Alma Mater Studiorum
Università di Bologna

II Facoltà di Ingegneria

Corso di Ingegneria Informatica

Laurea Magistrale in Sistemi Multi Agente

Molecules of Knowledge: architettura, implementazione ed esempi

Candidato
Mattia Occhiuto

Relatore
Prof. Andrea Omicini
Correlatore:
Ing. Stefano Mariani

Anno Accademico 2011/2012 - Sessione I

«Non cercate solo di superare i vostri
contemporanei o i vostri predecessori.
Cercate, piuttosto, di superare voi stessi.»
William Faulkner

Contents

Introduction

Il contesto sociale che stiamo vivendo è caratterizzato da una radicale evoluzione dei mezzi e delle modalità di interazione e socializzazione. Il ruolo sempre più centrale delle reti sociali (social network) sta portando a nuovi meccanismi di interazione e di socializzazione che pongono centralmente l'individuo, i suoi interessi, i suoi pensieri etc.

Compito delle scienze informatiche è quello di capire tali cambiamenti, adattarsi ad essi e generare modelli concettuali che ne permettano una gestione sempre più knowledge-oriented ed aderente al contesto ed alle nuove esigenze tecnologiche emergenti.

Tutto questo porta alla generazione di ambienti knowledge-intensive nei quali sfida centrale diventa la gestione di questa mole immensa di dati.

Oltre a questo è doveroso considerare la pervasività che questi sistemi sociali stanno avendo nella vita quotidiana delle persone; grazie a dispositivi tecnologici sempre più distribuiti e di: facile reperibilità, prezzi sempre più accessibili e utilizzo sempre più user friendly. Considerando la diffusione che stanno avendo sistemi come: smartphone, smartTV, PC, tablet, etc.; è facile intuire come tale contesto informatico-sociale sia sempre più legato ad ognuno di noi e come non sia possibile eseguire valutazioni e studi che esulino dal tenerli in considerazione.

La gestione di ambienti fortemente ricchi di informazioni stà diventando sempre di più un obiettivo sfidante per i sistemi software, si ha sempre di più la necessità di confrontarci con moli di dati in continuo aumento: provenienti da fonti differenti, con diversa struttura e con semantica non uniforme. Solo sistemi software autonomi possono essere in grado di affrontare un obiettivo sfidante come questo.

Sistemi adattativi e auto-organizzanti sembrano il solo approccio possibile

per riuscire a gestire problemi di tale entità: l'impredicibilità di tali sistemi, l'impossibilità di gestirli attraverso un controllo globale e l'assenza di soluzioni deterministiche valide, rappresentano le principali motivazioni che motivano la scelta di un modello alternativo.
Ambito di questa tesi è lo studio di un modello di coordinazione knowledge-intensive nature-inspired chiamato MoK, tale modello fornirà la base concettuale dalla quale partire per valutarlo come alternativa possibile ad alcuni classici paradigmi/modelli esistenti.
Modelli nature-inspired forniscono un chiaro esempio di come sia possibile ottenere ordine dal caos attraverso l'utilizzo di semplici meccanismi pervasivi basati su interazioni locali. Inoltre, modelli di coordinazione, come i centri di tuple, hanno già mostrato le loro capacità nell'ingegnerizzazione di sistemi software complessi knowledge-intensive, pervasivi e auto-organizzanti.
L'obiettivo di questa tesi è quello di studiare e valutare il modello nature-inspired knowledge-intensive MoK, valutare una possibile architettura implementativa applicabile cosiderando le infrastrutture a disposizione, analizzare le infrastrutture scelte, nel caso sia necessario estenderle, ed infinie proporne una prima implementazione che renda possibile, nella fase finale dell'elaborato, l'esecuzione di alcuni test pratici dando la possibilità di valutare tale modello rispetto a quelli esistenti.
Il lavoro sarà così suddiviso: nel capitolo uno verranno intodotte le caratteristiche di modelli nature-inspired ed in particolare si descriverà il modello MoK, nel secondo capitolo verranno analizzate le caratteristiche necesserie alla modellazione di sistemi biologici e ci si soffermerà in particolare su oaradigmi tuple-based del quale si fornirà un priomo mapping tra modello MoK, nel terzo capitolo verrà motivata la scelta di un infrastruttura tuple-based per implementare un sistema biologico come MoK, se ne considererereranno le caratteristiche ed i gap presenti rispetto al modello fornendone quindi un estensione, nel quarto capitolo si valuterà la simulazione di un ambiente chimico attraverso alcuni algoritmi noti e se ne fornirà un'implementazione funzionante e si completerà l'implementazione dell'architettura di MoK, nel quinto capitolo si valuterà un caso di studio ed infine si trarranno le conclusioni appropriate rispetto a quanto valutato e studiato e si proporranno possibili estensioni e migliorie al lavoro fatto.

Chapter 1

Modelli di coordinazione Nature-inspired

Come anticipato nell'introduzione di questo lavoro, il contesto sociale in cui viviamo è caratterizzato da un aumento esponenziale di device in grado di interagire gli uni con gli altri al fine di scambiare e condividere contenuti informativi sempre in continuo aumento. Tali device sono sia consumatori che produttori di informazioni - agenti proattivi -, la necessità portata da tale contesto prevede un'infrastruttura innovativa nella quale sia possibile far convergere ed integrare tale moltitudine di agenti (anche di natura diversa) al fine di riuscire ad orchestrarne l'interazione in un sistema che non sia vincolato alla tipologia di agente, alla dimensione del sistema o alla sua evoluzione costante.

Attualmente problematiche di questo tipo vengono affrontate estendendo le soluzioni esistenti con nuove funzionalità ma lasciandone invariato il paradigma alla base; quello che vogliamo proporre invece è un nuovo modello dedicato a risolvere questo set di problematiche tipiche di questo tipo di sistemi. La domanda a cui si cerca di rispondere può essere esposta come: *è possibile concepire un nuovo modello radicalmente diverso per integrare servizi network e il loro ambiente di esecuzione, in modo tale che problemi distinti come abilitare la pervasività, context-awareness, dependability, openess, diversity, robustezza e flessibilità all'evoluzione, possono essere uniformate una volta per tutte?*.

Proveremo a vagliare la possibilità di utilizzare sistemi nature-inspired po-

tendo forse favorire l'integrazione e la risoluzione delle problematiche che ci troviamo ad affrontare.

1 Metafore naturali

Le differenze chiave che riguardano i diversi approcci che possono essere presi in considerazione per la realizzazione di sistemi ispirati ad eco-law sta nella metafora utilizzata per modellare l'ecosystem, i sui individui e le sue leggi. Le metafore principali che possono essere prese in considerazione sono: metafora fisica, metafora biologica, metafora chimica e metafora ecologica.
Espliciteremo ora le caratteristiche che distinguono ogni metafora dalle altre al fine di percepirne le caratteristiche peculiari.

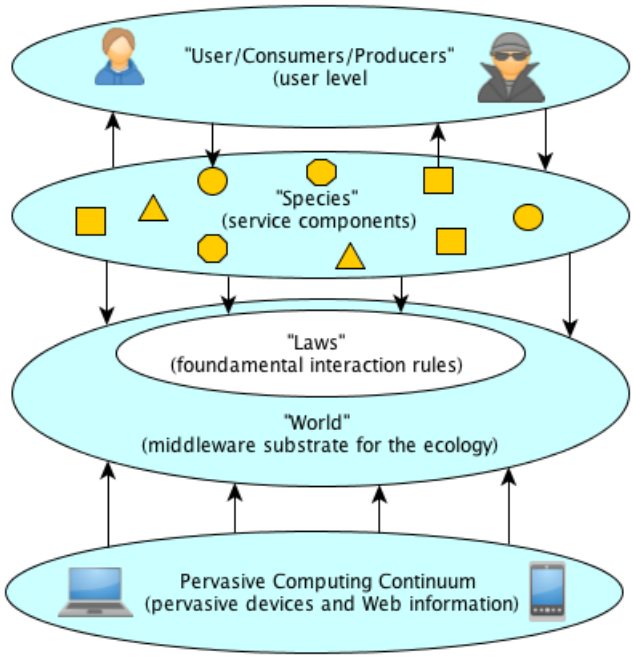
La **Metafora Fisica** considera che le specie dell'ecosistema sono una sorta di particelle computazionali, queste vivono in un mondo popolato da altre particelle, esistono poi campi virtuali computazionali i quali agiscono come i fondamentali media di interazione. Tutte le attività delle particelle sono guidate da leggi che determinano come le particelle possono essere influenzate dal gradiente locale e dalla forma del campo computazionale: essi possono cambiare il loro stato basandosi sulla percezione del campo, e possono muoversi o cambiare i dati navigando attraverso tale campo. Il mondo nel quale tali particelle vivono e nel quale il campo distribuisce e diffonde può essere molto semplice (euclide), o può essere imlementato come una sorta di mondo reale nel quale distribuzione e distanza nell'ambiente non sono inerenti agli individui ma sono gestiti dal campo (come ad esemio un campo gravitazionale spazio-tempo).

La **Metafora Chimica** considera che le specie dell'ecosistema siano una sorta di atomi/molecole computazionali, le cui proprietà vengono descritte da una descrizione formale che rappresenta la controparte della descrizione del comportamento e delle proprietà che legano gli atomi/molecole fisiche. Infatti, le leggi che guidano il comportamento generale del sistema sono una sorta di leggi chimiche, esse indicano come le reazioni chimiche correlano i componenti del sistema, queste possono portare alla creazione di nuovi ag-

gregati o di nuovi componenti. In questo caso, il mondo nel quale i componenti vivono è caratterizzato da un set di località, intese come le soluzioni nelle quali le reazioni chimiche possono verificarsi, inltre è previsto che i componenti possano diffondersi attraverso le diverse località per assicurare un'interazione globale.

La **Metafora biologica** fissa il focus sui sistemi biologici di piccola dimensione. Le specie sono quindi semplici cellule o animali senza intelligenza che agiscono sulla base di conportamenti molto semplici goal-oriented e vengono influenzati nelle loro attività dalla forza di segnali chimici presenti nel sistema. In maniera analoga a quanto visto per le metafore fisiche infatti i componenti sono in grado di spargere e diffondere segnali intorno, i quali possono influenzare il comportamente dei componenti. Le leggi presenti nell'ecosistema determinano come tali segnali si devono diffondere, e come esse sono in grado di influenzare il comportamento e le caratteristiche dei componenti. Il mondo nel quale i componenti vivono è tipicamente virtuale, è quindi possibile influenzare il processo attraverso il quale il segnale viene diffuso e la maniera con la quale i componenti possono muoversi sopra di esso.

La **Metafora Ecologica** pone il focus sui sistemi biologici al livello delle specie animali e delle loro interazioni. I componenti dell'ecosistem a sono una sorta di animali goal-oriented appartenenti ad una specifica specie, e che sono alla ricerca di cibo - risorsa necessaria per sopravvivere. Le leggi presenti nell'ecosistema determinano come possa essere realizzata la copertura di cibo, esse cioè determinano come e in quali condizioni agli animali sia permesso cercare il cibo, mangiare, e possibilmente riprodursi, in tale maniera l'intera dinamica del sistema e l'interazione tra gli individui di specie differenti viene influenzata. In maniera analoga a quanto visto per i sistemi chimici, lo scopo del mondo è quello di abilitare l'interazione e la diffusione di specie attraverso le nicchie.



2 Molecules of Knowledge

Verrà presentato nel seguito un modello nature-inspired, auto-organizzante e knowledge-oriented chiamato Molecule of Knowledge (MoK), esso si basa sull'astrazione di tuple space biochimico. In questo contesto le sorgenti di informazione produrranno nuovi atomi di conoscenza in compartimenti biochimici - spazio distribuito - i quali diffonderanno ed aggregheranno molecole attraverso l'applicazione di reazioni biochimiche, che agiranno sia localmente allo spazio sia interagendo con gli spazi vicini.

2.1 Introduzione al modello

In ogni metafora naturale disponibile in letteratura - fisica, chimica, biologica, sociale, etc. - è possibile riscontrare alcune caratteristiche comuni a tutti i modelli:*(i)* uno strato di ambiente comune che ne definisce *(ii)* leggi naturali fondamentali e funzioni disponibili sulle quali *(iii)* gli individui (possibilmente di specie differenti) possono vivere ed interagire; essi, al fine di portare a termine i propri obiettivi, interagiscono e competono nell'utilizzo dello spazio utilizzando le leggi sopra introdotte portando così ad un evoluzione dell'intero

ambiente. Queste caratteristiche dinamico evolutive sono specifiche di tutti i sistemi biologici che popolano il pianeta terra; la vita è nella sua concezione più amplia una sorta di sistema eterno eternamente adattativo: i meccanismi, i protocolli e l'infrastruttura base della vita nonè cambiata e non cambierà dalla sua nascita. Semplicemente, appaiono nuovi individui, questi cercano la propria strada nel sistema, e possibilmente danno luogo alla evidenza di nuove reazioni e nuove combinazioni di elementi, con lo scopo di fare emergere nuove forme di vita e nuove dinamiche ecologiche. La chimica della vita è eterna, le forme sotto la quale essa si manifesta dipendono dalle specifiche caratteristiche dell'ambiente, e dalle specie che lo popolano. La cosa più importante da capire è che: *non sono gli individui che evolvono, ma l'ecosistema come insieme.*

Lo strato degli individui è quello nel quale tutte le entità vengono interpretate con l'astrazione universale di "oggetti vivi" che popolano il sistema. Il livello ambiente invece determina il set di "eco-laws" responsabili di essere il mezzo di comunicazione attraverso il quale gli individui possono interagire gli uni con gli altri, proseguendo verso il decadimento - processo di selezione naturale nel quale si determinano individui vincitori che permangono maggiormente nell'ambiente e perdenti che, evolutivamente, non porteranno avanti la propria specie - intrinseco dell'ecosistema.

La dinamica del sistema è quindi così determinata, gli individui presenti agiscono autonomamente sullo spazio al fine di raggiungere i propri obiettivi personali, la loro interazione è soggetta (coordinata da) eco-laws environment-base presenti.

Poichè l'applicazione di tali leggi environment-base può essere soggetta alla presenza e allo stato di altri individui si ha l'identificazione di un *feedback loop* essenziale per abilitare proprietà di auto-*. Tra i tanti modelli di coordinazione environment-base, nature-inspired solo due hanno una rilevanza specifica per il modello di coordinazione **MoK**: *stigmercic coordination* e *field-base coordination.*

La *stigmergic coordination* è basata sull'utilizzo dell'ambiente come media

di coordinazione nel quale scrivere informazioni riguardanti l'effetto di comportamenti passati al fine di influenzare quelli futuri. Molti comportamenti sociali degli insetti , come la ricerca di cibo o il collective clustering, possono essere mappate su meccanismi di stigmergic. Anche comportamenti umani collettivi in sistemi sociali possono essere visti come meccanismi stigmergici: in questi casi però la traccia lasciata nell'ambiente dalle attività degli individui sono riconducibili anche ad un interpretazione simbolica, qui infatti la stigmergy e il processo cognitivo determinano insieme l'auto-organizzazione. Nella stigmercy cognitiva l'auto-organizzazione è essenzialmente un processo cognitivo distribuito dove l'ambiente, riempito dai produttori, con parti di conoscenza guidano l'individuo così come il comportamento complessivo dell'intero sistema.

La coordinazione *field-based* si affida alla computazione di campi virtuali utilizzando gravità ed elettromagnetismo come meccanismi base con i quali coordinare le attività attraverso un utilizzo dinamico e aperto di tali componenti applicativi. A differenza del mondo reale, qui i campi possono essere condivisi in accordo con ogni virtual law al fine di ottenere diversi pattern di coordinazione soddisfacendo così diversi obiettivi applicativi. Le strutture dati distribuite *campi* forniscono agli agenti le informazioni sul contesto per le loro attività e per la loro coordinazione.

Per concludere con questa introduzione al modello introduciamo il concetto di *centri di tuple biochimici* (biochemical tuple spaces), essi infatti rappresentano la fondamentale fonte di ispirazione per **MoK**. in questi modelli di coordinazione basati su spazi di tuple ogni tupla è associata ad un valore di *attività/pertinenza* il quale serve per rappresentare la concentrazione chimica e misura l'entità di influenza che tale tupla avrà rispetto allo stato di coordinazione. Le leggi chimiche incapsulate nello spazio, fanno evolvere la concentrazione delle tuple nel tempo simulando il comportamento tenuto dalle sostanze in una soluzione chimica. Inoltre, tali leggi, sono estese con un comportamento di *diffusione* che modella il passaggio di sostanze chimiche tra compartimenti biologici.

Le caratteristiche dei modelli biochimici di coordinazione tuple-based hanno mostrato di fornire un supporto effettivo per la gestione di sistemi pervasivi e auto-organizzati.

Estendendo l'interpretazione di tuple chimiche viste come contenitori di informazioni, **MoK** apre la strada ad una nuova classe di modelli di coordinazione knowledge-oriented, capaci di promuovere l'auto-organizzazione in ambienti knowledge-intensive.

2.2 Modello di coordinazione

Il modello *Molecules of Knowledge* si inspira alla metafora biochimica per riuscire a raggiungere un comportamento di auto-organizzazione della conoscenza. Le idee principali che stanno dietro a questo modello sono: *(i)* è la conoscenza che deve aggregarsi autonomamente e diffondersi al fine di raggiungere i consumatori e non questi ultimi che la devono andare a cercare (come solitamente avviene), *(ii)* la metafora biochimica è un modello adatto per l'auto-organizzazione della conoscenza in ambienti distribuiti knowledge-intensive.

Le astrazioni principale inserite dal modello MoK sono:

- **atomo**: la più piccola unità informativa presente nel modello, esso contiene informazioni anche riguardanti la sorgente che lo ha emesso e sta in un compartimento;
- **molecole**: sono aggregazioni di conoscenza, in particolare le *molecole* sono insiemi di *atomi*;
- **enzimi**: vengono emessi dai *catalizzatori*, gli *enzimi* influenzano le *reazioni* MoK, e questo ha effetto sull'evoluzione della conoscenza all'interno dei *compartimenti* di MoK;
- **reazioni**: lavorano ad uno specifico *rate*, le *reazioni* sono le leggi biochimiche che regolano l'evoluzione di ogni *compartimento* di MoK, governando l'*aggregazione* della conoscenza, la *diffusione* e la *decadenza* all'interno dei compartimenti MoK.

Altre astrazioni molto rilevanti introdotte da MoK sono aspetti come: topologia, produzione e consumo di conoscenza, in particolare:

- **compartimenti**: sono l'astrazione spaziale di MoK, i *compartimenti* rappresentano il concetto di luogo per tutte le entità MoK e fornisce al modello la nozione di *località* e di *vicinato*;
- **sorgenti**: ognuno è associato ad un *compartimento*, essi rappresentano l'origine della conoscenza, che viene inserita sotto forma di *atomi* ad un certo *rate* all'interno di un *compartimento*;
- **catalizzatori**: sono l'astrazione utilizzata per rappresentare i *consumatori* di informazioni, essi emettono un *enzima* ogni qual volta eseguono un'azione di consumo, questo ha lo scopo di avere un side effect sulla dinamica del compartimento al fine di crescere la probabilità di fornire ai *catalizzatori* informazioni rilevanti.

Nel proseguo di questa sezione verranno descritti più nello specifico le astrazioni principali introdotte da Mok.

Atomi Gli *atomi* sono la forma di conoscenza base presente nel modello. Un atomo Mok è prodotto da una fonte di conoscenza e contiene al suo interno un pezzo di conoscenza generato dalla sorgente. Un atomo non deve contenere solo informazione grezza, al suo interno si ritrovano infatti anche informazioni riguardanti all'origine , in maniera da preservare il significato dell'informazione stessa.
Un atomo MoK è sostanzialmente una tripletta nella forma:

$$atom(src, val, attr)_c$$

dove:

- *src* identifica la fonte che ha emesso l'informazione;
- *val* è il pezzo di informazione inserito;
- *attr* rappresenta essenzialmente un attributo del contenuto, cioè un informazione aggiuntiva che serva ad identificare meglio il contenuto - possibilmente espresso rispetto ad una specifica ontologia definita;
- c è la concentrazione corrente dell'atomo, inizializzata nel momento in cui l'atomo viene inserito nel compartimento e che evolverà nel tempo in accordo con l'evoluzione del sistema.

Molecole Un sistema MoK può essere visto come una collezione di atomi generati ad un certo rate che vagano per i compartimenti presenti e che probabilmente collidono all'interno del compartimento in cui risiedono. Queste collisioni danno come risultato *molecole di conoscenza*, cioè aggregazioni di atomi avvenute in maniera spontanea, stocastica e guidate dall'ambiente; queste aggregazioni vengono eseguite per reificare relazioni semantiche esistenti tra gli atomi che possibilmente possano aggiungere nuova conoscenza al sistema. Ogni molecola è semplicemente un insieme non ordinato di atomi semanticamente collegati. Una molecola MoK ha la seguente struttura:

$$molecule(Atoms)_c$$

dove:

- c è la concentrazione corrente della molecola;
- *Atoms* rappresenta la collezione di tutti gli atomi che correntemente sono all'interno della stessa molecola - che quindi rappresentano l'insieme di parti di conoscenza che in una certa catena di reazioni è stata aggregata durante la normale evoluzione del sistema.

Nella stessa maniera degli atomi, anche le molecole possono aggregarsi al fine di generare nuova conoscenza: esse infatti possono generare molecole più complesse basandosi su relazioni semantiche riscontrate da alcuni atomi presenti in esse.
A tale scopo è possibile vedere una relazione semplicistica della relazione che esiste tra atomi e molecole: è infatti possibile paragonare un atomo ad una molecola contenente un solo atomo, in questo modo avendo a disposizione un atomo a ed una molecola $m = molecule(a)$, è possibile considerare che $a = m$ ogni volta che questo dia una qualche utilità. A questo punto è possibile considerare l'intero sistema come una collezione di molecole che fluttuano nei compartimenti e che, reagendo tra di loro, generano autonomamente nuove informazioni aggregando informazioni presenti a seguito di reazioni presenti nell'ambiente, esse potranno poi diffondersi tra i compartimenti presenti.

Enzimi Una delle caratteristiche fondamentali del modello MoK è che il sistema è in grado di interpretare la azioni che i consumatori di informazioni eseguono rispetto alla conoscenza presente nel sistema - molecole, atomi - ed

interpreta tali azioni come feedback positivi che fanno si che la concentrazione relativa a tali atomi o molecole venga aumentata nel compartimento in cui risiede il consumatore. A tal fine il modello include l'astrazione di *catalizzatore* come consumatore che agisce sulla conoscenza - tipicamente un knowledge worker -, i quali producono *enzimi* ogni qual volta accedono un atomo od una molecola nel compartimento in cui risiedono. L'enzima può anche essere visto come uno strumento a disposizione del modello per aumentare la concentrazione di certi atomi o molecole al fine di produrre il feedback positivo necessario per abilitare l'auto-organizzazione della conoscenza.
Un *enzima* MoK ha la seguente struttura:

$$enzyme(Atoms)_c$$

dove:

- *Atoms* il set di atomi che un catalizzatore - consumatore - ha in qualche modo acceduto o per il quale ha comunque dimostrato un interesse in maniera inequivocabile;
- c rappresenta il valore di concentrazione che deve essere aggiunto alla concentrazione dell'atomo.

Reazioni Il comportamento di un sistema MoK è attualmente determinato dall'ultima astrazione fondamentale del modello, la sola che ne definisce un comportamento: la *reazione biochimica*. Le reazioni biochimiche all'interno del sistema guidano l'*aggregazione* di atomi e molecole, il loro *rinforzo*, il *decadimento* e la *diffusione*.

Mok function Essendo MoK un modello knowledge-oriented, il problema principale è determinare le relazioni semantiche che devono esistere tra gli atomi. A tal fine, per definire un sistema MoK, è necessario in prima istanza definire la *Mok function* base, vale a dire:

$$mok(atom_1, atom_2)$$

essa deve ritornare true se i due *atomi* presenti come argomenti della funzione sono semanticamente correlati - ciò significa che il concetto di relazione semantica non viene definita preliminarmente dal modello ma deve essere

cambiata a design time dall'utente a seconda delle sue esigenze, ciò rende il modello estremamente flessibile ed adattabile ad una moltitudine di casi di studio.
Il risultato di tale funzione tipicamente dipenderà dal valore attribuito al campo *attr* proprio di ogni atomo, essendo questo il solo attributo che può contenere informazioni riguardanti la semantica.

Aggregazione La *reazione di aggregazione* unisce atomi e molecole basandosi su relazioni semantiche:

$$molecule(Atoms_1) + molecule(Atoms_2) \longmapsto^{r\_agg} molecule(Atoms_1 \bigcup Atoms_2) + Residual(Atoms_1, Atoms_2)$$

dove:

- *r_agg* è il rate della reazione;
- $mok(atom_1, atom_2)$ vale per alcuni $atom_1 \in Atoms_1$, $atom_2 \in Atoms_2$
- $Residual(Atoms_1, Atoms_2)$ è il multiset di atomi ottenuti come differenza tra $(Atoms_1 \biguplus Atoms_2) \setminus (Atoms_1 \bigcup Atoms_2)$

Abbiamo quindi che ogni atomo appartenenti ad $Atoms_1$ oppure a $Atoms_2$ che non appartengono alle molecole risultato si identificano come *Residual*.
In breve, molecole più complesse vengono create ogni qualvolta si hanno degli atomi semanticamente correlati - funzione *mok*.

Rinforzo Il feedback positivo è ottenuto dalla reazione di *rinforzo* (Reinforcment reaction), la quale consuma una singola unità di enzima inserita da un catalizzatore al fine di produrre una singola unità di atomi/molecole rilevanti - a maggiore concentrazione.

$$enzyme(Atoms_1) + enzyme(Atoms_2)_c \longmapsto^{r\_reinf} molecule(Atoms_2)_{c+1}$$

dove:

- *r_reinf* è il rate della reazione;
- $enzyme(Atoms_1), molecule(Atoms_2)_c$ esistono nel compartimento con $c \neq 0$;
- $mok(atom_1, atom_2)$ vale per alcuni $atom_1 \in Atoms_1, atom_2 \in Atoms_2$.

Dinamiche spazio temporale Alcune reazioni biochimiche di MoK rappresentano l'evoluzione della conoscenza in accordo con alcuni pattern spaziali e temporali che tipicamente caratterizzano scenari spazio-temporali. Al fine quindi di arricchire il modello con il concetto di feedback negativo, le molecole dovrebbero scomparire al passare del tempo, perdendo quindi la loro concentrazione in accordo ad alcune *leggi di decadimento*. La rezione temporale che regola il *decadimento* è definita come segue:

$$molecule(Atoms)_c \longmapsto^{r\_decay} molecule(Atoms)_{c-1}$$

dove quindi una molecola scompare lentamente, ad un certo *rate r_decay*, la sua concentrazione viene decrementata di uno.
Analogamente a questo, un sistema distribuito auto-organizzato dovrebbe fornire una qualche forma di evoluzione spazio temporale. In accordo alla sua ispirazione biochimica, MoK adotta una *diffusione* come il meccanismo di migrazione della conoscenza. In questo contesto quindi la *migrazione* interessa un atomo o una molecola che migrano da un compartimento biochimico ad un altro, questo rappresenta il meccanismo base per la condivisione della conoscenza e lo spostamento autonomo in un sistema distribuito. In accordo con la metafora biochimica utilizzata, atomi e molecole si possono diffondere solo tra compartimenti *vicini*, simulando quindi il comportamento che si verifica tra le membrane delle cellule.
La reazione di diffusione adottata da MoK è quindi modellata come segue, assumendo che σ identifichi un compartimento biochimico, e $\|\|_\sigma$ rappresentano le molecole in un compartimento σ.

$$\|Molecules_1 \bigcup molecule_1\|_{\sigma^i} + \|Molecules_2\|_{\sigma^{ii}} \longmapsto^{r\_diffusion} \|Molecules_1\|_{\sigma^i} + \|Molecules_2 \bigcup molecule_1\|_{\sigma^{ii}}$$

dove:

- σ^i e σ^{ii} sono compartimenti vicini;
- *r_diffusion* rappresenta il rate di dffusione.

Il risultato di tale reazione è che $molecule_1$ passa da σ^i a σ^{ii}.
Grazie alla presenza del *decadimento* e del *rinforzo*, la *diffusione* assicura che le informazioni rilevanti vengano portate ai *consumatori* interessati; in

altre parole: la *diffusione* permette alle informazioni di viaggiare, il *rinforzo* aumenta la concentrazione delle informazioni rilevanti e il *decadimento* fa sparire quelle non rilevanti.

Chapter 2

Modellazione di sistemi biologici

Partendo dalle astrazioni proposte dal modello MoK sono state fatte alcune scelte tecnologiche per poter mappare tali concetti - atomi, molecole, enzimi, compartimenti - sullo strato tecnologico più adatto. Compito fondamentale di questa tesi è infatti quello di proporre un'implementazione del modello MoK quanto più fedele al modello concettuale descritto. Con questo si vuole intendere la necessità di ricercare la soluzioni migliore per mappare le astrazioni fondamentali del modello su uno strato infrastrutturale quanto più fedele possibile.

1 Nuove esigenze

L'importanza del modello MoK non è data solo dal suo approccio alla coordinazione e gestione della conoscenza descritto precedentemente, ma anche alla possibilità di applicare tale modello al contesto sociale che stiamo vivendo. La pervasività delle informazioni è ormai il punto focale su cui si basano tutti i prodotti tecnologici di nuova generazione; la fruizione di nuove informazioni non è ormai più vincolata al posto o allo strumento utilizzato, questo fa si che siano necessari modelli concettuali che permettano di rendere agevole e naturale tale paradigma di fruizione/condivisione/produzione delle informazioni.
Quello che si vuole fare è quindi identificare alcune caratteristiche fondamen-

tali che un modello di coordinazione distribuito come MoK debba avere e, al fine di utilizzare l'infrastruttura più adatta alla sua implementazione si verificherà se anche quest'ultima presenti tali caratteristiche.

- *Topology*, diventando l'obiettivo del nostro scenario sempre più quello del pervasive computing, sistemi multi-agente, Internet computing e intelligenza artificiale distribuita, possiamo assumere che l'intera applicazione venga distribuita in una rete topoligicamente-strutturata nel sistema. Ogni punto (i.e. Nodo) è direttamente connesso ad un set generalmente piccolo di vicini. I media di coordinazione e gli agenti sono anch'essi distribuiti nella rete, dove abbiamo che mentre gli agenti risiedono in una singola posizione della rete, i media di coordinazione possono essere distribuiti su un sottoinsieme di tutto il sistema;
- *Locality*, la dimensione topologica è strettamente connessa allo scopo dell'interazione. Un sistema coordinato può disporre due tipi di interazione: tra agente e medium di coordinazione e tra due medium di coordinazione. Va notato come il secondo tipo di interazione sia più difficile da riscontrare in un sistema, infatti tale capacità viene solitamente abilitata attraverso un agente che si prende carico di fare comunicare i due medium e non grazie ad una loro capacità intrinseca. Entrambi i tipi di interazione hanno luogo localmente, cioè tra sistemi che si trovano nello stesso luogo (agente-medium) o tra vicini definiti dalla topologia. In questo modo gli agenti hanno una visione parziale del sistema, per abilitare una comunicazione a "lunga distanza" è necessario abilitare una sequenza di interazioni locali - nello spazio e nel tempo;
- *On-line character*, i media di coordinazione tipicamente attivano delle regole attivate da specifiche interazioni in maniera tale che, le regole di coordinazione siano eseguite nel momento in cui avviene l'interazione. Con i sistemi auto-organizzanti invece, le regole di coordinazione reagiscono anche al passare del tempo, in altre parole un "comportamento di coordinazione" è attuato come un processo sempre attivo. Questo può essere visto come un servizio eseguito in background e sempre online, che oltre ad avere effetto sull'interazione del sistema, permette di evolvere il sistema con il passare del tempo. Questo processo associa gli agenti ad un comportamento autonomo, e grazie a questo definisce

un comportamento del sistema accurato e bilanciato. Ad esempio, nella coordinazione stigmergica il meccanismo della dissolvenza del feromone dipende strettamente dal rate al quali gli agenti si muovono e lavorano;

- *Time*, come la precedente proprietà ha introdotto, il processo di coordinazione auto-organizzata dipende fortemente dal tempo - come ogni altro processo auto-organizzato in presente in natura. In particolare, le regole di coordinazione sono generalmente temporali. Da una parte abbiamo che i servizi di coordinazione auto-organizzati devono essere forniti ad un certo rate: questo può essere attuato ad esempio facendo si che le regole di coordinazione si attivino reciprocamente quando scade un certo tempo (in un approccio ciclico). Similmente, certe primitive di coordinazione possono essere time-dependent, caratteristica tipicamente necessaria in contesti come Internet;
- *Probability*, il non-determinismo è un pattern della coordinazione, usato come potente principio di astrazione per separare il modello dalla sua implementazione. Per esempio, in LINDA ogni tupla che può essere letta matcha uno specifico template: comunque questo può causare che la stessa tupla venga presa più volte, fondamentalmente questo porta ad una gestione non corretta delle risorse. Un processo di auto-organizzazione è formato tipicamente da una grande quantità di istanze di azioni dello stesso tipo, le quali portano ogni volta ad effetti diversi. L'unico modo per descrivere il comportamento risultante è attraverso un modello stocastico. Può essere specificato che alcuni risultati sono migliori di altri, inoltre, l'alta adattatività di situazioni impredicibili implica la necessità di dare una bassa probabilità a certi comportamenti.

2 Un modello tuple-based

In un modello di coordinazione tuple-based gli agenti interagiscono attraverso lo scambio di tuple, le quali sono collezioni ordinate di informazioni (possibilmente eterogenee). Gli agenti comunicano, si sincronizzano e cooperano attraverso lo spazio condiviso scrivendo, leggendo e cosumando tuple in maniera associativa.

I principali benefici portati da un modello tuple-based sono *(i)* la separazione tra computazione e coordinazione, *(ii)* comunicazione generativa, *(iii)* accesso associativo allo spazio di interazione. La prima di queste features garantisce un architettura pulita mantendo computazione e coordinazione separate; la comunicazione generativa rende possibile il disaccoppiamento tra la dimensione degli agenti e lo spazio e il tempo; l'accesso associativo all'informazione di comunicazione rende possibile l'integrazione del modello tuple-based con sistemi eterogenei.
Il modello però aggiunge una complessità maggiore inserendo l'astrazione di regola biochimica - utilizzata sia per l'evoluzione dei rate che per l'aggregazione di conoscenza - per tale motivo appare evidente come un semplice spazio ti tuple non sia sufficientemente ricco per poter supportare un mapping del genere; la scelta è quindi stata quella di scegliere uno spazio di tuple che non fosse un mero contenitore di informazioni ma nel quale fosse possibile mappare le reazioni/leggi sopra descritte.
Per l'implementazione del modello è di centrale importanza avere una chiara visione di quello che l'infrastruttura valutata offre e non offre, obiettivo è infatti quello di esplicitare, ove presenti, i gap concettuali e tecnologici esistenti tra modello formale e astrazioni offerte dall'infrastruttura. In questo capitolo quindi non ci limiteremo ad introdurre le capacità e caratteristiche principali dei due tool TuCSoN e ReSpecT, infatti nel caso questi appaiano come la scelta più giusta verrano analizzati al fine di diminuire i gap concettuali presenti tra il modello ed essi e, a tal fine, verrano estesi nel caso appaia necessario.

3 TuCSoN

Uno dei principali vantaggi dell'interazione attraverso un spazio di tuple è che la coordinazione è guidata dall'informazione (*information-driven*): gli agenti si sincronizzano, cooperano e competono basandosi sulla disponibilità delle informazioni nello spazio condiviso attraverso l'accesso associativo, la consumazione e la produzione di informazioni.

Il problema principale dei modelli di coordinazione basati sugli spazi di tuple

è relativa al comportamento prefissato di questi medium di coordinazione. Dato un modello che adotti uno spazio di tuple come media di coordinazione, abbiamo che lo stato dello spazio è dato dall'effetto generato su di questo delle primitive di comunicazione applicate dagli agenti che condividono il mezzo.
Può inoltre accadere che uno spazio di tuple standard non sia in grado di supportare le politiche di coordinazione richieste dallo specifico caso, la quale non potrà quindi essere gestita nello spazio. Normalmente per sopperire a tale problematica si caricano i singoli agenti della conoscenza necessaria ad implementare la politica di coordinazione richiesta. Tale soluzione ha il grosso limite nel fatto che gli agenti devono essere *coordination-aware* e non possono quindi astrarre dai dettagli coordinativi.

3.1 Dagli spazi di tuple ai centri di tuple

Quello che un modello *centre-based* come TuCSoN porta è invece la possibilità di *(i)* mantenere l'interfaccia standard dello spazio di tuple *(ii)* potendo arricchire il comportamento dello spazio di tuple in termini di spazio di transizione generato a fronte di eventi comunicativi standard - in altre parole, un *centro di tuple* è uno spazio di tuple il cui comportamente a fronte di un evento comunicativo invocato su di esso non è standard ma definibile.
Un centro di tuple è quindi uno spazio di tuple arricchito con una *spacifica di comportamento*, essa viene espressa in termini di un *reaction specification language* ed associa ogni evento comunicativo possibile nel tuple centre ad un set di attività chiamate *reactions*. Più precisamente un reaction specification language:

- abilita a possibilità di inserire una definizione di computazione all'interno di un centro di tuple attraverso le reaction;
- permette di associare reazioni ad eventi comunicativi.

Ogni *reazione* può accedere e modificare lo stato corrente del centro di tuple (aggiungere o rimuovere tuple) ed accedere a tutte le informazioni riguardanti l'evento comunicativo che ne ha causato il triggering.
A questo punto la semantica delle primitive comunicative standard del centro di tuple non è più costretta ad essere tanto semplice quanto quella descritta

da modello Linda ma può essere tanto complessa quanto richiesta dalle particolari specifiche del problema corrente.
Mentre il classico modello di centro di tuple è indipendente dal tipo di tuple utilizzate, il centro di tuple TuCSoN adotta sia tuple logiche standard - tuple logiche Prolog-like senza alcun significato aggiuntivo rispetto alla mera funzione di contenitore di informazioni - che tuple ti specifica, cioè tuple aventi un significato comportamentale/reattivo specifico - tuple cioè che andranno ad arricchire il set di specifiche associate ad una determinata operazione comunicativa.
L'infrastruttura TuCSoN rende possibile lo sfruttamento del centro di tuple come servizio di coordinazione distribuito nella rete; in particolare lo spazio di coordinazione di TuCSoN è costituito di *nodi* TuCSoN, ognuno dei quali corrisponde ad un accesso ad Internet host o ad un server connesso ad internet.
Il *comportamento* del centro di tuple TuCSoN può essere personalizzato rispetto all'applicazione specifica definendo un set di *tuple di specifica*, o *reazioni*, che determinano come il centro di tuple deve reagire ad eventi uscenti/entranti.
Il linguaggio logico adottato da TuCSoN per programmare il centro di tuple è chiamato ReSpecT (*Reaction Specification Tuples*) ; attraverso tale linguaggio è possibile inserire nel sistema centro di tuple le specifiche di reazione che attiveranno i comportamenti reattivi richiesti.

3.2 Modello di coordinazione

In TuCSoN, gli agenti interagiscono attraverso una moltiplicità di media di coordinazione chiamati *centri di tuple* distribuiti nella rete. Gli agenti si scambiano informazioni sotto forma si tuple attraverso l'utilizzo di primitive Linda-like. Ogni centro di tuple è associato ad un nodo ed ha assegnato un nome univoco: in particolare, un centro di tuple può essere rappresentato da un riferimento globale o da un riferimento locale. In particolare un riferimento assoluto avente una forma del tipo *tc@node*, indica un centro di tuple avente nome *tc* situato all'interno del nodo avente indirizzo *node*. Da qui ne deriva la forma generale per utilizzabile per eseguire un operazione su un determinato centro di tuple *tc@node ? op(tuple)* dove viene chiesto al centro di tuple *tc* del nodo *node* di eseguire l'operazione *op* utilizzando la tupla *tuple*.

Come abbiamo precedentemente detto la caratteristica computazionale rappresentativa di un centro di tuple è quella di arricchire un semplice spazio di tuple con una specifica di comportamento: tale comportamento verrà sempre esplicitato a seguito di una operazione di comunicazione richiesta da un agente esterno. A seguito di ciò possiamo dire che assegnare un nuovo comportamento ad un centro di tuple significa una nuova transizione dello stato del centro a fronte dell'eseguirsi di un evento di comunicazione.
Ogni reazione associata ad un evento di comunicazione viene eseguito facendo riferimento ad una semantica transazionale: una reazione eseguita con successo modifica lo stato del centro di tuple in maniera atomica mentre un fallimento dell'esecuzione non porta alcun cambiamento.
In particolare operazioni sul centro di tuple come *out_s, in_s, rd_s* si comportano in maniera analoga ad operazioni di comunicazione escluso il fatto di essere in grado di triggerare nuove operazioni nella catena delle reazioni; infatti mentre le operazioni di comunicazione, come abbiamo visto riescono a gestire una catena di reazioni, questo non è verificato per le operazioni di specifica. L'insieme delle reazioni viene eseguita sequenzialmente e una catena di reazioni può venire eseguita correttamente o erratamente in maniera atomica, abbiamo cioè che l'esecuzione di una catena di reazioni tutte correlate tra di loro viene vista dall'agente esterno come un operazione atomica che può avere successo o fallire.

3.3 Topologia dell'infrastruttura

Lo spazio di interazione fornito da un sistema TuCSoN si realizza in una moltiplicità di centri di tuple distribuiti. In questo modo TuCSoN possiede i vantaggi di un modello basato su spazi di tuple distribuiti e unisce a questo la possibilità di incapsulare in questi leggi accesso e coordinazione. Questo porta ad introdurre la relazione tra modello di coordinazione e la sua topologia distribuita; ciò porta a discutere di due problematiche: *(i)* com'è modellato lo spazio in cui gli agenti vivono (modello della rete), *(ii)* come la conoscenza della struttura dello spazio è resa disponibile agli agenti (conoscenza della rete).
Problemi di questo tipo sono più evidenti quando ci troviamo ad affrontare problemi come la struttura di un dominio, infatti i nodi internet vengono

spessi suddivisi in cluster cioè soggetti con politiche di accesso assolutamente specifiche e selettive. Inoltre tali cluster possono essere raggruppati a loro volta da sub-cluster definendo in questo modo strutture gerarchiche. Questi cluster differenti in contesti come Internet possono determinare i priveligi di accesso e modifica dei dati presenti a seconda del tipo di utente che si è.
In contesti decentralizzati come questi è irreale poter pensare che tutti gli agenti abbiamo una visione completa della topologia della rete e della disponibilità di risorse all'interno di ogni singola parte. Reti come Internet sono spesso dinamiche e impredicibili ed è per questo impossibile prevedere una visione d'insieme valida e comune per tutti gli utenti. È perciò possibile pensare ad una struttura nella quale notizie riguardante la struttura della rete possono essere apprese in maniera dinamica ed incrementale attraverso le interazioni con l'ambiente. Questo fa si che parte della comunicazione effettuata dagli agenti debba riferirsi alla conoscenza della struttura topologica e rende la conoscenza della struttura della rete un problema di coordinazione.
In TuCSoN possiamo idealmente visualizzare dei gateway come astrazioni atte ad abilitare la comunicazione in una struttura gerarchica come quella che si sta palesando. Un gateway in particolare è, in questo contesto, un nodo con l'abilità di autenticare ed autorizzare l'accesso degli agenti a nodi distribuiti nella rete.
La topologia che TuCSoN mette a disposizione permette quindi di istanziare una struttura distribuita e potenzialmente innestata riuscendo ad emulare anche le più complicate reti internet provviste di firewall e regole di accesso; questo fa si che sia possibile gestire modelli di coordinazione di complessità molto maggiore rispetto ad un semplice sistema locale permettendo anche di simulare dinamiche di concorrenza e coordinazione su reti di maggiore entità (potenzialmente senza porre limiti alle dimensioni).
TuCSoN mostra di possedere le capacità di coordinazione auto-organizzata viste precedentemente:

- *Topology*, assumendo che la rete sia organizzata in una topologia strutturata distribuita nella rete, TuCSoN permette di creare uno o più centri di tuple localmente ad ogni specifico nodo nella rete;
- *Locality*, in TuCSoN una primitiva di coordinazione può essere eseguita su un centro di tuple fornendo il suo identificatore (nome e ind-

irizzo di rete) - sia per la comunicazione agente medium che medium medium. Per strutturare un sistema multiagente di coordinazione auto-organizzato, abbiamo semplicemente bisogno che gli agenti e i centri di tuple siano a conoscenza della lista degli identificativi dei centri di tuple che fanno parte del proprio vicinato;

- *On-line character and Time*, ReSpecT supporta reazioni temporali, cioè reazioni nelle quali l'evento di trigger è del tipo `time(T)`. Quando il centro di tuple raggiunge l'istante `T`, la reazione corrispondente viene eseguita. Inoltre, l'obiettivo di una reazione può anche essere del tipo `out_s(reaction(time(T),G,R))`, la quale inserisce la tupla `reaction(time(T),G,R)` nello spazio, la quale esegue una nuova reazione. Con un meccanismo di questo tipo è possibile realizzare anche servizi online che trasformano lo stato del sistema in funzione del passare del tempo;
- *Probability*, la probabilità è supportata in TuCSoN attraverso due strade. Da una parte possiamo utilizzare un generatore di numeri random per guidare le reazioni attraverso un processo di esecuzione, in modo tale che la trasformazione delle tuple può essere guidata da un processo probabilistico. Dall'altro lato sarebbe di notevole utilità poter disporre di una versione non deterministica delle operazioni presenti: ad esempio della `rd` la `urd`. La cui semantica sarebbe: tra tutte le tuple che matchano il template scegline una in maniera equiprobabilistica.

Allo stato attuale TuCSoN possiede tutte le proprietà sopra elencate tranne la possibilità di avere operazioni non deterministiche (del tipo `urd`), nel caso questo fosse il tool selezionato per l'implementazione del modello MoK sarà necessario risolvere tale mancanza attraverso l'utilizzo di un work-around o di una modifica all'interno del sistema.

4 ReSpecT

ReSpecT, quale linguaggio logico per programmare il centro di tuple TuCSoN, possiede una parte *dichiarativa* e una *procedurale*. Essendo un *linguaggio di specifica*, ReSpecT, permette di associare dichiarativamente eventi a reazioni nella forma di *tuple di specifica* la cui forma è `reaction(E,G,R)`: tale tupla associa una reazione $R\theta$ a Ev se $\theta = mgu(E, Ev)$ e la guardia `G` è true.

Una guardia è una sequenza di predicati guardie, cioè un ampio numero di condizioni riguardo un evento possono essere tutte verificate prima che una reaction venga triggerata in un centro di tuple: lo stato dell'evento, la sorgente, l'obiettivo, il tempo etc..

Come *reaction language*, ReSpecT , da la possibilità di essere definito in maniera procedurale in termini di una sequenza di reaction goals, ognuno dei quali può avere successo o fallire. La sequenza di reaction che popolano la specifica di un centro di tuple può essere vista come un unica reaction, essa infatti avrà successo se tutte le reazioni che la compongono avranno successo altrimenti, nel momento in cui una di esse fallisce, l'intera sequenza di reaction fallirà e dal punto di vista del centro di tuple non verranno percepiti alcune modifiche sullo stato del centro. Ogni reaction viene eseguita sequenzialmente con una semantica transazionale, per questo, il fallimento di una reazione non porta alcun effetto allo stato del centro.

Tutte le reazioni triggerate dallo stesso evento comunicativo sono eseguite prima di servire qualsiasi altro evento: in questo modo, l'agente percepisce il risultato della gestione dell'evento e dell'esecuzione di tutte le reaction insieme, viste cioè come una singola transizione dello stato del centro di tuple.

Essendo ReSpecT Turing-equivalent, è possibile utilizzarlo per eseguire qualsiasi computazione, quindi ogni legge di coordinazione può essere incapsulata in un centro di tuple ReSpecT. Per questo motivo ReSpecT può essere utilizzato come linguaggio general-purpose per la coordinazione: un linguaggio che è in grado di rappresentare ed applicare politiche e regole di ogni tipo di sistema collaborativo.

Adottando l'interpretazione dichiarativa delle tuple ReSpecT, un centro di tuple TuCSoN può avere una duplice natura:

- come teoria comunicativa - set di tuple ordinarie;
- teoria coordinativa - set di tuple di specifica.

In questo modo inoltre è possibile rappresentare in maniera univoca - fatti Prolog-like - sia tuple dichiarative che procedurali.

4.1 Semantica del linguaggio

Semantica Informale

Prima di passare alla descrizione formale del comportamento di un centro di tuple, ne descriveremo il comportamento in via informale.

Ogni qual volta viene invocata una primitiva del centro di tuple da un agente o da un artefatto viene generato un evento interno a ReSpecT il quale poi raggiunge il centro di tuple destinazione dove viene immediatamente inserito in una coda degli eventi chiamata *InQ*. Quando il centro di tuple è in stato *idle* (cioè non sono presenti attualmente reazioni da venire eseguite), il primo evento ϵ in *InQ* (in accordo alla politica FIFO) è caricato e spostato nel multiset *Op* delle richieste da essere servite: questo stadio è chiamato *request phase* dell'evento ϵ. Conseguentemente le reazioni alla richiesta di ϵ vengono attivate ($Z_E(\epsilon) \cup Z_E(n)$) aggiungendole al multiset *Re* delle reazioni attivate che attendono di essere eseguite.

Tutte le reazioni attivate presenti in *Re* sono poi eseguite in un ordine non deterministico. Ogni reazione viene eseguita sequenzialmente, con una semantica transazionale e può attivare altre reazioni, anch'esse aggiunte al set *Re*, come eventi di output scaturiti da una invocazione di collegameto (fatta da un artefatto e non da un agente): tali eventi sono aggiunti al multiset *Out* degli eventi uscenti e sono poi inseriti nella coda degli eventi di uscita *OutQ* alla fine dell'esecuzione della reazione - nel caso in cui questa abbia successo.

Solo quando *Re* è stato svuotato, le richieste in attesa di essere servite *Op* possono venire eseguite dal centro di tuple, e le operazioni vengono inviate indietro alle entità invocatrici. Questo ò dare luogo a reazioni future associate alla fase di *response* dell'invocazione originale, e può quindi dare luogo ad una nuova fase di esecuzione analoga a quella appena vista per la fase di *request*.

Questo è il ciclo principale che caratterizza ReSpecT, vedremo in seguito come questo è stato implementato a livello di codice per avere un'idea più chiara di come le funzionalità descritte dal modello siano più o meno presenti nella sua implementazione

Semantica Formale

Come linguaggio di specifica, ReSpecT, definisce la sintassi delle tuple di specifica che popolano lo spazio. In particolare ReSpecT definisce un *set di operazioni di comunicazione ammissibili*:

$$\mathcal{O} = \{ \texttt{out, in, inp, rd, rdp , set, get, no, nop}\}$$

e il set di *operazioni di reazione ammissibili*:

$$\mathcal{O}_s = \{\texttt{out\_s, in\_s, inp\_s, rd\_s, rdp\_s, set\_s, get\_s, no\_s, nop\_s}\}$$

,come le operazioni utili alla comunicazione, accesso e consumo di tuple nel centro di tuple. L'insieme di operazioni $\mathcal{O}^+ = \mathcal{O} \cup \mathcal{O}_s$ contiene tutte le operazioni ammissibili sul centro di tuple per un centro ReSpecT.

Un medium di coordinazione è adatto ad una caratterizzazione operativa espressa in termini di un sistema di transizione, nel quale lo stato della comunicazione è dato dallo stato del sistema, alcune transizioni sono azionate da eventi di interazione e alcune transizioni generano in output eventi di interazione. Detto questo e specificati i set di operazioni di *comunicazione eseguibili ammissibili* sullo spazio come $\mathcal{O}$ *e* $\mathcal{O}_s$ è possibile descrivere il comportamento del sistema di transizione.
Come abbiamo precedentemente descritto, la logica di una reazione inserita nel centro di tuple del tipo `reaction(E,G,B)` è quella per la quale a fronte dell'esecuzione di un certo evento di comunicazione appartenente ad $\mathcal{O}^+$ ed essendo la/le guardie `G` verificate, verranno eseguite tutte le operazioni presenti nel `B`.

Lo stato di un centro di tuple può essere espresso come la coppia $\langle T,\ W,\ Z \rangle_\sigma$, dove:

- T è il multi-set di tuple loogiche ordinarie che correntemente popolano lo spazio;
- W è il multi-set di query pendenti, tali che, la richiesta eseguita da un agente per certe tupe è stata accettata dal centro ma è ancora in attesa di essere eseguita;
- Z è il multi-set di triggered-reactions in attesa di essere eseguite

Dato un centro di tuple il cui stato sia $\langle T,\ W,\ Z\ \rangle_\sigma$, denotiamo come W_s il multiset di pending query in W tali che possano essere soddisfatte da alcune tuple T in accordo con il matching predicativo M. In particolare, W_s è definito in modo tale che se $o^a \downarrow_n \ \mathtt{t} \in W$, dove $o \in \{\mathtt{in,rd,inp,rdp}\}$, e $\exists$ t$\in\ T$ tale che $\mathtt{M}(\mathtt{t},t)$ è vero, allora $o^a \downarrow_n \ \mathtt{t} \in W_s$. Inoltre rappresentiamo con W_p il multi set delle pending query in W che corrispondono a operazioni di query predicative, tali che, possano essere identificati con una semantica di successo/fallimento. In particolare, W_p è definito in modo tale che se $o^a \downarrow_n \ \mathtt{t} \in W$ e $o \in \{\mathtt{inp,rdp}\}$, allora $o^a \downarrow_n \ \mathtt{t} \in W_p$.
Il comportamento operazionale di un centro di tuple può essere ora modellato in termini di un sistema di transizione con tre tipi di transizioni ammissibili:

- *listening* ($\longrightarrow_l$), prendendo l'evento di comunicazione triggerato dall'agente come input;
- *speaking* ($\longrightarrow_s$), ritornando la risposta all'agente come output;
- *reacting* ($\longrightarrow_r$), gestendo l'esecuzione della reazione.

Ogni qualvolta non ci sono task da compiere, e quando non ci sono ne query soddisfabili pendenti ($W_s = \emptyset$), ne query predicative in attesa di risposta (W_p) $= \emptyset$, ne triggered reaction da eseguire ($Z = \emptyset$), un centro di tuple è in attesa di un evento comunicativo dall'agente: in questo stato, possiamo dire che il centro di tuple è *listening*. Quando un evento di comunicazione raggiunge il centro di tuple, una transizione di *listening* viene triggerata, essa può assumere una delle seguenti forme, a discrezione dell' event-operation generato:
if $W_p = W_s = \emptyset\ and\ o = \mathtt{out}$,

$$\langle T, W, \emptyset\rangle_\sigma \xrightarrow{o^a \downarrow_n t}_l \langle T \uplus \{t\},\ W, \mathtt{Z}_\sigma(o^a \downarrow_n t)\rangle_\sigma$$

if $W_p = W_s = \emptyset\ and\ o \in \{\mathtt{in,rd,inp,rdp}\}$,

$$\langle T, W, \emptyset\rangle_\sigma \xrightarrow{o^a \downarrow_n t}_l \langle T,\ W \uplus \{o^a \downarrow_n t\}\ , \mathtt{Z}_\sigma(o^a \downarrow_n t)\rangle_\sigma$$

In particolare, $\mathtt{Z}_\sigma(o^a \downarrow_n t)$ rappresenta il multi set delle reazioni triggerate dall'evento $\{o^a \downarrow_n t\}$ in accordo con la specifica di comportamento σ.
Quando non sono presenti reazioni triggerabili, ma ci sono invece pending query ($W_s \neq \emptyset$) oppure una query predicativa con nessun risultato soddisfacente ($W_s = \emptyset \wedge\ W_p \neq \emptyset$), una transizione *speaking* è triggerata, in accordo ad una delle seguenti forme:

if $o \in \{\texttt{in,inp}\}$ and $\texttt{M}(t,t')$,

$$\langle T \uplus \{t\},\, W \uplus \{o^a \downarrow_n t\}, \emptyset \rangle_\sigma \xrightarrow{o^a \uparrow_n t'}_s \langle T, W, Z_\sigma(o^a \uparrow_n t') \rangle_\sigma$$

if $o \in \{\texttt{rd,rdp}\}$ and $\texttt{M}(t,t')$,

$$\langle T \uplus \{t\},\, W \uplus \{o^a \downarrow_n t\}, \emptyset \rangle_\sigma \xrightarrow{o^a \uparrow_n t'}_s \langle T \uplus \{t'\}, W, Z_\sigma(o^a \uparrow_n t') \rangle_\sigma$$

if $W_s = 0$ *and* $o \in \{\texttt{in,inp}\}$,

$$\langle T,\, W \uplus \{o^a \downarrow_n t\}, \emptyset \rangle_\sigma \xrightarrow{o^a \not\uparrow_n t}_s \langle T, W, Z_\sigma(o^a \not\uparrow_n t) \rangle_\sigma$$

Tutte le regole sopra elencate forniscono il risultato in un evento di output inviato indietro all'agente che ne ha scatenato il triggering a come risposta alla sua precedente query, precedentemente registrata come una pending query $o^a \downarrow_n t$ da una transizione di *listening*. In particolare, le prime due regole corrispondono ad una risultato positivo di gestione della query, mentre invece la terza rappresenta il caso di fallimento di una query predicativa.
In fine, ogni qualvolta una reazione di triggering sta per essere eseguita, una transizione *reacting* viene eseguita, in accordo con la seguente forma:

$$\text{if } z \in Z,\ \langle T, W, Z \uplus \{z\} \rangle_\sigma \longrightarrow_r \langle T', W, Z \uplus Z' \rangle_\sigma$$

dove $(T',Z') = \texttt{E}_\sigma(z,T)$ risulta dall'esecuzione della reazione z in accordo con la funzione di valutazione della reazione E.

Può essere notato come un centro di tuple non riceverà eventi esterni e nemmeno servirà query pendenti fino a quando non saranno eseguite tutte le razioni. In altre parole, tutte le reazioni triggerate da un evento di comunicazione in accordo alla specifica inserita nel centro di tuple sono triggerate dal centro di tuple prima di gestire ogni richiesta futura proveniente dall'agente. Come conseguenza di tutto questo, gli agenti percepiscono il risultato di ogni evento di comunicazione e degli effetti generati dalle reazioni a questi associate come un tutt'uno, cioè come una singola transizione dello stato del centro di tuple.

5 MoK su infrastruttura TuCSoN

Ciò che TuCSoN introduce di rivoluzionario nell'ambito della coordinazione multi agente, è la possibilità di avere centri di tuple *programmabili distribuiti* su diversi nodi della rete.
Punto focale da ricordare è che TuCSoN è completamente svincolato dal linguaggio utilizzato per programmare il comportamento reattivo dei centri di tuple, la scelta di utilizzare ReSpecT rappresenta quindi solo una delle molteplici possibilità presenti.
Appare quindi evidente come un'analisi di tale infrastruttura possa essere divisa in due scope principali:

- TuCSoN ha l'ownership di occuparsi dell'infrastruttura distribuita dei centri di tuple;
- ReSpecT si occupa di definire la programmabilità dei centri e di interfacciarsi ad essi

Possiamo visualizzare il sistema TuCSoN ReSpecT come un architettura divisa sostanzialmente in tre livelli principali: al livello più basso troveremo i centri di coordinazione tuple-based, al livello intermedio vi è ReSpecT che si occupa di interagire con i centri sia attraverso primitive Linda-like che attraverso specifiche di reazione, al livello più alto vi è TuCSoN che si occuperà di gestire l'infrastruttura distribuita dei centri di tuple.
Dal momento che i centri di tuple logic-based ReSpecT sono adatti per la gestione della rappresentazione della conoscenza, la prima implementazione di MoK è costruita basandosi su TuCSoN. In particolare, TuCSoN fornisce facilmente i due ingredienti principali per la coordinazione biochimica: *(i)* l'evoluzione stocastica delle tuple inspirate ad un modello chimico - ottenuto grazie all'implementazione del noto algoritmo di Gillespie per la simulazione di ambienti biochimici come una specifica ReSpecT -, *(ii)* il matching possibile tra leggi biochimiche e tuple dello spazio - ottenibile interpretando le tuple come individui del sistema biochimico, ed utilizzando dei template per rappresentare i reagenti nelle reazioni biochimiche.
Passeremo ora ad elencare i mappi individuati tra modello MoK ed infrastruttura TuCSoN al fine di reificare i concetti visti teoricamente in parti di un contesto tecnologico conosciuto come i centri di tuple.

Il primo mapping naturale dei concetti visti in MoK sull'infrastruttura TuCSoN è il seguente:

$$Individuals \longrightarrow Atoms/Molecules/Enzymes \longrightarrow (Logic)\ Tuples$$

Quando i sistemi computazionali sono visti come *eco-sistem*, gli *individui* sono gli abitanti dell'ambiente, essi diventano sia i soggetti delle leggi sia, allo stesso tempo, le entità proattive agiscono sull'ambiente. In MoK gli individui sono tutte le parti di conoscenza che risiedono nei compartimenti biochimici: i soggetti e gli attivatori delle reazioni biochimiche. In TuCSoN tutte queste astrazioni vengono tradotte in tuple logiche.

La rappresentazione del substrato-ambiente in termini del centro di tuple TuCSoN è chiaro: l'ambiente contiene sia gli individui che le leggi naturali allo stesso modo in cui un centro di tuple mantiene tuple e regole di coordinazione:

$$Environmental\ Substrate \longrightarrow Compartments/Catalysts \longrightarrow \text{TuCSoN}\ Tuple\ Centres$$

Dal momento che i catalizzatori sono gli utenti, e che ogni utente ha un proprio compartimento nel quale lavorare, sia i catalizzatori che i compartimenti sono mappati su un centro di tuple.

In un compartimento biochimico sono poi presenti leggi biochimiche, tale ruolo viene svolto da reazioni biochimiche che a loro volta vengono mappate in TuCSoN come reazioni ReSpecT:

$$Laws\ of Nature \longrightarrow Biochemical\ Reactions \longrightarrow \text{ReSpecT}\ Reactions$$

Ogni centro di tuple TuCSoN rappresenta un compartimento MoK programmato per lavorare come una virtual machine biochimica che implementi l'algoritmo chimico di Gillespie, e che adotti le reazioni biochimiche di MoK come reazioni ReSpecT.

5.1 Mapping dei concetti

In questa sezione ci occuperemo di vedere operativamente quali dei concetti proposti dai modello di TuCSoN e ReSpecT sono stati implementati; avendo deciso di utilizzare tale infrastruttura per eseguire l'implementazio del modelo MoK è ora necessario capire se tutti i concetti alla base dei quali si affida il mapping di MoK sull'infrastruttura siano presenti o se sia necessario introdurre alcune modifiche al fine di avere un sistema quanto più completo.

Operativamente TuCSoN fornisce un'infrastruttura capace di permettere ad agenti software indipendenti e distribuiti nella rete di comunicare e di coordinarsi. Al fine di rendere chiaro quanto appena detto, si specifica che un agente software non è altro che un componente software che incapsula non solo il suo stato e il suo comportamento ma anche il suo flusso di controllo.
Per poter comunicare con un centro di tuple TuCSoN un agente deve in prima istanza acquisire un ACC (*agent coordination context*) questo rappresenta operativamente un 'interfaccia che un agente deve acquisire al fine di poter invocare qualsiasi operazione su un centro di tuple. Acquisito un ACC, un agente, è in grado di comunicare e coordinarsi attraverso il centro di tuple, idealmente l'astrazione di ACC va associata ad un contesto più avanzato, possiamo infatti pensare di voler assegnare a determinati agenti un determinato/limitato set di operazioni applicabili/permesse. In questo modo possiamo vedere l'ACC come l'astrazione fondamentale grazie alla quale è possibile gestire gli accessi e i permessi dei singoli agenti che accedono ai centri di tuple.
Acquisito un ACC, un agente, è in grado di invocare operazioni sul centro di tuple; essendo però una rete a topologia distributa è chiaro che il centro di tuple non possa riesedere localmente all'agente, ciò viene gestito attraverso la presenza di un'entità software locale all'agente che si occupa di accogliere le richieste di quest'ultimo e di inoltrarle al centro indicato nella specifica.
Possiamo quindi, partendo dal processo di acquisizione di un ACC, vedere il flusso base che un agente deve eseguire per poter comunicare con un centro di tuple e le entità che entrano in gioco: innanzi tutto viene acquisito l'ACC, successivamente viene invocata un'operazione tra quelle disponibili nel contesto acquisito, tale richiesta arriva ad un *proxy* lato nodo, il proxy

inoltra la richiesta al centro di tuple a cui fa riferimento, da questo punto in poi proxy e ACC si scambieranno informazioni in rete e l'agente le acquisirà grazie all'acquisizione dell'ACC.

Come abbiamo detto una richiesta partita dall'agente attraversa l'ACC, la rete, il proxy, ed arriva al centro di tuple specificato. Nel momento in cui il centro di tuple riceve l'evento generato a fronte della richiesta di operazione vista questo lo inserisce nella coda degli eventi *InQ* e fa partire il processo descritto precedentemente.

Nell'ambito dell'implementazione di ReSpecT vedremo invece come il sistema di transizione descritto precedentemente e la gestione degli eventi e delle code associate venga gestito a livello software. Non ci soffermeremo a descrivere parti di codice ma introdurremo solo una breve presentazione sulle classi principali.

Partiremo ora ad elencare le classi principali e le loro respondabilità:

ReSpecT VM

La classe *RespectVM* rende possibile avere un comportamento iterativo all'interno del *motore* ReSpecT, tale classe è infatti un thread che rimane sempre in ascolto di nuovi eventi generati a fronte della richiesta di nuove operazioni da parte di agenti esterni. Tale classe si occupa principalmente di permettere che il motore ReSpecT sia sempre in grado di ricevere e gestire eventi generati a fronte dell'invocazione di operazioni e che questi non vengano persi ma vengano correttamente gestiti e quindi inseriti all'interno all'interno della coda *InQ*.

La macchina a stati ReSpecT

Ciò che implementa operativamente quanto descritto brevemente nella sezione di *semantica informale*, cioè la gestione degli eventi presenti nella coda di input *InQ* e l'attivazione delle reazioni presenti nello spazio generando quindi un ciclo continuo, è una macchina a stati implementata basandosi sul sistema di transizione visto. Essendo questo il cuore dell'infrastruttura ReSpecT passeremo ad una descrizione dettagliata.

Gli stati presenti nella macchina a stati sono sei, i principali di questi sono ricavati dal mapping uno-a-uno con le tre transizioni viste nel capitolo prece-

dente, a questi sono stati aggiunti tre ulteriori stati necessari per la corretta gestione:

- *ResetState*, stato iniziale nel quale la macchina a stati si trova solo al primo avvio;
- *IdleState*, stato in cui si trova il sistema quando essa è in idle, quindi non vi sono eventi da gestire ne reaction da computare;
- *ListeningState*, la macchina a stati è in attesa di un evento proveniente dall'agente esterno ;
- *ReactingState*, stato del sistema raggiunto nel momento in cui sono presenti reazioni triggerabili;
- *FetchEnvState*, stato raggiunto nel caso in cui siano percepiti eventi dall'environment;
- *SpeakingState*, stato che si occupa della gestione dell'esecuzione dell'operazione richiesta dall'agente.

Ad ognuno degli stati visti corrisponde una classe avente lo stesso nome che ne implementa il comportamento. Per chiarezza verrà mostrata una macchina a stati rappresentante la logica sopra vista. **RespectVMContext**
Di particolare inportanza per quel che riguarda l'interfacciamento a basso livello di ReSpecT con lo spazio di tuple che utilizza per contenere le due diverse tipoligie di tuple è la classe *RespectVMContext*, tale classe infatti contiene al suo interno due funzionalità molto importanti:

- la gestione a basso livello dell'inserimento/rimozione/lettura delle tuple presenti nello spazio;
- le funzionalità indispensabili per verificare la presenza di reaction che abbiamo event triggering uguali a quello attualmente arrivato al sistema.

Appare evidente quindi come le funzionalità messe a disposizione di questa classe siano di assoluta importanza, in particolare qual'ora si volessero inserire nel sistema nuove operazioni di comunicazione con il centro di tuple sarà necessario operare direttamente anche in tale classe - tale valutazione è fatta rispetto a quanto verrà descritto nei capitoli futuri.

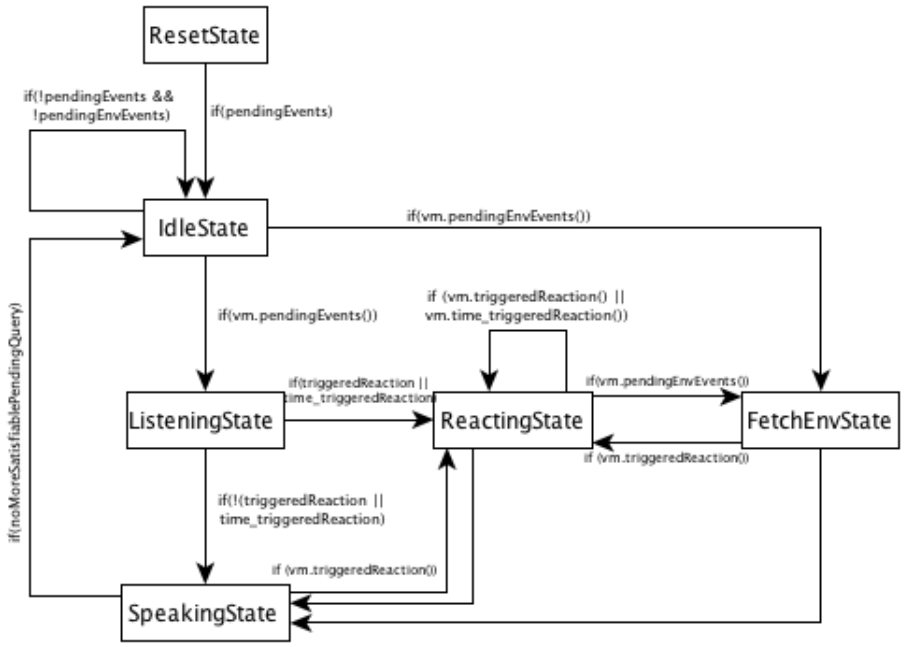


Figure 2.1: Macchina a stati ReSpecT

Contesti ReSpecT

L'interazione tra agente e engine ReSpecT può essere abilitata attraverso due paradigmi comunicativi differenti: *bloccante* e *non bloccante*. È di fondamentale importanza esplicitare in maniera chiara la differenza che si ha tra comunicazione agente-centro bloccante e di operazione di comunicazione del centro di tuple bloccante.

La differenza tra operazione di comunicazione bloccante e non bloccante è molto chiara, un operazione di rimozione bloccante come la `in` non restituirà altro risultato se non una tupla che soddisfi la semantica di tale operazione, nel caso in cui non sia presente nello spazio alcuna operazione accettabile essa si bloccherà in attesa; la duale non-bloccante della in e l'operazione `in_p` essa, a differenza della `in`, ha due risultati possibili a fronte della sua esecuzione, se esiste già una tupla che possa essere ritornata come risultato dell'interrogazione allora questa verrà restituita all'agente, nel caso in cui invece questa non sia presente tale operazione darà come risultato un esito negativo.

Appare evidente quindi come la prima `in` abbia una semantica bloccante che

le impedisce di terminare l'esecuzione fino a quando la tupla cercata non è reperita, mentre la seconda operazione `in_p` ha una semantica non bloccante: essa infatti nel caso in cui la tupla cercata non sia presente non si blocca ma prosegue la sua esecuzione ritornando un risultato negativo.

Il secondo livello in cui è possibile incontrare una differenza di questo tipo è invece nella comunicazione tra agente e centro di tuple ReSpecT: nel momento in cui un agente vuole interfacciarsi ad un centro di tuple deve infatti acquisire un *ACC* che gli permetta di eseguire operazioni sul centro. In ReSpecT esistono due tipi di *ACC*: *BlockingContext* e *NonBlockingContext*, tali contesti, come si intuisce dai nome delle classi che li implementano, definiscono che tipo di comunicazione deve esistere tra agente e centro. Nel caso in cui il contesto acquisito sia il *BlockingContext* la comunicazione tra agente e centro di tuple è di tipo bloccante, questo significa che indipendentemente dal tipo di operazione eseguita dall'agente questo si bloccherà in attesa del suo completamento; nel caso in cui invece il contesto sia il *NonBlockingContext* la comunicazione sarà di tipo non bloccante, questo fa si che l'agente al momento dell'invocazione dell'operazione passi un riferimento, attraverso tale riferimento l'agente verrà notificato nel momento in cui l'esecuzione dell'operazione verrà completata.

Porebbe sembrare che i due casi sopra descritti possano essere equivalenti, se però immaginiamo che un certo agente acquisisca il contesto non bloccante ed esegua un operazione bloccante sul centro, apparirà evidente come sia netta la differenza tra i due livelli comunicativi; in questo caso infatti nell'eventualità in cui non sia presente la tupla cercata l'agente non si bloccherà ma potrà preseguire il suo flusso, non appena la tupla verrà trovata esso verrà notificato direttamente all'agente.

Le operazioni ReSpecT

Tutte le operazioni invocabili da un agente esterno o da un artefatto sul centro di tuple presenti nell'insieme $\mathcal{O}^+$, visto anche precedentemente, sono rappresentate in ReSpecT con la classe *RespectOperation*. Tale classe non fa altro che bindare il tipo di operazione invocato dall'agente in una struttura dati più articolata. Tale meccanismo è di particolare importanza per la definizione formale delle operazioni che fanno parte del set di operazioni

possibili dalle altre, è inoltre a questo livello che viene esplicitato e registrato il riferimento all'agente che ha invocato l'operazione utile nel caso il contesto utilizzato sia non bloccante.

5.2 Limiti ed estensioni possibili

In questo capitolo è stato cercato di mostrare ed esplicitare i principali motivi per i quali sia corretto utilizzare un infrastruttura come TuCSoN per mappare ed implementare il modello MoK; è opportuno però valutare i limiti che questa infrastruttura possiede e le principali modifiche portabili al fine di diminuire il gap tecnologico/concettuale presente tra modello MoK - modello TuCSoN/ReSpecT e modello TuCSoN/ReSpecT - implementazione TuCSoN/ReSpecT. Cercheremo quindi di valutare modifiche inseribili per migliorare il mapping tra concetti MoK e astrazioni TuCSoN/ReSpecT e per migliorare come il modello TuCSoN/ReSpecT è stato implementato, vedremo infatti che alcuni concetti presenti nel modello di TuCSoN e ReSpecT non sono realmente presenti nella loro implementazione.

Per quanto riguarda ReSpecT possiamo vedere come il modello manchi di alcuni concetti fondamentali al fine di eseguire operazioni di tipo bulk o stocastiche. Se infatti volessimo cercare tutte le tuple presenti nello spazio che matchano un certo template e leggere/rimuoverle noteremo che non sarà possibile farlo con una singola primitiva ma che, se necessario, dovrà essere implementato ad alto livello una specifica scritta in questo linguaggio. Altro tipo di operazione mancante è quella per la quale sia possibile estrarre/leggere una tupla in maniera non deterministica dallo spazio.
Se pensiamo al modello che ci accingiamo ad implementare appare evidente come operazioni di questo tipo non solo siano utili ma possano diventare quasi indispensabili, infatti come verrà spiegato nel capitolo 4 saranno presenti numerose risorse da gestire in maniera bulk e sarà altresì necessario gestirne altre in maniera stocastica - algoritmo di scelta delle leggi da eseguire.
Possiamo dire quindi che le modifiche proposte a livello di ReSpecT vadano ad incidere esclusivamente il modello, dovranno essere inserite quindi nell'insieme delle operazioni $\mathcal{O}$ un nuovo set di primitive che abilitino operazioni con com-

portamenti analoghi a quelli descritti.
Le modifiche necessarie all'infrastruttura TuCSoN sono causate sia in conseguenza alle modifiche effettuare a ReSpecT che per via di alcune mancanze a livello implementativo di concetti ed astrazioni proprie del modello teorico di TuCSoN. Abbiamo ad esempio che:

- attualmente non è possibile istanziare più nodi sullo stesso host;
- la presenza degli ACC dovrebbe fornire un accesso strutturato e regolamentato ai centri di tuple, generando una gerarchia di ACC che specificano il tipo di operazioni eseguibili a seconda dell'ACC acquisito; l'implementazione attuale gestisce un solo ACC dal quale può essere invocata qualsiasi operazione;
- la comunicazione tra ACC lato agente e lato nodo (Proxy) non prevede la gestione di deadlock, che infatti si genera in seguito all'applicazione di alcuni pattern distribuiti.

Nel prossimo capitolo verranno quindi spiegate nel dettaglio le modifiche portate al fine di vedere come risultato una nuova e più completa release dell'inffrastruttura TuCSoN ReSpecT.

Chapter 3

Estensioni a TuCSoN e ReSpecT

Visto quanto preannunciato nell'ultima sezione del capitolo precedente, passeremo ora a descrivere le modifiche portate all'infrastruttura TuCSoN e a ReSpecT.

Le modifiche che vedremo sono state considerate a fronte di due requirement:

1. avere un infrastruttura implementata quanto più fedele al modello teorico proposto, facendo in modo cioè che ogni concetto descritto nel modello sia mappato nell'infrastruttura reale;
2. estendere l'infrastruttura attuale al fine di fornirne una versione più agevole per mappare il modello MoK su tale infrastruttura.

Vedremo quindi due tipi di estensioni, estensioni a livello di implementazione ed estensioni a livello, di modello e poi di implementazione. Per meglio distinguere tali estensioni apportate a TuCSoN e a ReSpecT seguiremo le modifiche fatte seguento le due tecnologie separatamente.

1 Le nuove primitive ReSpecT

Le primitive ReSpecT attualmente disponibili nel modello e nella sua implementazione non sono altro che tutte le operazioni presenti nell'insieme $\mathcal{O}^+$ visto a livello di modello, cioè l'insieme delle primitive di comunicazioni ammissibili $\mathcal{O} = \{$ `out, in, inp, rd, rdp , set, get, no, nop`$\}$ e il set di

operazioni di operazioni di reazioni ammissibili $\mathcal{O}_s = \{$`out_s, in_s, inp_s, rd_s, rdp_s, set_s, get_s, no_s, nop_s`$\}$.
In generale la lettura/rimozione di tuple è permessa solo adottando operazioni Linda-like, ciò rappresenta un vincolo nel momento in cui si ha la necessità di maneggiare/gestire grandi quantità di dati attravero paradigmi non standard.

In particolare esiste storicamente la necessità di gestire due tipi di modelli di fruzione dell'informazione particolarmente importanti:

1. la possibilità di estrarre/leggere una tupla che matchi un certo template scegliendola non deterministicamente tra quelle presenti che soddisfano il template passato in ingresso;
2. la possibilità di estrarre/leggere tutte le tuple presenti nel centro che matchino un certo template.

Gestire tale necessità significa aumentare l'espressività del linguaggio ReSpecT attraverso un nuovo set di primitive di comunicazione che abilitino questi due nuovi modelli di comunicazione con il centro di tuple.
L'importanza di tali estensioni sta nel fatto che storicamente per sopperire a tali mancanze era necessario creare specifiche ad alto livello che attraverso le operazioni standard a disposizione - quelle del set $\mathcal{O}^+$ - dessero come risultato un comportamento analogo a quello richiesto, spostando quindi l'ownership del problema non più a livello di infrastruttura ma a livello di agente. d

1.1 Operazioni Bulk

Le operazioi di tipo *Bulk* sono quelle operazioni che danno come risultato tutte le tuple presenti nello spazio che matchino un certo template specificato.
In particolare aggiunte da tale estensione sonole seguenti:

- `in_all{ Tin, Lout }`;
- `rd_all{ Tin, Lout }`;
- `no_all{ Tin, Lout }`.

Com'è evidente, tutte le operazioni di tipo *Bulk* hanno due parametri. In generale il primo rappresenta il parametro di ingresso, e il secondo quello

di uscita. La differenza sostanziale tra queste operazioni e le classiche operazioni già presenti nei due set è che tali operazioni non devono dare come risultato una singola tupla che unifichi con il tamplate passato in ingresso ma in generale devono ritornare una lista di tuple tutte unificanti con il template passato. Da tale trattazione potrebbe sembrare che non sia possibile avere solo una tupla come risultato dell'esecuzione di un operazione *Bulk*, ciò è naturalmente errato, nel caso infatti si abbia una singola tupla come risultato dell'esecuzione questa sarà comunque l'unico elemento presente nella lista delle tuple di ritorno.

Passando ora alla descrizione delle operazioni sopra elencate descriveremo per prima cosa il significato dei due parametri: il primo parametro *Tin* rappresenta il tuple template d'ingresso che deve matchare le tuple cercate, il secondo parametro *Lout* invece rappresenta la lista di tuple ottenute dal match del template.

Per quanto riguarda la **semantica delle operazioni** invece abbiamo un mapping abbastanza intuitivo tra la loro sintassi ed il loro significato/comportamento: `in_all` e `rd_all` rappresentano rispettivamente le due operazioni di rimozione e lettura di tuple, esse, analogamente alla `in` e alla `rd`, rimuovono o leggono tutte le tuple presenti nel centro di tuple che matchino il template passatogli in ingresso. A differenza delle operazioni standard, $\mathcal{O}$, però le *in_all* e la *rd_all* non sono bloccant, infatti: se sono presenti tuple che matchino il template indicato queste vengono inserite nel secondo parametro sotto forma di lista di uscita, nel caso in cui invece non sia presente alcuna tupla che matchi viene restituita una lista vuota. È molto importante individuare le caratteristiche di tale semantica, innanzi tutto tali operazioni non si bloccheranno mai in attesa di tuple che soddisfino il template; in questo senso potremmo quindi vederle come operazioni non bloccanti, che nonostante l'esito della ricerca tornano comunque un esito positivo sotto forma di lista vuota o lista popolata.

La semantica dell'operazione `no_all` è sostanzialmente inversa a quella delle due operazioni sopra descritte: essa restituisce infatti tutte le tuple presenti nello spazio che non matchano il template in ingresso; come le precedenti ha sempre successo e ritorna una lista vuota nel caso tutte le tuple matchino il

template o una lista popolata da tutte le tuple che violano tale template.

1.2 Operazioni Stocastiche

Le operazioni di tipo *stocastico* sono quelle operazioni che danno come risultato, ammesso che esso sia presente, una tupla scelta in maniera non deterministica dal centro di tuple.
Il modello stardard di selezione delle tuple presente in ReSpecT ha una logica ben precisa; nonostante infatti tale concetto non venga trattato normalmente, il modello utilizzato per cercare tuple che matchino un certo template non è casuale ma gestito attraverso una lista Java. Questo significa che ogni qualvolta venga invocata un'operazioni di lettura/rimozione di una tupla questa viene cercata nella lista di tuple che popolano il centro attraverso una politca FIFO.
Questo modello di gestione delle tuple toglie il concetto di casualità utilie in molti paradigmi di coordinazione implementabili attraverso tale infrastruttura. Per questo motivo sono state introdotte le seguenti operazioni *stocastiche* che portano il concetto di stocasticità nell'ambito della comunicazione tra agente e centro di tuple.
Le operazioni inserite sono:

- `uin( Templ );`
- `urd( Templ );`
- `uinp( Templ );`
- `urdp( Templ );`
- `uno( Templ );`
- `unop( Templ );`

Come possiamo notare in questo caso le operazioni inserite hanno un unico parametro rappresentante sia tuple template che tupla di ritorno; questo porta ad una sua semantica analoga a quella delle operazioni standard, nelle quali a fronte di una tupla che matchi il template questa verrà unificata con tale parametro.

La **semantica delle operazioni** in questo caso è presto detta, le prime due infatti `uin` e `urd` rappresentano la forma *stocastica* delle operazioni `in` e `rd`, in questo caso ciò che avviene è la garanzia che la tupla venga letta/rimossa in maniera casuale e con un paradigma bloccante. Quindi possiamo dire che le prime quattro operazioni dell'elenco sopra mostrato hanno una semantica identica alle operazioni presenti nell'insieme stardad di operazioni con in più la garanzia di avere un comportamento stocastico.
Per quello che riguarda le ultime due operazioni invece abbiamo che sia la `uno` che `unop` hanno successo se non sono presenti tuple che matchino il template passato in ingresso, entrambe, nel caso sia presente una tupla che matchi tale template, falliscono, nel caso invece non sia presente almeno una tupla che matchi hanno successo. In particolare la prima operazione, essendo non predicativa, è bloccante mentre la seconda non lo è. Altra differenza fondamentale tra le due operazioni sta nel paradigma utilizzato per ritornare le tuple esempio che fanno falllire la ricerca: nel caso della `uno` la tupla viene rimossa attraverso il modello standard ReSpecT mentre nel caso della `unop` la tupla viene rimossa in maniera stocastica tra tutte quelle che unificano il template e che fanno fallire l'operazione.

Al fine di descrivere meglio la semantica delle operazioni inserite è stato introdotto un **nuovo formalismo** per rappresentarla in maniera quanto più chiara, essa si basa sulla grammatica standard utilizzata per descrivere la sintassi degli operatori base presenti in ReSpecT. In particolare in tale grammatica è presente lo scopo TCCycleResult che rappresenta il risultato dell'esecuzione di un operazione sul centro di tuple. Rispetto a tale elemento presente nella grammatica standard ne è stata esteso il significato per poter rappresentare anche informazioni riguardanti la semantica delle operazioni. Tale lavoro è ha dato come risultato tale specifica:

Sintassi:

TCCycleResult ::= OpResult, TupleResult, TupleListResult, StartTime, EndTime
OpResult ::= success | failure | undefined
TupleResult ::= tuple | tupleTemplate | ⊥
TupleListResult ::= {tuple}

Semantica:

out $\longrightarrow$ *TCCycleResult = [success, tuple, null]*
in = uin = rd = urd $\longrightarrow$ *TCCycleResult = [success, tuple, null]*
inp = uinp = rdp = urdp $\longrightarrow$ *TCCycleResult = [success / failure, tuple / tupleTemplate, null]*
no = uno $\longrightarrow$ *TCCycleResult = [success, tupleTemplate, null]*
nop = unop $\longrightarrow$ *TCCycleResult = [success / failure, tupleTemplate / tuple, null]*
in_all = rd_all = no_all $\longrightarrow$ *TCCycleResult = [success, tupleTemplate, {tuple}]*
get = set $\longrightarrow$ *TCCycleResult = [success, null, {tuple}]*

1.3 Nuovo Set disponibile

L'introduzione di nuove operazioni avrà un triplice effetto sull'infrastruttura ReSpecT:

1. sarà possibile invocare tali operazioni di comunicazione sul centro di tuple;
2. sarà possibile attivare delle reazioni a fronte dell'esecuzione di tali operazioni;
3. sarà possibile inserire nel body di una reazione questo nuovo set di primitive.

Come risultato finale di tale estensione è ora possibile definire un nuovo set di operazioni $\mathcal{O}_o = \{$`rd_all, in_all, no_all, uin, urd, uinp, urdp, uno, unop`$\}$; questo nuovo set, insieme ai due già presenti $\mathcal{O}$ e $\mathcal{O}_s$ vanno a creare un nuvo insieme di operazioni $\mathcal{O}^{++}$:

$$\mathcal{O}^{++} = \mathcal{O} \cup \mathcal{O}_s \cup \mathcal{O}_o$$

2 Un nuovo TuCSoN

A seguito di quanto brevemente detto sono state apporta a TuCSoN quattro modifiche sostanziali, come detto alcune a seguito di modifiche al modello ReSpecT, altre a seguito di mismatch tra modello e implementazione e altre date da problemi presenti e non risolti.

2.1 Più nodi, un host

Prima modifica tecnologica portata all'infrastruttura TuCSoN è quella di poter istanziare più nodi all'interno dello stesso host. Tale estensione porta numerosi vantaggi a livello architetturale permettendo di creare reti a topologia non-banala anche in uno stesso punto fisico.
Un estensione di questo tipo ha generato un side-effect a livello sintattico nel momento in cui si vuole invocare una operazione specificando nodo e centro di tuple; originariamente infatti nel momento in cui si voleva specificare il percorso in cui era localizzato il centro di tuple sul quale eseguire l'operazione era sufficiente la seguente sintassi: *tcname @ ipAddress*, dove *tcname* rappresenta il nome del centro di tuple e *ipAddress* l'indirizzo in cui era istanziato il nodo. A fronte però delle modifiche portate la nuova sintassi è divenuta:

tcName @ ipAddress : port

in tale contesto è stato quindi inserito il concetto di porta, essa rappresenta la porta logica sulla quale è stato attivato il servizio nodo sull'host già specificato.

2.2 Gerarchia di ACC

La presenza dei centri di tuple dovrebbe servire sopratutto a garantire che: un agente sia in grado di eseguire operazioni sul centro di tuple e che la gestione degli accessi alle primitive sia gestita in maniera strutturata. Si intende per gestione strutturata una gestione che permetta di suddividere in set precisi le operazioni che ogni agente può eseguire sul centro. L'utilità di tale meccanismo è garantire che, ad esempio, solo certi agenti possano modificare le specifiche presenti del centro di tuple, o che solo certi agenti/utenti possano eseguire particolari "interrogazioni" sul centro.
In questo modo è possibile gestire gli agenti attraverso una sorta di dominio nel quale ogni agente ha assegnato un certo account con specifici privilegi e permessi.
Avendo chiarito quali siano i set di operazioni applicabili lato agente vedremo ora la struttura introdotta per gestire la gerarchia degli ACC.

La struttura gerarchica proposta quindi prevede i seguenti ACC:

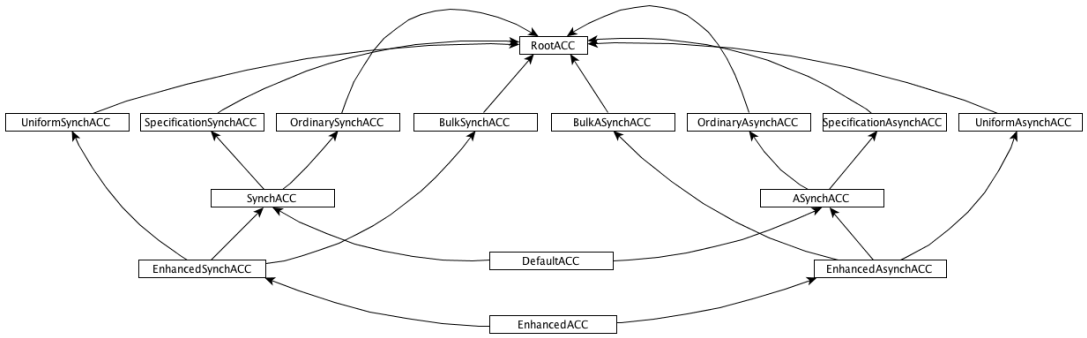


Figure 3.1: Gerarchia ACC

- UniformSynchACC e UniformAsynchACC, per rappresentare gli ACC riguardanti le nuove operazioni stocastiche viste precedentemente;
- SpecificationSynchACC e SpecificationAsynchACC, per rappresentare gli ACC adibiti alla gestione delle tuple di specifica;
- OrdinarySynchACC e OrdinaryAsynchACC, per rappresentare gli ACC che abilitano l'utente ad eseguire le operazioni base (per internderci quelle presenti nel set $\mathcal{O}$);
- BulkSynchACC e BulkAsynchACC, sono un estensione del set di operazioni che includono anche il set di operazioni bulk prima identificate con il set $\mathcal{O}_o$;

Possiamo dire che questi sono gli ACC base, da questi vengono poi creati dei macro ACC che raggruppano le operazioni di più sotto set.
L'ACC acquisito il quale un agente ha modo di eseguire qualsiasi operazione è l'EnhancedACC, mentre l'ACC standard presente anche prima dell'inserimento delle nuove operazioni bulk e stocastiche è il DefaultACC. Possiamo quindi vedere come sia estremamente più agevole ed intuitivo gestire l'associazione tra agente-operazioni permesse attraverso una struttura di questo tipo.
A livello di codice avremo un mapping analogo tra la gerarchia degli ACC mostrati e la gerarchia delle classi create. In tale contesto possiamo vedere il RootACC come "mera classe di partenza" dalla quale partire per definire poi successivamente tutte le altre.
In tale descrizione non si è parlato della differenza tra la versione sincrona e quella asincrona degli ACC e quindi anche delle operazioni da essi gestite, questo verrà introdotto dal prossimo paragrafo.

2.3 Comunicazione tra *ACC - Proxy - Centro di tuple*

Una qualsiasi operazione presente nel set $\mathcal{O}^{++}$, può avere un comportamente sincrono o asincrono (come si può anche vedere dalla divisione effettuata sugli ACC. Tale divisione è indipendente dal tipo di operazione invocata sul centro e riguarda solamente l'aspetto sincrono o asincrono dell'agente rispetto all'attesa del risultato e non la semantica dell'operazione (vedere capitolo precedente per chiarimenti).
Un flusso di comunicazione generato a fronte della richiesta di operazione da parte di un agente era precedentemente costituita da tale processo: l'agente, acquisito un ACC, invocava un'operazione permessa da tale interfaccia, tale richiesta giungeva al proxy che a sua volta acquisiva un nuovo ACC (lato centro di tuple, *dello stesso tipo dell'ACC acquisito dall'utente*) per comunicare con il tc attraverso il quale si interfacciava con esso.
Gestire la comunicazione tra proxy e tc attraverso un'ulteriore ACC, per giunta dello stesso tipo di quello acquisito dall'agente, è un errore implementativo non presente nel modello originale di TuCSoN. Tale gestione può generare situazioni di deadlock nel momento in cui il proxy, bloccato per via di una richiesta gestita in maniera asincrona, riceve unl'ulteriore richiesta dall'ACC.

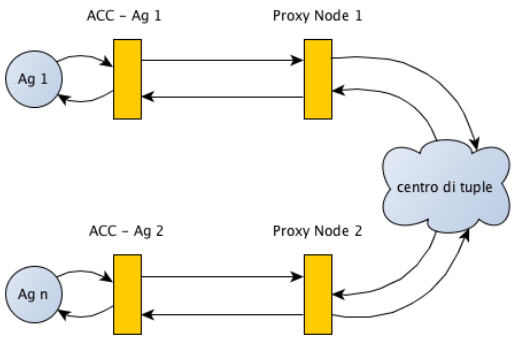


Figure 3.2: Comunicazione Agente - Tuple Centre in TuCSoN

Per eliminare tale problematica è stata inserita una nuova logica di comunicazione tra agente e tc, in particolare: l'agente acquisisce uno degli ACC sopra visti, attraverso esso invia una richiesta sincrono o asincrona di operazione, tale richiesta giungerà al proxy il quale non dovendo più occuparsi

di acquisire un ACC, comunicherà con il tc attraverso un pattern asincrono indipendentemente. Tale modifica, seppur semplice, elimina definitivamente la possibilità di deadlock.

2.4 Esecuzione remota di primitive

Ciò che fino a questo momento era presente nel modello concettuale ma non nella sua implementazione è la possibilità di eseguire primitive su centri di tuple remoti (diversi da quello locale). La sintassi precedentemente mostrata nella quale si specificava nodo, centro di tuple e operazione da eseguire è stata fin'ora un concetto solo formale mai realmente esistente/funzionante.
Quello che è stato fatto ha riguardato quindi l'implementazione di quanto visto come prima modifica portata al sistema cioè abilitare la possibilità di invocare operazioni attraverso la sintassi: *tcname @ ipAddress : port*. Tale possibilità è stata messa a disposizione non solo all'utilizzo dell'agente ma anche nel body di una reaction. Tale modifica renderà molto più verosimile l'implementazione del motore chimico (descritto nel prossimo capitolo) inquanto abilita la possibilità di un atomo/molecola di diffondere in un compartimento vicino.

Oltre alle modifiche qui portate ne sono sta aggiunte molte altre, sono state qui riportate solo quelle significative ai fini dello scopo di tale lavoro.

Chapter 4

Il Motore Chimico

Lo scopo principale di questo elaborato di tesi è di analizzare la proposta attualmente presente di implementazione del modello MoK, valutando tecnologie scelte implementative al fine di proporre nuove soluzioni e approcci al fine di generare un'infrastruttura più robusta e di diminuire il gap concettuale - di cui abbiamo parlato all'inizio - tra astrazioni proposte dal modello e proposte infrastrutturali offerte dalle tecnologie scelte.

Abbiamo a tal fine descritto le due principali infrastrutture su cui si basa l'implementazione di MoK: TuCSoN *e* ReSpecT, esse infatti si occuperanno dello strato più basso dell'architettura del sistema MoK, cioè la comunicazione con i centri di tuple programmabili e l'architettura distribuita del sistema. Fissati tali livelli infrastrutturali è ora necessario introdurre l'astrazione che permette alle due tecnologie utilizzate di simulare un vero e proprio sistema chimico e il mezzo utilizzato per itegrare questi due sistemi.

Il modello MoK basa tutte le sue astrazioni su modelli di coordinazioni biochimici, ciò sottintende che siano disponibili meccanismi informatici che rendano possibile la simulazione di un ambiente di questo tipo (dando per scontato l'aspetto insfrastrutturale distribuito reso possibile da TuCSoN).

1 L'algoritmo di Gillespie

Esistono sostanzilamente due formalismi matematici per descrivere un sistema chimico spaziale omogeneo: l'*approccio deterministico* che considera l'evoluzione del tempo come continua, un processo completamente predicibile governato da una coppia da un insieme di coppie di equazioni differenziali (le reaction-rate equation); l'*approccio stocastico* invece considera l'evoluzione del tempo come un processo composto di passi random governato da una semplice equazione differenziale (la "master-equation").
La formulazione stocastica utilizzata per descrivere la cinetica di sistemi chimici è basata su un modello più vicino alle basi fisiche rispetto a quella deterministica, malgrado ciò la master-equation dell'approccio stocastico ha una complessita molto maggiore ed è spesso intrattabile.
Nonostante ciò esiste un metodo per eseguire il calcolo esatto all'interno del framework stocastico senza dover affrontare la risoluzione della master-equation.

Consideriamo un volume V contenente una soluzione uniforme di N speci chimiche che interagiscono attraverso la presenza di M specifiche di canali di reazioni chimiche, quindi

dato il numero di molecole di ogni specie presente ad uno stato iniziale quale sarà il livello di popolazione delle molecole ad ogni istante successivo?

1.1 L'approccio tradizionale

L'approccio tradizionale a tale problema lo tradurrebbe nel linguaggio matematico delle equazioni differenziali ordinarie, più precisamente se assumiamo che le molecole di una certa specie i_esima in V al tempo t può venire rappresentata da una funzione continua, ad un valore $X_i(t)$ $(i=1,...,N)$, e se consideriamo anche che ognuna delle M reazioni chimiche può essere considerata come un processo a rate continuo, allora è possibile costruire facilmente un set di coppie del primo ordine di equazioni differenziali ordinarie della forma:

$\mathrm{d}X_1/\mathrm{d}t = \mathrm{f}_1(X_1,...,X_N)$
$\mathrm{d}X_2/\mathrm{d}t = \mathrm{f}_2(X_1,...,X_N)$

.....
$\mathrm{d}X_N/\mathrm{d}t = \mathrm{f}_N(X_1,...,X_N)$

Queste equazioni vengono chiamate *reaction-rate*, risolte queste per le funzioni $X_1(t)$,...,$X_N(t)$ soggette alle condizioni iniziali descritte precedentemente è equivalente a risolvere il problema dell'evoluzione temporale posto precedentemente.
Il problema principale dato da questo approccio alle equazioni differenziali per simulare la cinetica di un sistema chimico si basa su due presupposti troppo labili che si distaccano dal modello fisico reale, questo approccio assume infatti che: l'evoluzione temporale di un sistema chimico sia *continuo* e *deterministico*. Entrambi i presupposti su cui si basa queso modello sono errati infatti: l'evoluzione temporale di un sistema chimico di reazioni non è un processo continuo inquanto il livello di popolazione delle molecole può variare solo ad intervalli discreti di tempo inoltre l'evoluzione rispetto al tempo non è un processo deterministico. Oltre a tutto questo va aggiunto l'impossibilità di predirre il numero esatto di molecole che popoleranno il sistema in un istante di tempo futuro senza conoscere dati esatti come la posizione e la velocità di tutte le molecole appartenenti al sistema.
In altre parole l'evoluzione di un sistema chimico può essere considerato un processo deterministico solo se nel caso in cui si compiano specifiche semplificazioni al modello riguardo sopratutto il numero di molecole e la conoscenza di informazioni come velocità e posizione; nel caso in cui si voglia considerare un modello quanto più generale ed applicabile ad una moltitudine di casi diversi è necessario approssimare il sistema ad un processo *non deterministico*.

1.2 Formulazione stocastica della cinetica chimica

Una reazione chimica ha luogo ogni volta che due o più molecole, di specifici tipi, collidono in uno specifico modo.
La *formulazione stocastica* della cinetica chimica è semplicemente una conseguenza del fatto che: *in un sistema chimico nel quale sono presenti molecole di diverso tipo ed in equilibrio chimico, accadono in maniera non deterministica - random.*

Collisioni tra molecole: consideriamo un sistema composto da due gas S_1 e S_2 in equilibrio termico posti all'interno di un certo volume V. Per semplicità consideriamo inoltre che le molecole che costituiscono i due composti siano assimilabili a sfere rigide di raggi r_1 e r_2 rispettivamente. Avremo quindi che una collisione tra due molecole dei due composti avverrà non appena la distanza tra le due sfere/molecole non sarà inferiore a $r_{12} = r_1 + r_2$. Quello che vorreo fare è quindi cercare di riuscire a calcolare il rate al quale tali collisioni accadono all'interno di V. Consideriamo ora due molecole 1 e 2, avremo che v_{12} sarà la velocità della molecola 1 rispetto alla molecola 2; osserviamo inoltre che se nel prossimo intervallo temporale δt la molecola 1 viene spazzata via relativamente al volume di collisione calcolato rispetto alla molecola 2 - $\delta V_{coll} = \pi r_{12}^2 * v_{12}\ \delta t$ - allora le due molecole hanno colliso nell'intervallo temporale (t, $t + \delta T$). La procedura classica prevede che si consideri il numero di molecole di S_2 il cui centro stia all'interno di δV_{coll}, si divida tale numero per δt e se ne faccia il limite tale che $\delta t \rightarrow\ 0$ per ottenere il rate al quale le molecole di S_1 collidono con le molecole S_2. Questa procedura però soffre di un grosso difetto, infatti nel momento in cui il volume di collisione $\delta V_{coll} \rightarrow\ 0$ la probabilità che una molecola S_2 stia al suo interno sarà 0 o 1. Essendo il sistema in equilibrio termico, avremo che le molecole saranno, per tutto il tempo, distribuite in maniera *random* e *uniforme* all'interno del volume V. Perciò la probabilità che il centro di una molecola arbitraria S_2 si all'interno del volume di collisione δV_{coll} nell'istante t è data dal semplice rapporto di $\delta V_{coll}/ V$, questo sarà vero anche al diminuire del volume di collisione.

Quello che appare evidente è l'mpossibilità di calcolare rigorosamente il numero di collisioni 1-2 che accadono in V in un intervallo infinitesimo di tempo; l'unica cosa che siamo in grado di calcolare è la *probabilità* che una collisione 1-2 accada in V in ogni intervallo infinitesimo di tempo. Questo è il motivo per il quale le collisioni in un sistema chimico rappresentano un processo stocastico Markoviano e non un processo a rate deterministico, questo significa che dovremmo caratterizzare il sistema di molecole in equilibrio chimico attraverso una *probabilità di collisione per unità di tempo* - questo sarà il coefficiente $\mathbf{d}t$.

Fin'ora abbiamo valutato la probabilità che due molecole collidano solo at-

traverso considerazioni spaziali - la distanza tra i due centri delle molecole -; consideriamo però che il volume V contenga una mistura spazialmente omogenea di molecole X_i di specie S_i (i = 1,...,N), e supponiamo inoltre che queste N specie possano interagire attraverso M specifici canali chimici $\mathbf{R}_\mu$ (μ = 1,...,M). Possiamo allora affermare l'esistenza di M costanti c_μ (μ = 1,...,M), che dipendono solo dalle proprietà fisiche delle molecole e dalla temperatura del sistema, tale che: $\boldsymbol{c}_\mu\ d\boldsymbol{t}$, è la probabilità media che una particolare combinazione $\mathbf{R}_\mu$ di molecole reagenti reagiranno insieme nel prossimo intervallo infinitesinale dt.
Consideriamo tale probabilità come "media" inquanto, se eseguiamo la moltiplicazione $c_\mu\ dt$ per il numero totale di combinazioni distinte di molecole reagenti $\mathbf{R}_\mu$ presenti in V all'istante t, otterremo la probabilità che una reazione $\mathbf{R}_\mu$ avvenga da qualche parte all'interno del volume V nel prossimo intervallo infinitesimo (t, t + dt).
Quest'ultima equazione può essere considerata sia come la definizione della *costante di reazione stocastica* c_μ, e sia come *ipotesi fondamentale* della formulazione stocastica della cinetica chimica.

Calcolo del tempo evolutivo in maniera stocastica

Il comportamento temporale di una soluzione spazialmente omogenea di N specie di molecole che interagiscono attraverso M canali chimici di reazione è governata solo attraverso l'*ipotesi fondamentale* vista precedentemente. Non è del tutto ovvio però come tale formula debba essere utilizzata al fine di calcolare questo comportamento temporale. Esistono sostanzialmente due approcci per affrontare questo problema: l'approccio tradizionale *master equation* e il nuovo approccio di *simulazione stocastica*; va comunque tenuto sempre in mente come entrambi gli approcci, nonostante siano completamente differenti, siano dirette conseguenze dell'*ipotesi fondamentale*.

Master Equation

L'approccio della *Master Equation* è la tecnica tradizionale utilizzata per calcolare l'evoluzione temporale stocastica di un sistema chimico reattivo e si basa sulla risoluzione di una *master equation* collegata al sistema.

Il punto fondamentale su cui si basa il formalismo della *master equation* è la *grand probability function*:

$P(X_1, X_2, ..., X_N; t)$

essa rappresenta la probabilità che ci saranno in v all'istante t X_1 molecole della specie S_1 e X_2 molecole della specie S_2, e così via X_N molecole della specie S_N. La conoscenza di tale funzione vuole evidentemente fornire una caratterizzazione completa dello stato stocastico del sistema all'istante t.
La *master equation* diventa quindi semplicemente l'evoluzione temporale della funzione $P(X_1, X_2, ..., X_N; t)$. Essa può esseere calcolata dall'*ipotesi formale* utilizzando le leggi di moltiplicazione ed addizione della teoria delle probabilità per scrivere $P(X_1, X_2, ..., X_N; t + dt)$ come la somma delle probabilità delle $M+1$ differenti strade attraverso le quali il sistema può arrivare allo stato $(X_1, X_2, ..., X_N)$ all'istante $t + \mathrm{d}t$.
Non scendendo ulteriormente nei dettagli di questa soluzione "classica" possiamo dire che essa rappresenta un approccio non soddisfacente al problema che stiamo affrontando, in particolare essa non è in grado di risolvere qualsiasi tipo di sistema in maniera analitica (il numero di sistemi che non può risolvere sono più di quelli in cui riesce), inoltre il calcolo della soluzione dell'equazione così ricavata non è per niente banale.

Reaction Probability Density Function

La *Reaction Probability Density Function* invece cerca di valutare l'evoluzione stocastica del sistema chimico reattivo. Ipotizziamo di avere il sistema nello stato $(X_1, ..., X_N)$ all'istante t, per poter rispondere alla necessità di "poter muovere il sistema avanti nel tempo" è necessario poter far fronte a due problematiche: *quando si verificherà la prossima reazione ?* e *quale tipo di reazione sarà ?*. Essendo il processo studiato di tipo stocastico, ci si aspetta che la risposta che si può dare a tale domanda sia basata solo su valori probabilistici.
Partendo da tale presupposto introduco il concetto di funzione $\mathrm{P}(\tau, \mu)$ definita da:

$P(\tau, \mu)d\tau$

come la probabilità che, dato lo stato $(X_1, ..., X_N)$ all'istante t, la prossima reazione in V avverrà nell'intervallo temporale infinitesimo $(t+\tau, t+\tau+d\tau)$, e sarà una reazione di tipo R_μ.
Tale funzione di probabilità è denominata *reaction probability density function*, infatti in terminologia matematica essa rappresenta una funzione di densità di probabilità congiunta nello spazio delle variabili continue τ $(0 \leq \tau < \infty)$ e la variabile discreta μ $(\mu = 1, 2, ..M)$. Va notato come le due variabili τ e μ sono quantità il cui valore rappresesenta la risposta alle due domande precedentemente poste. Punto centrale di tale processo è dato quindi dal'assegnazione di valori a tali variabili, il primo passo sarà quindi quello di derivare tali valori partendo dall'*ipotesi fondamentale* per derivare poi l'espressione analitica di $\mathrm{P}(\tau, \mu)$.
A tal fine partiamo definendo per ogni reazione R_μ una funzione h_μ definita come il numero di molecole reagenti R_μ distinte presenti nello stato $(X_1, X_2, ..., N_N)$ $(\mu = 1, ..., M)$. Quindi se R_μ è nella forma $S_1 + S_2 \rightarrow$ "qualsiasi cosa", allora avremo $h_\mu = 1/2X_1(X_1 - 1)$. In generale h_μ sarà una funzione combinatoria delle variabili $X_1, X_2, .., X_N$; avendo definito h_μ possiamo definire:

$$a_\mu dt \equiv h_\mu c_\mu dt$$

come la probabilità che una reazione R_μ si verifichi in V in $(t, t+dt)$, dato che il sistema parta nello stato $(X_1, ...; X_N)$ all'istante t $(\mu = 1,..,\mathrm{M})$. A questo punto possiamo calcolare la probabilità $P(\tau, \mu)d\tau$ come il prodotto di: $P_0(\tau)$, probabilità che dato lo stato $(X_1, ..., X_N)$ all'istante t, nessuna reazione avverrà nell'intervallo temporale $(t, t+\tau)$; e l'stante $a_\mu d_\tau$, come la probabilità conseguente che la reazione R_μ avvenga nell'intervallo temporale $(t+\tau, t+\tau+d\tau)$, avremo quindi:

$$P(\tau, \mu)d\tau = P_0(\tau) * a_\mu d\tau$$

Per trovare un'espressione per $P_0(\tau)$, va innanzi tutto notato che [1 - $\sum_\nu a_\nu d\tau'$] è la probabilità che nessuna reazione avverrà nell'intervallo $d\tau'$ dallo stato $(X_1, .., X_N)$, perciò si avrà che:

$$P_0(\tau' + d\tau') = P_0(\tau')[1 - \textstyle\sum_{\nu=1}^{M} a_\nu d\tau']$$

per la quale viene dedotto che:

$P_0(\tau) = exp[-\sum_{\nu=1}^{M} a_\nu \tau]$

Inserendo 1.2 nella 1.2, possiamo concludere affermando che la funzione di densità di probabilità delle reazioni definita in 1.2 è data da:

$$P(\tau, \mu) = \begin{cases} a_\mu \ exp(-a_0 \tau) & \text{, se } 0 \leq \tau < \infty \text{ e} \\ 0 & \text{, altrimenti} \end{cases}$$

dove:
$a_\mu \equiv h_\mu c_\mu \ (\mu = 1, ..., M)$
e
$a_0 \equiv \sum_{\nu=1}^{M} a_\nu \equiv \sum_{\nu=1}^{M} h_\nu c_n u$

Va notato come tale tecnica sia tanto rigorosa quanto la *master equation*, conseguenza dell'*ipotesi fondamentale* vista precedentemente, va infatti notato come $P(\tau, \mu)$ dipenda da tutte le reazioni costanti e sui membri correnti delle molecole di tutte le specie reagenti.

1.3 Algoritmo di simulazione stocastica

Nella precedente sezione è stato osservato che ciò che occorre per simulare l'evoluzione temporale di un sistema chimico sia sostanzialmente una tecnica per specificare quando avverrà la prossima reazione e di quale tipo essa sarà. Possiamo, a questo punto, formalizzare maggiormente tale requiremente attraverso una visione matematica: ci serve un metodo per prelevare una coppia (τ, μ) dal set di coppie la cui funzione di probabilità è $P(\tau, \mu)$.
Genrare un algoritmo significa, in questo contesto, trovare un metodo automatico che possa essere gestito anche da un calcolatore; in particolare sarà dato per assodata la disponibilità di un generatore di numeri random.
Il nostro obiettivo è quello di generare una coppia random di numeri (τ, μ) in accordo alla funzione di probabilità :

$$P'(\tau, \mu) = \begin{cases} 1 & \text{, se } 0 \leq \tau \leq 1 \text{ e } 0 \leq \mu \leq 1 \\ 0 & \text{, altrimenti} \end{cases}$$

quindi possiamo generare semplicemente due numeri random r_1 e r_2 utilizzando un generatore random di numeri e prendere: $\tau = r_1$ e $\mu = r_2$.

Nonostante questo il nostro obiettivo è quello di generare una coppia random (τ, μ) in accordo alla funzione di densità di probabilità; esiste infatti una procedura formale per prendere due numeri random $r_1 e r_2$ dall'intervallo unitario della distribuzione uniforme e, costruire da esso, una coppia random (τ, μ) da un set descritto da ogni coppia di probabilità specificata dalla funzione di densità.
L'algoritmo per simulare l'evoluzione temporale stocastica di un sistema reattivo chimico peò venire oral formalizzato nel seguente flusso di step:

1. (*inizializzazione*). Inserire i valori desiderati per i valori delle M costanti delle reazioni $c_1, ..., c_M$ e il numero iniziale N di molecole che popolano il sistema $X_1, ..., X_N$. Impostare la variabile tempo te il contatore delle reazioni n a zero. Inizializzare il generatore di numeri random;
2. Calcolare e scrivere le M quantità $a_1 = h_1 c_1, ..., a_M = h_M c_M$ per il numerdo della popolazione corrente di molecole, dove h_ν è questa funzione $X_1, ..., X_N$ definita precedentemenete. Inoltre si calcoli e si scriva come a_0 la somma degli M valori $a\nu$;
3. Generare due numeri random $r_1 e r_2$ e calcolare $\tau e \mu$ in accordo alle regole viste precedentemente;
4. Usare i valori $\tau e \mu$ ottenuti allo step precedente, incrementare t di τ , e aggiustare il livello di popolazione delle molecole per riflettere l'occorrenza di una reazione R_μ; e.g., se R_μ è la reazione triggerata , allora vanno decrementati $X_1 e X_2$ di 1 e il contatore delle reazioni n va incrementato di 1, poi si può tornare allo step 1.

Valutazioni dell'approccio stocastico

L'approccio stocastico, descritto in questa sezione, di simulazione di evoluzione temporale di un sistema chimico ha numerosi vantaggi, rispetto all'approccio della *master equation*, tra i quali:

1. La simulazione stocastica è esatta, in questo senso si intende che tiene in considerazione delle fluttuazioni e correlazioni implicate nell'*ipotesi fondamentale*. Questa è una conseguenza data dal fatto che, come la

spazio omegenea *master equation*, l'algoritmo di simulazione stocastica è derivato dall'ipotesi fondamentale attraverso un procedimento matematicamente formale;

2. Diversamente dal metodo numerico che risolve equazioni differenziali, l'algoritmo di simulazione stocastica non approssima mai l'incremento infinitesimale dt da uno step finito $\Delta\ t$. Questo porta i vantaggi maggiori quando ci si trova a trattare con sistemi nei quali la il livello di popolazione delle molecole può cambiare improvvisamente e nettamente nel tempo;
3. L'implementazione e l'esecuzione di un algoritmo software che implementi tale procedura non richiede l'utilizzo di una grande quantità di memoria. Per N specie di molecole e M canali reattivi, le variabili principali che devono essere scritte sono: NX_i valori, Mc_nu valori e $M+1a_\nu$, per un totale di $N + 2M + 1$;
4. In termini di "isieme statistico dei sistemi" visto nell'approccio della *master equation*, l'algoritmo di simulazione stocastica ci fornisce essenzialmente di informazioni sul comportamento temporale dei singoli individui che popolano l'esempio, nella stessa maniera in cui è possibile osservarlo in un esempio reale in laboratorio;
5. Quando la media d'insieme non può essere calcolata dalla master equation, l'approccio di simulazione stocastica offre un chiaro approccio numerico per stimare tale valore.

2 Gillespie per MoK

Nell'ambito dei modelli di coordinazione basati su *Biochemical Tuple Space for Self-Organasing Coordination* è di centrale importanza ricordare come lo stato del sistema evolva, in accordo al modello biochimico a cui si ispira, riflettendone il cambiamento nella concentrazione associata a ciascuna tupla che popoli il sistema.
La *concentrazione*, misura l'attività della tupla a cui è associato e ne determina la rilevanza che avrà all'interno del sistema. La possibilità di avere un valore di *concentrazione* associato ad ogni tupla fa si che sia possibile applicare modelli stocastici all'evoluzione del sistema e di far evolvere lo stato del

sistema attraverso regole di ispirazione biochimica. Di particolare importanza in questo ambito è il lavoro fatto da Gillespie, il quale è riuscito a proporre un modello formale in grado di simulare l'evoluzione di un sistema biochimico attraverso l'utilizzo di un sistema di *Continous-Time Markov Chains*; esso infatti attraverso l'*algoritmo di Gillespie* ha proposto un estensione alle *Catene di Markov Tempo Discrete* nella quale gli archi anzichè dai rate sono contrassegnati dalla probabilità e le transizioni non necessitano di un tempo continuo per essere triggerate.
L'astrazione fondamentale su cui si basa il modello di *Gillespie* è la *transizione*, esse rappresentano rezioni di stampo chimico che regolano l'evoluzione dell'ecologia del sistema e non sono altro che il mapping che utilizza il modello per rappresentare un singolo step evolutivo del composto biochimico. Il rate di una transizione rappresenta la frequenza media, mentre la distanza di tempo tra due occorrenze di una stessa transizione segue una distribuzione di probabilità esponenziale negativa.
Ogni rate ha quindi un proprio rate che ne determina la frequenza con la quale può essere triggerata, l'attivazione di una transizione però non è soggetta a solo questo valore (rate cinetico), essa è soggetta anche alla concentrazione dei reagenti d'ingresso.
Considerando una generica transizione

$$X + Y \text{ - } r \longrightarrow Z$$

il rate markoviano R ad essa associato è calcolato dal prodotto del numero di molecole di X, per il numero di molecole di Y, per il rate cinetico r. Ad ogni step quindi il sistema calcola i rate markoviani di ogni le reazioni Ri e la loro somma $Rtot$, e verrà scelta una solo transizione tra quelle presenti scelta con probabilità $Ri/Rtot$.
L'intero processo ha una periodicità variabile, ad ogni esecuzione viene calcolato attraverso la funzione $\log(1/\tau)/Rtot$ dove τ è un numero casuale compreso tra 0 e 1, da questo si evince come la frequenza della simulazione viene influenzata non solo dai rate cinetici delle leggi ma anche dal variare delle concentrazioni dei reagenti.
Leggi chimiche e *reagenti* sono reificati nel centro di tuple sotto forma di tuple logiche opportunamente progettate e gestite attraverso le normali operazioni di comunicazione introdotte precedentemente.

L'**algoritmo di Gillespie** entra nell'ambito di questa tesi inquanto implementa il comportamento evolutivo tempo discreto del sistema chimico su cui si basa l'infrastruttura di coordinazione alla base del progetto. Implementando l'*algoritmo di Gillespie* potremmo simulare il modello MoK senza preoccuparci dello strato inferiore, cioè potremmo implementare le regole di coordinazione di MoK come leggi chimiche che popolano un certo sistema chimico simulato da Gillespie. In questo modo sarà garantita la selezione stocastica delle leggi da triggerare ad ogni dt istante tempo discreto del sistema chimico.
In questo modo potremo osservare e valutare le caratteristiche evolutive e coordinative che ci aspettiamo di osservare in un sistema chimico-biochimico reale. Il sistema di coordinazione, organizzato in un set di centri di tuple e caratterizzato da una precisa struttura topologica, prevede che ogni nodo possa avere un numero arbitrario di centri di tuple vicini, cioè direttamente collegati ad esso, che rappresentano i solo centri con il quale ogni singolo centro può comunicare.
L'interazione tra questi centri di tuple vicini è basata sul linkability-model ed è reificata nel sistema attraverso una tupla specifica nominata *firing tuple*, ovvero quelle tuple che dovranno essere propagate in uno dei centri che costituiscono il vicinato del centro corrente scelto in maniera equiorobabilistica. In questo modo è possibile far si che la concentrazione dei reagenti di un certo tuple centre non vari sono a causa di reazioni avvenute localmente ma anche per effetto di reazioni avvenute in centri di tuple che costituiscono il suo vicinato.
Quest'ultimo meccanismo permette di reificare il concetto di *compartimento* caratteristico dei sistemi biologici e del modello MoK studiato, ciò viene mappato agilmente sull'infrastruttura TuCSoN attraverso il concetto di nodo e di centro di tuple.

2.1 Motore chimico ReSpecT

Come detto ampiamente precedenemente utilizzando un infrastruttura come quella di TuCSoN si hanno numerosi vantaggi architetturali, tra i principali si ha che l'architettura topologica di un sistema chimico può venire mappata naturalmente su un'infrastruttura TuCSoN nella quale ad ogni centro di tu-

ple si fa corrispondere un compartimento chimico, oltre a questo abbiamo la possibilità di arricchire tali centri di tuple con specifiche reazioni che reagiscono ad eventi predeterminati. Tutto questo fa si ce venga naturale scegliere questa tecnologia come tool per implementare il modello MoK.
In particolare l'implementazione dell'*algoritmo di Gillespie* sarà costituita da una specifica ReSpecT che verrà inserita nei centri di tuple che costituiscono il sistema, in questo contesto chiameremo informalmente tale specifica *motore chimico* intendendo quella serie di regole e procedure che danno luogo ad una infrastruttura che permetta di simulare un sistema chemical-inspired.

A fronte delle modifiche portate alle operazioni di comunicazione eseguibili su un centro ReSpecT la specifica è stata modificata notevolmente rispetto a versioni precedentemente implementate nelle quali, l'assenza di tali operazioni, obbligava l'inserimento di dinamiche che gestissero liste e altre strutture dati non pertinenti all'algortimo specifico ma dovute a mancanze del linguaggio. Avendo quindi diminuito il gap concettuale inizialmente descritto, ho potuto fornire una specifica molto più snella, sia computazionalmente che a livello di lettura (da parte di un lettore interessato).
Nel seguito quindi verranno descritte le carattestiche funzionali e architetturali principali della specifica per meglio visualizzare il mapping avvenuto tra *algoritmo di Gillespie* e linguaggio ReSpecT. Per prendere visione della specifica completa si faccia riferimento all'appendice di questo documento.

Astrazioni principali

Nella specifica sono presenti tre astrazioni principali che devono essere mappate e sulle quali si basa poi l'applicazione dell'algortimo evolutivo, queste sono: *leggi chimiche, reagenti, vicinato*. La gestione di tali "individui" all'interno del motore chimico è centrale per il corretto funzionamento di quest'ultimo; possiamo addirittura affermare che il corretto funzionamento del motore chimico dipende dalla corretta gestione di tali entità.
Avendo deciso di implementare tale algoritmo attraverso il linguaggio ReSpecT tali astrazioni principali sono state mappate in tuple debitamente progettate.
Le *leggi* che popolano il nostro sistema/motore chimico sono sostanzialmente di due tipi, abbiamo leggi: ordinarie e temporali.

Le *leggi ordinarie* sono quelle leggi che per essere triggerate, oltre a dover essere selezionate dal motore, devono garantire la presenza dei reagenti di ingresso; tali leggi saranno, all'interno della specifica, reificate all'interno di una tupla avente la seguente sintassi:

```
law(Reagenti, Rate, Prodotti)
```

dove:

- *Reagenti* rappresenta la lista di reagenti che occorrono alla legge per essere eseguita, e che la legge consuma;
- *Rate* è il rate intrinseco associato alla legge, che rappresenta l'influenza di tale legge rispetto alle altre presenti;
- *Prodotti* sono tutti i prodotti della reazione.

Nelle precedenti versioni del motore chimico la presenza di tali leggi nello spazio non era sufficiente a garantire il corretto svolgimento della simulazione, il motore infatti non aveva la possibilità di ispezionare tutte le leggi presenti nello spazio e doveva quindi mantenere una lista aggiornata di tutte le leggi presenti nel sistema da dover aggiornare ad ogni inserimento o rimozione di nuove leggi. Questo portava quindi, oltra a dover mantenere una struttura dati aggiuntiva, ad avere un set di reazioni specifiche che si dovessere occupare della gestione di tale lista di leggi. L'estensione portata al linguaggio attraverso l'inserimento delle nuove specifiche ha semplificato notevoltemente la specifica limitandola alla presenza delle sole tuple `law` precedentemente descritte.
Le *leggi temporali* sono quelle leggi alle quali non viene associato un rate ma un intervallo di tempo passato il quale devono andare in esecuzione, nel caso in cui tutti i reagenti necessari siano presenti nello spazio. La struttura della tupla logica che reifica tale astrazione è la seguente:

```
timedLaw(Reagenti, Tempo, Prodotti)
```

dove:

- *Reagenti* ha lo stesso significato visto precedentemente;
- *Tempo* rappresenta il Δt dopo il quale deve essere eseguita la reazione;
- *Prodotti* ha lo stesso significato visto precedentemente.

Le considerazioni fatte nell'ambito della gestione delle leggi vista precedentemente è da considerarsi ancora valida, infatti anche in questo caso è stato possibile snellire il motore chimico dalla gestione di strutture dati più complesse grazie alla presenza del nuovo set di operazioni di comunicazione disponibili. Nell'ambito della descrizione delle legge disponibili vanno citate anche le leggi istantanee, queste sono leggi che hanno rate infinito, la loro sintassi è analoga a quella delle leggi standard con l'unica eccezione che hanno al posto del rate la tupla *inf* e non un valore numerico.

```
law(Reagenti, inf, Prodotti)
```

Fin'ora abbiamo considerato solo strutture dati che modellano la trasformazione e l'evoluzione del sistema, ora descriveremo le strutture dati che modellano gli individui fondamentali: *reagenti, vicini, firing.*
I *reagenti* che sono presenti nel sistema vengono rappresentati anch'essi da tuple logiche, in questo caso la tupla assume la sintassi:

```
reactant(Tipo, Quantita)
```

dove:

- *Tipo* rappresenta il reagente che stiamo inserendo nel sistema;
- *Quantita* rappresenta la quantità di reagente che è stata iniettata nel sistema.

Il concetto di vicinato: con vicinato si intendono tutti i centri di tuple che sono raggiungibili da un certo centro di tuple, in particolare un certo centro di tuple avrà al suo interno tante tuple *neighbour* quanti sono i suoi vicini "raggiungibili". Per fare in modo che un certo centro di tuple sia fisicamente abilitato a comunicare con un altro centro di tuple localizzato nella rete è necessario che vengano matenute alcune informazioni riguardanti tale vicino:

```
neighbour(Id, Address, Port)
```

dove:

- *Id* rappresenta il nome del tuple centre che rappresenta il nostro vicino;
- *Address* rappresenta il nodo di rete in cui tale centro è localizzato;

- *Port* rappresenta la porta di rete nella quale è relamente localizzato il centro (partendo dall'address).

Per fare in modo che un agente software possa comunicare al centro di tuple quali tuple devono essere inviate ad un vicino, esiste un'ultima astrazione: le *firing* tuple. Un agente che wrappa un reactant o una law, di qualsiasi tipo essa sia, all'interno di una firing tuple sta comunicando al motore che desidera inviare tale informazione ad un centro di tuple presente nel suo vicinato.

```
firing(Element)
```

dove:

- *Element* rappresenta una law o un reactant che voglia essere inviato ad un centro di tuple vicino.

Come si è potuto notare da questa breve introduzione, gli effetti delle modifiche portate all'infrastruttura TuCSoN ReSpecT ha fatto si che sia stato possibile mappare logicamente concetti del motore chimico, si considera ad esempio un singolo individuo *neighbour* il quale esplicitando Id, Address e Port reifica perfettamente quanto detto su centri di tuple all'interno di uno stesso dominio fisico.
Altre considerazioni riguardanti i benefici delle modifiche portate verrano discusse di seguito, passeremo ora a trattare la logica di esecuzione del motore.

Comportamento del motore

Il comportamento del motore piò essere suddiviso logicamente in quattro fasi principali quali:

1. Selezione della legge, che si divide in:
   (a) calcolo dei rate di tutte le leggi;
   (b) scelta della legge triggerabile basandosi sul metodo stocastico visto di Gillespie;
2. Esecuzione di una legge:
   (a) aggiornamento reagenti;

(b) aggiornamento dello stato del sistema.

Questi due passi costituiscono le due azioni principali che deve compiere il motore chimico in maniera iterativa e continua fino a quando sono disponibili leggi e/o reagenti. Ad ogni iterazione, infatti, il motore chimico valuta i rate globali - *rate globale = rate intrinseco * concentrazione*, dove *concentrazione = quantità reagente/quantità tutti reagenti* - di ogni legge, in base ai valori di tali rate viene applicato l'algoritmo di scelta stocastica di Gillespie per selezionare qualemettere in esecuzione.
La specifica implementa, sotto forma di reazioni, i meccanismi necessari per eseguire la legge che è stata selezionata, ad ogni esecuzione è fondamentale consumare tutti i reagenti utilizzati dalla legge ed evolvere lo stato del sistema.
I cicli di esecuzione del processo che seleziona la legge da mettere in esecuzione vengono scanditi da una legge temporale, tale legge viene triggerata ogni Δt tempo settato a partire dall'esecuzione dell'ultima legge, in maniera perfettamente aderente a quanto specificato nell'algoritmo di Gillespie; va inoltre sottolineato come tale intervallo di tempo non sia fisso ma vari in base al valore della somma di tutti i rate delle leggi chimiche presenti nel tuple centre e dalla generazione di un altro numero preso casualmente.
È previsto inoltre che il motore chimico possa raggiungere uno stato di *stop* momentaneo dato dall'assenza di leggi chimiche eseguibili o dei reagenti d'ingresso necessari per l'esecuzione; per uscire da tale stato è necessario che, a seguito di un evento di comunicazione sul centro di tuple, venga modificato lo stato del sistema in modo tale che i due requisiti che l'avevano posto in stato di stop non siano più verificati.
Volendo mantenere il motore chimico una quantopiù fedele implementazione dell'algoritmo di Gillespie è stato introdotta un ulteriore funzionalità per la quale si ha il seguente comportamento: se nel periodo d'attesa tra un ciclo del motore e il successivo lo stato del sistema cambia a fronte dell'esecuzione di un operazione di comunicazione che ha modificato lo stato dei reagenti o delle leggi, il motore ricalcola l'intervallo Δt associato alla legge che scandisce l'istante di tempo in cui dovrà essere verificata la presenza di una nuova legge eseguibile. Nel caso in cui il nuovo intervallo di tempo sia maggiore del successivo, viene aggiornata la legge temporale con il nuovo valore, nel caso

in cui invece il nuovo intervallo sia minore del precedente e già trascorso, la legge temporale viene cancellata e il motore viene stimolato per eseguire istantaneamente la fase di selezione di una legge. I cambiamenti di stato che devono provocare una rivalutazione dell'intervallo Δt sono: l'inserimento o la rimozione di una legge, l'aggiornamento o l'inserimento di un reagente; nel caso in cui venga aggiunta una legge istantanea o le cui condizioni siano già verificate questa viene automaticamente scelta e messa in esecuzione.

3 La specifica MoK

Quello che rimane da descrivere arrivati a questo punto dell'elaborato è capire come le astrazioni proposte in MoK possano essere gestite dal motore chimico creato attraverso la specifica ReSpecT.
La definizione del motore chimico ha fatto si che si sia alzato ulteriormente il livello di astrazione attraverso il quale si programmerà la specifica MoK. In particolare siamo partiti da reazioni ReSpecT e predicati logici Prolog per implementare il motore chimico e siamo ora giunti ad una nuova sorta di linguaggio definito in termini di law, timedLaw e reactant. Dovremo quindi reificare le astrazioni del modello MoK utilizzando tali concetti proosti dal motore chimico.
Atom, appare evidente come il concetto di atomo possa essere facilmente mappato su quello di reactant, in particolare essendo un atomo l'unità base di informazioni avremo che ogni atomo sarà contenuto all'interno di un reactant attraverso una sintassi del tipo:

```
reactant(atom(Src,Val,Attr),Conc)
```

dove:

- *Src*, rappresenta la sorgente che ha pubblicato l'atomo;
- *Val*, rappresenta l'informazione vera e propria contenuta nell'atomo;
- *Attr*, le informazioni aggiuntive previste;
- *Conc*, la concentrazione dell'atomo.

Molecule, la molecola, come ricorderemo, non è altro che una lista di atomi che per qualche motivo si è ritenuto opportuno aggregare (match sintattico, semantico - funzione MoK), essa rappresenta quindi, allo stesso modo

dell'atomo, un'unità informativa presente nella nostra "sostanza chimica". Il mapping tra tale astrazione e le astrazioni proposte dal motore chimico è quindi presto detta, avremo infatto che:

$$reactant(molecule([atom(Src_1, Val_1, Attr_1), ..., atom(Src_n, Val_n, Attr_n)]), Conc)$$

dove:

- la lista di atomi rappresentano gli atomi aggregati all'interno della stessa molecola;
- *Conc* rappresenta la concentrazione della molecola.

Enzyme, l'enzima segue lo scopo di dare un feedback positivo al sistema che faccia aumentare la concentrazione di atomi e molecole presenti. Questo, come già descritto, avviene a seguito di interazioni effettuate dall'utente/agente che maneggiando tale risorsa comunica implicitamente al sistema la sua rilevanza. La reificazione di tale concetto all'interno del motore chimico è:

```
reactant(enzyme(atom(Src,Val,Attr)),Conc)
```

Il significato di tale predicato è assolutamente intuitivo, esso infatti incapsula all'interno di un reactant una tuple enzima contenente l'atomo/molecola visitata/maneggiata dall'utente/agente. In questo contesto il parametro *Conc* rappresenta la concentrazione dell'enzima. Al fine di garantire che la presenza nel centro dell'enzima crei come effetto l'incremento della concentrazione dell'atomo/molecola a cui si riferisce è garantito da una law ulteriore inserita nel centro:

```
law([enzyme(atom(Src,Val,Attr)),atom(Src,Val,Attr)], Rate,
          [atom(Src,Val,Attr),atom(Src,Val,Attr)])
```

La semantica della legge è molto semplice: ogni qualvolta sia presente nel centro di tuple una tupla enzima avente la stessa sintassi specificata nell'atomo presente come secondo elemento della lista (enzyme) questa viene consumata (insieme all'atomo template), e quindi rimossa, e al suo posto viene emessa una singola tupla relativa all'informazione riferita all'interno dell'enzima aumentandone quindi la concentrazione (l'inserimento del secondo atomo identico serve per bilanciare la rimozione dell'atomo emplate). Come detto, tale

meccanismo può venire applicato anche nel caso delle molecole semplicemente pubblicando un enzym che riferisca ad una di queste ed inserendo nel centro una law relativa.

Passeremo ora alla descrizione del mapping tra le reazioni MoK proposte dal modello concettuale e la loro reificazione nell'engine chimico qui implementato.

Decay: la reazione del decadimento presentata da MoK è centrale nella simulazione del processo chimico naturale. Questa infatti permette di simulare la naturale decadenza di unità quali molecole o atomi nel momento in cui esse non siano di alcun interesse e rilevanza per gli utenti/agenti. Quello che accade quindi è una diminuzione costante della concentrazione degli atomi/molecole presenti nel centro di tuple.

```
law([atom(_,_,_)],Rate,[]) law([molecule(_,_,_)],Rate,[])
```

Provando ad interpretare il comportamento di tale legge avremo che: qual'ora essa venga selezionata dal motore chimico per venire eseguita, cerca un atomo/molecola presente nel centro di tuple e, una volta travata una, consuma un unità di questa diminuendone la concentrazione.

Aggregation, l'aggregazione è quel comportamento proattivo proposto dal modello MoK che fa in rende possibile generare nuova conoscenza, precedentemene non presente nel compartimento, aggregando unità informative secondo una qualche funzione MoK. Per essere più chiari possiamo dire che, avendo specificato i termini per i quali due unità informative devono essere considerate aggregabili - scopo della funzione MoK - la legge non deve fare altro che cercare n atomi/molecole che possano soddisfare tale funzione ed unirli all'interno di un unica molecola. Come proposto dal modello infatti, lo scopo delle molecole è appunto quello di contenere al proprio interno un lista di atomi; tale lista di atomi non sono selezionati a caso ma sono inseriti basandosi sulla funzione MoK del caso. Tale funzione MoK può quindi considerare semplici match sintattici a livello dei campi si source, valore o attributo; oppure match semantici più sofisticati e di maggiore valore aggiunto. Nel contesto di questa specifica MoK considereremo il caso più semplice del match sintattico verificato tra due atomi.

```
law([atom(Src,Val,Attr1),atomo(Src2,Val,Attr2)],Rate,
[molecule([atom(Src,Val,Attr1),atom(Src2,Val,Attr2)]))
```

Appare evidente quindi che qualora la legge venga selezionata essa cercherà nel centro due tuple aventi la stessa sorgente di informazione e, trovati e consumati, varrà generata una nuova molecola contente entrambi i due atomi. In questo contesto si è deciso di utilizzare il campo Src per eseguire il match sintattico, questo è naturalmente una scelta fatta nel contesto di questa implementazione e non un requirement.

Propagation, la reazione di diffusione fa si che il nostro centro non evolva autonomamente e localmente ma fa si, in accordo alla metafora a cui si ispira, che molecole ed atomi presenti in un compartimento chimico possano migrare in quello vicino in maniera probabilistica. In particolare abbiamo che la legge di diffusione estrae un atomo o molecola dal compartimento in cui si trova e lo diffonda in un compartimento vicino.

```
law([atom(Src,Val,Attr)],Rate,[firing(atom(Src,Val,Attr))])
   law([molecule(Atoms)],Rate,[firing(molecule(Atoms))])
```

Come appare nelle due leggi qui portate il comportamento è semplice: a fronte della scelta non deterministica di un atomo o di una molecola, questa viene consumata - diminuita la concentrazione - e viene diffusa in un compartimento vicino. La scelta del compartimento è anch'essa non deterministica; in accordo alla sintassi del motore chimico vediamo come il funtore *firing* palesi il fatto che tale tupla verrà diffusa in un altro centro di tuple/compartimento chimico.

Chapter 5

Caso di studio

Al fine di poter valutare concretamente i vantaggi che un modello di coordinazione e gestione della conoscenza knowldege-oriented basato su una metafora biochimica come MoK possa realmente fornire vedremo, nel seguente capitolo, un caso di studio che fornisca un utilizzo pratico dell'infrastruttura implementata.
Essendo fine ultimo di questa tesi quello non solo di studiare e valutare il modello teorico ma anche di fornirne una sua implementazione, è di grande significatività al fine di fornire un lavoro quanto più completo ed in linea con la mission iniziale ricercare risultati pratici da poter valutare.

1 Repository di articoli distribuito

Il contesto in cui si pone il caso di studio è quello di un team di ricercatori. Poniamo che questo gruppo di colleghi si trovi a lavorare, come spesso capita, ad ambiti diversi di ricerca; fonte inesauribile di conoscenza, aiuto e stimolo è sicuramente il lavoro portato a termine dai colleghi ricercatori ed in particolare dalle pubblicazioni da questi prodotti. Il livello di complessità raggiunge però un livello maggiore nel momento in cui si considera un contesto di lavoro in cui non siano presenti server locali sui quali poter condividere tali fonti di conoscenza con i colleghi. Appare subito evidente come l'unica soluzione per poter condividere tali documenti possa essere una rete ad architettura distribuita dove ogni nodo della rete è rappresentato dai songoli pc dei ricercatori (ed eventualmente dai loro collaboratori) che comunicano costituendo

una rete di host distribuita per la rete.
Questo grupo di ricercatori ha inoltre la buona abitudine di associare ad ogni documento una serie di meta dati eventualmente utili per poterne eseguire uno scanning automatico al quale potrà poi seguire una qualche ricerca etc. A tutto questo si aggiunge la necessità, di tali ricercatori, di volere maggiore vsibilità di tutti quei documenti prodotti eventualmente dai colleghi che per qualche ragione intersecano i propri campi di ricerca o che in qualche modo possono veicolare meglio le idee e gli spunti personali di questi ultimi al fine di concretizzare nuovi lavori. L'idea poi che documenti provenienti da fonti diverse, ma legati in qualche modo da meta-informazioni, possano "aggregarsi" autonomamente ed essere visualizzati dal singolo utente come una nuova informazione aggiuntiva appare più come sogno che come requisito.
Definiti gli ambiti di interesse di ciascun ricercatore, e palesati al sistema, sarebbe di assoluto valore poter ricevere notizia di lavori analoghi inerenti presenti negli altri nodi della rete in maniera proattiva, senza cioè che il singolo utente esegua esplicitamente la ricerca ma in maniera tale che definiti gli ambiti siano poi i documenti a "raggiungerlo" autonomamente.

Analizzando attentamente il contesto presentato appare evidente come siano presenti una moltitudine di criticità difficilmente gestibili da una classica infrastruttura di condivisione di documenti. Possiamo in particolare riconoscere alcune caratteristiche/requisiti fondamentali:

1. *ambiente distribuito*, abbiamo infatti che ogni ricercatore dovrà porre i propri documenti all'interno del pc personale volendo in ogni modo permettere ai colleghi di poterne fruire liberamente;
2. la natura dinamica dei articoli che gli permette di *migrare* autonomamente dal pc di un ricercatore ad un altro permettendogli di venire a conoscenza di questo;
3. la possibilità di consentire ad ogni singolo ricercatore di specificare ambiti di interesse;
4. la natura dinamica delle informazioni che *autonomamente si aggregano* in riferimento agli ambiti di interesse specificati.

Leggendo attentamente il caso di studio proposto appare piuttosto palese come un sistema di questo tipo possa essere modellato attraverso un'implementazione

del modello MoK, non solo, ma vedremo come tutti i requisiti di maggiore complessità qui proposti, difficilmente gestibili da paradigmi classici, assumano la caratteristica di una naturale feature del nostro sistema.

2 $APICe^2$

Il sistema, nominato $APICe^2$, (riferendosi palesemente al repository utilizzato dai ricercatori dell'università di ingegneria di Cesena per mantenere, tra le altre cose, i propri lavori, chiamato APICe) sarà quindi un sistema atto alla condivisione di informazioni relative a articoli didattici, distribuito, con un comportamento autonomo che permette alle informazioni di vagare tra i nodi della rete in maniera stocastica e che palesa un comportamento evolutivo/auto-organizzativo tipico di sistemi naturali/chimici, nei quali sono gli individui più forti che sopravvivono più a lungo. In questo contesto vengono definiti individui più forti (quelli che meglio passeranno la prova evolutiva) quelli rappresentanti gli ambiti di interesse di ciascun ricercatore.
Vediamo quindi il mapping tra caso di studio $APICe^2$ e MoK:

- ogni singolo ricercatore della rete avrà sul proprio pc istanziato un centro di tuple TuCSoN sul quale sarà debitamente caricata la specifica ReSpecT del motore chimico, ogni pc di altri ricercatori rappresenterà un compartimento chimico raggiungibile dagli altri e verso i quali potranno diffondere le informazioni;
- ogni qualvolta un ricercatore pubblicherà un nuovo lavoro questo lo inserirà in una certa cartella del proprio pc dal quale un agente autonomamente estrarrà le meta informazioni annesse dal ricercatore e le pubblicherà nel proprio sistema chimico sotto forma di atomi del tipo:

  ```
  atom('www.myPersonalRepo.com/metaChim/MoK',mariani, author)
  atom('www.myPersonalRepo.com/metaChim/MoK',2011,year)
  atom('www.myPersonalRepo.com/metaChim/MoK',gillespie,keywords)
  ```

  In questo modo quindi per ogni nuovo articolo inserito/pubblicato saranno presenti informazioni nel relativo centro;

- gli atomi così formati potranno aggragarsi (nel nostro esempio rispetto al campo valore) generando molecole di nuova conoscenza prima non presenti, in questo modo il sistema in forma autonoma aggregerà con buona probabilità tutti gli articoli scritti nello stesso anno o tutti quelli aventi lo stesso autore o le stesse keywords. Valore aggiunto è dato dal fatto che prenderanno parte a queste nuove forme di conoscenza non solo gli atomi pubblicati dal nodo stesso ma anche quelli conosciuti grazie al processo di diffusione;
- attraverso delle azioni che palesino l'interesse di ogni agente ricercatore presente su ogni nodo, come lettura/rimozione di informazioni, verranno generate nel singolo centro/compartimento chimico enzimi che automaticamente faranno in modo che sia palesato come certi ambiti siano di maggiore rilevanza per l'agente e per i quali sarà quindi più difficile subire l'effetto della decadenza ed eliminazione tipico dei sistemi evolutivi (saranno quelle informazioni maggiormente resistenti all'evoluzione);

Abbiamo in questo modo sottolineato come ogni astrazione proposta dal modello MoK venga mappata nel nostro caso di studio garantendo evidenti vantaggi alla sua implementazione e gestione.

Interazioni con lo spazio

Per poter simulare efficacemente il risultato della computazione del nostro caso di studio è stato necessario simulare le interazioni degli utenti attravero specifici agenti e comportamenti standard insiriti all'interno dello law.
Avremo in particolare due tipi di agenti:

- *LawAndSourceAgent*, come l'agente incaricato di eseguire la fase di init del motore chimico, in tale fase in particolare verranno inseriti le law che abilitano il comportamento evolutivo del motore;
- *UserAgent*, l'agente atto a simulare le interazioni di lettura/rimozione delle informazioni effettuate da un utente esterno (quelle cioè che causeranno la pubblicazione di atomi enzima).

La presenza di questi due agenti è sufficiente per simulare il nostro caso di studio, nell'ambito del quale sono stati utilizzati tre nodi differenti posti sulla

stessa macchina fisica ma localizzati su tre porte logiche diverse; all'avvio del main del progetto questo si occuperà quindi di istanziare un agente *LawAndSourceAgent* e un *UserAgent* per ogni centro di tuple istanziato.

Pubblicazione di atomi

Per gestire la pubblicazione costante di atomi all'interno di ogni compartimento chimico sono state utilizzate law costruite ad hoc che simulassero la presenza di un untente esterno, in particolare una di queste leggi potrebbe apparire come:

```
law([],0.9,[atom('www.myPersonalRepo.com/metaChim/MoK',mariani, author)])
```

una law di questo tipo infatti non fa altro che pubblicare nel compartimento chimico corrente un atomo relativo ad un certo articolo specificandone ad esempio l'autore. Specificando law diverse all'interno dei file di configurazione di ogni centro di ogni centro di tuple è stato possibile specificare diverse law di "pubblicazione atomi" in modo da simulare l'azione di inserimento di nuovi articoli all'interno del proprio pc continuativamente nel tempo.
Per effettuare i test sono stati codificati in tuple *atom* le informazioni di una ventina di articoli recuperate sul repository APICe, in particolare per ogni articolo sono state inserite informazioni inerenti a:

- autore dell' articolo o, nel caso ce ne fossero più di uno, tutti gli autori;
- l'anno di pubblicazione;
- alcune keywors specificate.

Enzimi

Come abbiamo più volte detto la lettura di un atomo da parte di un agente deve generare nel compartimento la creazione di un enzima, questo nel contesto del caso di studio è stato simulato agento da due lati. Da lato agente si è fatto in modo che ad una certa frequenza l'agente simulasse la lettura di una tupla:

```
int chosen = r.nextInt(4);
switch(chosen){
  case 0: acc.rd(tcid, LogicTuple.parse("reactant(atom(_,omicini,_),_)"), (Long)null);
  case 1: acc.urd(tcid, LogicTuple.parse("reactant(atom(_,omicini,_),_)"), (Long)null);
  case 2: acc.rd(tcid, LogicTuple.parse("reactant(molecule([atom(_,omicini,_)|T]),_)"),(Long)null);
  case 3: acc.urd(tcid, LogicTuple.parse("reactant(molecule([atom(_,omicini,_)|T]),_)"),(Long)null);
  default: break;
```

```
  }
 }
```

Come visibile dal codice sopra riportato si è gestito la scelta dell'operazione di lettura da invocare sul centro di tuple in maniera casuale, va però notato come il parametro Val (in questo caso) sia stato debitamente settato ad un valore di interesse. Palesando tale valore si comunica esplicitamente quali siano le informazioni di nostro interesse e quali quindi ci aspettiamo/speriamo di trovare in maggiore concentrazione nel nostro compartimento. Per poter gestire "lato specifica" il comportamento che a fronte di un operazione di lettura da parte dell'agente il compartimento generi autonomamente un enzima, sono state utilizzate delle reaction; in particolare nel caso di una rd, si avrà:

```
reaction(rd(reactant(X,C)),(from_agent,post),(out(enzyme(reactant(X,10)))))
```

Caratteristiche ricercate

L'implementazione del caso di studio proposto mira ad osservare il verificarsi di due comportamenti specifici del modello MoK, in particolare vogliamo osservare che:

1. vengano generate molecole che rispettino la funzione di match sintattico utilizzata e che tali atomi non siano tutti stati pubblicati direttamente nel tuple centre di interesse ma che possano provenire anche da altri centri di tuple/compartimenti chimici attraverso reazioni di diffusione;
2. ogni singolo utente accedendo ad atomi o molecole di suo interesse presenti nel proprio centro di tuple, fa si che il sistema reagisca generando enzimi relativi a tali informazioni accedute. Tale meccanismo garantisce che nel sistema aumenti la concentrazione di atomi/molecole relative a notizie di maggior interesse per l'utente lasciando quindi decadere quelle meno importanti.

Per quanto riguarda la specifica MoK completa utilizzata per la fase di init di ogni compartimento chimico si faccia riferimento all'appendice B nella quale è riportata interamente.

3 Risultati ottenuti

I test sono stati eseguiti mediamente per 30 minuti utilizzando una singola macchina fisica nella quale venivano lanciate tre 3 jvm in parallelo (una per ogni centro di tuple). Si è notato che dopo un periodo di questo tipo i risultati ricercati (molecole contenenti atomi provenienti da centri diversi e, per ogni centro, una maggiore concentrazione di molecole contenenti atomi relativi alla preferenza esplicitata dall'utente) sono stati verificati.
Come detto è stata simulata la presenza di tre ricercatori localizzati su tre porte logiche diverse, ogni ricercatore pubblica informazioni relative ad articoli di cui è l'autore o a cui a partecipato. Ogni autore sarà interessato degli articoli scritti da un altro collega e per questo cercherà di reperirine informazioni eseguendo delle rd sul proprio centro.
Quello che questa simulazione vuole verificare è quindi il verificarsi delle situazioni sopra descritte in un contesto operativo.
Mostreremo di seguito alcuni print screen ottenuti a seguito di alcuni test effettuati localmente; si ricorda che per questi test è stata utilizzata una singola macchina sulla quale sono stati istanziati tre nodi diverse (porte: 20504, 20505, 20506), ognuno dei quali conteneva al proprio interno un singolo centro di tuple. All'avvio ogni centro è stato inizializzato con la stessa specifica (riportata all'appendice B) eccezion fatta per i dati relativi ai vicini ed al contenuto degli atomi. Si è simulato in questo modo che ogni nodo contenga autonomamente solo un certo set di informazioni riguardanti uno specifico insieme di articoli e che ogni altra informazione provenga da nodi vicini.

Aggregazione di atomi

Come primo risultato mostreremo un come due atomi, pubblicati dall'agente "proprietario del compartimento", possano aggregarsi generando una molecola:

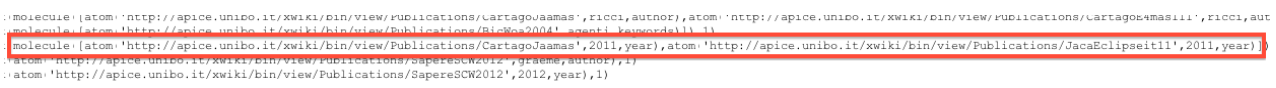


Figure 5.1: Aggregazione - 1

Come secondo esempio di aggregazione di atomi consideriamo invece il caso in cui venga pubblicata una molecola nella quale un atomo era già presente nel

compartimento mentre altri provengono da compartimenti esterni (ottenuti per diffusione):

In questo caso in particolare l'articolo SelforgcoordKer25years è giunto al

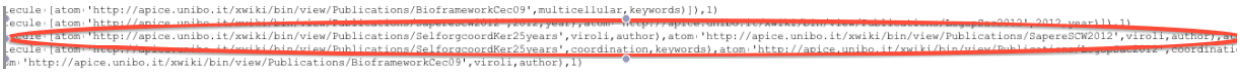


Figure 5.2: Aggregazione - 2

compartimento corrente a seguito di una diffusione di questo

Effetto enzimi

Come ultimo esempio verrà mostrato come l'effetto della lettura di certi particolari atomi/molecole, e quindi la successiva presenza di enzimi, aumenti la concentrazione di tutti quegli atomi/molecole fedeli a tale pattern: in questo

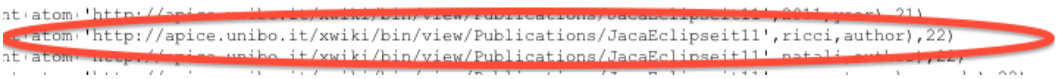


Figure 5.3: Aumento concentrazione a seguito della presenza di enzimi

esempio in particolare l'interesse era puntato sugli articoli pubblicati da un certo ricercatore, ed appare subito evidente come la concentrazione degli atomi relativi ad esso sia aumentata velocemente.

Conclusioni e possibili estensioni

Attraverso questo elaborato di tesi si è cercato di valutare ed implementare concretamente il modello MoK testandone quindi le possibili applicazioni/ complessità/ potenzialità reali attraverso il suo sviluppo concreto ed un caso di studio che ne mettesse alla prova i limiti/valori. Si è valutata quindi, a tal fine, un'opportuna tecnologia che ne permettesse un mapping coerente con le astrazioni trattate, essa è poi stata, ove ve ne fosse la necessità, estesa e ampliata al fine di diminuire il gap concettuale tra modello e infrastruttura. Si è infine valutato un algoritmo di simulazione stocastica di sistemi chimici proposto da Gillespi e se ne è fornita un'implementazione concreta che fosse utilizzabile sulla nuova infrastruttura creata abilitando quindi il sistema a simulare l'evoluzione di un sistema chimico.

Tutto questo ha portato ad un sistema sufficientemente stabile e sicuro su cui poter iniziare lo sviluppo/ test di nuovi pattern evolutivi/ distribuiti/ concorrenti etc. Le possibilità teoriche fornite dal modello sono innumerevoli, si considera di particolare importanza poter valutare e testare come ambienti, classicamente strutturati come sistemi centralizzati, possano apparire/ comportarsi/ evolvere passando da un modello centralizzato ad uno MoK-like. Possibili scenari futuri potrebbero quindi prevedere di modellare sistemi conosciuti classicamente come, ad esempio, architetture client-server e fornirne una versione basata sul modello MoK.

Come è stato detto nell'introduzione infatti, il contesto sociale ci sta portando sempre di più a sistemi socio tecnolgici nel quale punto centrale sia la dinamicità delle informazioni, queste essendo dinamiche devono poter vagare

per lo spazio alla ricerca di un utente interessato e qui proliferare. Dal mio punto di vista infatti è questa la strada che deve essere intrapresa al fine di poter fornire sempre di più tecnologie e modelli concettuali che vadano di pari passo con quello che è il contesto socio-culturale-tecnologico.

Appendice A - Specifica motore chimico

In questa appendice verrà riportata la specifica completa del motore chimico che simula l'algoritmo di Gillespie utilizzato nella presente tesi. La struttura in cui è riportato è molto semplice: nella parte iniziale *reazioni* sono riportate tutte le reaction ReSpecT e nella seconda parte *predicati* sono riportati i predicati Prolog utilizzati. Per comprendere pienamente il funzionamento e le scelte fatte si consiglia di leggere la specifica facendo particolarmente riferimento ai commenti riportati e al capitolo quattro nel quale viene descritta e motivata.

```
[( (out(engine_trigger(_,_)),endo,(
        in(engine_trigger(_,_)) ,
        urd(law(IL,inf,OL)),
        current_time(Now),
        out(execution(law(IL,_,OL),Now,true))
)),
(out(engine_trigger(_,Flag)),endo,(
        no(law(_,inf,_)),
        in(engine_trigger(_,Flag)),
        isAlreadyTriggered(Flag)
)),
(out(engine_trigger(_,_)),endo,(
        no(law(_,_,_)),
        in(engine_trigger(_,_)),
```

```
        out(engine_suspended)
)),
(out(execution(law([X—T],_,OL),Time,Flag)),endo,(
        in(execution(law([X—T],_,OL),Time,Flag)),
        removeReactant(reactant(X,1)),
        out(execution(law(T,_,OL),Time,Flag))
)),
(out(execution(law([] ,_,[X—T]),Time,Flag)),endo,(
        in(execution(law([] ,_,[X—T]),Time,Flag)),
        processOutput(X,Time) ,
        out(execution(law([],_,T),Time,Flag))
)),
(out(execution(law([] ,_,[]) ,Time,Flag)),endo,(
        triggerIfNecessary(Flag) ,
        in(execution(law([],_,[]),Time,Flag))
)),
(out(reactant(X,C)),(from_agent,post),(
        in(reactant(X,C)) ,
        updateReactants(reactant(X,C))
)),
(in(reactant(X,C)),(from_agent,post), (
        out(reactant(X,C)),
        removeReactant(reactant(X,C)),
        out(engine_trigger(0,true))
)),
(out(law(IL,R,OL)),(from_agent,post),(
        not(var(R)),
        in(law(IL,R,OL)),
        updateLaws(law(IL,R,OL))
)),
(in(law(IL,R,OL)),(from_agent,post),(out(engine_trigger(0,true)))),
(inp(law(IL,R,OL)),(from_agent,post),(out(engine_trigger(0,true)))),
(out(timedLaw(IL,Dt,OL)),(response,exo),(
        in(timedLaw(IL,Dt,OL)),
        event_time(Time) , Time2 is Time + Dt ,
```

```
        out(timedLaw2do(IL2,Time2,OL)),
        out_s(time(Time2), endo, (
                in(timedLaw2do(IL2,Time2,OL)),
                out(execution(law(IL2,_,OL),Time,true)) ))
)),
( out(tosend(neighbour(Id,Address,Port),B)),endo,(
        in(tosend(neighbour(Id,Address,Port),B)),
        Id@Address:Port ? out(B)
)
)]
isAlreadyTriggered(true):-
        in_s(time(_),_,out(engine_trigger(Time,_)) ), !,
        rd_all(law(_,_,_),LL), calcRates(LL,NL,Rtot,0), not(Rtot==0),
        rand_float(Tau) , NewDt is round(log(1/Tau)/Rtot*1000 ) ,
        TimeNew is Time + NewDt,
        checkTime(Time,TimeNew).

isAlreadyTriggered(_):-
        chooseLaw(law(IL,Ra,OL),Rtot), !,
        rand_float(Tau) , Dt is round(log(1/Tau)/Rtot*1000) ,
        event_time(Time) , Time2 is Time + Dt ,
        out(execution(law(IL,Ra,OL),Time,false)),
        ( Ra==Rtot; out_s(time(Time2),endo,
        out(engine_trigger(Time,false))) ).

chooseLaw(Law, Rtot):-
        no(laws(_,inf,_)),
        rd_all(law(_,_,_),LL),
        calcRates(LL,NL,Rtot,0),
        not(Rtot==0),
        sortLaws(NL,SL),
        rand_float(Tau),
        chooseGillespie(SL,Rtot,Tau,Law).

calcRates(LL,NL,Rtot,Test):- calcRates(LL,[] ,NL,0,Rtot).
```

```
calcRates([Law—T] ,PL,NL,SR, Rtot ):- calcRate(Law,R),!, SR2 is SR+R,
                calcRates(T,[(R,Law)—PL],NL,SR2,Rtot).
calcRates([Law—T] ,PL,NL,SR, Rtot ):- calcRates(T ,PL,NL,SR, Rtot ).
calcRates([] ,PL,PL,SR,SR).

calcRate(law([],R,_),R):- !.
calcRate(law(IL,R,_),GR):- calcRate(IL,R,GR,_,0).

calcRate([H—T],_,0,H,N):- rd(reactant(H,NH)), NH=¡N, !.

calcRate([H—T],PR,GR,H,N):- ! ,rd(reactant(H,NH)), M is N+1,
            PR2 is PR*(NH-N)/M,
            calcRate(T,PR2,GR,H,M).

calcRate([H—T],_,0,X,_):- rd(reactant(H,NH)), NH= 0, !.
calcRate([H—T] ,PR,GR,X,_):- rd(reactant(H,NH)) ,
             PR2 is PR*NH,
             calcRate(T,PR2,GR,H,1) .
calcRate([],R,R,_,_).

sortLaws([],[]) .
sortLaws([(X,Law)—Xs],Ys):-
        partition(Xs,(X,Law) ,Bs,Ls) ,
        sortLaws(Bs,BOs) ,
        sortLaws(Ls,LOs) ,
        append(BOs,[(X,Law)—LOs] ,Ys).

partition([],_,[],[]) .
partition([(X,LawX)—Xs],(Y,LawY) ,[(X,LawX)—Bs],Ls):- X¿=Y, !,
                        partition(Xs,(Y,LawY),Bs,Ls).
partition([(X,LawX)—Xs],(Y,LawY),Bs,[(X,LawX)—Ls]):- partition(Xs,(Y,LawY),Bs,Ls).

checkTime(Time,TimeNew):-
        current_time(Now), TimeNew¿Now, !,
        out_s( time(TimeNew), endo, out(engine_trigger(Time,false)) ).
```

```
checkTime(Time,_):- out(engine_trigger(Time,false)).

chooseGillespie(SL,Rtot,Tau,Law):- checkGillespie(SL,Rtot,Tau,0,Law).
checkGillespie([(Ri,Lawi)—_],Rtot,Tau,PR,Lawi):- Tau =¡ (Ri/Rtot)+PR, !.
checkGillespie([(Ri,Lawi)—T],Rtot,Tau,PR,Law):-PR2 is ((Ri/Rtot)+PR),
                    checkGillespie(T,Rtot,Tau,PR2,Law).
checkGillespie([(Ri,Lawi)],Rtot,Tau,PR,Lawi).

processOutput(firing(law(A,R,B)),_):-
        no(neighbour(_,_,_)), ! ,
        processOutput(law(A,R,B),_) .

processOutput(firing(law(A,R,B)),_):-
        urd(neighbour(Id,Address,Port)),!,
        out(tosend(neighbour(Id,Address,Port),law(A,R,B))).

processOutput(law(A,R,B),_):-!,addLaw(law(A,R,B)).

processOutput(firing(timedLaw(A,Dt,B)),_):-
        no(neighbour(_,_,_)),!,
        out(timedLaw(A,Dt,B)).

processOutput(firing(timedLaw(A,Dt,B)),_):-
        urd(neighbour(Id,Address,Port)),!,
        out(tosend(neighbour(Id,Address,Port),timedLaw(A,Dt,B))).

processOutput(timedLaw(A,R,B),_):-!,out(timedLaw(A,R,B)) .

processOutput(firing(reactant(X,Y)),_):-
        urd(neighbour(Id,Address,Port)),!,
        out(tosend(neighbour(Id,Address,Port),reactant(X,Y))).

processOutput(X,Time):- addReactant(reactant(X,1)).
```

```
triggerIfNecessary(_):-in(engine_suspended),!,out(engine_trigger(0,false)).
triggerIfNecessary(true):-!,out(engine_trigger(0,true)).
triggerIfNecessary(false).

updateReactants(reactant(X,C)) :-
        no(reactant(_,_)) ,
        no(law(_,_,_)), ! ,
        out(reactant(X,C)).

updateReactants(reactant(X,C)) :-
        no(reactant(_,_)),!,
        out(reactant(X,C)),
        out(engine_trigger(0,false)).

updateReactants(reactant(X,Cin)) :-
        addReactant(reactant(X,Cin)),
        out(engine_trigger(0,true)).

addReactant(reactant(X,Cin)) :-
        ( (inp(reactant(X,Cold)), Cnew is Cold+Cin); Cnew = Cin ),
        out(reactant(X,Cnew)).

removeReactant(reactant(X,C)) :- no(reactant(X,_)).

removeReactant(reactant(X,C)):-
        in(reactant(X,Cold)), Cold ¿= C,
        Cnew is Cold - C,
        ( Cnew==0; out(reactant(X,Cnew)) ).

updateLaws(law(Re,Ra,Pr)):-
        no(law(_,_,_)),no(reactant(_,_)),!,
        out(law(Re,Ra,Pr)).

updateLaws(law(Re,Ra,Pr)):-
        no(law(_,_,_)), !,
```

```
        out(law(Re,Ra,Pr)).

updateLaws(law(Re,Ra,Pr)):-
        no(law(Re,Ra,Pr)),!,
        out(law(Re,Ra,Pr)),
        out(engine_trigger(0,true)).

updateLaws(law(Re,Ra,Pr)).

addLaw(law(Re,Ra,Pr)):-
        list(Re),not(var(Ra)),list(Pr),
        no(law(Re,Ra,Pr)),!,
        out(law(Re,Ra,Pr)).

removeLaw(law(Re,Ra,Pr)):- in(law(Re,Ra,Pr)), out(engine_trigger(0,true)).
```

Appendice B - Specifica MoK

In questa appendice verrà riportata la specifica di law utilizzata per modellare le reazioni descritte da modello MoK. (aggregazione, decadimento, aggregagazione, diffusione) In essa saranno presenti alcuni valori di rate associati alle reazioni, non si considerino tale law vincolate a tali valori si è semplicemente voluto riportare la specifica così com'è stata utilizzata in fase di test del sistema.

```
 %%%%% AGGREGAZIONE %%%%%
 % Aggragazione di atomi e molecole aventi lo stesso campo Val,
 % tale aggregazione serve per aumentare il numero di atomi
 % contenuti all'interno di una stessa molecola
law([atom(A,B,C),molecule([atom(D,B,E)|T])],2,[molecule([atom(A,B,C),atom(D,B,E)|T]

% Aggragazzione di due atomi, matchanti nel campo Val, al fine
% di generare una nuova molecola
law([atom(A,V,B),atom(C,V,D)],1.5,[molecule([atom(A,V,B),atom(C,V,D)])])

%%%%% FEED BACK POSITIVO %%%%%
% Gestione della presenza degli enzimi nello spazio, ogni
% qualvolta ci sia un enzima questo viene rimosso e viene
% aumentata la concentrazione dell'atomo a cui si riferiva
law([enzyme(atom(A,B,C))],0.2,[atom(A,B,C)])

%%%%% DIFFUSIONE %%%%%%
% Ogni qualvolta la law venga eseguita questa consuma un
% atomo/molecola e lo fa diventare un firing(atom), intendendo
% come questo debba essere diffuso ad un vicino.
```

```
law([atom(A,B,C)],0.15,[firing(atom(A,B,C))])
law([molecule(A)],0.15,[firing(molecule(A))])

%%%%% DECADIMENTO %%%%%
% Ogni qualvolta la legge venga eseguita questa diminuisce la
% concentrazione di un atomo/molecola  presente nello spazio

law([atom(A,B,C)],0.09,[])
law([molecule(A)],0.09,[])
```

Bibliography

[1] Stefano Mariani (2011), *"Molecules of knowledge: a new approach to knowledge production, management and consumption"*;

[2] Stefano Mariani, Andrea Omicini *"Self-Organising News Management: The Molecules of Knowledge Approach"*;

[3] Stefano Mariani, Andrea Omicini *"Molecules of Knowledge: Self-Organisation in Knowledge-Intensive Environments"*;

[4] Franco Zambonelli, Mirko Viroli *"Architecture and Metaphors for Eternally Adaptive Service Ecosystems"*;

[5] Mirko Viroli, Matteo Casadei, Andrea Omicini *"A Framework for Modelling and Implementing Self-Organising Coordination"*;

[6] Marco Cremonini, Andrea Omicini, Franco Zambonelli *"Coordination and Access Control in Open Distributed Agent Systems: The TuCSoN Approach"*;

[7] Andrea Omicini,Enrico Denti *"From tuple spaces to tuple centres"*, Science of Computer Programming 41 (2001) 277-294;

[8] Andrea Omicini, Enrico Denti *"Formal* ReSpecT*"*, Electronic Notes in Theoretical Computer Science 48 (2001);

[9] Andrea Omicini *"Formal* ReSpecT *in the A&A Perspective"*, Electronic Notes in Theoretical Computer Science 175 (2007) 97-117;

[10] Matteo Casadei, Andrea Omicini *"Situating A&A* ReSpecT*: Reactivity to Environment Events"*;

[11] Daniel T. Gillespie *"Exact Stochastic Simulation of Coupled Chemical Reactions"*, The Journal of Physical Chemistry, Vol, 8 1, No.25, 1977;

[12] Francis Heylighen, Carlos Gershenson, Gary William Flake, David M. Pennock, Daniel C. Fain, David De Roure, Karl Aberer, Wei-Min Shen, Olivier Dousse, Patrick Thiran (2003), *"Neurons, Viscose Fluids, Freshwater Polyp Hydra and Self-organising Information System"*, IEEE Intelligent Systems, July/August, 72-86;

[13] Marco Mamei, Ronaldo Menezes, Robert Tolksdorfand, Franco Zambonelli, *"Case Studies for Self-Organization in Computer Science"*;

[14] Matteo Casadei, Mirko Viroli (2008), *"Applying Self-Organizing Coordination to Emergent Tuple Organization in Distributed Networks"*, Second IEEE International Conference on Self-adaptive and Self-organising Systems, 213-222;

[15] Mirko Viroli, Franco Zambonelli (2010), *"A biochemical approach to adaptive service ecosystems"*, Information Sciences 180, 1876-1892;

[16] Mirko Viroli, Matteo Casadei (2009), *"Biochemical Tuple Spaces for Self-organising Coordination"*, COORDINATION 2009, LNCS 5521, pp. 143-162, 2009;

[17] Kenneth Lange (2010), *"Continuous-Time Markov Chains"*, APPLIED PROBABILITY, Springer Texts in Statistics, Volume 0, 187-215;

[18] J. Fisher, T.A. Henzinger (2007), *"Executable cell biology"*, Nat. Biotechnol. 25, 1239-1249;

[19] C. Priami (1995), *"Stochastic pi-calculus"*, Comput. J. 38 (7) 578-589;

[20] G. Paun, (2002), *Membrane Computing An Introduction*, Springer-Verlag;;

[21] http://alice.unibo.it/xwiki/bin/view/Tuprolog/;

[22] Elena Nardini, Mirko Viroli, Emanuele Panzavolta (2010), *"Coordination in Open and Dynamic Environments with TuCSoN Semantic Tuple Centres"*, SAC'10 March 22-26;

[23] Marco Dorigo, Mauro Birattari, and Thomas Stutzle (2006), *"Ant Colony Optimization Artificial Ants as a Computational Intelligence Technique"*, IEEE COMPUTATIONAL INTELLIGENCE MAGAZINE;

[24] Matteo Casadei, Mirko Viroli, Luca Gardelli (2009), *"On the collective sort problem for distributed tuple spaces"*, Science of Computer Programming 74, 702-722;

[25] S. Camazine, J.-L. Deneubourg, N.R. Franks, J. Sneyd, G. Theraulaz, E. Bonabeau (2001), *"Self-Organization in Biological Systems, Princeton Studies in Complexity"*, Princeton University Press, 41 William Street, Princeton, NJ 08540, USA;

[26] Mirko Viroli (2009/2010), *Formal Grammars*, Corso di Linguaggi e Modelli Computazionali, II Facoltà di Ingegneria, Cesena;

[27] Andrea Omicini (2007), *"Formal* ReSpecT *in the A&A perspective"*, Eletronic Notes in theoretical computer science 175 (97-117);

[28] Marco Sbaraglia (2009), *"Coordinazione space-based di ispirazione biochimica per la piattaforma bioinformatica Cellulat"*;

[29] http://apice.unibo.it/xwiki/bin/view/TuCSoN/Overview;

Ringraziamenti

Ringrazio innanazi tutto la Pisin, Luck e Stevo (mamma e fratelli) che in questi anni di università mi sono stati accanto sopportandomi durante le numerose crisi universitarie.

Ringrazio Claudia, che piu di tutti mi è stata vicina supportandomi, incoraggiandomi e cantandomi un sacco di belle canzoni per invogliarmi a studiare e dimostrarmi quanto ci tiene a me.

Ringrazio Ste per il suo enorme aiuto datomi per lo svolgimento di questa tesi, per la sua disponibilità precisione e costanza (mi ha insegnato tanto)...non dimenticando poi tutte le chiacchierate fatte durante le lezioni e nella preparazione degli esami ;).

Ringrazio tutti i miei amici che in questi anni sono stati compagni di studio (in valle o alla centrale :)) e con i quali ho condiviso crisi universitarie/sociali/politiche/etc..insomma ringrazio tutta casa Chris!!

Ringrazio tutti i professori, che in questi anni hanno fatto parte della mia vita universitaria, per la loro grinta, passione e impegno messi (e che mettono tuttora) nell'insegnamento - sono stati stimolo costante di miglioramento sia a livello scolastico che personale.
In particolare ringrazio il professor Omicini per il grande aiuto datomi nell'esecuzione di questo lavoro e per la fiducia dimostratami.

Ringrazio tutte le persone che sono qui oggi per festeggiare con me o che sarebbero volute essere qui.

Ringrazio infine chi non c'è più ma sarebbe dovuto essere qui, questo lavoro è dedicato anche a lui.