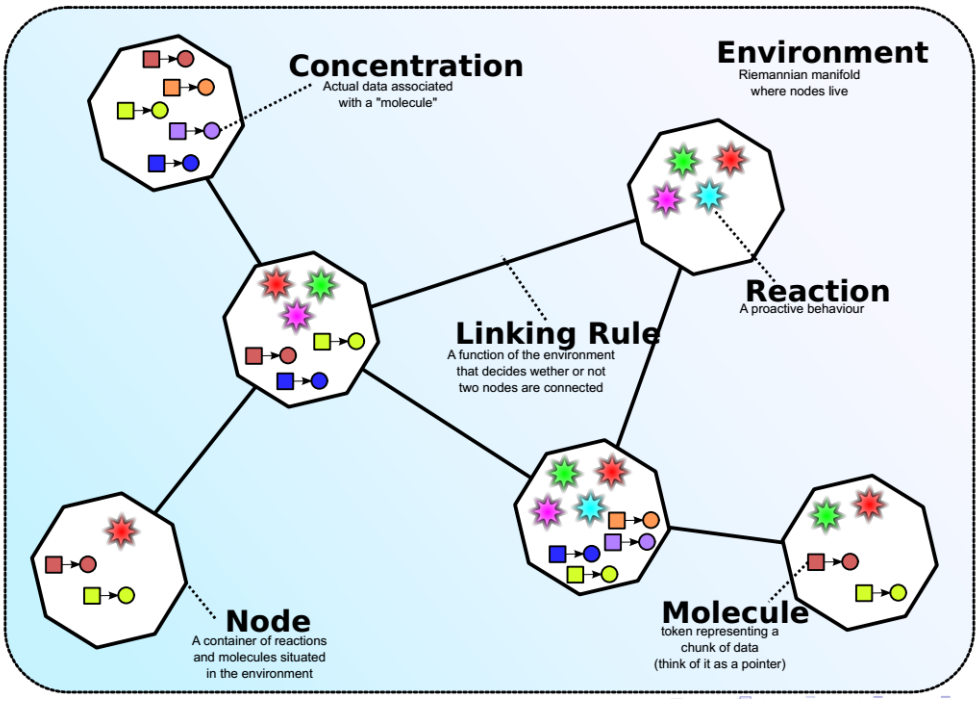

Figura 2.2: Una rappresentazione grafica delle astrazioni usate da Alchemist e di come sono in rapporto fra di loro [20]

definizione univoca alle astrazioni definite sopra, in modo che esse rappresentino le entità che è di interesse simulare. Ad esempio, in biochemistry, le molecole rappresentano le biomolecole e la loro concentrazione, ovviamente, rappresenta la loro quantità in una cellula (che viene rappresentata dal nodo); invece in sapere le molecole rappresentano programmi in esecuzione su un dispositivo (rappresentato dal nodo) e la loro concentrazione rappresenta il numero di tuple riconducibile ad un determinato template.

Vediamo quindi come questi pochi elementi possano rappresentare e definire anche realtà estremamente lontane fra di loro, e come quindi questi possano dare ad Alchemist la generalità di un simulatore general-porpouse.

2.2.1 Le reazioni

Le reazioni assumono una posizione di rilevanza fra le astrazioni elencate nella sezione precedente, in quanto sono quella parte del modello che specifica *come il sistema deve evolvere nel tempo*, e che quindi ha il più profondo rapporto con il motore di simulazione. Come possiamo vedere in Figura 2.3, una reazione è costituita da tre elementi fondamentali:

- Un rate statico, impostato dall'utente.
- Un insieme, anche vuoto, di *condizioni* (conditions), che determinano quando e quanto è probabile che una reazione venga eseguita. Ogni condizione infatti è una classe che, in un certo istante, è caratterizzata da due valori: uno booleano, che ci dice se la condizione è soddisfatta o meno, e un `double`, che rappresenta la *propensità* della condizione. Una reazione può essere eseguita solo se tutte le sue condizioni sono valide e, se questo è verificato, il prodotto delle propensità di tutte le condizioni, moltiplicate per il suddetto rate statico, definisce la propensità della reazione stessa;
- Un insieme, anche vuoto, di *azioni* (actions), che definiscono come la reazione cambia lo stato del sistema. Ad esempio, un'azione può modificare la concentrazione di una o più molecole in un nodo, può cambiare la sua posizione, rimuoverlo o duplicarlo;
- Una distribuzione temporale, impostata dall'utente;

L'ultimo elemento è stato introdotto per permettere al simulatore di gestire anche eventi non Markoviani. Infatti ci sono situazioni in cui una variazione nel sistema può intervenire solo in un certo momento fissato, o in maniera periodica (come l'iniezione di una sostanza esterna). Per permettere alle reazioni di modellare anche fenomeni di questo tipo quindi il tempo di occorrenza della reazione non viene più calcolato solamente come se la sua distribuzione nel tempo fosse poissontiana, cioè attraverso la formula:

$$\tau = \frac{1}{a_0} ln(\frac{1}{z2}) \tag{2.1}$$

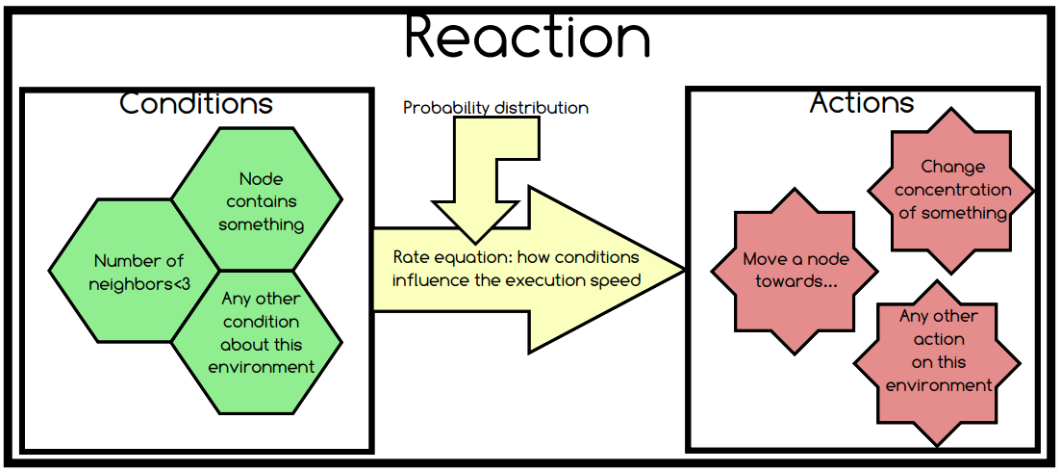

Figura 2.3: Schema rappresentate la composizione di una reazione in Alchemist [20]

ma in base, appunto, alla sua distribuzione temporale, che determina la manira di calcolare il τ. Ad esempio in Alchemist è possibile programmare reazione con una distribuzione temporale a pettine (δ-Dirac Comb), cioè che avvengono periodicamente nel tempo.

2.3 Il motore di simulazione

Come abbiamo già avuto modo di dire, il motore di simulazione è la parte del software che definisce l'evoluzione del sistema, quindi l'esecuzione delle reazioni. É una parte che è presente in ogni simulatore e può utilizzare approcci molto diversi, sia per quanto riguarda lo scorrere del tempo, sia per come si determina lo stato futuro a partire da quello presente. Scopo di questa sezione sarà spiegare alcuni di questi approcci, mettendo in luce quali sono stati adottati in Alchemist e i motivi che hanno spinto a questa scelta.

2.3.1 L'algoritmo di simulazione

Come abbiamo già detto, Alchemist è un simulatore stocastico, il che significa che la scelta dello stato futuro a partire da un certo stato

avviene casualmente, attraverso l'algoritmo di Gillespie (già esposto nel primo capitolo). Ma come può una simulazione basata su scelte casuale fornire dei risultati sensati? Non si tratta certo di fortuna.

Questo è possibile attraverso il metodo Monte Carlo, che viene adottato dal motore di simulazione. L'idea è molto semplice e può essere ben espressa dal seguente esempio: poniamo che si voglia conoscere la somma delle aree delle due figure mostrate in Figura 2.4. Utilizzare metodi geometrici sarebbe assai complesso, vista l'alta irregolarità delle figure; un metodo più semplice, basato su un procedimento stocastico, sarebbe selezionare moltissime volte un punto casuale all'interno del rettangolo che contiene le due figure, sapendo per ognuno se si trova all'interno o all'esterno delle suddette figure. Sommando poi tutti i punti che sono risultati interni e mettendo a rapporto tale valore con la totalità dei punti estratti otteniamo una *stima del rapporto fra l'area desiderata e l'area del rettangolo*, che invece è facilmente calcolabile.

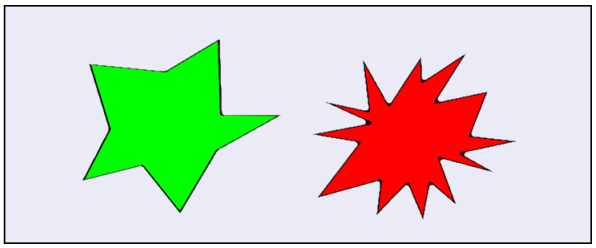

Figura 2.4: Il metodo Monte Carlo non si applica solo alle simulazioni, ma a qualsiasi tipo di problema, come il calcolo di queste aree.

In sostanza, il metodo Monte Carlo si basa su una procedura stocastica che ci permette di esplorare una parte del sistema che ci interessa (un punto, nel caso precedente), che viene ripetuta moltissime volte in modo da ottenere un'esplorazione completa di quest'ultimo. Quello che viene fatto dal motore di simulazione di Alchemist, quindi, è ripetere l'algoritmo di Gillespie varie volte, in modo da ottenere una collezione di simulazioni casuali, che quando vengono mediate ci forniscono una ricostruzione realistica di come il sistema si è evoluto. La scelta di questo particolare algoritmo stocastico si deve anche al modo in cui gestisce lo scorrere del tempo che, come sto per spiegare, permette al motore di simulazione di essere particolarmente veloce.

Lo scorrere del tempo in Alchemist

I simulatori possono essere divisi in due categorie: *continui* e *discreti*. Nei primi il tempo è una variabile continua, che viene aggiornata periodicamente facendo evolvere il sistema simulato con continuità; nel secondo caso invece si "scatta" da un istante all'altro e, allo stesso modo, i cambi di stato non avvengono in maniera continua, ma discreta, cioè tra un istante di tempo e l'altro. Questo secondo approccio è quello maggiormente utilizzato dai simulatori computerizzati, Alchemist compreso, e vede due varianti:

- i simulatori *Time Driven*;
- i simulatori a *Discrete Events Simulations*.

Nei primi la discretizzazione del tempo è diretta: il tempo viene diviso in intervalli di ampiezza Δt (detto *tick*) e il sistema viene aggiornato in maniera conseguente. Ciò significa che se in un tick devono avvenire due reazioni, queste vengono considerate contemporanee, mentre nel caso non vi sia alcuna reazione al tick successivo il sistema non avrà subito variazioni di stato. Questo approccio presenta il suo punto debole nel dover scegliere un tick fissato all'interno della simulazione: nel caso questo sia troppo breve, il sistema verrà aggiornato molte volte senza alcun cambiamento effettivo di stato, con un grosso spreco di risorse e un abbassamento della velocità di simulazione; nel caso in cui si scelga un intervallo troppo lungo, si avrà una certamente una simulazione più veloce, ma con una precisione molto inferiore. É esattamente come se dovessimo cuocere una torta e, per controllarne la cottura, decidessimo di assaggiarla periodicamente. Controllarla ogni minuto ci permetterebbe di averla esattamente al grado di cottura che preferiamo, ma ci farebbe perdere molto tempo; d'altro canto, controllarla ogni mezz'ora sarebbe molto meno impegnativo, ma farebbe aumentare di molto il rischio di trovarla bruciata. A questo punto potremmo fare due cose per ottenere una cottura perfetta con il minimo sforzo: o trovare, attraverso molti tentativi, una durata perfetta dell'intervallo o, molto più semplicemente, informarsi sul tempo di cottura e impostare un timer. Questo ci permetterebbe di controllare la torta una sola volta e trovarla ben cotta, con il minimo dispendio di energie. Questa è proprio la filosofia che viene adottata dall'approccio

a *Discrete Events Simulations*, che non effettua una discretizzazione del tempo a priori, ma sulla base degli eventi che avvengono durante una simulazione. Una volta che una reazione è stata eseguita e lo stato del sistema aggiornato, il tempo viene aumentato di un intervallo corrispondente alla durata della reazione, e così per ogni reazione della simulazione. Questo secondo approccio è quindi evidentemente più evoluto del primo, il Time Driven; tuttavia, quest'ultimo viene utilizzato dalla maggior parte dei simulatori esistenti, perché presenta un'implementazione estremamente più semplice. Invece Alchemist può usufruire del Discrete Time Simulation senza alcuna difficoltà, perché è l'approccio che viene già implementato all'interno dell'algoritmo di Gillespie dove, come abbiamo visto, il tempo viene aggiornato alla fine di ogni reazione. Il nostro simulatore può quindi contare già a priori su una gestione del tempo più sofisticata, dunque su una maggiore rapidità.

Le estensioni all'algoritmo

L'algoritmo originale ha dovuto subire alcune estensioni per supportare le astrazioni (e le ambizioni) di Alchemist. Infatti, se alcune parti del modello sono state prelevate da Gillespie ed estese per renderle capaci di supportare fenomeni diversi da quelli chimici (si veda sezione 2.2), altri sono stati introdotti ex-novo e non potrebbero assolutamente essere supportati dall'algoritmo originale. In particolare, Alchemist supporta sistemi composti da più compartimenti, mentre l'algoritmo di Gillespie no. Questo problema è stato gestito semplicemente introducendo, per le reazioni, dei contesti di input e di output, che determinano quanto "allargare" il compartimento che si sta modificando a seconda della reazione che si sta considerando. Mi spiego meglio: nell'algoritmo di Gillespie originale le reazioni non necessitavano di un contesto; essendo il sistema mono-compartimentale, non vi era necessità di specificare dove si trovassero le molecole reagenti e dove dovevano finire i prodotti. In Alchemist invece, nel caso più generale, si devono considerare diversi nodi, ognuno con diverse reazioni, che possono avere reagenti provenienti da nodi diversi e analogamente i prodotti. La maniera per specificare da dove possono provenire i

reagenti e dove devono essere piazzati i prodotti è proprio il *contesto*, che può essere:

- *local*, che comprende solo il nodo in cui la reazione è contenuta;
- *neighborhood*, che comprende i nodi vicini al nodo in cui la reazione è contenuta (dove la vicinanza, si ricorda, è definita dalla *linking-rule* della simulazione);
- *global*, che comprende tutti i nodi che partecipano alla simulazione.

Ora, prendiamo il caso in cui il motore di simulazione stia eseguendo una reazione appartenente ad un nodo N. Se questa ha contesto di input e di output *local*, vorrà dire che l'intera reazione non influisce su altri nodi se non N, quindi il "mono-compartimento" in cui la reazione si svolge viene fatto corrispondere al solo nodo N. Se invece la reazione avesse il contesto di input *neighborhood* o *global* questo verrebbe esteso, rispettivamente, alla somma di tutti i vicini di N o a tutti i nodi, perché la reazione che si sta considerando viene influenzata da tutti questi compartimenti. Si può quindi dire che le reazioni vengono sempre eseguite come se si trovassero in unico compartimento, che viene però allargato o ristretto a seconda del numero di nodi che la reazione può influenzare o da cui può essere influenzata.

Ottimizzazione

Altre modifiche invece non sono state fatti per motivi di flessibilità, ma di performance. In particolare, sono state introdotte delle ottimizzazioni sull'aggiornamento dello stato dei nodi, che avviene solo per il gruppo di nodi influenzati dalla reazione appena eseguita, e per ridurre le estrazioni di numeri casuali (algoritmo di Slepoy), che rappresentano sempre un grosso peso per il calcolo della simulazione.

Il risultato è quindi un algoritmo stocastico flessibile ed estremamente veloce, che permette ad Alchemist di superare lo scoglio delle performance con cui tutti gli altri simulatori *general-porpouse* si erano scontrati. Come mostra l'immagine 2.5, infatti, mentre per un ABS (Repast, nel caso considerato; http://repast.sourceforge.net/) risulta

molto difficile mantenere buone performance all'aumentare del numero di agenti, Alchemist mantiene la sua velocità di esecuzione molto bassa, anche cambiando in numero di nodi di un ordine di grandezza.

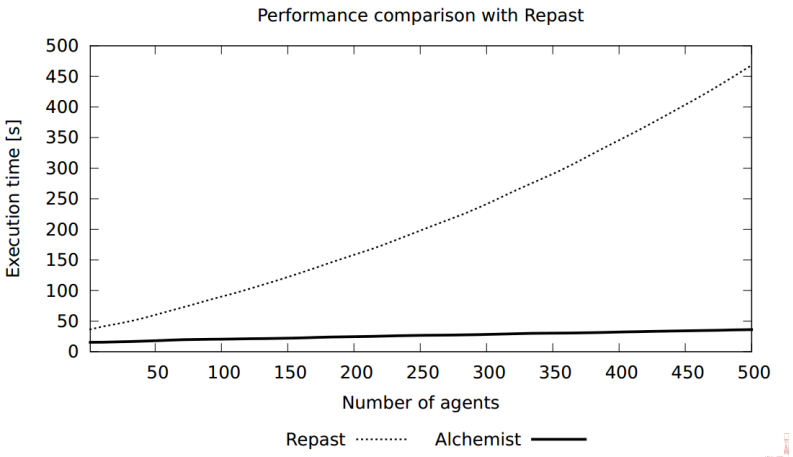

Figura 2.5: Confronto fra Alchemist e Repast, un simulatore ad agenti molto utlizzato.

2.4 Simulare in Alchemist

Ora che abbiamo visto come è strutturato il modello di una simulazione e come quest'ultima si evolve nel tempo, non ci resta che mostrare come l'utente può inserire in Alchemist il proprio modello è simularlo. L'interfacciamento fra utente e simulatore avviene attraverso la scrittura di un file, che specifichi:

- l'incarnazione che si vuole utilizzare (obbligatorio);
- il modello che si vuole simulare, quindi il numero e la posizione dei nodi, le reazioni, le molecole, eccetera.

Questo, nella prima relase di Alchemist (1.*) veniva fatto in XML (file *.xml), che però è stato presto abbandonato a causa della sua lontananza dal linguaggio umano e della sua estrema verbosità. Al

suo posto viene ora utilizzato il linguaggio Yaml, che permette invece un'espressone molto semplice ed intuitiva dei concetti.

2.4.1 Linguaggio Yaml

Lo Yaml (Yaml Ain't a Markup Language) è un linguaggio pensato per fornire uno standard semplice e facilmente comprensibile per la serializzazione di dati. Il risultato è uno strumento molto potente ed estremamente flessibile, che permette di definire in maniera semplice una gerarchia di entità correlate fra loro. Ciò che fa questo linguaggio infatti è definire una *mappa*, cioè un insieme di coppie chiave-valore, dove la chiave non può essere ripetuta mentre il valore si. Una mappa è quindi una *suriezione* che ha, come dominio, l'insieme di tutte le chiavi e, come codominio, l'insieme di tutti i valori. Un semplice esempio potrebbe essere l'insieme delle coppie (codice fiscale, nome) dei dipendenti di un'azienda: il codice fiscale rappresenta la *chiave*, in quanto per ogni dipendente è unico (nell'ipotesi che i dipendenti siano tutti italiani), mentre il nome è il valore associato, che può essere anche ripetuto.

In Yaml una chiave è sempre seguita dal simbolo ':' e il rispettivo valore è indicato successivamente. Si noti che sia chiavi che valori possono essere qualsiasi cosa: stringhe, numeri, liste e anche altre mappe.

```yaml
# Un esempio di file Yaml

## Un gruppo di chiavi associate a diversi valori
key1: stringa
key2: 2
key3: una stringa piu lunga

## Una chiave che ha come valore un'altra mappa
key:
  madre: laura
  padre: marcello
  sorella: elena

## Una chiave che ha come valore una lista
```

```yaml
key1:
  - 1
  - 2
  - 3
  - stella

key2: [1, 2, 3, stella]
```

Ci sono poi dei valori speciali, detti *anchors*, che possono essere definiti in una parte del testo per essere riutilizzati in un'altra parte. Sono contraddistinte dal simbolo '`&`' e possono essere richiamate attraverso l'asterisco '`*`'. Questi spesso servono a rendere il testo più leggibile e ad evitare problemi di indentazione spesso frequenti in Yaml, ma rientrano sempre all'interno del meccanismo di chiave-valore che è alla base di questo linguaggio.

```yaml
# Un esempio di utilizzo di anchors
## Definisco una variabile
variabile: &lista_della_spesa
  - carote
  - peperoni
  - passata di pomodoro
  - articolo: scatolette di tonno
    quantita: 2
  - articolo: pasta
    tipo: spaghetti
    quantita: 3

## La riutilizzo per accorciare una parte del codice
cose_da_fare:
  mattina:
    attivita: spesa
    lista: *lista_della_spesa
  pomeriggio: palestra
  sera: cinema
```

Quello che Alchemist fa, una volta che l'utente ha completato la scrittura della simulazione, è cercare delle chiavi precise, delle quali i valori sono utilizzati per costruire la simulazione. Queste chiavi sono

molte e servono a specificare ogni aspetto del modello; in particolare, le chiavi fondamentali, che costituiscono le *keywords* del linguaggio Alchemist sono:

- `incarnation`, che specifica l'incarnazione che si vuole utilizzare;
- `seeds`, che serve a inizializzare la generazione dei numeri random. Serve a fare in modo che questi vengano generati sempre allo stesso modo, affinchè si ottengano sempre simulazioni ripetibili.
- `environment`, che serve a inizializzare l'ambiente di simulazione;
- `network-model`, per specificare la *linking-rule*;
- `variables`, per indicare, sotto forma di mappa, quali parametri del modello si intende far variare nel corso delle simulazioni. É una funzione molto utile nel caso si voglia controllare che risultati otteniamo da una simulazione variando alcuni valori, come i rate delle reazioni;
- `displacements`, cioè la chiave che serve a indicare:
  - il tipo dei nodi, se l'incarnazione supporta più di un tipo;
  - come i nodi devono essere posizionati nell'ambiente;
  - cosa devono contenere i nodi (nel nostro caso, le molecole);
  - le reazioni che devono esserci in ogni nodo.

  Infatti questa chiave ha come valore una mappa che contiene diverse altre keywords, in paricolare `nodes`, `contains` e `programs`.
- `export`, che serve a indicare quali valori della simulazione devono essere esportati, ad esempio il tempo di simulazione, o la quantità di una determinata molecola. I dati vengono esportati sotto forma di file .txt.

Capitolo 3

I fondamenti di ALCHEMIST-`biochemistry`

Chiariti i concetti di base del simulatore Alchemist e i meccanismi che ne permettono la grande flessibilità, mi accingo ora a spiegare come sono state edificate le fondamenta della nostra incarnazione, `biochemistry`, la parte del simulatore dedicata allo studio di ambienti multicellulari. Vista la pluricitata derivazione di Alchemist dall'algoritmo di Gillespie, il nostro progetto ha rappresentato una specie di ritorno alle origini per questo simulatore, in quanto molte delle sue astrazioni ritrovano in questa incarnazione il loro significato originale (si pensi, ad esempio, alle astrazioni di concentrazione, di molecola e di reazione). Questo non ci ha evitato, però, di fare delle precise scelte modellistiche e di implementazione, scelte che in questo capitolo cercherò di spiegare e giustificare, sia per completezza, sia per rendere la lettura dei capitoli successivi più comprensibile. Tutte queste scelte sono state fatte e portate a compimento nell'ambito del progetto di tesi di Gabriele Graffieti, uno studente di Ingegneria e Scienze Informatiche che, prima di me, ha deciso di dedicarsi a questo progetto. Per informazioni più approfondite sui concetti trattati in questo capitolo, che io affronterò solo brevemente, rimando dunque al suo lavoro [10].

3.1 Requisiti

Innanzitutto, per poter cominciare la progettazione dell'incarnazione è stato necessario un'attenta analisi dei requisiti, cioè che cosa doveva essere supportato nella simulazione di un ambiente che coinvolge più cellule. Si è deciso di dividere queste necessità su tre livelli: quello cellulare, quello multi-cellulare e quello di interfacciamento con l'utente.

3.1.1 Requisiti cellulari

Le cellule dovranno, innanzitutto, essere identificabili e separate l'una dall'altra, in modo da poterne gestire eventuali differenze nel comportamento. Ogni cellula dovrà poter contenere più molecole, che potranno prendere parte a reazioni. Le molecole dovranno rappresentare atomi singoli o aggregati e, evidentemente, le reazioni rappresenteranno le reazioni chimiche in cui possono essere coinvolte, quindi dovranno sicuramente prevedere:

- dei reagenti;
- dei prodotti;
- un rate.

Bisognerà quindi tenere in conto che, in un certo momento, la stessa reazione potrà essere eseguita o meno a seconda della cellula considerata, che presenterà quantità di reagenti e un rate differenti dalle altre. Inoltre le reazioni dovranno supportare anche fenomeni di più alto livello rispetto alla semplice aggregazione e disgregazione di molecole. Ad esempio, dovranno poter supportare azioni come il movimento cellulare, la duplicazione, l'apoptosi e la formazione di giunzioni.

3.1.2 Requisiti multicellulari

Ogni cellula dovrà poi interagire con il mondo esterno e, in particolare, con due elementi fondamentali: le cellule vicine e l'ambiente. Le simulazioni in questa incarnazione, quindi, dovranno per forza avere

un'estensione spaziale, affinché si possa determinare se due cellule sono vicine o lontane. Due cellule si considereranno vicine se distano meno di una certa distanza prefissata, e si dirà *vicinato* di una cellula l'inseme delle cellule con cui quest'ultima può interagire. Il vicinato può anche variare nel corso della simulazione, ad esempio a seguito di spostamenti della cellula. Un'ultimo requisito riguarda poi, di nuovo, le reazioni, che dovranno essere in grado di gestire le interazioni della cellula con il vicinato e con l'ambiente.

3.1.3 Requisiti di interfacciamento

Infine, i requisiti di interfacciamento riguardano il rapporto che l'utente deve poter avere con il simulatore. Questo infatti dev'essere il più possibile semplificato; ogni parte del modello dovrà essere definibile in maniera semplice da parte dell'utente e, visto l'ambito di cui si tratta, con un linguaggio più simile possibile a quello utilizzato in biologia. Infine, i risultati della simulazione dovranno essere presentati in modo chiaro sia attraverso interfaccia grafica (GUI), sia sotto forma di dati numerici.

3.2 Mapping dei requisiti su Alchemist

Spiegati i requisiti che il nostro simulatore doveva supportare, in questa sezione mostrerò come questi sono stati adattati alle astrazioni di Alchemist, che abbiamo visto nel capitolo 2.

3.2.1 L'incarnazione

La prima cosa da fare era implementare l'estensione dell'interfaccia `Incarnation` relativa alla nostra incarnazione. Questa infatti è la struttura dati che fa da collegamento fra le astrazioni generiche e il modello specifico (si veda la Figura 2.1), che viene espresso dall'utente all'interno del file Yaml dato in input al simulatore. Nella sezione 2.4 abbiamo detto, infatti, che la maniera di specificare come costruire una simulazione è unica e fa uso sempre delle medesime keywords; ma allora, come si fa ad essere sicuri che inizializzando dei nodi, delle molecole o delle reazioni queste vengano trattate coerentemente con

l'incarnazione che abbiamo indicato? Questa certezza ci è data proprio dalla struttura dati di cui si parlava sopra, `Incarnation`: questa viene caricata coerentemente con quanto indicato nel file (se si scrive, ad esempio: `incarnation: sapere`, si andrà a caricare la classe `Incarnation` specifica per questo ambiente) e, come mostrato in Figura 3.1, possiede metodi per fornire ogni tipo di astrazione presente all'interno di Alchemist.

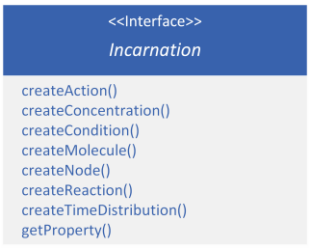

Figura 3.1: L'interfaccia `Incarnation` [10], in cui gli argomenti dei metodi non sono stati riportati per semplicità. Come si può notare, vi sono metodi per inizializzare qualsiasi entità possa essere richiesta: reazioni, concentrazioni, molecole, eccetera

Queste astrazioni, come sarà mostrato nelle sezioni successive, corrispondono ad altrettante interfacce, ognuna con implementazioni diverse a seconda del concetto che deve rappresentare (per esempio, l'interfaccia di `Molecule` è una sola, ma esiste una sua diversa implementazione per ogni incarnazione). La classe `Incarnazion` si occupa, quindi, di fornire *esattamente l'implementazione dell'interfaccia di cui si necessita in quel contesto*, garantendo la coerenza con quanto richiesto dal modellista. Nel nostro ambito, tale classe specifica è `BiochemistryIncarnation`, che provvederà a fornire le implementazioni adatte a simulare ambienti multicellulari.

3.2.2 Le molecole

Come abbiamo già detto nel capitolo 2, le molecole in Alchemist devono rappresentare atomi singoli o aggregati, con una determinata concentrazione. Per la concentrazione si è scelto di utilizzare un valore `Double`, solo positivo, per garantire la massima precisione. Per le

modecole invece si è creata una nuova implementazione, che estendesse l'interfaccia `Molecule`: la classe `Biomolecule`. In questa implementazione le molecole sono identificate semplicemente da una stringa che ne identifica il nome, senza alcuna restrizione. Questo per garantire la massima libertà all'utente: se quest'ultimo volesse, anche solo per semplicità, indicare una reazione di questo tipo:

$$A + B \rightarrow C$$

dovrebbe essere libero di farlo, anche se in natura non esistono molecole indicate con il simbolo chimico A.

3.2.3 Le cellule

Le cellule sono state, probabilmente, l'entità più complessa da mappare sul simulatore, non tanto per le difficoltà implementative, ma per le scelte modellistiche che siamo stati costretti a fare per garantire la maggiore vicinanza possibile con la realtà. Come ho già detto, le entità in grado di contenere reazioni e molecole, evidentemente di prima necessità per la cellula, sono i nodi. Ci si aprivano dunque davanti due strade, ognuna con i suoi vantaggi e difficoltà:

- considerare le cellule come un insieme di nodi, accomunate da un identificativo;
- far coincidere la cellula con un nodo.

Il primo approccio sacrificava quasi completamente l'identità della cellula, ma consentiva una modellazione molto precisa delle sue diverse parti. Non è infrequente, infatti, che in zone diverse della cellula vi siano concentrazioni differenti di una certa molecola; inoltre, molti processi e strutture sub-cellulari, come gli organuli, hanno una precisa locazione nella cellula, non ne occupano tutto il volume. Tuttavia, questa scelta implicava anche un altro, grosso problema: in quante sottoparti dovrebbe essere divisa la cellula? E cosa rappresenterebbe una sua sottoparte? Un organulo o semplicemente una parte del citoplasma?

Queste domande, unite alla nostra indisposizione a considerare la cellula come un insieme di entità indipendenti, ci hanno quindi portato a seguire la seconda strada, in cui la cellula possiede un'identità

unica. Questo significa che significa che in biochemistry le molecole sono assunte *sempre uniformemente distribuite all'interno della cellula* e, allo stesso modo, *non possono avvenire processi limitati a una sola una parte del citoplasma*. Siamo coscienti dei limiti che queste ipotesi comportano, ma il abbiamo ritenuti accettabili, ritenendo più importante preservare la possibilità di caratterizzare la cellula come sistema piuttosto che una sua singola parte. Inoltre, non è escluso che un giorno questo aspetto venga modellato e implementato all'interno del simulatore.

Abbiamo quindi implementato una nuova classe, estensione dell'interfaccia `Node` di Alchemist: `CellNode`. Questa non doveva essere semplicemente una trasposizione del nodo, ma aggiungergli qualcosa in più. In particolare, la prime aggiunte che sono state fatte sono servite a garantire il supporto delle giunzioni e dell'estensione spaziale, di cui però parlerò nel prossimo capitolo.

3.2.4 Le reazioni

Le reazioni sono state sicuramente la parte più impegnativa nell'implementazione dell'incarnazione. Infatti il lavoro di adattamento ha richiesto due fasi:

- nella prima sono stati adattati i concetti propri delle reazioni di Alchemist alle reazioni chimiche, che sono scritte in termini di reagenti e prodotti;
- nella seconda invece si è definito un linguaggio di scrittura delle reazioni apposta per biochemistry, dove queste vengono scritte in maniera simile alle reazioni chimiche.

Condizioni e Azioni

Come abbiamo detto nel capitolo 2, per motivi di flessibilità le reazioni in Alchemist sono definite da un insieme di condizioni e uno di azioni, che indicano, rispettivamente, quando la reazione può accadere e come questa modifica il sistema modellato. Anche queste astrazioni corrispondono, a livello software, a delle interfacce (si veda Figura 3.3),

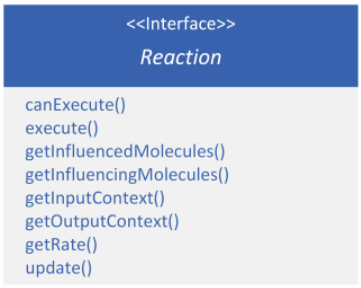

Figura 3.2: L'interfaccia `Reaction` di Alchemist [10].

Figura 3.3: Le interfacce `Action` e `Condition` di Alchemist [10].

che devono trovare un'implementazione particolare per ogni incarnazione, compreso il nostro caso. Si sono dovute creare, quindi, delle nuove condizioni e azioni che si adattassero al concetto di prodotto e di reagente e che, contemporaneamente, mantenessero il parallelismo con l'algoritmo originale di Gillespie per quanto riguarda la propensità della reazione, che deve essere, detto r il rate della reazione:

- $r[S_1]$ per una reazione del tipo $S_1 \xrightarrow{c_j} prodotti$;
- $r\frac{1}{2}[S_1]([S_1]-1)$ per una reazione del tipo $2S_1 \xrightarrow{c_j} prodotti$;
- $r[S_1][S_2]$ per una reazione del tipo $S_1 + S_2 \xrightarrow{c_j} prodotti$.

Sono state quindi create delle condizioni per modellare la richiesta di un reagente da parte di una reazione:

- `BiomolPresentInCell` che, data una concentrazione e una molecola, risulta verificata se tale molecola è presente nella cellula a una concentrazione maggiore uguale a quella richiesta.

- `BiomolPresentInNeighbor` che è analoga a quella sopra, ma la verifica viene fatta su un vicino della cellula in cui è contenuta la reazione.

- `NeighborhoodPresent` che verifica la presenza di almeno un vicino per la cellula corrente.

Le prime due condizioni sono accomunate dal fatto di avere una propensity coerente con quella definita dall'algoritmo di Gillespie [8]. Questa viene infatti calcolata secondo la formula del coefficiente binomiale:

$$\binom{q_c}{q_r} = \frac{q_c!}{q_r! \cdot (q_c - q_r)!} \tag{3.1}$$

dove q_c rappresenta la concentrazione della molecola presente nella cellula, mentre q_r la quantità richiesta nella condizione. Ricordando che l'algoritmo di Gillespie considera reazioni al massimo bimolecolari, la quantità richiesta dalla condizione può essere al massimo 2. Calcolando il coefficiente binomiale con $q_r = 1$ o $q_r = 2$ abbiamo, rispettivamente:

$$\binom{q_c}{1} = q_c \qquad \text{e} \qquad \binom{q_c}{2} = \frac{1}{2} \cdot q_c \cdot (q_c - 1)$$

che sono esattamente le formule per la propensità usate dall'algoritmo per reazioni con reagenti di un solo tipo. Nel caso, invece, di una reazione con due molecole diverse, semplicemente vi saranno due condizioni di tipo `BiomolPresentInCell`, che ritorneranno ognuna una propensità pari a q_c, che saranno poi moltiplicate risultando in una propensità complessiva di $q_{c1} \cdot q_{c2}$.

Successivamente, sono state create le azioni corrispondenti alla formazione di un prodotto nella cellula o in un suo vicino, che sono:

- `ChangeBiomolConcentrationInCell` che, data in ingresso una molecola e un qualunque valore δ, somma tale valore alla concentrazione della molecola nella cellula. Questa potrebbe salire o scendere in base a δ, che può essere sia positivo che negativo. Ovviamente, se la somma tra δ e la concentrazione dovesse risultare in un numero negativo, quest'ultima verrà settata a zero.

- `ChangeBiomolConcentrationInNeighbor`, che è analoga all'azione precedente, con l'unica differenza viene compiuta su una cellula vicina (se esistente).

Queste azioni vengono chiamate quando si forma un prodotto, ad esempio in conseguenza all'esecuzione della seguente reazione:

$$H + OH \rightarrow H2O$$

ma anche per rimuovere i reagenti che si sono uniti per formare il prodotto. La reazione sopra, infatti, implica che dopo la sua esecuzione vi siano una molecola di idrogeno e un gruppo ossidrile in meno della cellula. Per una reazione del genere quindi l'azione `ChangeBiomolConcentrationInCell` dovrà essere eseguita tre volte: una per rimuovere un atomo di H, una per rimuovere una molecola di OH e una per aggiungere una molecola di acqua.

Concludo questa piccola sezione precisando che le azioni e le condizioni descritte non sono le uniche implementate nel corso del lavoro di Gabriele Graffieti. Io ho incluso solamente quelle necessarie a spiegare al lettore come si possa passare da una reazione chimica, composta da reagenti e prodotti, a una fatta di condizioni e azioni, che sono concetti introdotti solo con Alchemist. Si segnala in particolare la presenza di un gran numero di classi per la gestione delle giunzioni cellulari, che saranno spiegate nel capitolo successivo; per il lettore che desiderasse una lista precisa di tali classi, rimando al lavoro del mio predecessore [10].

Il DSL per le reazioni

All'interno del file Yaml (si veda sezione 2.4) le reazioni vengono inserite come stringhe scritte in un linguaggio specifico (DSL, Domain Specific Language), che sono poi interpretate da un *parser* specifico, dei quali dettagli di implementazione saranno, per brevità, trascurati in questo elaborato. Basti sapere che il processo effettua una traduzione automatica della stringa in una reazione di Alchemist, associando automaticamente a quest'ultima le azioni e le condizioni necessarie e permettendo così all'utente di utilizzare un linguaggio simile a quello delle reazioni chimiche. Per spiegare come avviene questa traduzione, partirò subito da una stringa esemplificativa, che rappresenta una reazione valida nel DSL di biochemistry:

```
[A] + [B in neighbor] --> [C]
```

Innanzitutto una stringa valida deve presentare sempre il simbolo `-->`, che separa le *condizioni* (a sinistra) e le *azioni* (a destra) della reazione. Ad esempio, nel caso della sopra, nella reazione di Alchemist risultante dovranno sicuramente comparire:

- condizione 1: una molecola di A presente nella cellula;
- condizione 2: una molecola di B presente in un vicino;
- azione 1: aggiungi una molecola di C nella cellula.

A queste poi, il traduttore aggiungerà automaticamente altre azioni, nel caso sia necessario bilanciare la reazione. Sempre considerando l'esempio sopra, saranno aggiunte le azioni:

- azione 2: rimuovi una molecola di A dalla cellula;
- azione 3: rimuovi una molecola di B dal vicino (lo stesso per cui era soddisfatta la condizione 2).

Vediamo poi che nella stinga compaiono delle parentesi quadre; queste servono a separare i vari contesti delle condizioni e delle azioni. Si prenda ad esempio questa reazione:

```
[A + B in neighbor] --> [C]
```

In questo modo si specifica che entrambi i reagenti si devono trovare in un vicino, senza doverlo scrivere due volte. Si possono utilizzare tre keywords all'interno delle parentesi quadre:

- `cell`, che indica la cellula in cui si trova la reazione. Può anche essere omesso, come nel caso della reazione sopra dove, come prodotto, ho scelto di scrivere solo `[C]`, non `[C in cell]`;
- `neighbor`, che indica una cellula vicina;
- `env`, che indica l'ambiente circostante. Le azioni e le condizioni utilizzate nel caso si utilizzi questa keyword non sono state ancora spiegate, poichè incluse nel prossimo capitolo.

Esiste poi un'altra keyword specifica del linguaggio, che però non serve a indicare un contesto, bensì una giunzione cellulare, la cui implementazione in questa incarnazione sarà discussa brevemente nel prossimo. Una giunzione è indicata dalla keyword `junction`, seguita da una stringa che indica le molecole che fanno parte della giunzione. Questo aspetto sarà spiegato nel dettaglio nel capitolo successivo; in questa sezione, per completezza, riporterò solamente un esempio di come, in biochemistry, si può scrivere una reazione di formazione di una giunzione:

```
[A] + [B in neighbr] --> [junctionA-B]
```

e di come distruggerla:

```
[junctionA-B] --> []
```

Infine, in una reazione è possibile indicare delle *custom conditions* e delle *custom actions*, delle classi speciali che permettono di ampliare le possibilità di modifica dello stato della cellula da parte delle reazioni. Per ora infatti, non abbiamo visto un modo per richiedere alla cellula di compiere azioni complesse, come muoversi, ad esempio. Questo può essere fatto specificando direttamente il nome della classe dell'azione o della condizione desiderata nella reazione, che saranno poi istanziate tramite *reflection* [15].

Le custom actions sono indicate in maniera simile alle azioni normali, solo che invece di essere molecole sono classi Java. Un esempio è la reazione:

```
[] --> [BrownianMove(0.1)]
```

che, ogni volta che viene eseguita, sposta casualmente la cellula in una posizione distante al massimo 0,1 dalla sua poszione originale. `BrownianMove`, infatti, è una classe Java che implementa l'interfaccia `Action` (si veda sopra), costruita appositamente per muovere casualmente in nodo in cui è contenuta di una distanza compresa fra zero e il valore indicato tra parentesi.

Le custom conditions, invece, sono indicate a seguito delle azioni della reazione, precedute dalla keyword `if`. Ad esempio la reazione:

```
[A] --> [A in neighbor] if JunctionPresentInCell
```

può avvenire solo se la condizione implementata dalla classe Java `JunctionPresentInCell` è realizzata.

Capitolo 4

I requisiti per il movimento cellulare

Il movimento cellulare, come sarà spiegato meglio nel prossimo capitolo, è un fenomeno molto complesso ed estremamente variabile a seconda del tipo di cellula e dalla situazione specifica. Infatti, una delle più grosse difficoltà che abbiamo riscontrato in durante la sua realizzazione è stata trovare un modello sufficientemente generico da poter comprendere tutti i diversi connotati che questo fenomeno può assumere. Vi erano delle cose però che, a prescindere dal modello che avremmo adottato per il moto, era necessario implementare all'interno di Alchemist. La prima di queste era un meccanismo che facesse in modo che le cellule, nel muoversi, non si sovrapponessero. Infatti, i nodi non hanno un volume specifico, perciò potrebbero trovarsi in qualsiasi punto nello spazio, anche in uno già occupato da un altro nodo. Questa libertà nel posizionamento dei nodi non sarebbe stata verosimile per la nostra incarnazione; come tutti i corpi dotati di fisicità, le cellule infatti occupano uno spazio ben definito, che non può essere occupato da altre cellule. Il posizionamento e lo spostamento dei nodi rappresentati le cellule, quindi, andava regolato in qualche modo; noi, come vedremo nel corso di questo capitolo, lo abbiamo fatto conferendo alle cellule una forma circolare regolabile, che definisce una regione di spazio non occupabile da altre cellule. Approfondendo lo studio del movimento, altri due elementi rivelatisi di fondamentale per qualsiasi tipo di movimento cellulare sono le giunzioni e l'ambiente

extracellulare. Come spiegherò nel capitolo 5, infatti, è anche grazie alle giunzioni e alla composizione nell'ambiente che la cellula riesce a regolare il suo moto.

Queste considerazioni ci hanno convinto del fatto che fosse necessario lavorare sull'implementazione di questi tre elementi prima che del movimento cellulare. Sarà quindi scopo di questo capitolo spiegare come sono stati modellati e implementati, a partire dalle giunzioni cellulari.

4.1 Giunzioni e adesione cellulare

Contribuire alla modellazione delle giunzioni è stato il mio primo vero e proprio incarico nel progetto dell'incarnation biochemistry. Infatti è stato mio compito reperire tutte le informazioni sui meccanismi molecolari che portano alla loro sintesi e distruzione, durante la vita della cellula. Queste informazioni, tutte raccolte dal testo *Biologia molecolare della cellula* [1], sono poi servite alla stesura di un modello che è stato poi implementato e testato da Gabriele Graffieti [10].

4.1.1 I tipi di giunzione

Le giunzioni cellulari possono essere divise in quattro categorie [1]:

- Giunzioni di ancoraggio;
- Giunzioni serrate, o strette;
- Giunzioni comunicanti;
- Giunzioni che trasmettono segnali.

Tuttavia le ultime dell'elenco sono un po' particolari, in quanto non sono dei complessi molecolari ma, più precisamente, delle vere e proprie zone adibite della cellula adibite alla comunicazione. Per fare un esempio, le sinapsi neuronali vengono classificate in questa categoria. Queste zone in realtà non sono altro che adesioni formate da giunzioni di ancoraggio, unite a diversi complessi molecolari di segnalazione e recezione, attraverso le quali le cellule comunicano in modo

più complesso di quanto non possano fare tramite le altre giunzioni. Questo tipo di collegamento quindi è stato immediatamente scartato dal panorama di fenomeni che dovevamo modellare con il concetto di giunzione nelna nostra incarnazione, poichè lo abbiamo ritenuto di un livello molto più alto rispetto a quello che si forma con gli altri tipi di giunzione. Ovviamente questo non implica che le giunzioni che trasmettono segnali non possano essere simulate: possono essere modellate attraverso l'unione di entità più semplici, come appunto le altre giunzioni o i recettori. In questa sezione quindi saranno trattate solo le prime tre categorie di giunzione, quelle di ancoraggio, le serrate e le comunicanti.

4.1.2 Giunzioni di ancoraggio

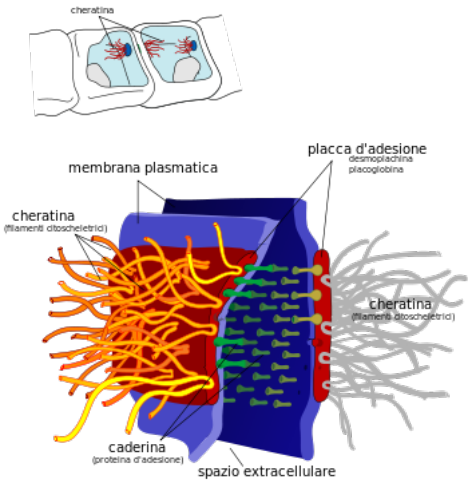

Figura 4.1: Una rappresentazione grafica di una giunzione di ancoraggio. É stato evidenziato come il citoscheletro (in giallo) si leghi alle caderine che formano la giunzione, formando quasi una continuità fra i citoscheltri delle due cellule [9]

Le giunzioni di ancoraggio sono delle connessioni *fra cellula e ambiente esterno* realizzate da particolari proteine transmembrana, che sono in grado di legarsi sia internamente che esternamente con altre proteine specifiche che a loro volta si legano:

- Dal lato della cellula con il citoscheletro della stessa;
- Dal lato del matrice extracellulare con altre strutture specifiche.

Il motivo per cui ho evidenziato che queste giunzioni formano una connessione fra cellula e ambiente esterno è che esse non collegano solo cellule diverse fra loro, ma anche la cellula alla matrice extracellulare in cui è immersa. In particolare si parla di giunzioni cellula-cellula nel primo caso e di giunzioni cellula-matrice nel secondo. Tuttavia in questa tesi approfondirò soltanto le prime, essendo le più studiate in letteratura [1] e quelle che eravamo più interessati a modellare.

Le caderine

La maggior parte delle giunzioni cellula-cellula animali sono mediate da proteine integrali dette *caderine*, che possono essere di moltissimi tipi differenti, anche a seconda del tipo cellulare. Sono suddivise in due macro-categorie: caderine che legano i filamenti intermedi del citoscheletro, come desmogleina e desmocollina, e caderine che legano i filamenti di actina. Questa divisione dà origine a due categorie di giunzioni cellula-cellula: le *giunzioni aderenti* e i *desmosomi*. Inoltre le caderine possono essere divise anche in "*classiche*" e "*non classiche*", a seconda che siano state scoperte più o meno recentemente.

Nonostante l'enorme varietà che esiste fra le caderine, queste si comportano tutte allo stesso modo:

- Tutte le caderine necessitano di calcio nell'ambiente extracellulare per formare l'adesione con la cellula. Infatti la porzione extracellulare delle caderine è formata da diverse ripetizioni di un motivo peptidico particolare, detto *dominio delle caderine*, che sono collegati fra loro da "cerniere". Queste, solitamente, non sono rigide e non permetterebbero una salda adesione fra le cellule, ma il legame con il calcio irrobustisce la porzione extracellulare della caderina, rendendola simile a un'asticella rigida sporgente dalla membrana. In questo modo il complesso possiede la forza necessaria per formare il legame con un'altra molecola di giunzione.

- Tutte le caderine mediano un'adesione *omofilica*, il che significa che una caderina si può legare solamente con una caderina identica. Questo implica tra l'altro che le giunzioni di ancoraggio impongano una distanza fra le cellule pari al doppio della lunghezza di una molecola di caderina.

- Tutti i legami caderinici sono piuttosto deboli. Infatti, in una giunzione cellulare solitamente non sono coinvolte solo due molecole, ma tante caderine legate in parallelo. Questo meccanismo permette di creare un "effetto velcro" grazie al quale la giunzione si dimostra resistente agli stress meccanici, pur rimanendo semplice da spezzare in caso di riarrangiamenti nell'organizzazione fra le cellule. Infatti in tal caso basta andare a rompere uno a uno, mediante enzimi specifici, i legami fra le molecole di caderina.

- Tutte le caderine generano un collegamento diretto fra i citoscheletri di cellule diverse, svolgendo un ruolo determinante nel coordinamento motorio fra esse.

- Molte caderine svolgono un ruolo fondamentale, se non coinvolte in un adesione cellulare, nella segnalazione extracellulare.

Viene quindi da chiedersi il motivo per il quale in natura si siano osservate così tante varietà di questa molecola. In realtà questo è un aspetto fondamentale per questo tipo di giunzioni, poiché, essendo le caderine in grado di formare solo legami omofilici, la loro grande varietà si traduce in un'altissima specificità nelle giunzioni di ancoraggio, che svolgono infatti un ruolo fondamentale nella separazione dei vari tessuti durante l'embriogenesi. Infatti le cellule nel tempo si differenziano, sviluppandosi in quelle che saranno poi le cellule muscolari, epiteliali, cerebrali, eccetera. Differenziandosi cambiano anche le caderine che possono esprimere, quindi le adesioni che possono formare. Cellule simili andranno, perciò, ad unirsi saldamente fra loro, creando il nucleo di quello che a sviluppo terminato sarà un tessuto a sè stante, separato dagli altri. Un altro esempio in cui questa specificità è importante, anche se non mediata solo da caderine, è nella formazione della sinapsi, in cui l'assone deve saper riconoscere qual'è la porzione giusta della cellula in cui deve legarsi.

Altre proteine

Le caderine sono le proteine più importanti fra le quelle che mediano le giunzioni di ancoraggio cellula-cellula, ma non sono le uniche. Da citare vi sono anche le *selettine* (o selectine), le *integrine*, che, pur essendo le principali mediatrici di giunzioni cellula-matrice, possono prendere parte anche ad adesioni cellula-cellula. Vi sono poi le proteine della super-famiglia delle *immunoglobuline*.

Selettine e integrine si comportano in modo piuttosto simile: entrambe necessitano di calcio nell'ambiente extracellulare per formare giunzioni; entrambe formano un'adesione *eterofilica* (le selettine con carboidrati presenti sulla superfice delle altre cellule, mentre le integrine con proteine specifiche); entrambe formano legami molto deboli, svolgendo quindi un ruolo importante solo in adesioni temporanee fra le cellule. Sono molto importanti nel sistema circolatorio, dove capita spesso che una cellula debba legarsi solo temporaneamente ad un'altra.

Le immunoglobuline invece, essendo una super-famiglia, si comportano in moltissimi modi diversi. Alcune sono eterofiliche, altre omofiliche e quindi in grado di contribuire alla specificità di legame fra cellule simili. Possono presentare una o più catene di domini extracellulari di tipo Ig. Infine, spesso sono espresse contemporaneamente alle caderine, ma formano legami molto più deboli di quest'ultime. In esperimenti sui topi infatti si è visto che la non espressione delle caderine porta alla morte prematura del feto, mentre la non espressione sie tipi più comuni di immunoglobuline sviluppano solo lievi danni cerebrali.

4.1.3 Le giunzioni strette

Alcune volte la semplice coesione fra le cellule non è sufficiente a caratterizzare un tessuto; questo è particolarmente vero nei tessuti epiteliali che, dovendo funzionare da barriera fra porzioni di spazio differenti, devono controllare con la massima precisione la transizione di sostanze, sia per via intracellulare, sia per via paracellulare (cioè fra le cellule stesse). A questo scopo esistono le giunzioni strette: queste sono giunzioni che minimizzano la distanza fra le membrane plasmatiche delle cellule che legano, rendendo sostanzialmente impraticabile la via paracellulare da parte delle molecole. Questa impermeabilità però non è

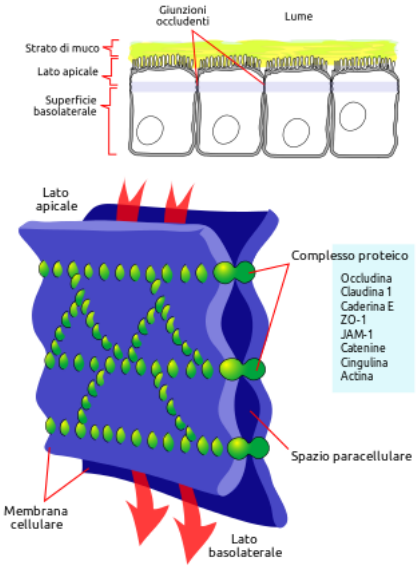

Figura 4.2: Rappresentazione grafica delle giunzioni strette. Viene anche evidenziato come esse si comportano nei tessutti empiteliali [9]

assoluta: si è visto che alcuni piccoli atomi riescono comunque ad attraversare le cellule unite da giunzioni serrate. Infatti a seconda delle proteine che formano la giunzione stessa il tipo e la quantità di questi atomi varia. In ogni caso, quindi, è comunque realizzato un efficiente controllo dello spostamento di queste particelle.

Il complesso proteico che realizza le giunzioni serate è molto diverso da quello delle giunzioni di ancoraggio. Esso infatti si basa su filamenti sigillanti formati, principalmente, da lunghe catene di *claudine* e *occludine*, che sono proteine transmembrana che si legano in maniera omofilica tra membrane vicine. Quello che si ottiene è un effetto trapuntato, quasi come se le cellule fossero cucite fra loro da lunghi filamenti che rendono lo spazio fra loro quasi nullo (si veda Figura 4.2).

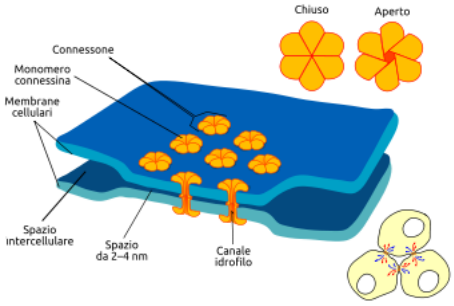

Figura 4.3: Rappresentazione grafica di una giunzione comunicante (fonte: Wikipedia).

4.1.4 Le giunzioni comunicanti

Le giunzioni comunicanti, negli animali, sono giunzioni che connettono direttamente il citsol delle cellule che collegano, permettendo così il passaggio di piccole molecole. Sono formate da due tipi di proteine: le connessine, le più comuni nei vertebrati, e le integrine. Sono entrambe proteine integrali a 4 passaggi, che si combinano in gruppi di 6 in una struttura tubolare detta *connessone* e vengono poi trasportate fino alla membrana cellulare, nella quale vengono incluse per esocitosi (in modo simile a molte altre proteine di membrana). Quando due connessoni di cellule diverse sono allineati, possono unirsi a formare un canale acquoso spesso circa 1,5 nm, da cui è consentito il passaggio di particelle. In modo analogo ad altre giunzioni che abbiamo visto però non è la proteina generata dall'unione dei due connessoni ad essere detta gap-junction, ma gruppi di esse messe tutte in parallelo. Questo dà alle gap-junction un aspetto caratteristico, che è stato definito "setaccio molecolare" [1]. Queste placche di connessoni inoltre non sono fisse, ma in continuo mutamento: infatti le connessine hanno un'emivita molto breve (circa due ore) e quindi sono continuamente incluse e sostituite nella giunzione. Per quello che riguarda la permeabiltà delle gap-junction, come già detto, queste consentono il passaggio solo a molecole molto piccole, come ioni inorganici, zuccheri, amminoacidi, nucleotidi, vitamine, eccetera. La quantità e il tipo delle molecole che vengono trasportate può variare da cellula a cellula. Infatti esistono diversi tipi di connessine che le cellule possono esprimere e, a

seconda di come queste si combinano nei connessoni, la permeabilità della giunzione risultante cambia. Si noti, però, che l'utilizzo da parte di una cellula di certe combinazioni di connessine non forza le cellule vicine a usare le stesse combinazioni: connessoni di cellule diverse possono combinarsi indipendentemente dal tipo di connessine da cui sono formati (queste giunzioni sono caratterizzate infatti da bassissima specificità). Inoltre, la permeabiltà delle gap-junction, come la permeabilità dei canali ionici, è in una certa misura regolabile da parte della cellula: determinati eventi intracellulari possono ridurre o annullare reversibilmente la permeabilità di una gap-junction, come anche aumentarla.

Per concludere, possiamo dire che l'utilità di questo tipo di giunzioni è soprattutto l'attuazione di un meccanismo di scambio cellulare veloce e affidabile. Infatti è un tipo di giunzione molto presente nelle cellule eccitabili elettricamente, che devono trasmettere la correte elettrica alle cellule vicine in maniera diretta. Ma è anche un meccanismo molto utle per fare in modo che gruppi estesi di cellule mantengano la concentrazione di certe sostanze allo steso livello; questo, a volte, è necessario per garantire una risposta univoca da parte di una colonia di cellule.

4.1.5 Il modello

Il nostro scopo, come già affermato, era modellare questi tre tipi di giunzione, che sono però tutti estremamente diversi fra loro. Si è deciso, tuttavia, di modellare le giunzioni come un'unica entità. Il motivo di questa scelta è che le reazioni programmabili all'interno della cellula possono coinvolgere direttamente le giunzioni, permettendo di modellarne il comportamento a discrezione dell'utente. Se ad esempio fosse necessario includere una giunzione comunicante che permette il passaggio di calcio da cellula a cellula, basterà scrivere:

```
[junctionA-B + Ca] --> [Ca in neighbor] + [junctionA-B]
```

Ciò che rende possibile questa flessibilità è stata la scelta di *modellare la giunzione come una molecola*, creando l'estensione `Junction` della classe `Biomolecule`. Queste, infatti, saranno sempre caratterizzate da

un nome, che però non sarà a discrezione dell'utente, come per le normali molecole (si veda la sezione 3.2.2). Il nome della giunzione dovrà cominciare sempre con la stringa `"junction"`, seguita a un'ulteriore stringa caratterizzante, che indica le molecole da cui la giunzione è formata. Prendiamo l'esempio sopra, dove viene indicata la giunzione `junctionA-B`: la stringa `"A-B"` sta a indicare che questa è formata da una molecola di A della parte della cellula in cui la reazione è contenuta e da una molecola di B dalla parte del vicino. Se invece avessimo la stringa `"junction6ConnessinaA-3ConnessinaB:3ConnessinaC"`, questa indicherebbe una giunzione formata da sei molecole di ConnessinaA dal lato della cellula in cui è contenuta la reazione e da 3 ConnessinaB e 3 ConnessinaC dal lato della cellula vicina. In generale quindi, nella nomenclatura scelta, il segno "`-`" divide le molecole delle due cellule, mentre il segno "`:`" divide molecole della stessa cellula. Un'altra differenza con le normali molecole è che in realtà una giunzione è formata da *due istanze* della classe progettata (`Junction`), una per ognuna delle cellule legate dalla giunzione stessa. Questo perché una giunzione deve far parte di entrambe le cellule giunte, non solo di una. Ovviamente però il nome delle de molecole sarà invertito: se in una molecola la giunzione si chiama `junctionA-B`, nell'altra si dovrà chiamare `juntionB-A`.

La giunzione in sè però non era l'unico aspetto che doveva essere modellato: era necessario anche considerare anche la formazione e la distruzione delle giunzioni a seconda della distanza della cellula vicina. Come si potrà immaginare infatti le cellule non possono formare legami con tutte le altre cellule presenti, ma solo con quelle sufficientemente vicine. Tuttavia dare una definizione precisa di "vicinanza sufficiente" non è certo banale: ci sono giunzioni di lunghezza e resistenza diversa, tanto che sarebbe risultato estremamente complesso stendere un modello che considerasse tutti i possibili comportamenti delle giunzioni, soprattutto considerando che le cellule possono muoversi relativamente fra loro. Si è quindi scelto, per ora, di permettere alle cellule di formare giunzioni solo con le cellule appartenenti al vicinato, che ricordiamo essere definito dalla linking-rule. Quando una cellula si trova nel vicinato di un'altra cellula può formare una giunzione con essa, quando ne esce la giunzione si spezza.

L'implementazione, come avevo anticipato all'inizio della sezione, è

stata completamente svolta da Gabriele Graffieti, che ha anche testato la realizzazione, ottenendo ottimi risultati. Per maggiori informazioni quindi rimando alla sua tesi di laurea [10].

4.2 La forma cellulare

Come ho spiegato nell'introduzione, per una corretta resa del moto cellulare nell'incarnazione `biochemistry` era necessario trovare un meccanismo che regolasse la posizione dei nodi in relazione a quella degli altri, facendo in modo che non potesse esservi sovrapposizione fra questi. Tuttavia, questo meccanismo non doveva corrispondere ad un'implementazione del volume cellulare: questa avrebbe richiesto un lavoro molto più consistente, perché il volume cellulare rappresenta un aspetto molto complessa della cellula. Quello che ci interessava era solo conferire una forma semi-verosimile alla cellula, in modo tale che vi fosse un controllo più accurato sulla sua posizione.

Fra tutte le forme semplici, quella più vicina alla realtà biologica ci è sembrata il cerchio, visto che in genere le cellule (o almeno quelle non vegetali), presentano una forma rotondeggiante. Il primo passo è stato quindi fare in modo che ad ogni cellula venisse associato un'area circolare, che ne definisse la forma.

Il passo successivo è stato realizzare il meccanismo che evitasse la sovrapposizione fra questi cerchi. Per farlo, abbiamo implementato una classe Java, chiamata `BioRectEnvironmentNoOverlap`, che estende l'interfaccia `Environment` di Alchemist. Si ricorda, infatti, che nel nostro simulatore "*l'environment*" rappresenta semplicemente la struttura dati che si occupa di gestire la posizione e lo spostamento delle cellule (si veda sezione 2.2); non va quindi confuso con l'ambiente extracellulare discusso nella sezione 4.3, che è un'astrazione propria dell'incarnation `biochemistry`.

4.2.1 La forma cellulare

Per associare alle cellule una forma circolare, abbiamo innanzitutto creato un'interfaccia chiamata `CellWithCircularShape`, per separare concettualmente le cellule generiche (rappresentate da `CellNode`) da quelle dotate di forma circolare; questa distinzione è stata fatta in

vista di una possibile aggiunta futura di forme cellulari differenti da quella circolare. L'interfaccia contene due semplici metodi:

- `getDiameter();`
- `getRadius();`

con il compito di fornire la dimensione del cerchio associato alla cellula. Abbiamo poi fatto implementare questa interfaccia dalla classe `CellNodeImpl`, già creata durante il lavoro di G. Graffieti [10], arricchendola con i metodi descritti sopra e con un campo di tipo `double`, `diameter`, che rappresenta il diametro della cellula. Questo campo viene definito nel momento in cui la classe viene istanziata attraverso un costruttore apposito, che prende in ingresso tale valore.

4.2.2 `BioRectEnvironmentNoOverlap`

Come abbiamo detto, l'ambente in Alchemist rappresenta l'entità che si occupa di gestire il posizionamento e lo spostamento dei nodi. Infatti, come possiamo vedere in Figura 4.4, questo possiede diversi metodi che vengono chiamati quando la cellula viene posizionata o spostata. Come per ogni interfaccia Java, sarà poi scopo delle implementazioni `Environment` definire con precisione questi metodi. Il

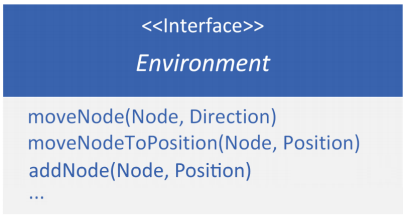

Figura 4.4: Interfaccia `Environment`; sono stati riportati solo i metodi correlati al posizionamento e allo spostamento dei nodi.

nostro compito è stato, quindi, realizzare un ambiente che estendesse `BioRectEnvironment`, l'ambiente principale dell'incarnation biochemistry [10], in modo tale da sovrascriverne due metodi:

- `nodeShouldBeAdded(Node, Position)`, un metodo utilizzato da `addNode(Node, Position)` per verificare che la posizione data in ingresso sia occupabile dal nodo;
- `moveNodeToPosition(Node, Position)`, che dato un nodo e una posizione richiesta (`Position`), sposta il nodo della posizione più vicina possibile a quella richiesta.

`nodeShouldBeAdded(Node, Position)`

Come suggerisce il nome, questo metodo ritorna un booleano, che sarà `true` nel caso in cui la posizione richiesta sia libera e `false` nel caso contrario. Per il nostro ambiente, quindi, doveva restituire in uscita `false` nel caso in cui la posizione richiesta portasse la cellula a sovrapporsi con qualcuna delle altre. Per fare ciò è stato implementato il seguente algoritmo:

1. Data la posizione richiesta p, si controlla in un suo intorno, di ampiezza parti alla somma fra il raggio della cellula e il raggio della cellula più grande contenuta nell'ambiente, se sono presenti delle cellule. Questo intorno, infatti, rappresenta lo spazio massimo in cui può trovarsi una cellula che può presentare delle sovrapposizioni con la cellula da posizionare. Se tale intorno è vuoto, si ritorna immediatamente `true`;
2. Se invece vi sono delle cellule, si controlla se queste si trovano una distanza tale da non sovrapporsi con la cellula che dev'esser posizionata; questo corrisponde a controllare se la distanza di ognuna delle cellule da p è maggiore o uguale a $r_{cell} + r_{cellToAdd}$, dove r_{cell} è il raggio della cellula considerata, mentre $r_{cellToAdd}$ è il raggio della cellula da aggiungere. Se nessuna di queste si sovrappone, il metodo restituisce `true`, altrimenti `false`.

`moveNodeToPosition(Node, Position)`

Questo metodo gestisce lo spostamento della cellula (nel nostro caso) ed è uno dei più importanti per quel che riguarda questo elaborato. Infatti, sarà questo metodo che verrà richiamato da tutte le `Action` che gestiscono il moto cellulare (si veda ultimo capitolo) nella nostra

incarnazione. Questo doveva essere implementato in maniera tale che, data una posizione richiesta, il nodo fosse spostato il più vicino possibile a quest'ultima considerando tutti gli altri nodi come ostacoli, cioè fermandosi se avesse incontrato un altro nodo (si veda Figura 4.5). Si noti che questo comportamento non rappresenta con naturalezza

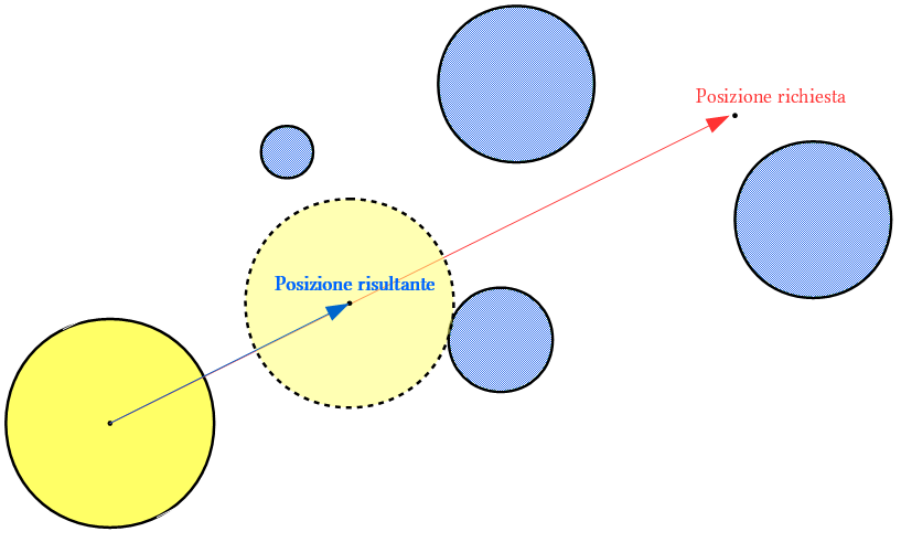

Figura 4.5: Rappresentazione grafica del funzionamento di `moveNodeToPosition(Node, Position)` nell'ambiente implementato. In giallo è rappresentata la cellula da spostare, in violetto le altre. Si noti che le cellule possono essere di varie dimensioni.

il movimento di una cellula, perché in natura le cellule possono interagire meccanicamente, spostandosi a vicenda. In questa fase della progettazione, però, non era questo l'aspetto che ci interessava: noi dovevamo solo fare in modo che le cellule non si sovrapponessero. L'interazione meccanica fra le cellula è stata poi modellata e implementata separatamente (si veda sezione 5.4).

L'algoritmo attraverso il quale questo spostamento viene compiuto è piuttosto lungo e complicato, quindi, per non dilungarmi troppo su aspetti tecnici, non lo spiegherò nel dettaglio in questo elaborato. Semplificando, si può dire che la determinazione della posizione risultante avviene in quattro passi:

1. Si controlla se vi sono nodi fra la posizione originale della cellula e quella richiesta;

2. Si verifica quali e quanti di questi potrebbero costituire un ostacolo;

3. Si isola il più vicino alla posizione iniziale della cellula;

4. Si calcola la posizione risultate attraverso la formula:

$$p_{new} = \vec{u} \cdot [||(p_{int} - p_{old})|| - c]$$

$$\text{con c pari a } c = \sqrt{d_{co}^2 - d_{oi}^2} \tag{4.1}$$

dove tutti i valori sono esposti Figura 4.6.

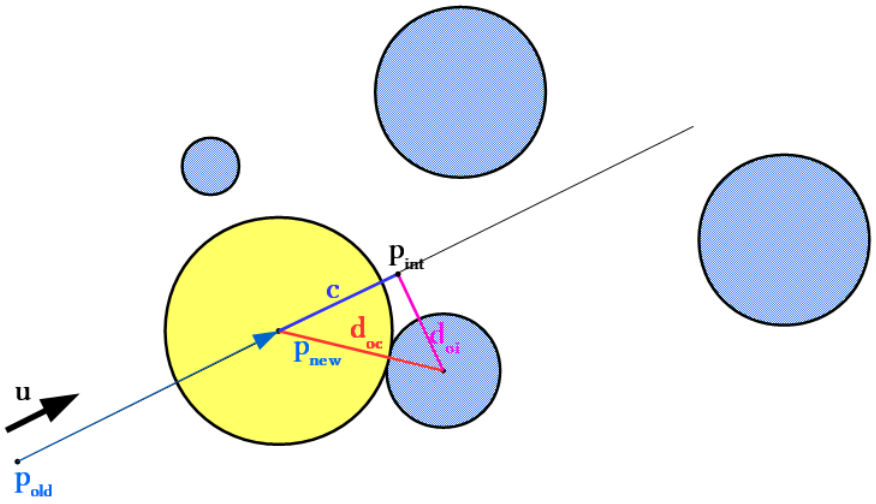

Figura 4.6: Rappresentazione grafica del metodo utilizzato per determinare la posizione risultante p_{new}. Tutti le distanze e le posizioni mostrate in figura vengono determinate nelle varie fasi dell'algoritmo. $\vec{u}$ rappresenta un versore nella direzione definita da $(\vec{p_{new}} - \vec{p_{old}})$

4.3 L'ambiente extracellulare

In natura non esiste un corrispettivo univoco a quello che noi qui chiameremo "ambiente": in ambito fisiologico corrisponderebbe alla matrice extracellulare (ECM), ma in generale delle cellule potrebbero trovarsi in contatto con qualsiasi cosa, dalla semplice acqua a un composto sintetico. Si può dire quindi che l'ambiente extracellulare di questa incarnazione nasce per modellare qualsiasi cosa, diversa da una cellula, con la quale la cellula possa entrare in contatto.

Come è immaginabile, la relazione fra la cellula e ciò che la circonda è di importanza fondamentale in moltissimi ambiti della vita cellulare. Si pensi solo al potenziale di membrana e delle sue variazioni, che sono principalmente causate da uno scambio continuo con l'ambiente di ioni. Di importanza fondamentale è inoltre per il movimento cellulare, poichè la cellula, come sarà spiegato meglio, prende sempre come riferimento l'ambiente per capire che direzione prendere nel suo moto. L'introduzione di un'interazione diversa da quella che la cellula aveva già con le altre cellule, quindi, era prioritaria nello sviluppo di questa incarnazione. Abbiamo perciò sviluppato un modello che fosse sufficientemente generico da abbracciare le diverse esigenze del modellista, poi l'abbiamo implementato in Alchemist e testato con alcune semplici simulazioni.

4.3.1 Il modello

Sebbene i contesti in cui un gruppo di cellule può trovarsi siano infiniti, le nostre necessità erano semplici e ci siamo posti l'obbiettivo di creare un'implementazione che fosse il più flessibile possibile. Infatti, necessitavamo di un modello che supportasse semplicemente lo scambio di molecole fra cellula e esterno e il moto libero e programmabile delle molecole contenute nell'ambiente da una zona all'altra (in particolare per simulare la diffusione di sostanze). Per fare ciò era necessario un meccanismo semplice e non troppo pesante dal punto di vista computazionale, che però tenesse conto delle continue variazioni dell'ambiente causate dalle attività delle cellule e dal movimento delle molecole.

Una delle prime ipotesi fu quella di considerare l'ambiente come un nodo, esattamente come una cellula. L'ambiente sarebbe stato modellato, quindi, come un semplice compartimento, capace però di interagire con tutte le cellule, non solo con quelle vicine. Tale approccio sarebbe stato molto vicino a quello dei classici modelli a compartimenti e, perciò, ci avrebbe probabilmente permesso di ottenere buoni risultati dal punto di vista quantitativo; il suo punto debole, però, risiedeva nell'incapacità di supportare la distribuzione spaziale delle molecole. I nodi di Alchemist, infatti, hanno un'estensione spaziale pari a quella di un punto. Come definire quindi un gradiente di concentrazione di una particolare molecola? Come associare un atomo a una posizione spaziale? Qualsiasi soluzione sarebbe stata troppo pesante computazionalmente, quindi abbiamo quasi subito abbandonato questa idea.

Un'altra possibilità sarebbe stata quella di introdurre delle entità completamente nuove in Alchemist, dette Layer (letteralmente "strato"). Questi sarebbero dovute essere strutture dati con il compito di implementare la distribuzione spaziale di una molecola. In pratica, data una posizione o un'area nello spazio, sarebbe stato compito del Layer associare a quest'ultima una certa concentrazione della molecola. Questo approccio, tuttavia, presentava una pecca non indifferente: non garantiva una gestione semplice della dinamicità dell'ambiente. Il motore di simulazione di Alchemist, infatti, ammette variazioni runtime (cioè durante la simulazione) solo all'interno dei nodi, perché solo all'interno di questi sono contenute le reazioni. Rendere i Layer dinamici al pari dei nodi, però, avrebbe richiesto radicali modifiche al motore di simulazione, che sarebbero state troppo complesse e dispendiose in termini di tempo. Ci saremmo , quindi, dovuti accontentare di un'ambiente statico, incapace di mutare e di rispondere alle variazioni delle cellule nel tempo. Nemmeno questa strada dunque è stata percorsa fino in fondo.

La nostra scelta finale è stata quella di modellare l'ambiente extracellulare non come una singola entità, bensì come una griglia di nodi capaci di interagire sia fra di loro, che con le cellule. Concettualmente, ognuno di questi nodi, detti "ambientali", corrisponderebbe quindi a una "porzione di ambiente", come se quest'ultimo fosse diviso in tanti piccoli quadratini, ognuno con una sua identità.

Questa idea è decisamente in contrasto con i comuni modelli di ambiente e anche con l'idea stessa di ambiente extracellulare, che è più facile immaginare come un'entità unica e continua in ogni sua parte. Questo approccio, però, garantisce all'ambiente extracellulare una flessibilità senza pari. Innanzitutto, in questo modo è possibile simulare sia ambienti uniformi, più comuni, che ambienti che presentano comportamenti diversi in zone diverse. Così, se si desidera un ambiente uniforme basta programmare tutti i nodi ambientali allo stesso modo, con le stesse reazioni e le stesse concentrazioni iniziali delle molecole; se, invece, si desidera un ambiente che presenta comportamenti differenti a seconda della zona, si possono programmare i nodi in modo da fare quanto richiesto. Ad esempio, si potrebbe simulare la generazione spontanea di una molecola in una piccola parte della matrice semplicemente programmando un gruppo di nodi col compito di produrre quella molecola. Oppure, si potrebbero inserire valori di concentrazione crescenti per una molecola in modo da modellarne un gradiente nello spazio. Infine, con questo approccio sono pienamente supportati lo scambio di molecole da ambiente a cellula e il moto di una molecola nella matrice, perché entrambi questi processi corrispondono a uno scambio di molecole fra nodi.

Lo svantaggio principale di questo approccio, però, risiede nella necessità di aggiungere moltissimi nodi, che rendono la simulazione più complessa. Alchemist, tuttavia, non presenta grandi variazioni di performance all'aumentare del numero di compartimenti (si veda Figura 2.5), perciò questa complessità non pesa troppo sul tempo di simulazione.

4.3.2 L'implementazione

L'implementazione di questo ambiente "reticolare" ha richiesto l'aggiunta di quattro nuove classi Java e di una interfaccia:

- L'interfaccia `EnviornmentNode`, che estende la già presente interfaccia `Node`;
- La classe `EnvrironmentNodeImpl`, implementazione della precedente interfaccia;
- L'azione `ChangeBiomolConcentrationInEnv`;

- La condizione `BiomolPresentInEnv`;
- La condizione `EnvPresent`.

Le prime due corrispondono all'implementazione dei nodi ambientali, che non sono molto diversi dai nodi cellulari (`CellNode`). L'unica differenza, infatti, è che questi ultimi non possono sovrapporsi fra loro (come vedremo), mentre i nodi ambientali si.

La `Action` implementata serve sostanzialmente a fare in modo che una reazione possa avere come effetto l'aggiunta o la rimozione di una o più molecole in un nodo ambientale. Se, ad esempio, si desiderasse specificare una reazione in cui una molecola viene rilasciata nell'ambiente attraverso un canale ionico, questa si potrebbe scrivere come:

```
[channel + Ca] --> [channel] + [Ca in env]
```

Che semplicemente controlla se le molecole "channel" e "Ca" sono nella cellula e, se lo sono, rimuove una molecola di Ca dalla cellula e ne aggiunge una nel nodo ambientale più vicino. Se i nodi ambientali vicini fossero tutti alla stessa distanza, allora viene scelto quello che presenta la minore concentrazione di Ca.

Le `Condition` servono a fare in modo che una reazione possa avvenire solo nel caso siano presenti dei nodi ambientali intorno alla cellula (`EnvPresent`) o nel caso una qualche molecola si trovi nell'ambiente `BiomolPresentInEnv`. Ad esempio la reazione contraria a quella sopra:

```
[channelOpen] + [Ca in env] --> [channelOpen] + [Ca]
```

dovrebbe poter avvenire solo se ci sono dei nodi ambientali nelle vicinanze e se, in almeno uno di questi, è presente una molecola calcio. Quando si scrive a sinistra della reazione [Ca in env], infatti, si richiamano implicitamente le condizioni `BiomolPresentInEnv` e `EnvPresent`, che si occupano dei controlli necessari.

Capitolo 5

Il movimento cellulare

La modellazione e l'implementazione del movimento cellulare sono stati i miei più sostanziali contributi all'incarnazione biochemistry. Quando ci imbarcammo in questa impresa, non ci aspettavamo che dietro a un fenomeno apparentemente così semplice potesse nascondersi una tale complessità. I meccanismi di regolazione del movimento, infatti, sono estremamente variabili tra cellule di tipo diverso; inoltre, si tratta di un fenomeno che, nel complesso, è stato poco studiato: quasi tutti i dati reperibili derivano, in realtà, da studi che riguardano solo marginalmente il movimento cellulare. Nonostante ciò, tale fenomeno è di fondamentale importanza in un gran numero di contesti di interesse scientifico e patologico, primi fra tutti l'embriogenesi [5] e la risposta immunitaria [6]. Tuttavia, alla fine siamo riusciti a trovare un modello che riteniamo sufficientemente generico da contenere qualsiasi tipo di movimento cellulare, che abbiamo poi applicato alla realizzazione del movimento casuale e di un moto di maggiore rilevanza biologica: la *chemiotassi*. Utilizzando lo stesso modello poi, siamo riusciti anche a realizzare un meccanismo per rendere possibile l'interazione meccanica fra le cellule in Alchemist, fenomeno che, nella prima parte del progetto, era stato completamente trascurato. Scopo di questo capitolo sarà quindi spiegare le fasi che hanno permesso la stesura di tale modello, a partire dalla biologia del movimento cellulare. Infine, spiegherò come sono stati implementati i tre tipi di movimento supportati, per ora, in Alchemist: quello casuale, quello dato dalla chemiotassi e quello causato dalla spinta di altre cellule.

5.1 Il fenomeno

In questa prima sezione mi dedicherò alla spiegazione del movimento cellulare come fenomeno biologico, al fine di rendere comprensibile le scelte modellistiche e implementative che ne hanno poi reso possibile la realizzazione nell'incarnazione biochemistry. É doveroso precisare, però, che il movimento di cui parleremo nel corso di questo capitolo sarà solo di tipo *ameboide* (*crawling*); infatti, abbiamo scelto di trascurare altri tipi di movimento, come il quello flagellare e quello ciliale, in quanto tipici di situazioni in cui la cellula si trova isolata, non dei sistemi multicellulari che è nostro obbiettivo simulare [5].

Per cominciare, esporrò i meccanismi base del *crawling* cellulare, poi spiegherò brevemente cos'è la chemiotassi e perché, fra tutti i tipi di movimento cellulare, abbiamo scelto di modellare proprio questo.

5.1.1 Il nano-machinario della locomozione cellulare

Il *nano-macchinario della locomozione cellulare* [5] è un termine utilizzato dallo scienziato inglese Jamie A. Devis per indicare quell'insieme di meccanismi che sono realizzati in tutte le cellule che praticano il movimento ameboide. Questo fenomeno, infatti, si fonda sempre su tre fenomeni principali:

- la *protrusione*, che avviene in accordo con una di una determinata polarizzazione della cellula, come vedremo successivamente;
- lo scorrimento della maggior parte del citoplasma, organuli e nucleo compresi;
- la ritrazione della parte posteriore della cellula in direzione del moto.

Queste tre fasi si succedono per dare luogo alla locomozione cellulare, ma non avvengono in modo perfettamente ciclico: spesso si svolgono in sovrapposizione o addirittura in contemporanea fra di loro, poiché ognuno di questi fenomeni è sostanzialmente indipendente dagli altri [5]. Inoltre, ogni fase viene regolata in maniera separata nella cellula, tanto che si sono osservati casi in cui, a causa della mancata espressione

di alcuni geni, avveniva la protrusione senza che vi fosse lo scorrimento del citoplasma e viceversa [14] [5].

Protrusione

La protrusione è la prima fase dell'avanzamento della cellula, che avviene per avanzamento di piccole parti del citoscheletro in seguito a una certa *polarizzazione*, cioè ad un'attività differenziata di determinati enzimi rispetto alla zona della cellula. La protrusione, infatti, non è di per sè ciò che determina la direzione del moto: in mancanza di stimoli esterni che provochino una polarizzazione, può avvenire in maniera casuale in tutte le direzioni, portando la cellula a muoversi casualmente. Ciò che determina tale direzione è la polarizzazione; questa, infatti, non è altro che una differenza di attività degli enzimi che regolano l'attività protrusiva, che si concentrano in una determinata zona, favorendone (o sfavorendone) lo svolgimento e determinando, quindi, dove si muoverà la cellula. Le zone in cui viene favorita l'attività protrusiva vengono chiamate in due modi, a seconda della loro forma forma: *filopodia* e *lamellipodia*.

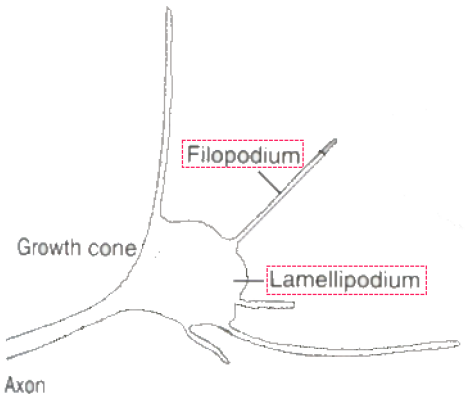

Figura 5.1: La differenza fra un *filopodium* e un *lamellipodium* in neurone [5].

Lamellipodium. Il lamellipodium è una zona protrusa della cellula di forma piatta, della quale struttura interna è per la maggior parte occupata da micro-filamenti proteici. I filamenti seguono una

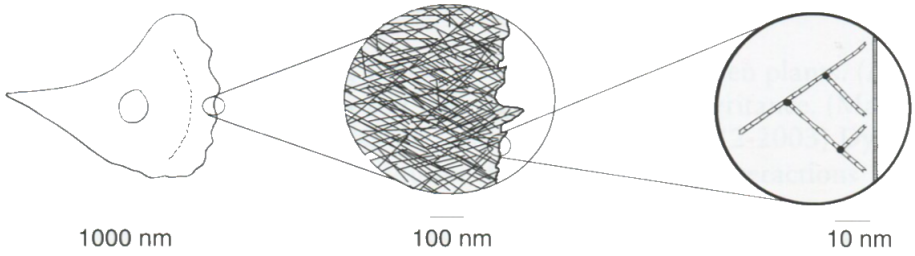

Figura 5.2: La composizione di un lamellipodium, mostrata a diversi livelli di dettaglio [5].

struttura molto precisa: si dispongono sempre in maniera tale da formare un angolo fra i 60 e i 40 gradi con la membrana cellulare (si veda Figura 5.2); inoltre, sono dotati di una grande flessibilità, che gli permette di essere facilmente piegati dalle interazioni con le altre molecole del citoplasma. Questi soli due fattori realizzano quasi completamente il nano-macchinario della protrusione: quando i filamenti vengono deformati, appositi enzimi ne accrescono la lunghezza, aggiungendo proteine alle loro terminazioni; quando l'attrazione molecolare non è più sufficiente a mantenere la deformazione, il filamento torna alla sua posizione originale, rilasciando repentinamente tutta l'energia accumulata. La forza elastica si traduce in energia meccanica, che spinge, letteralmente, in avanti la membrana cellulare (si veda Figura 5.3). Questo meccanismo, applicato a tutti i filamenti del lamellipodium, fa procedere il citoplasma della cellula, realizzando quindi la protrusione.

Filopodium. I filopodia sono simili ai lamellipodia, in quanto anch'essi sono formati da lunghi filamenti proteici, ma presentano con questi ultimi alcune piccole differenze. Innanzitutto, nei filopodia i filamenti si dispongono perpendicolarmente con la membrana, non in direzione obliqua. Questi crescono formando una protrusione lunga ed esile, in cui i filamenti proteici sono tenuti insieme da apposite proteine (si veda Figura 5.4). Unendosi in questo modo la rigidità dei filamenti aumenta, permettendogli di non deformarsi e, anzi, di spingere la membrana cellulare ad assumere la forma descritta. Inoltre, sembra che il loro sviluppo non avvenga aggiungendo proteine al termine del

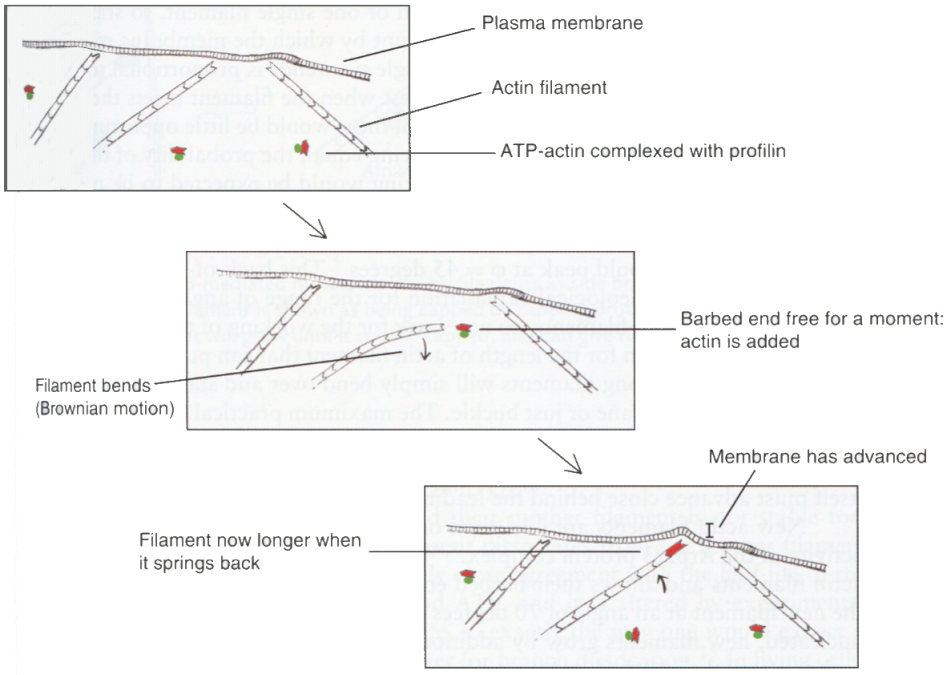

Figura 5.3: Rappresentazione del meccanismo che permette l'avanzamento dei lamellipodia, che è stato anche osservato in vivo grazie a proteine fluorescenti [5]. Nell'esempio mostrato i microfilamenti sono di actina, che sono i più comuni, ma possono essere anche basati su un'altra proteina.

filopodia, come avviene nel caso dei lamellipodia, ma aggiungendole da sotto, dove il filamento proteico ha avuto origine.

Avanzamento del corpo cellulare

La protrusione però non può bastare da sola a muovere l'intera cellula; la fisica elementare, infatti, ci insegna che questa può spostarsi in avanti solamente spingendo indietro l'ambiente che la circonda, cioè trascinandosi, letteralmente, da un posto all'altro (il termine inglese *crawling*, infatti, indica proprio l'azione di strisciare, di trascinarsi). Per potersi trascinare però la cellula necessita di un modo per "aggrapparsi" alla superficie su cui poggia. Questa adesione viene realizzata

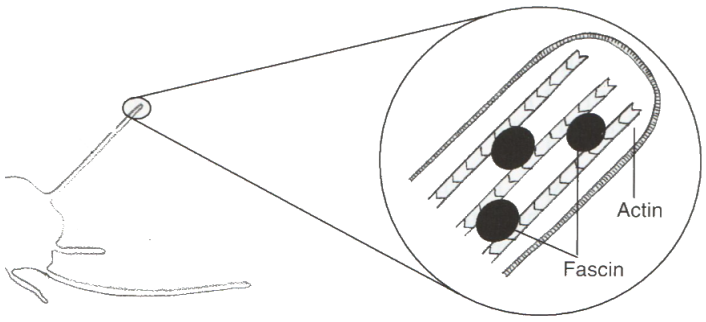

Figura 5.4: La composizione interna di un filopodium [5]. Anche qui sono stati presi in esempio filamenti di actina, unite da proteine di *fascina*, ma queste potrebbero variare da cellula a cellula.

attraverso delle giunzioni cellulari (si veda sezione 4.1). All'interno dei lamellipodia, infatti, si formano nuove, piccole adesioni, che però sono molto più resistenti delle vecchie [5] (si veda Figura 5.5). Al fronte di protrusione sono poi locati speciali complessi (spesso formati da ponti actino-miosinici), che fanno contrarre la cellula in maniera simile a un miocita. Conseguentemente a questa contrazione le giunzioni più vecchie, che abbiamo detto essere meno resistenti delle nuove, si spezzano, in modo tale che il risultato sia un netto avanzamento della cellula.

Ritrazione del retro della cellula

L'ultima fase del movimento cellulare riguarda la ritrazione della parte posteriore della cellula, che deve seguire l'avanzamento del corpo principale. Questa non avviene, però, solo in maniera attiva, come per la trazione del corpo principale. Si pensa, infatti, che avvenga per la combinazione di due fattori: un'effettiva contrazione, mediata da complessi molecolari del tipo actina-miosina (in maniera simile a quanto spiegato nella sezione precedente) e una contrazione passiva, dovuta soltanto alla degradazione del citoscheletro in questa parte della cellula. Nel retro della cellula, infatti, il citoscheletro non si sviluppa come sul fonte di protrusione; questo causa una sua veloce degradazione, che si traduce in una ritrazione della membrana cellulare sotto il peso della sua stessa tensione superficiale che, non essendo più contrastata

dal citoscheletro, porta la membrana a contrarsi. Il contributo di quest'ultima fase al movimento dipende quindi solamente dall'entità del termine "attivo" della ritrazione, che può cambiare a seconda dello stato della cellula. Alcune volte può essere sufficientemente grande da dare un contributo misurabile all'avanzamento, mentre altre può essere quasi nullo, e quindi non contribuisce al moto.

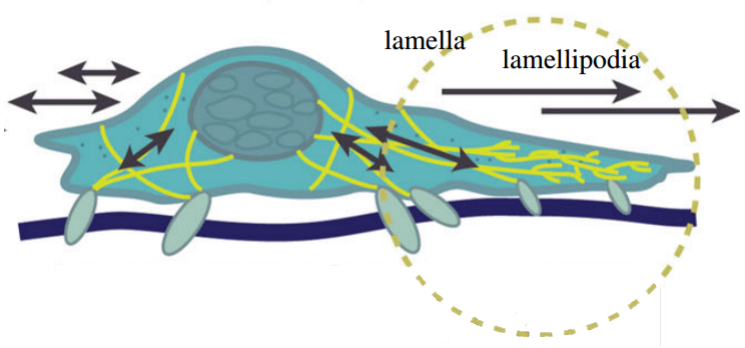

Figura 5.5: Raffigurazione del movimento cellulare, compresa la formazione delle giunzioni nei lamellipodia [11]. Si noti che tali giunzioni sono più piccole di quelle che si trovano nel corpo centrale della cellula.

5.1.2 La chemiotassi

Abbiamo detto che la direzione del movimento, nel caso del *crawling* cellulare, viene determinata dalla polarizzazione, che avviene conseguentemente alla recezione di uno stimolo che porta la cellula a mettere in moto il nano-macchinario del moto (sezione 4.1.1). Questi stimoli possono essere molteplici e sono tutti dovuti a particolari condizioni ambientali: la galvanotassi, ad esempio, è il tipo di movimento che si verifica in presenza di campo elettrico; la navigazione "per contatto" [5], invece, avviene in matrici extracellulari solide a causa delle loro piccole imperfezioni, o della loro composizione chimica. Tuttavia, nella nostra incarnazione solo uno si questi movimenti è stato implementato: la chemiotassi, cioè il tipo di movimento cellulare che si osserva in risposta ad un gradiente chimico presente nell'ambiente extracellulare. Il motivo di questa scelta è che, fra tutti i tipi di movimento, questo ci è sembrato uno dei più importanti, vista la sua rilevanza in fenomeni

come la fecondazione umana e la risposta immunitaria [5] [14]. Inoltre, era uno dei più agevoli da modellare e implementare. Formatosi il gradiente, infatti, le cellule possono muoversi solamente in due direzioni: risalendo il gradiente, dirigendosi verso concentrazioni maggiori della molecola, o discendendolo, andando verso concentrazioni minori.

5.2 La modellazione

Chiariti i meccanismi biologici alla base del movimento cellulare, posso ora procedere con la spiegazione del modello che abbiamo adottato per realizzarne un'implementazione sufficientemente generica da supportare ogni forma in cui questo può manifestarsi. Un tale modello non era strettamente necessario all'implementazione della chemiotassi, che era il nostro obbiettivo principale. Comunque, abbiamo preferito mantenerci il più possibile generici, in modo da facilitare un'eventuale implementazione futura degli altri tipi di movimento cellulare.

5.2.1 Premesse

Per realizzare un modello che tenesse conto sia gli aspetti qualitativi, che di quelli quantitativi del moto, la conoscenza di come avviene il fenomeno a livello cellulare non era sufficiente. Serviva infatti, chiarire alcune questioni di carattere tecnico, che possono essere riassunte in unica, fondamentale domanda:

In che modo i vari fattori che danno luogo al movimento influenzano la sua velocità?

Sapere quale velocità dev'essere associata alla cellula in moto a seconda del tipo di movimento, infatti, è un requisito base della modellazione. In generale, per rappresentare realisticamente un movimento non basta sapere dove il corpo in moto si sta dirigendo, ma serve anche un modo per determinare a che velocità sta andando. La domanda indicata sopra racchiude in sè almeno altre due domande, corrispondenti ad altrettanti aspetti da determinare:

- Le forze di contrazione che agiscono sulla cellula, provocandone il moto, ne causano un'accelerazione o ne determinano semplicemente una velocità?

- Quali sono le relazioni che legano l'intensità della polarizzazione, la forza di protrusione e la forza di contrazione alla velocità della cellula?

Accelerazione o velocità? Rispondere alla prima domanda equivale a determinare se, nei sistemi multicellulari, prevalgano gli effetti viscosi ($\sum F = \lambda \vec{v}$) o quelli inerziali ($\sum F = m\vec{a}$). La conoscenza della risposta è, quindi, di grande importanze per la modellazione, poiché determina quale delle due grandezze (accelerazione o velocità), bisogna far variare a seconda delle condizioni.

Su questo punto la comunità scientifica si trova abbastanza in accordo: prevalgono gli effetti viscosi, poiché le masse in gioco sono talmente piccole da non generare effetti inerziali misurabili [7]. Questo significa che il movimento dovrà essere sempre specificato in termini di velocità.

Relazioni fra i vari fenomeni concorrenti nel movimento e velocità della cellula. Nonostante le lunghe ricerche, reperire delle misure chiare su come ogni fase del *crawling* cellulare influisca sulla velocità del moto è risultato impossibile. Infatti, pare che non solo questa sia variabile a seconda del tipo di cellula, ma anche a seconda dello stato attuale della cellula stessa. Questo è probabilmente dovuto al fatto che, come abbiamo detto, le tre fasi del movimento sono fenomeni indipendenti, che possono occorrere con un'intensità e una frequenza diversa a seconda di una miriade di fattori ambientali e intra-cellulari; perciò, se effettivamente influiscono sulla velocità, estrarre la relazione precisa che colleghi questi fenomeni alla velocità risultante sarebbe molto difficile, anche solo restringendosi al caso della chemiotassi.

Questo però è un problema decisamente non trascurabile per il modellista: non avere tali relazioni significa non poter associare alla cellula, dato il suo stato interno e l'ambiente che la circonda, una velocità, rendendo così impossibile un modello quantitativo preciso per il moto. Come spiegherò nella sezione successiva, questo problema è stato superato cambiando l'approccio di modellazione, abbandonando l'ambizione di realizzare un modello che definisse ogni aspetto del moto in favore di una soluzione più semplice e più flessibile.

5.2.2 Modello

In mancanza di sufficienti informazioni per definire quantitativamente ogni tipo di moto possibile, abbiamo deciso, semplicemente, di adottare un modello che lasciasse l'utente completamente libero di definire a piacimento gli aspetti del moto che gli interessa simulare. Per farlo, abbiamo deciso di modellare il movimento non come un'unica entità, ma come effetto dell'occorrenza indipendente di due fenomeni separati:

- La determinazione della direzione del moto, che sarà diversa a seconda del tipo di moto simulato (si veda sezione 5.1.2);

- Lo spostamento vero e proprio, nella direzione definita nella fase precedente. In particolare, la grandezza e la frequenza di questo spostamento dovrebbe essere definibili a piacimento dall'utente, in modo tale che possa un moto della velocità che desidera.

Crediamo che questo approccio comporti dei grandi vantaggi:

- Maggiore coerenza con la realtà biologica, dove questi eventi sono effettivamente separati e correlati solo nel momento in cui si realizza il moto.

- Possibilità di definire separatamente ogni aspetto dellla locomozione, dalle condizioni per cui deve avvenire al modo in cui deve avvenire. Infatti ci potrebbero essere delle situazioni in cui la polarizzazione avviene, ma il moto non si realizza, ad esempio perché non si sono realizzate le giunzioni, e viceversa (sezione 4.1.1).

- Maggiore semplicità nel combinare insieme i vari stimoli che tendono a spostare la cellula. Infatti se il moto fosse modellato come un'unica entità, con una sola causa e un solo effetto, la compresenza di più stimoli non sarebbe facile da gestire. La cellula si sposterebbe ogni tanto seguendo uno stimolo, ogni tanto seguendo l'altro. Se si avessero invece più cause di polarizzazione, ognuna che definisce una direzione, con unico movimento effettivo sarebbe facile combinarle per poi fare in modo che la cellula si muova tenendo conto di tutte le cause.

Tuttavia, come ogni modello, questa scelta trascura alcuni aspetti che abbiamo discusso nei capitoli precedenti. Prima di tutto, si sono trascurate tutte le deformazioni che avvengono nella cellula durante il movimento. Questa è stata questi una scelta obbligata visto che, in biochemistry, le cellule non supportano ancora le deformazioni attive, solo quelle passive (si veda sezione 5.4). Tuttavia, non abbiamo ritenuto questo aspetto di fondamentale importanza, non avendo la deformazione in se un influenza reale sul moto. Inoltre, mentre le prime due fasi del *crawling* possono essere considerate modellate con le fase descritte sopra, l'ultima fase, quella di ritrazione del retro, è stata trascurata, vista la sua minore rilevanza nel moto.

5.3 Implementazione

Nella sezione precedente ho spiegato che il movimento cellulare è stato modellato in due fasi: una che ne determina la direzione e una di effettivo spostamento. La determinazione della direzione è diversa a seconda del tipo di moto, mentre lo spostamento avviene allo stesso modo per tutti i casi. Da questo consegue che l'implementazione dev'essere svolta su due livelli: uno generale, corrispondente alla programmazione dello spostamento della cellula, e uno relativo allo stimolo che causa lo spostamento, cioè al modo in cui dallo stimolo si giunge alla determinazione della direzione di spostamento. Nelle sezioni successive, quindi, spiegherò la realizzazione della locomozione cellulare nella nostra incarnazione, partendo dal livello più alto, corrispondente all'implementazione dello spostamento, per poi scendere alla spiegazione di come viene determinata la direzione del moto. Ovviamente, quest'ultima parte sarà spiegata solo in relazione ai due tipi di moto supportati: la chemiotassi e il movimento random.

5.3.1 Aspetti generici: lo spostamento

Essendo lo spostamento della cellula un evento che cambia lo stato del sistema simulato, è stato implementato attraverso una `Action` (o meglio, di una *custom action*; si veda sezione 3.2.4), chiamata `CellMove`. Si tratta di un'azione estremamente semplice: possiede un solo costruttore, che prende in ingresso due parametri, un `double` e un `boolean`.

Il primo determina di quanto si deve muovere la cellula, mentre il secondo indica come il primo parametro dev'essere interpretato. Infatti, abbiamo lasciato la possibilità di esprimere la distanza a cui la cellula si deve muovere in due modi: o in percentuale del raggio della cellula, o come distanza assoluta. Se si programma la reazione in questo modo:

```
[] --> [CellMove(true, 0.1)]
```

La cellula si muoverà per una distanza pari al *dieci per cento del diametro cellulare*; se invece si scrive:

```
[] --> [CellMove(false, 0.1)]
```

La cellula si muoverà di 0.1 rispetto al sistema di riferimento, a prescindere dal proprio raggio. La prima scelta non può essere effettuata, ovviamente, per le cellule di diametro 0: in tal caso qualsiasi valore del booleano venga inserito si ricadrà sempre nel secondo caso.

Si noti che l'utilizzo di una custom action per modellare il solo spostamento ci permette di renderlo influenzabile a piacimento, a seconda del tipo di movimento che vogliamo modellare. Per esempio, nel caso si volesse modellare uno spostamento dato da una contrazione actino-miosinica del citoscheletro, che come abbiamo detto è un caso molto comune, si dovrà tenere in conto che tale contrazione avverrà solo in presenza di calcio nella cellula, come nel caso della contrazione muscolare. Per specificare questo aspetto, basta programmare nella cellula una reazione tipo:

```
[Ca] --> [CellMove(true, 0.1)] + [Ca]
```

dove, tra l'altro, la quantità di calcio presente nella cellula influenza la frequenza con cui questa reazione verrà eseguita. Inoltre, rende la velocità del movimento controllabile dall'utente. Nota infatti l'entità dello spostamento, per determinare la velocità con cui avviene il moto basta impostare per la reazione il rate desiderato. Se, ad esempio, fosse nell'interesse dell'utente programmare un movimento per il quale la cellula, in un istante di tempo, si sposta della metà della sua grandezza, basterebbe scrivere:

```
   time-distribution:  1
[] --> [CellMove(true, 0.5)]
```

oppure:

```
    time-distribution:  10
[] --> [CellMove(true, 0.05)]
```

A seconda che si voglia che tale movimento sia istantaneo (primo caso) o graduale (secondo caso). Infatti, la prima reazione rappresenta uno spostamento della cellula pari alla metà del suo diametro che viene eseguito *circa* una volta per istante di tempo; perciò la velocità del moto è:

$$v = \frac{spazio}{tempo} \simeq \frac{0.5 \cdot d_{cell}}{1} = 0.5 \cdot d_{cell}$$

Il "circa" è dovuto al fatto che la determinazione dell'esecuzione delle reazioni è stocastica, quindi la velocità non sarà esattamente $0.5 \cdot d_{cell}$; corrisponderà a tale valore solo *in media*. La seconda reazione, analogamente, realizza uno spostamento pari a un decimo della metà del diametro cellulare, che però viene schedulato *circa* 10 volte per istante di tempo; la velocità perciò è:

$$v = \frac{spazio}{tempo} \simeq \frac{0.05 \cdot d_{cell}}{0.1} = 0.5 \cdot d_{cell}$$

5.3.2 Aspetti relativi alla causa del moto: la polarizzazione

Tuttavia, per effettuare praticamente lo spostamento, la `CellMove` non deve conoscere solo la distanza di cui spostare la cellula, ma anche la direzione in cui questo spostamento deve avvenire. Questa viene determinata sempre attraverso delle custom actions, che calcolano la direzione del moto della cellula sotto forma di vettore. Il vettore viene poi memorizzato all'interno della cellula stesso, in un apposito campo aggiunto appositamente per supportare il movimento cellulare: `polarizationVersor`. Questo, come dice il nome, è un versore e non un vettore; questo perché non deve dare alcuna informazione di tipo quantitativo: è semplicemente la direzione in cui l'azione `CellMove` dovrà spostare la cellula. `polarizationVersor` è implementato come un campo di tipo `Position` (classe corrispondente a un vettore,

in Alchemist) all'interno di `CellNodeImpl`. Quando la classe viene istanziata questo campo ha valore (0, 0); per modificarlo, sono stati aggiunti i metodi:

- `setPolarization(Position v)`, che semplicemente sostituisce il valore di `polarizationVersor` con `v`;
- `addPolarization(Position v)`, che invece somma (in senso vettoriale) il valore di v al valore già contenuto in `polarizationVersor`. Se il risultato della somma non dovesse essere un versore, in `polarizationVersor` viene inserito un versore con la stessa direzione del vettore risultato.

Il primo viene utilizzato solo nel caso la direzione che si vuole inserire sia assoluta. Ad esempio, viene utilizzato in `CellMove` per riportare il versore a zero dopo che è avvenuto lo spostamento, in modo che la direzione in cui muoversi successivamente sia ricalcolata da capo. Il secondo metodo invece è stato aggiunto per tenere in conto di più cause di movimento in contemporanea, che agiscano congiuntamente per muovere la cellula. É il metodo che viene utilizzato, infatti, dalle azioni che implementano la polarizzazione, che sono `ChemiotacticPolarization` e `RandomPolarization`.

`ChemiotacticPolarization`

Questa azione è l'unica realmente legata alla chemiotassi e, come anticipato, si occupa di calcolare in base all'ambiente intorno della cellula l'orientazione del versore di polarizzazione da inserivi. L'azione richiede di inserire una stringa, rappresentante il nome della molecola che bisogna seguire, e un'altra stringa, che serve a indicare se la cellula dovrà polarizzarsi nella direzione in cui la concentrazione della molecola cresce (`"up"`) o nella direzione in cui la molecola decresce (`"down"`). Quindi se, ad esempio, si scrive la seguente reazione:

```
[] --> [ChemiotacticPolarization(Ca, up)]
```

si otterrà un'azione che fa polarizzare la cellula in direzione della crescita di concentrazione di calcio; si otterrà invece un effetto opposto programmando nella cellula la seguente azione:

```
[] --> [ChemiotacticPolarization(Ca, down)]
```

Per calcolare il versore di polarizzazione poi, che è il punto centrale, abbiamo utilizzato questo metodo:

1. Si raccolgono tutti gli `EnvironmentNode` che fanno parte del vicinato della cellula;

2. Per ognuno, si crea un versore che punta nella direzione di quest'ultimo rispetto alla cellula da polarizzare (ad esempio, se la cellula da polarizzare si fosse trovata in $(0, 0)$ e il vicino in $(2, 0)$, tale versore sarebbe dovuto essere $(1, 0)$);

3. Ogni versore ottenuto viene moltiplicato per la concentrazione della molecola seguita all'interno nodo ambientale che punta (nel caso di prima, se il nodo vicino avesse avuto una concentrazione 10, il vettore calcolato sarebbe diventato $(10, 0)$).

4. Si sommano tutti i vettori ottenuti e si normalizza il risultato in modo da ottenere un versore.

5. Tale versore sarà poi aggiunto al versore di polarizzazione già presente nella cellula attraverso `addPolarization`.

`RandomPolarization`

`RandomPolarization` è un'azione implementata per fornire alll'utente un meccanismo per specificare un movimento casuale della cellula. Non è infrequente, infatti, che si osservi tale fenomeno nei sistemi multicellulari. L'algoritmo con cui il versore casuale viene determinato è estremamente semplice:

1. Si scelgono casualmente due numeri tra 0 e 1;

2. Si genera il versore con la medesima direzione del vettore derivato che ha per componenti i due valori casuali;

3. Si aggiunge tale versore al versore di polarizzazione già presente nella cellula attraverso `addPolarization`.

Considerazioni sull'utilizzo

Implementare la polarizzazione come una `Action` comporta gli stessi vantaggi che avevamo visto per `CellMove`, cioè la possibilità per l'utente di condizionare questo fenomeno a seconda del sistema che vuole simulare e di definirne a piacere la frequenza attraverso il rate. Quest'ultimo punto diventa particolarmente importante nel caso della polarizzazione, perché non influisce sulla velocità del moto, bensì sull'*effetto che una determinata causa esterna ha rispetto alle altre*. Mi spiego meglio: come abbiamo già detto, sia `RandomPolarization` che `ChemiotacticPolarization` non modificano la polarizzazione della cellula attraverso `setPolarization`, ma attraverso `addPolarization`. Questo comporta che, nel caso esistano più stimoli che concorrono contemporaneamente nel polarizzare la cellula, quello che verrà considerato maggiormente sarà, generalmente, quello che presenta una maggiore frequenza. Poniamo, ad esempio, di trovarci nella necessità di modellare un particolare tipo di cellula che risponde contemporaneamente a due gradienti, uno di una molecola A e uno di una molecola B. Per quello che abbiamo detto fin'ora, la cellula dovrebbe presentare queste due due reazioni:

```
[] --> [ChemiotacticPolarization(A, up)]
[] --> [ChemiotacticPolarization(B, up)]
```

Se si desidera conferire una maggiore sensibilità rispetto a uno dei due gradienti, basterà associare alla relativa reazione un rate più alto; in tal modo il versore computato dalla `ChemiotacticPolarization` sarà sommato più volte al vettore polarizzazione, assumendo maggior peso nella somma che terminerà, infine, la direzione del moto.

L'implementazione realizzata, quindi, garantisce la massima flessibilità nella determinazione della direzione del moto. Unendo a questa la flessibilità nella specificazione della velocità della cellula, spiegata in precedenza, otteniamo un'implementazione semplice ma molto potente per simulare il moto cellulare nell'incarnation bichemistry, nonché aperta all'eventuale implementazione futura di altri tipi di moto.

5.4 L'interazione meccanica

Sebbene la determinazione della direzione e della velocità sia l'aspetto principale nella modellazione del movimento cellulare, non bisogna dimenticare che la maggior parte delle volte le cellule si spostano in un ambiente dove sono presenti altre cellule, che condizionano necessariamente il moto. Questa interazione meccanica, sebbene in passato venisse poco studiata, è tutt'altro che trascurabile; anzi la sua rilevanza è cresciuta sempre più negli studi recenti sul movimento. Infatti pare che le interazioni meccaniche non condizionino solo la dinamica del moto cellulare, come un insieme di semplici ostacoli, ma anche lo stato interno della cellula, modificandolo in maniera non trascurabile. Si può quindi dire che, per una cellula, muoversi fra le sue simili sia come per noi camminare in una piazza affollata: le persone condizioneranno sicuramente il nostro percorso, poiché nel muoverci dovremo evitarle, ma potrebbero condizionare anche il nostro stato d'animo; con le loro parole, i loro gesti o semplicemente i loro sguardi, queste interagiscono continuamente noi, condizionandoci e magari cambiando completamente la nostra giornata. Era quindi di primaria importanza includere questo meccanismo nel nostro simulatore, soprattutto perché fino ad ora questo aspetto era completamente trascurato nell'incarnation biochemistry. Quando una cellula trovava un'altra cellula sul suo cammino, semplicemente si fermava, come se si fosse trovata di fronte un ostacolo invalicabile (si veda sezione 4.2.2). Sebbene non ci sia stato il tempo di realizzare un'implementazione che tenesse in conto di tutti i complessi fenomeni che intervengono nelle interazioni meccaniche tra le cellule, siamo riusciti a realizzare un modello che le riproducesse in maniera qualitativa, permettendo alle cellule di "spingersi" e, in una certa misura, anche di deformarsi se sottoposte a uno stress da parte delle altre cellule. Si è cercato, inoltre, di lasciare l'utente sufficientemente libero di adattare il fenomeno alla realtà che deve simulare.

In questa sezione cercherò di spiegare come è stata realizzata quest'ultima componente del movimento: partirò da alcuni approcci trovati in letteratura, soffermandomi in particolar modo sul simulatore MecaGen [7], quello che ha avuto la maggiore influenza, per poi concentrarmi sull'effettiva implementazione adottata in Alchemist.

5.4.1 La modellazione della biomeccanica cellulare

L'interazione meccanica fra le cellule rappresenta uno degli aspetti più complessi del moto cellulare, poiché nel suo ambito vi sarebbero da considerare una miriade di aspetti meccanici, come:

- la rigidità della cellula, che può variare moltissimo a seconda del suo stato interno [7];
- la forza adesiva data dalle giunzioni, anch'essa variabile a seconda del tipo di giunzione [1], della sua posizione [5] e della fase vitale della cellula [1];
- lo stato di moto dei fluidi che circondano la cellula [14];
- influenza di fattori meccanici esterni [14] [7];
- proprietà meccaniche del sistema multicellulare nel suo complesso, in particolar modo se si tratta di un tessuto [7].

Questo ha portato gli studiosi ad utilizzare moltissimi approcci per modellare e studiare la biomeccanica cellulare, che possono essere divisi in due, grandi categorie: macroscopici e microscopici [7]:

> **Approcci macroscopici.** Questi non modellano la biomeccanica cellulare caratterizzando la cellula come elemento a se stante, ma focalizzandosi sulle proprietà meccaniche dell'intero sistema multicellulare. É particolarmente usato per la biomeccanica dei tessuti e, spesso, riutilizza approcci modellistici tipici della meccanica dei solidi e dei liquidi per caratterizzarne il comportamento.
>
> **Approcci microscopici.** Questi, invece, si concentrano sulla caratterizzazione della cellula come elemento a sé stante per definire sia la sua meccanica come elemento, sia la meccanica del sistema multicellulare complessivo. Questi si dividono sostanzialmente in 4 categorie, a seconda della libertà di movimento della cellula e del numero di parti utilizzate per modellarla [7]:

i) approccio a *cellular-automata*, già citato nel corso di questo elaborato, in cui le cellule sono modellate come entità uniche e fissate a una griglia, in cui possono muoversi solo da un'intersezione all'altra;

ii) modelli basati su più elementi per cellula, che vengono caratterizzati individualmente, sempre fissati su una griglia. Membri principali di questa categoria sono il *Cellular Potts Model* e i suoi derivati;

iii) modelli basati su *Delunay-Object-Dynamics*, in cui le cellule sono modellate come entità a se stante libere di muoversi nello spazio;

iv) modelli analoghi al Cellular Potts Model, ma con elementi liberi di muoversi ovunque nello spazio (tra i quali il più famoso è sicuramente il *Subcellular Elements Model*).

Vediamo, quindi, che i modelli più facilmente adattabili al nostro simulatore erano quelli microscopici, in particolare quelli appartenenti alla prima e alla terza categoria. Infatti, proprio cercando fra questi, abbiamo trovato in letteratura il modello che ci avrebbe fatto da principale esempio per la nostra implementazione; questo non solo per la sua vicinanza ad Alchemist (è, infatti, basato su *Delunay-Object-Dynamics*), ma anche per la sua estrema potenza e semplicità.

Sebbene sia un modello che tiene in conto di molti degli elementi elencati all'inizio di questa sezione, per mancanza di tempo ci siamo accontentati di adattare per Alchemist solo la parte relativa allo scambio di forze che avviene fra le cellule in seguito ad un urto, in modo che le cellule potessero "spingersi" a vicenda. In seguito, quindi, spiegherò la parte di questo modello relativa solamente a quest'ultimo aspetto; per maggiori informazioni, rimando al lavoro compiuto da J. Delile et al. nell'ambito del loro simulatore [7], che ho davvero molto apprezzato.

Il modello Mecagen

MecaGen [7] è un simulatore open-source sviluppato specificatamente per lo studio dei primi stadi dell'embriogenesi, ambito nel quale la meccanica cellulare ha un ruolo di primo piano. Infatti che il nome

MecaGen è stato scelto proprio a causa del desiderio di creare un simulatore che unisse la biomeccanica cellulare (prefisso *Meca*) con la più nota e studiata genetica della cellula (suffisso *Gen*). Sebbene sia un simulatore limitato ad un solo ambito di studio, presenta molti punti di incontro con l'incarnazione biochemistry: anche MecaGen, come già accennato, considera le cellule come singole entità; inoltre, anche in questo simulatore le cellule venivano considerate circolari (o meglio, sferiche, dato che supporta anche ambienti tridimensionali), assunzione che anche noi avevamo fatto durante l'implementazione dell'ambiente `BioRectEnvronmentNoOverlap` (si veda sezione 4.2).

Tuttavia, ciò che più di tutto mi ha colpito in questo modello e che mi ha poi convinto nella scelta di ispirarsi a esso è la sua grande semplicità. Pur tenendo conto di fenomeni estremamente complessi si basa su semplici considerazioni geometriche. In MecaGen infatti la forza

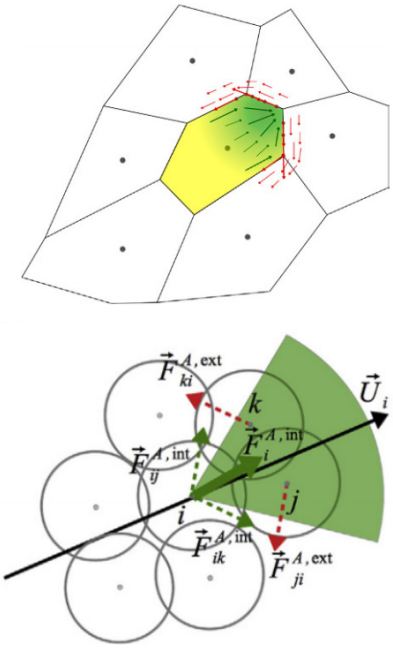

Figura 5.6: Confronto fra fenomeno simulato e modello in MecaGen. Le cellule sono rappresentate come cerchi in grado di sovrapporsi e una cellula, nel muoversi, può imporre nelle altre una forza di una certa direzione e intensità.

di repulsione meccanica fra due cellule viene semplicemente considerata *proporzionale alla sovrapposizione fra i due cerchi rappresentanti le due cellule*. Sebbene possa sembrare un'assunzione fin troppo semplice, è stata supportata da ingenti dati sperimentali [7] ed è, a mio avviso, molto fedele alla natura di corpi non perfettamente rigidi come le cellule. Cosa provoca una deformazione (e di conseguenza uno stress meccanico) se non l'invasione dello spazio fisico di una cellula da parte di un corpo estraneo?

MecaGen quindi utilizza un modello di grandissima semplicità concettuale, che si accompagna a una grande semplificazione a livello implementativo, poiché non esplicita al suo interno concetti di alto livello come la deformazione, o la tensione meccanica a cui una cellula può essere sottoposta da uno stress. Deriva tutti questi aspetti da un fenomeno molto più semplice: quello del movimento di cerchi in base alla loro sovrapposizione.

É doveroso aggiungere, però, che questa realizzazione è stata possibile in MecaGen principalmente grazie alla ristrettezza del campo di studio: trattando il simulatore di cellule non troppo dissimili fra loro, tutti i parametri del modello (come il coefficiente di proporzionalità fra area di sovrapposizione e forza generata) potevano essere fissati. Il mio compito quindi non è stato solo quello di adattare questo modello all'infrastruttura di Alchemist, ma anche di estenderlo per renderlo in grado di supportare cellule diverso tipo, il tutto programmabile in maniera sufficientemente semplice da parte dell'utente.

5.4.2 Il modello in `biochemisty`

Il modello che abbiamo formulato per la nostra incarnazione è, come abbiamo detto, estremamente simile a quello di MecaGen. La forza di repulsione fra due cellule viene considerata direttamente proporzionale all'area di sovrapposizione fra i due cerchi che le rappresentano, e varia di intensità a seconda della loro *rigidità*. Questo significa che due cellule più rigide, a parità di area di sovrapposizione, si imporranno a vicenda una forza repulsiva maggiore rispetto a quella di due cellule meno rigide. La rigidità viene considerata costante e attribuito immodificabile della cellula nella durata della simulazione; tuttavia, può essere liberamente impostata dall'utente all'inizio della simulazione.

Questo in realtà non è perfettamente coerente con la realtà biologica, visto che le cellule possono variare la loro deformabilità anche in maniera consistente a seconda del loro stato interno. Aggiungere questo aspetto al movimento, però, avrebbe richiesto una fase di modellazione a sè stante, che avrebbe richiesto troppo tempo rispetto a quello che avevamo. Si è quindi ignorato, per ora, questo connotato della rigidità cellulare, lasciandolo per lavori futuri.

Sotto la precedente ipotesi, un'assunzione che abbiamo ritenuto verosimile per modellare la deformabilità delle cellule quando sono sottoposte a una tensione meccanica, è stata di considerare quest'ultime munite di due aree circolari, invece che di una sola: una minima e una massima (si veda Figura 5.7). La prima delle due rappresen-

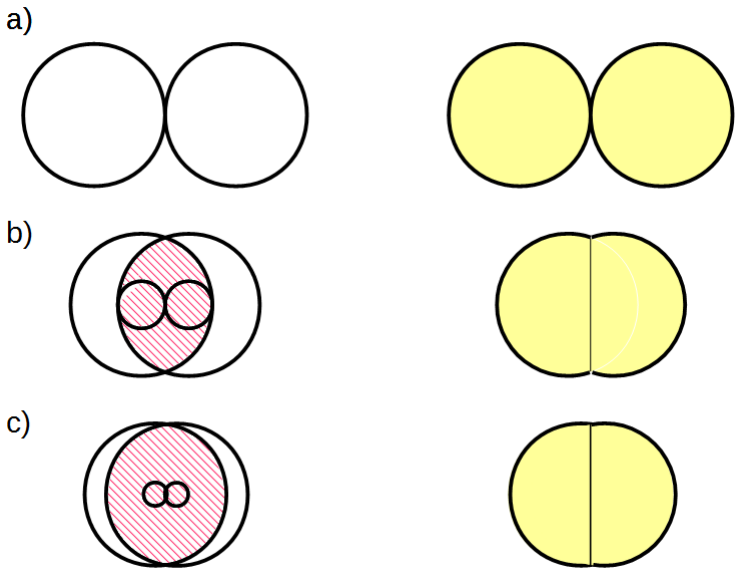

Figura 5.7: Dimostrazione visiva di come si può ottenere l'effetto di cellule diversamente rigide variando la dimensione del diametro interno rispetto a quello esterno: a) cellule perfettamente rigide, il diametro esterno è uguale a quello interno e non si ottiene quindi nessuna deformazione; b) e c) variando invece la dimensione si hanno cellule maggiormente deformabili. Si noti che in tutte queste situazioni la forza repulsiva fra le cellule è massima.

ta la dimensione minima che la cellula può assumere se sottoposta a uno stress, quindi il limite massimo alla sua deformazione; la seconda invece rappresenta la dimensione che la cellula assume quando non è sottoposta af alcuno stress. Questo vuol dire che, date due cellule con area minore di raggio, rispettivamente, r_{min1} e r_{min2}, la distanza minima a cui queste due cellule possono trovarsi è $(r_{min1} + r_{min2})$; le aree "minime" di due cellule non possono mai sovrapporsi. Invece le aree "massime" sono libere di sovrapporsi, ed è proprio la sovrapposizione di queste che viene presa in considerazione per il calcolo della forza. Se le due cellule si trovano a una distanza maggiore di $((r_{max1} + r_{max2}))$, non si imporranno nessuna forza repulsiva a vicenda; se, invece, la sovrapposizione fra queste due aree è massima (cioè le due cellule si trovano a distanza $(r_{min1} + r_{min2})$), la tensione che verrà imposta su ognuna sarà massima, anche nel caso l'area di sovrapposizione massima sia zero (caso di cellule perfettamente rigide, si veda Figura 5.7). Se il motivo di quest'ultima scelta non appare chiara, si pensi alle cellule perfettamente rigide come a delle sferette di ceramica: quando vengono in contatto durante il moto, queste si respingono come se si urtassero in maniera perfettamente elastica, senza che avvenga alcuna deformazione significativa.

5.4.3 Implementazione

L'implementazione del modello che ho descritto nella sezione precedente ha seguito due fasi distinte: una in cui le cellule sono state equipaggiate della doppia area e un'altra dove è stato realizzato il movimento dovuto alla tensione che le cellule si impongono a vicenda per sovrapposizione. Per questa seconda fase si è scelto di seguire lo stesso approccio che era stato utilizzato nella realizzazione degli altri tipi di movimento cellulare (si vedano sezioni 5.2 e 5.3), quindi di realizzare una `Action`, chiamata `CellTensionPolarization` che determinasse solo la direzione del moto, lasciando poi a `CellMove` il compito di effettuare lo spostamento effettivo. Il motivo di questa scelta è stato che, in questo modo, si manteneva la possibilità che la cellula determini la direzione in cui muoversi in base a più fattori (si veda il paragrafo "Considerazioni sull'utilizzo" nella sezione 5.3.2): se, ad esempio, viene sottoposta a tensione, ma contemporaneamente

attratta da un gradiente chimico, riteniamo più verosimile che la direzione di movimento venga determinata tenendo conto di entrambi i fattori, che non solo di uno dei due. Infine si è implementata una condizione `TensionPresent` che tiene conto della tensione complessiva a cui è sottoposta la cellula.

`CirularDeformableCell`

Per implementare la "doppia dimensione" delle cellule di cui ho parlato nella sezione precedente, ho aggiunto ad Alchemist una nuova interfaccia, `CirularDeformableCell`, che estende `CellWithCircularArea` con due nuovi metodi, cioè:

- `getMaxRadius()`;
- `getMaxDiameter()`;

Che si aggiungono ai metodi `getDiameter()` e `getRadius()` di `CellWithCircularNode`, implementando la dimensione "rilassata" della cellula. L'interfaccia viene poi implementata dalla classe `CircularDeformableCellImpl`, che presenta un costruttore per definire i due diametri dei due cerchi che caratterizzano la cellula. Questi però non sono definiti entrambi in maniera diretta; il costruttore di `CircularDeformableCellImpl` presenta, infatti, due valori di ingresso, entrambi `double`:

- il primo rappresenta il diametro *maggiore* della cellula, definendone quindi la dimensione "rilassata";
- il secondo invece rappresenta la *rigidità* della cellula, e può assumere valori da 0 a 1. Se tale valore è zero la cellula non avrà alcuna rigidità, se invece è 1 la cellula sarà perfettamente rigida, quindi indeformabile.

Il secondo diametro, quello che definisce la dimensione minima della cellula, viene calcolato a partire da i due valori dati in ingresso attraverso la semplice formula:

$$d_{min} = r \cdot d_{max} \tag{5.1}$$

dove r rappresenta sopracitata rigidità della cellula.

`CellTensionPolarization`

Analogamente a `ChemiotacticPolarization`, è l'azione che definisce la direzione in cui si deve muovere la cellula se si trova in tensione. Per determinare questa direzione si devono fare due cose:

- identificare il gruppo dei nodi che mettono in tensione la cellula, quindi quelli che si sovrappongono con essa;
- convertire la sovrapposizione in una misura della tensione che ognuno di questi nodi esercita sulla cellula.

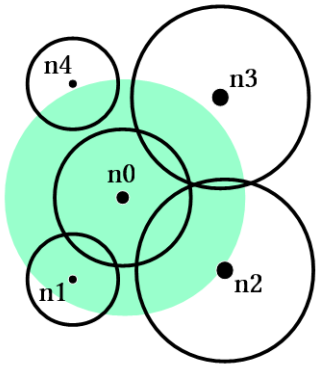

Figura 5.8: I nodi che mettono in tensione la cellula non sempre fanno parte del vicinato. Notiamo infatti come nel vicinato di n0, evidenziato in verde, non sia presente il nodo n3.

Si noti che il gruppo di nodi che mette in tensione la cellula non corrisponde necessariamente al vicinato di quest'ultima, come mostrato in Figura 5.8. Per essere sicuri di identificare correttamente tale gruppo, quindi, era necessario conoscere il massimo diametro (grande) presente fra tutte le cellule, in modo da poter controllare in un range pari a quel diametro se vi fossero nodi che mettono in tensione la cellula. Infatti, dati due nodi, se questi sono della dimensione massima, la minima distanza a cui possono presentare una sovrapposizione è pari alla somma dei loro due raggi maggiori, cioè il loro diametro maggiore; se questo è vero per due cellule della dimensione massima, sarà poi vero a maggior ragione per due cellule più piccole (si veda Figura 5.9). Per conoscere questo valore si avevano due possibilità:

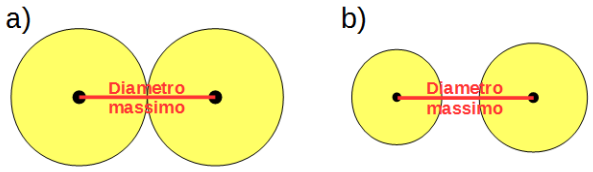

Figura 5.9: a) se le due cellule sono della massima dimensione, non possono presentare sovrapposizione a una distanza maggiore del loro diametro (in figura, *diamtero massimo*); b) a maggior ragione, questo vale se le cellule sono più piccole.

- far controllare all'azione stessa, ogni volta che veniva chiamata, fra tutti i nodi quale fosse il diametro massimo;

- memorizzare nell'ambiente quale fosse il nodo con il massimo diametro, in modo da rendere l'ambiente stesso in grado di fornire su richiesta questo valore.

Costringere una `Action` a compiere un controllo su tutte le cellule dell'ambiente ad ogni sua esecuzione era decisamente troppo oneroso in termini di performance, quindi abbiamo optato per la seconda opzione. Ho creato una nuova interfaccia, che estende `Environment`, chiamata `EnvironmentSupportingDeformableCells`, con un solo metodo in più rispetto alla sua super-classe: `getMaxDiameterAmongDeformableCell()`. L'ambiente viene reso cosciente di tale valore all'inizio della simulazione, in fase di inizializzazione: in questa fase infatti tutti i nodi vengono aggiunti all'ambiente, e ogni volta che questo succede si controlla se tale nodo sia di tipo `CircularDeformableCell`; se lo è, si controlla se fra i nodi inseriti se esiste un nodo con il diametro maggiore di questo. Se la risposta è positiva, tale nodo viene salvato all'interno di una variabile, altrimenti si va avanti con l'aggiunta dei nodi.

Per misurare la tensione alla quale una cellula viene sottoposta da un'altra, invece, ho deciso di non utilizzare propriamente il calcolo dell'area sovrapposta fra i due cerchi rappresentati le cellule, come in MecaGen, ma la distanza fra i due nodi. Noto infatti il diametro di ognuno dei due nodi, l'area di sovrapposizione è evidentemente legata univocamente alla distanza fra i due. In particolare, questa sarà:

- massima quando i due nodi si trovano a una distanza pari alla somma dei loro due raggi minimi;
- zero quando i due nodi si trovano a una distanza maggiore alla somma dei loro raggi massimi;
- non nulla per qualsiasi altra distanza.

Per scavalcare quindi la complessità del calcolo dell'area quindi ho deciso di quantificare da 0 a 1 la tensione che due cellule possono imporsi a vicenda, che viene calcolata attraverso la formula:

$$\frac{(r_{max1} + r_{max2}) - d_{12}}{(r_{max1} + r_{max2}) - (r_{min1} + r_{min2})} \tag{5.2}$$

dove d_{12} è la distanza fra le due cellule, r_{max1} e r_{max2} sono i loro raggi maggiori e r_{min1}, r_{min2} quelli minimi. Si noti che questa formula dà come risultato 1 se le cellule sono al massimo della sovrapposizione e 0 non si sovrappongono. Il caso particolare in cui entrambe le cellule siano perfettamente rigide (si veda Figura 5.7, *a*) viene gestito separatamente.

La `CellTensionPolarization`, quindi, non fa altro che prendere ogni nodo che può essere sovrapposto al nodo in cui è contenuta l'azione e associare ad ognuno di questi un vettore, con direzione opposta a quella del nodo considerato e modulo calcolato secondo la formula 5.2. Questa serie di vettori viene poi sommata, e il risultato determina quindi la direzione in cui il nodo si dovrà muovere (si veda Figura 5.10).

`TensionPresent`

Questa condizione, invece, serve a determinare se la cellula sia sottoposta a stress o meno e, in tal caso, ha una propensità correlata all'intensità dello stress. Questa infatti viene calcolata determinando, per ogni cellula che presenta una sovrapposizione non nulla con quella in cui è contenuta questa condizione, il valore di tensione relativo utilizzando la formula 5.2, e infine sommando tutti questi valori (si veda Figura 5.11).

Si noti che per un'implementazione minima di interazione meccanica sarebbe bastata anche la creazione della sola azione

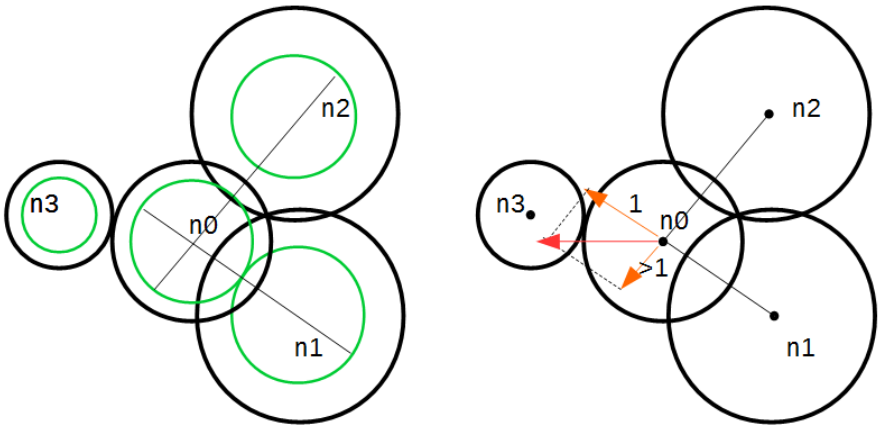

Figura 5.10: Esempio di determinazione della direzione del moto a seconda della distanza fra le cellule. Come vediamo il nodo n1 e n0 si trovano a distanza minima, quindi il vettore associato ha modulo 1. Essendo la sovrapposizione fra n0 e n2 è minore, il vettore ha modulo minore.

`CellTensionPolarization`, ma ho deciso di introdurre comunque questa condizione per due motivi:

- il primo, più legato all'interazione meccanica, era che mi sembrava più naturale che ci fosse un modo per regolare la velocità del movimento dovuto alle interazioni meccaniche in base alle intensità di queste ultime. Infatti, se nella cellula vengono programmate la reazioni:

```
[] --> [CellTensionPolarization()]
[TensionPresent()] --> [CellMove(true, 0.5)]
```

  la seconda reazione verrà schedulata con frequenza sempre crescente all'aumentare dello stress a cui si trova. Quindi, il movimento, del quale direzione viene determinata dalla prima azione, risulterà più veloce all'aumentare della tensione a cui è sottoposta la cellula.

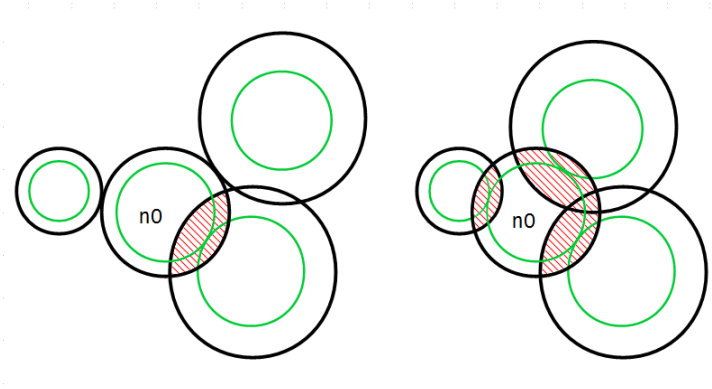

Figura 5.11: Determinazione della propensity di `TensionPresent` in due casi. In quello più sinistra la propensity è pari a 1, mentre in quello più a destra è 3.

- il secondo, invece, è legato al concetto generale di tensione cellulare. Come dicevamo all'inizio di questo capitolo, la tensione in una cellula può non portare solo al movimento, ma anche a modifiche dello stato interno della cellula. Una condizione come `TensionPresent` facilita quindi l'espressione di reazioni che non dipendono solo dalla presenza di alcune molecole, ma anche dallo stato tensionale della cellula.

Capitolo 6

Test

I test rappresentano una parte fondamentale dello sviluppo di un software, poiché ne garantiscono la correttezza e ne facilitano enormemente la modifica. Non è infrequente, infatti, che nella modifica di una parte del programma si vada involontariamente a condizionare altre sezioni di codice, causando degli errori. L'esecuzione dei test a seguito di ogni modifica permette di individuare subito tali sviste, garantendo sempre la massima funzionalità del programma. Anche Alchemist fa ingente utilizzo di test; durante tutta l'implementazione, infatti, le nuove parti implementate sono state continuamente testate. Per aiutarsi in questo lavoro si è fatto ricorso a test automatizzati realizzati attraverso il framework JUnit [12]. I I suddetti test sono tutti consultabili nella cartella `src/test` all'indirizzo ufficiale del repository GitHub di Alchemist [2].

Questi test però non sono tutto. La correttezza di un software non viene definita solo sulla base del fatto che non vi siano errori, ma anche sul suo utilizzo pratico. Per certificare la validità di questo lavoro, quindi, abbiamo riportato nelle sezioni successive alcuni esempi di simulazioni effettuate con Alchemist utilizzando le novità introdotte nel corso di questo progetto di tesi.

6.1 Test di diffusione di sostanze nell'ambiente

Come primo test per l'ambiente extracellulare, abbiamo eseguito delle simulazioni in cui siamo riusciti a riprodurre correttamente la diffusione di una sostanza chimica nell'ambiente. Per farlo, è stato simulato un ambiente privo di cellule, in cui in un punto era stata posizionata una sorgente molecolare, dove veniva prodotta in maniera continua una molecola (Figura 6.1).

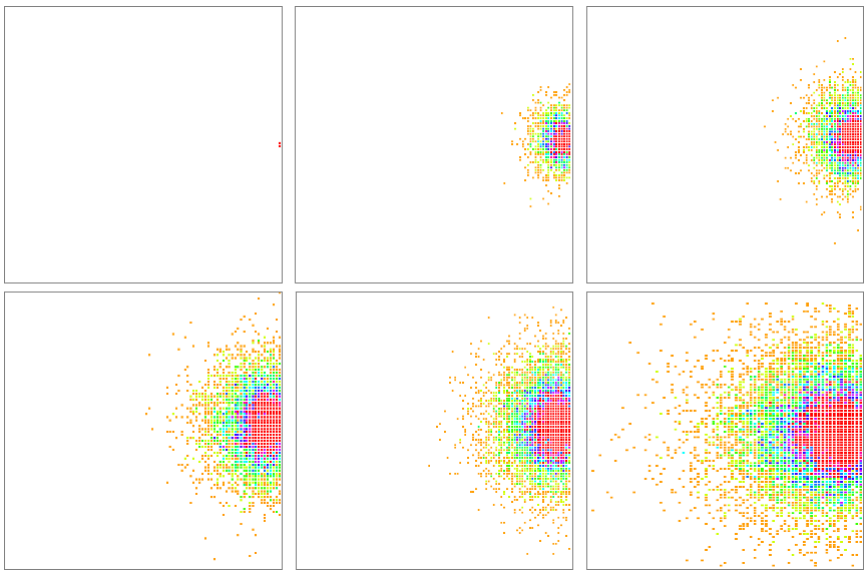

Figura 6.1: Diffusione di una molecola nell'ambiente, simulata attraverso Alchemist; il rosso rappresenta il luogo dove la molecola si trova maggiormente concentrata, l'arancione dove si trova meno concentrata. Come possiamo vedere, all'inizio della simulazione la molecola si trova solo in un punto dell'ambiente (immagine in alto a sinistra), poi diffonde velocemente andando a occupare una porzione sempre maggiore dell'ambiente.

Per ottenere questo risultato, abbiamo programmato in tutti i nodi ambientali che compongono l'ambiente il seguente set di reazioni:

```
programs:
```

```
  - time-distribution: 100
    program: >
      [S] --> [S] + [X] # generazione spontanea di A
  - time-distribution: 1
    program: >
      [A] --> [A in env] # diffusione di A
  - time-distribution: 0.001
    program: >
      [A] --> [] # degradazione di A
```

Dove la molecola S (che sta per "*source*", sorgente), che determina la formazione spontanea della molecola A, è stata posizionata solo in due di tutti i nodi ambientali, in modo da dare l'effetto di una sorgente ristretta.

Abbiamo poi verificato come, cambiando i rate delle reazioni, si possa facilmente controllare il tempo di diffusione. Abbiamo eseguito di nuovo il test precedente, in un caso lasciando il set di reazioni intatto:

```
# set di reazioni a
programs:
  - time-distribution: 100
    program: >
      [S] --> [S] + [X] # generazione spontanea di A
  - time-distribution: 1
    program: >
     [A] --> [A in env] # diffusione di A
  - time-distribution: 0.001
    program: >
     [A] --> [] # degradazione di A
```

e nel secondo riducendo i rate di un ordine di grandezza:

```
# set di reazioni b
programs:
  - time-distribution: 100
    program: >
      [S] --> [S] + [X] # generazione spontanea di A
  - time-distribution: 0.1
    program: >
```

```
    [A] --> [A in env] # diffusione di A
- time-distribution: 0.0001
  program: >
    [A] --> [] # degradazione di A
```

I risultati sono riportati in Figura 6.2.

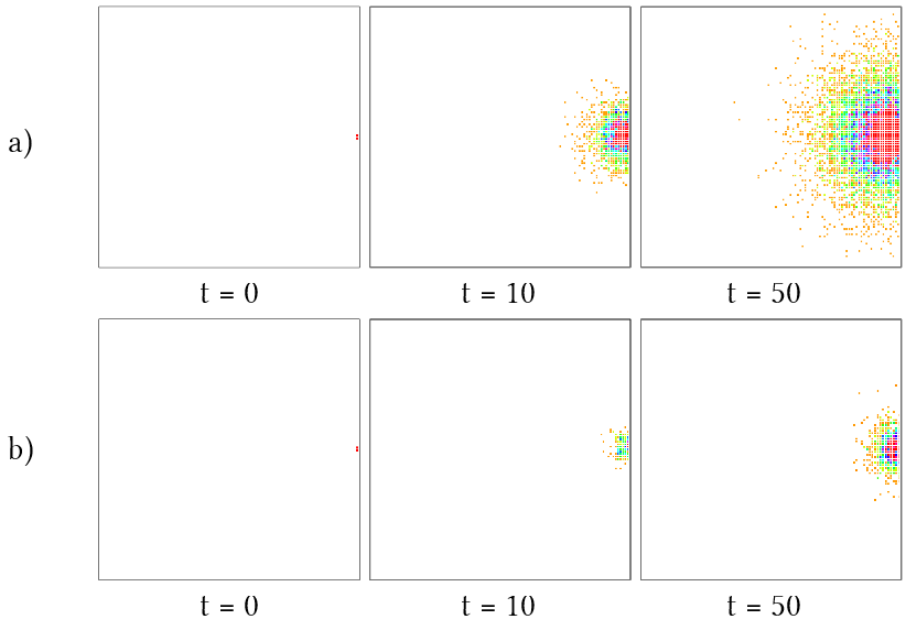

Figura 6.2: a) la simulazione della Figura 6.1 (set di reazioni a); b) la stessa simulazione, con i rate ridotti (set di reazioni b)

6.2 Test sulla chemiotassi

Come simulazione di test per la chemiotassi abbiamo deciso di riproporre una simulazione fatta attraverso la piattaforma EPISIM [22], caricata su Youtube in forma video dagli stessi sviluppatori della piattaforma. In questo modo, non solo abbiamo verificato il corretto funzionamento della chemiotassi, ma ci siamo anche confrontati con un altro simulatore. Alcuni frame del video sono riportati in Figura **??**fig:chem1). Come si può vedere, vi è un nucleo di cellule centrali

in cui viene prodotta una sostanza chemioattraente, che poi diffonde della matrice nell'ambiente circostante formando un gradiente; le cellule intorno, poi, percepiscono tale sostanza e ne vengono attratte, motivo per cui si spostano man mano vero il nucleo di cellule fino a circondarlo completamente.

Il risultato della riproduzione è mostrato in Figura 6.4. Il fenomeno viene riprodotto in maniera perfetta dal punto di vista qualitativo da Alchemist, che si dimostra quindi paragonabile a EPISIM in questo senso (si veda anche confronto in Figura 6.5). Purtroppo però, non essendo a conoscenza del modello adottato dagli sviluppatori di EPISIM per simulare il fenomeno, non siamo sicuri che la resa quantitativa (cioè in termini di tempo e velocità del fenomeno) corrisponda perfettamente a quella di questo simulatore.

6.3 Test sull'interazione meccanica

Per la biomeccanica cellulare invece abbiamo realizzato due simulazioni: la prima è stata costruita ad hoc per verificare che la resa dell'interazione meccanica fosse corretta (quindi non ispirata a nessun fenomeno biologico); la seconda invece è ispirata a un fenomeno realmente osservato.

Prima simulazione. Nella prima simulazione, della quale alcuni frame sono mostrati in Figura 6.6, possiamo osservare due gruppi di cellule: uno giallo e uno grigio. Il primo è stato programmato per rispondere a un gradiente che viene generato all'estrema destra dell'ambiente, mentre l'alto no. Il gruppo di cellule "inerti" è stato disposto per formare una barriera fra la sorgente del chemioattraente e il primo gruppo "non inerte"; in questo modo, questo secondo gruppo di cellule avrebbe dovuto "spingere via" le altre per poter raggiungere la sorgente del chemioattraente, dandoci modo di controllare se l'interazione meccanica venisse resa in maniera corretta dal simulatore.

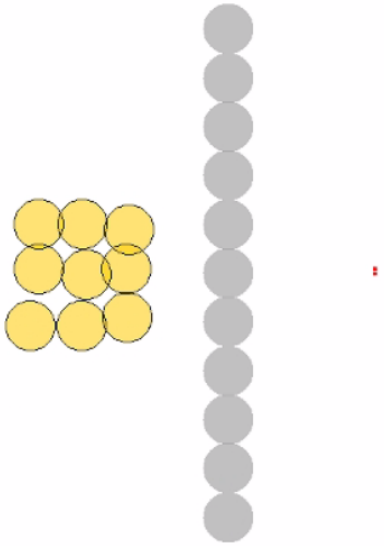

Figura 6.6: La simulazione di test n. 1 prima di essere avviata

Il risultato si è dimostrato da subito positivo. Come vediamo in Figura 6.7, infatti, le cellule gialle riescono a rompere la disposizione delle cellule grigie, raggiungendo la sorgente del gradiente. Tra l'altro, questo test ci ha dato modo di notare quanto l'implementazione dell'interazione meccanica tra le cellule abbia aumentato la verosimiglianza delle simulazioni. Prima, infatti, le cellule del primo gruppo si sarebbero immediatamente fermate una volta incontrato il muro.

Seconda simulazione. Per la seconda simulazione invece ci siamo ispirati a un fenomeno legato alla risposta immunitaria, che è stato simulato attraverso la piattaforma EPISIM [22], ma anche direttamente filmato da Roger Davis nel 1950. Il fenomeno avviene in corrispondenza dell'ingresso di un batterio nell'ambiente dove si trova un globulo bianco. Quest'ultimo si accorge della presenza nell'ambiente del batterio e si muove rapidamente verso di esso per fagocitarlo; il batterio, in risposta, si accorge del macrofago e fugge da esso, dando luogo a un bizzarro inseguimento fra neutrofilo e batterio (si veda Figura 6.8).

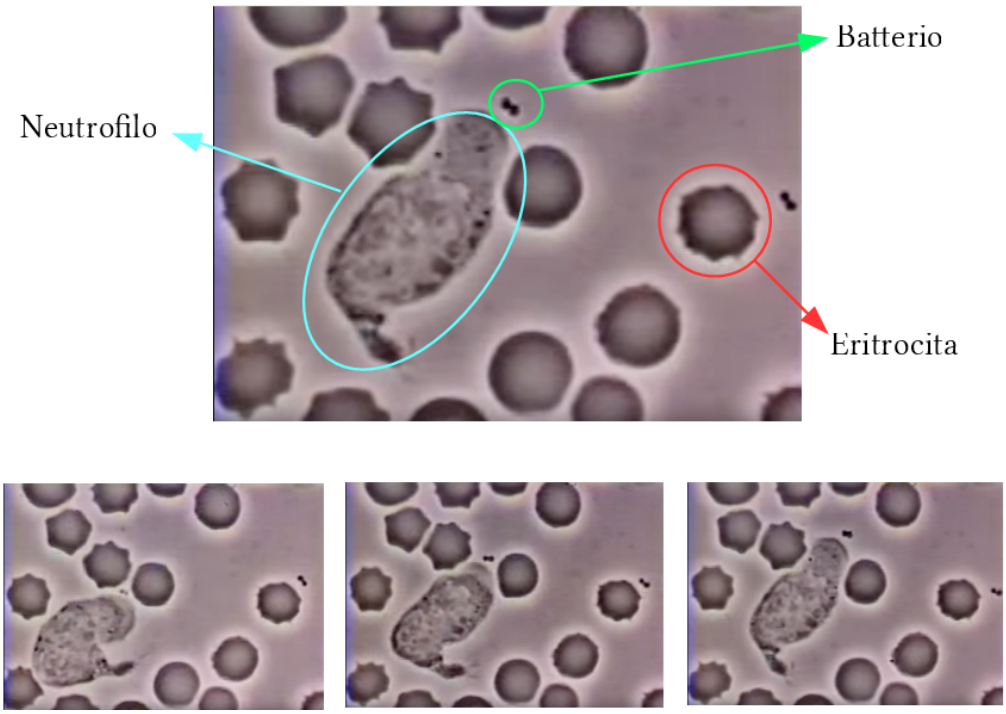

Figura 6.8: Alcune scene del video registrato da Roger Davis, dove possiamo vedere il globulo bianco che si fa strada fra due globuli rossi per raggiungere il batterio, che in risposta fugge.

Sebbene non siano del tutto chiarite le ragioni del fenomeno osservato, gli sviluppatori della piattaforma EPISIM hanno scelto di modellarlo come il risultato di due chemiotassi congiunte:

- La prima che vede il macrofago attratto da una sostanza che viene diffusa nell'ambiente dal batterio;
- La seconda che vede, viceversa, il batterio allontanato da una sostanza chemiorepellente, che viene diffusa nell'ambiente dal macrofago.

Il risultato ottenuto con EPISIM è mostrato in Figura 6.9 attraverso alcuni frame prelevati dal video che gli sviluppatori hanno caricato su Youtube.

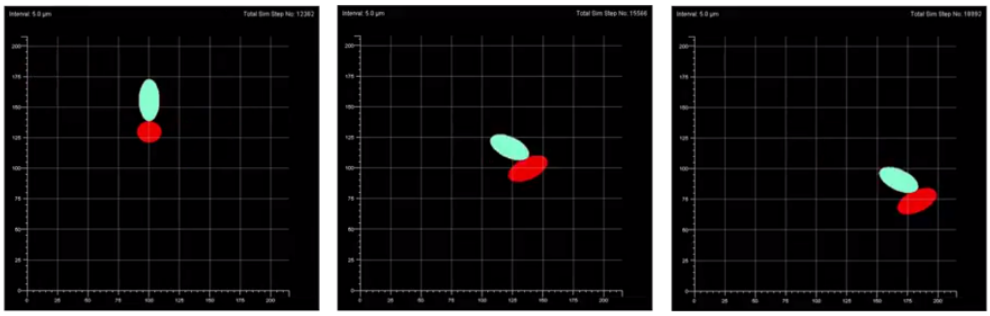

Figura 6.9: Il fenomeno simulato attraverso EPISIM. Nella simulazione è stata trascurata la presenza degli eritrociti.

Abbiamo scelto di modellare anche noi il fenomeno allo stesso modo, in modo da poterci confrontare con EPISIM, aggiungendo però anche la presenza dei globuli rossi, per verificare che il neutrofilo interagisse meccanicamente con questi similmente al quanto registrato da Roger Davis.

Il risultato è mostrato in Figura 6.10. La simulazione riproduce in maniera abbastanza verosimile l'evento spiegato: il globulo bianco segue il batterio, anche spostando i globuli rossi che incontra sulla sua strada; il batterio, in risposta, si allontana dal macrofago.

Figura 6.10: Il fenomeno simulato attraverso Alchemist. In blu è riportato il neutrofilo, in verde il batterio e in viola i globuli rossi.

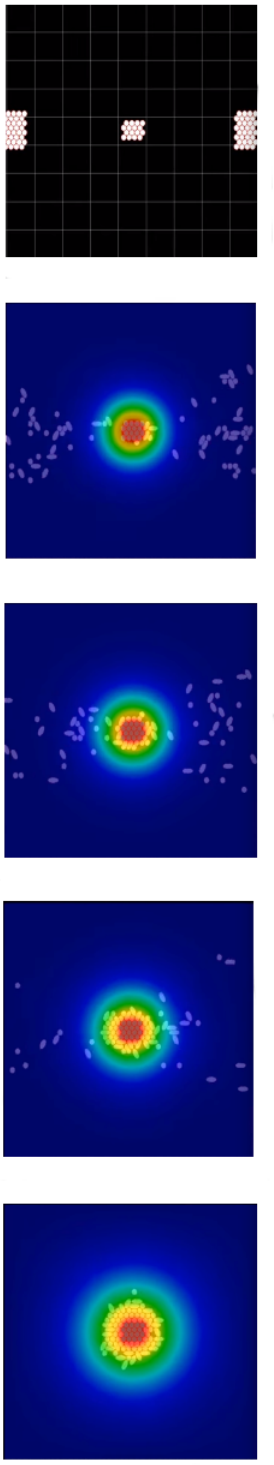

Figura 6.3: Alcuni frame del video caricato su Youtube dagli sviluppatori di EPISIM [22], che mostrano lo sviluppo del fenomeno.

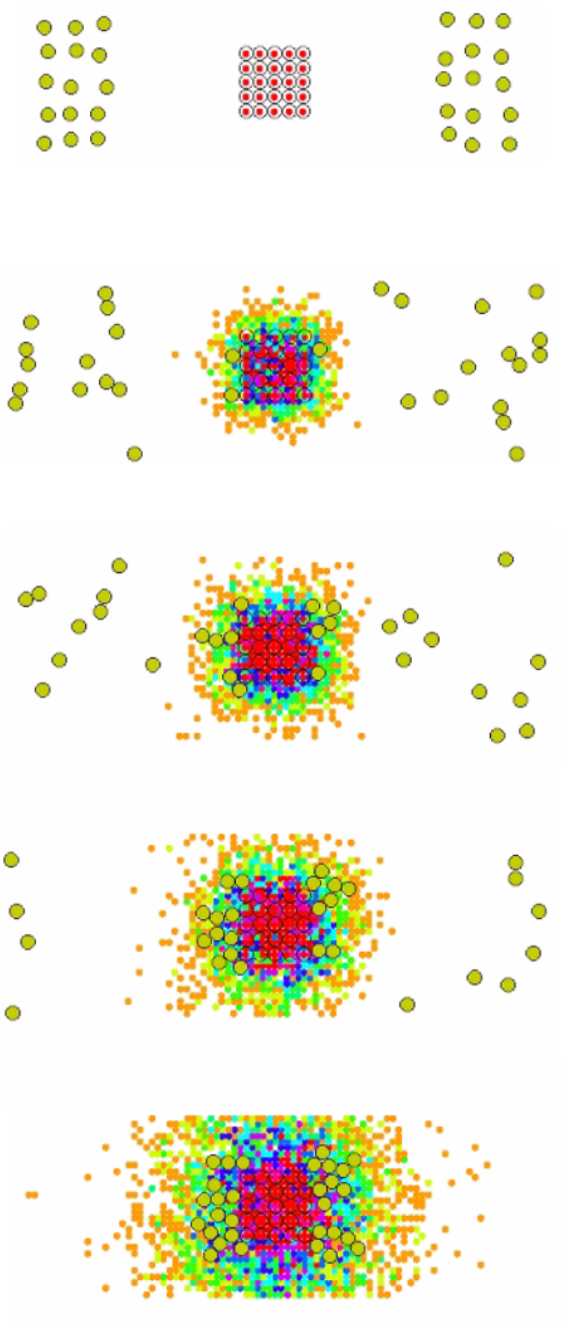

Figura 6.4: Lo stesso fenomeno simulato mostrato in Figura 6.3, simulato su Alchemist.

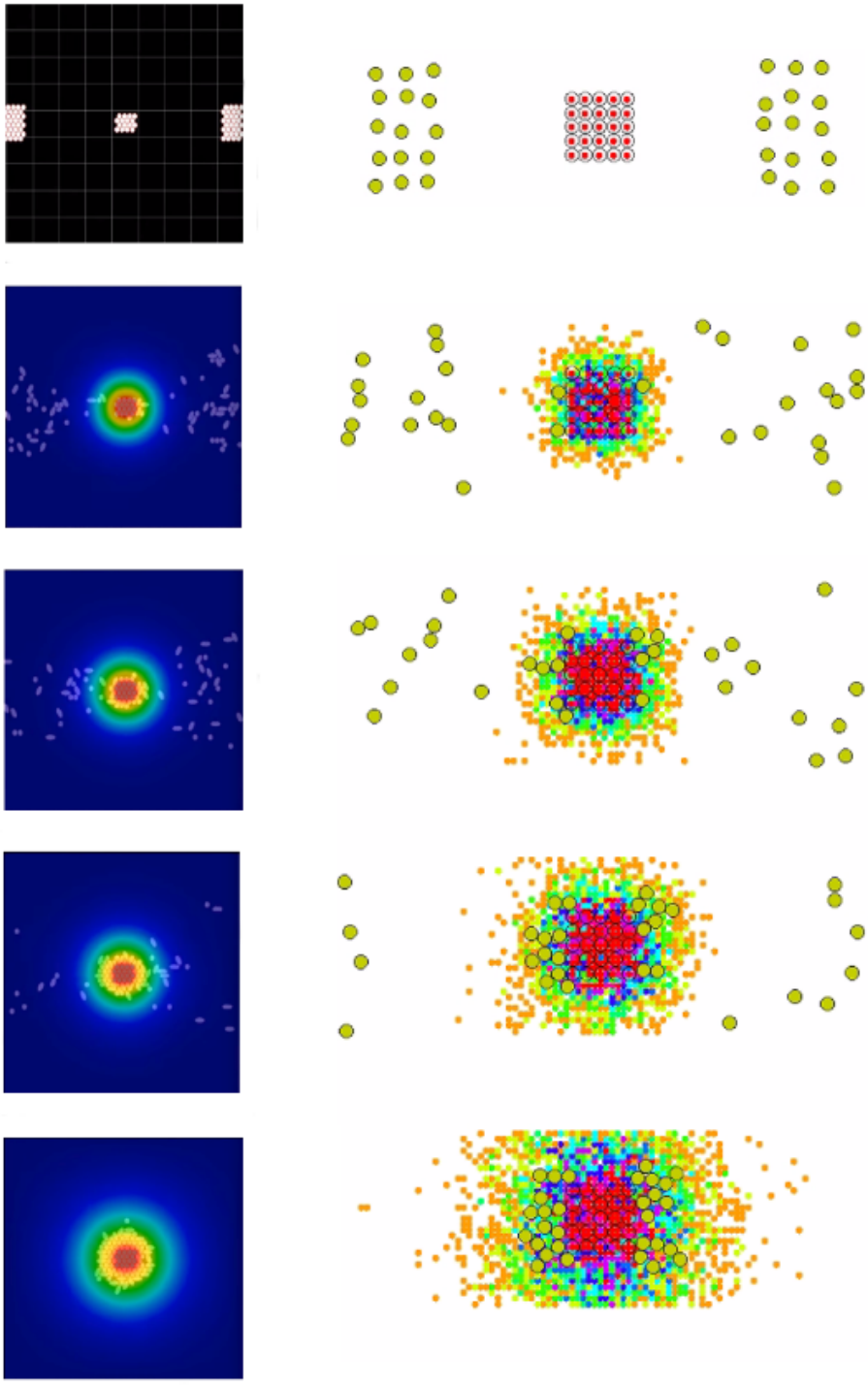

Figura 6.5: Confronto fra Figura 6.3 e Figura 6.4. Come possiamo vedere, la resa qualitativa del fenomeno è identica in entrambi i simulatori.

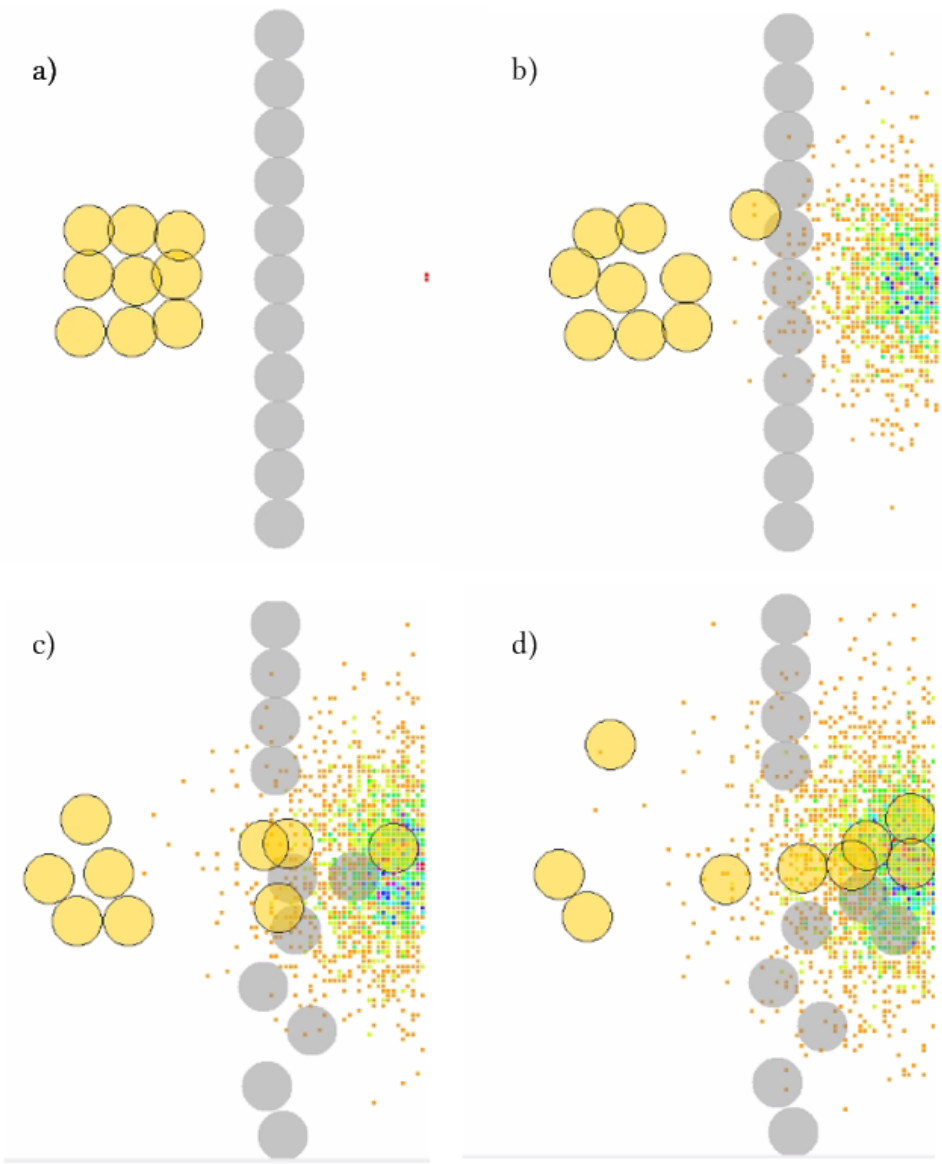

Figura 6.7: Lo sviluppo della simulazione di test n. 1; vediamo che il gruppo di cellule gialle, spinte dalll'attrazione verso la sorgente della sostanza diffusibile, riscono a rompere il muro formato dalle cellule inerti.

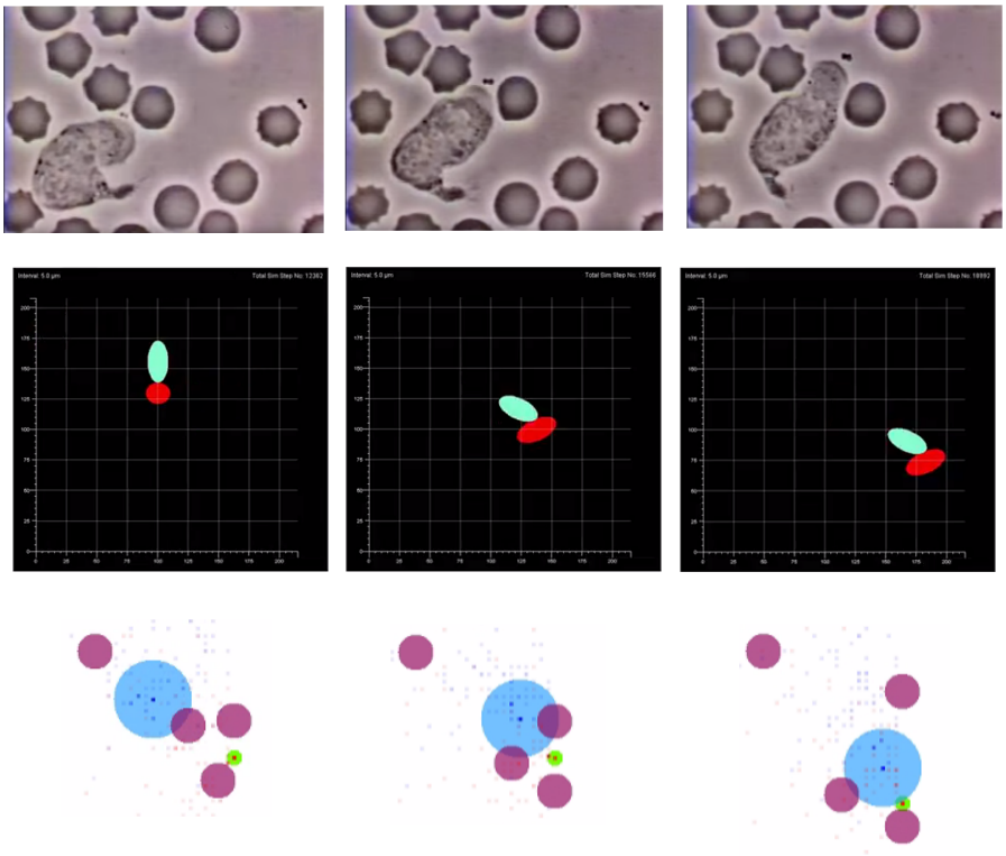

Figura 6.11: Le emozionanti scene dell'inseguimento confrontate insieme. Nella riga 1 è mostrato il video del fenomeno reale; nella riga 2 la simulazione in EPISIM; nella riga 3 la simulazione in Alchemist.

Conclusioni

In conclusione, possiamo dirci molto soddisfatti dei risultati raggiunti da questo progetto, che sono stati pienamente all'altezza delle aspettative. Alchemist è stato esteso in modo da supportare la modellazione dell'ambiente extracellulare e del movimento cellulare.

> L'ambiente è stato realizzato affinché possa interagire con le cellule, permettendo la diffusione di sostanze chimiche da queste ultime all'esterno e viceversa, ma non solo. Infatti, può anche essere caratterizzato a seconda delle necessità dell'utente; è possibile, ad esempio, creare sorgenti molecolari in qualsiasi punto dello spazio per simulare gradienti, caratterizzare diverse zone dello spazio con differenti popolazioni di atomi, e molto altro.
>
> Il movimento cellulare è stato modellato e implementato per realizzare la chemiotassi, il movimento random e l'interazione meccanica, ottenendo buoni risultati nelle simulazioni. Inoltre, è stato realizzato in maniera tale da facilitare la sua estensione anche ad altri tipi di movimento cellulare, dei quali la progettazione è lasciata a lavori futuri.

Tuttavia, i lavori sull'incarnazione biochimica di Alchemist si possono dire tutt'altro che completati. Infatti vi sono fenomeni biologici di estrema rilevanza che non possono essere ancora modellati con il nostro simulatore. In particolare, riteniamo che le estensioni su cui si dovranno concentrare maggiormente i lavori futuri saranno quelle per supportare la riproduzione cellulare e l'apoptosi, che non sono ancora simulabili in Alchemist. Un'altro aspetto di particolare rilevanza, sempre dal punto di vista biologico, è il volume cellulare, che non è stato ancora definito in maniera precisa: allo stato attuale le cellule

occupano uno spazio, che gli permette di interagire e di non sovrapporsi completamente fra loro, ma si dovrà presto definire una modello, che permetta di simulare le variazioni di volume e di deformabilità che possono avvenire nella cellula come conseguenza dello scorrere del tempo e di fattori esterni. Vi sono poi altri aspetti, più legati alla implementazione, che andranno migliorati prossimamente: innanzitutto, saranno da risolvere dei problemi legati alle performance, osservati, ad esempio, all'avvio della simulazione; dovrà essere rinnovata l'interfaccia grafica, ora non particolarmente adatta a rappresentare ambienti multicellulari in maniera dettagliata. Infine, si ritiene importante l'aggiunta di un'estensione che permetta di caricare modelli scritti con lo standard SBML[21] (Systems Biology Markup Language), per rendere il sistema più fruibile ai modellisti. Questo standard, infatti, è molto utilizzato nella Computational Systems Biology, poiché definisce un linguaggio unico per tutti i simulatori che lo supportano, facilitando così la specifica dei sistemi da simulare da parte degli utenti.

Lasciamo tutte queste cose, che sono state riassunte brevemente in quest'ultimo capitolo, nelle mani dei futuri sviluppatori di Alchemist. Anche io spero di poter contribuire ancora a questo innovativo e stimolante progetto, nella speranza che possa, un giorno, essere in grado di supportare in maniera concreta la ricerca dei sistemi biologici.

Ringraziamenti

Sebbene, in generale, abbia visto come il sistema della laurea triennale e magistrale faccia un po' passare il "gusto" di laurearsi (poiché non rappresenta un vero termine del corso di studi; non nel mio caso, almeno) non posso fare altro che constatare di aver raggiunto un traguardo importante, terminando questa tesi. Se non posso gioire di essermi laureato, posso gioire di questa meravigliosa esperienza che ho intrapreso, lavorando a questo progetto; se non posso gioire di aver partecipato a un progetto appassionante, posso gioire di aver scelto di studiare questa materia, e di aver intrapreso questo bellissimo percorso durato tre anni; se non posso gioire nemmeno di questo, posso semplicemente godermi il momento che, al di là di tanti discorsi che si possono fare, e tante parole che possono essere scritte, rappresenta qualcosa di importante per me.

Però, come per ogni cosa che ho avuto nella, per questo devo ringraziare davvero molte persone che mi hanno permesso di raggiungere questo traguardo:

> Innanzitutto, devo ringraziare il mio relatore, il Prof. Mirko Viroli, per avermi permesso di partecipare a questo progetto e anche per la fiducia che ha avuto in me;
>
> Poi la mia correlatrice, la Dott.ssa Sara Montagna, per avermi condotto, supportato e aiutato per tutta la durata di questo percorso, durato quasi sei mesi, rivelandosi la migliore correlatrice che si potesse avere;
>
> Infine, il Dott. Danilo Pianini, per aver sopportato le mie lacune informatiche e avermi aiutato a colmarle.

Poi, è dovuto un ringraziamento a tutte quelle persone che non mi hanno sostenuto solo per questa tesi, ma ogni giorno, ogni ora, ogni minuto, mi danno il loro sostegno, spronandomi a dare sempre il meglio di me:

> AI miei genitori e alla mia esplosiva sorella.
>
> Alla mia famiglia: i miei nonni, i miei zii e i miei cugini.
>
> A tutti gli incredibili e pazzi fratelli che ho adottato in questi anni: Ciaramitra, Coach Biagio (conosciuto da Nenne come "Rubensh"), Gangaman (detto anche "Allora, ganj?"), Garidos Rosso Alapa (si, ALAPA) Shiny, Gavin (detto anche "Babyano"), Gianni Caccia (detto anche Cocchi, ma nessuno lo chiama così...), Juli, Kovacicc, La Chiave, Lawrence D'Arabia, Mega-Proteo (per chi non lo conoscesse, è il fratello gemello del David), Scaglio, Sira Giulia Gatta Fox;
>
> A tutte le meravigliose e dolcissime sorelle adottive (per cui ho solo normali, per fortuna): Agghy, Babi, Carla, Fede, Gio, Glori, Kety, Ludo, Marti.
>
> E infine, alla mia fidanzata, che mi sopporta e supporta ogni giorno, che mi ascolta ripetere fino a tardi, che perdona le mie dimenticanza e mi ama, nonostante tutto. Grazie, Nini.

Nel corso dei miei studi per questo elaborato mi sono imbattuto in una meravigliosa metafora sulla morfogenesi, che associa l'embrione ad *"una tela che si dipinge da sola"* [4], dove i colori sono rappresentati dalle cellule che si caratterizzano nello sviluppo. Leggendola la prima volta mi sorse spontaneamente questa domanda: qual'è il momento preciso in cui smettiamo di essere una "tela"? Sebbene la metafora si riferisse allo sviluppo embrionale, noi siamo per tutta la vita un po' "embrioni", essendo potenza di ciò che saremo. Anche una volta nati il nostro sviluppo continua, prima dal punto di vista fisico e poi sempre più dal punto di vista emotivo. Ma allora cos'è che ci definisce, quali sono i colori e tratti che caratterizzano la nostra tela? Ognuno può avere un opinione diversa, ma io ritengo che siano le persone che amiamo ciò che ci definisce e ci modella più di ogni

altra cosa; le parole, i sentimenti, le opinioni, il carattere e in generale ciò che siamo, sono tutte conseguenze di chi abbiamo incontrato. Se quindi posso permettermi di estendere la metafora di Enrico Coen, io credo che siano le persone che amiamo che, sostituendosi alle semplici cellule, finiscano per rappresentare i colori della nostra vita, definendo le tonalità di un quadro cangiante che mostriamo al mondo e a noi stessi.

Bibliografia

[1] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Biologia molecolare della cellula*. Zanichelli, 2013.

[2] `https://github.com/AlchemistSimulator/Alchemist`. Alchemist, official GitHub repository, offical website; last visited on September 2016.

[3] S. Bornholdt. Boolean network models of cellular regulation: prospects and limitations. *Journal of The Royal Society Interface*, 5(Suppl 1):S85–S94, 2008.

[4] E. Coen. *The Art of Genes: How Organisms Make Themselves*. Oxford Univerity Press, 2000.

[5] J. A. Davies. *Mechanisms of morphogenesis*. Elsevier, 2005.

[6] W. De Back. Multicellular systems biology. `http://walter.deback.net/multicellular-systems-biology/`. Last visited on September, 2016.

[7] J. Delile, R. Doursat, and N. Peyriéras. *Computational Modeling and Simulation of Animal Early Embryogenesis with the MecaGen Platform*, chapter 16. Elsevier, 2013.

[8] D. T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22(4):403 – 434, 1976.

[9] Giunzione cellulare. `https://it.wikipedia.org/wiki/Giunzione_cellulare`. Wikipidia, last visited on September 2016.

[10] G. Graffieti. Progettazione e implementazione di una incarnazione biochimica per il simulatore alchemist, 2016.

[11] K. Haase and A. E. Pelling. Investigating cell mechanics with atomic force microscopy. *Journal of The Royal Society Interface*, 12(104), 2015.

[12] `http://junit.org/junit4/`. JUnit, offical website; last visited on September 2016.

[13] H. Kitano. Computational systems biology. *nature*, 2002.

[14] S. Li, J.-L. Guan, and S. Chien. Biochemistry and biomechanics of cell motility. *Annual Review of Biomedical Engineering*, 7(1):105–150, 2005. PMID: 16004568.

[15] G. McCluskey. Using java reflection. `http://www.oracle.com/technetwork/articles/java/javareflection-1536171.html`. Last visited on September 2016.

[16] S. Montagna. Multi-level models and infrastructures for simulating biological system development, 2011.

[17] S. Montagna and A. Omicini. Simulation and multi-agent systems. an introduction. `http://campus.unibo.it/203921/1/8-simulation.pdf`. Slide del corso di Sistemi Autonomi.

[18] M. J. North, N. T. Collier, J. Ozik, E. R. Tatara, C. M. Macal, M. Bragen, and P. Sydelko. Complex adaptive systems modeling with repast simphony. *Complex Adaptive Systems Modeling*, 1(1):1–26, 2013.

[19] D. Pianini. `https://alchemistsimulator.github.io/`. Alchemist, offical website; last visited on September 2016.

[20] D. Pianini. Stochastic simulation in alchemist. `https://core.ac.uk/download/pdf/31079323.pdf`. Slide del corso ISAC.

[21] `http://sbml.org/Main_Page`. SBML, offical website; last visited on September 2016.

[22] T. Sütterlin, C. Kolb, H. Dickhaus, D. Jäger, and N. Grabe. Bridging the scales: semantic integration of quantitative sbml in graphical multi-cellular models and simulations with episim and copasi. *Bioinformatics*, 29(2):223–229, 2013.

[23] B. Veleirinho and al. *Protein Kinase A and Protein Kinase C Connections: What Could Angiogenesis Tell Us?*, chapter 7. InTech, 2015.

[24] U. Wilensky and CCL. `http://ccl.northwestern.edu/netlogo/index.shtml`. NetLogo home page. Last visited on September, 2016.