ALMA MATER STUDIORUM
UNIVERSITÀ DEGLI STUDI DI BOLOGNA

Seconda Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

LA PIATTAFORMA BTSSOC-CELLULAT PER LA SIMULAZIONE DI SISTEMI BIOCHIMICI

Elaborata nel corso di: Sistemi Multi-Agente

Tesi di Laurea di:
ANDREA BOCCACCI

Relatore:
Prof. ANDREA OMICINI

Correlatori:
Dr PEDRO PABLO GONZÁLEZ PÉREZ
Dr Ing SARA MONTAGNA

ANNO ACCADEMICO 2010–2011
SESSIONE III

PAROLE CHIAVE

Biochemical Tuple Space

TuCSoN

Agenti

Simulazione Cellulare

A...

Indice

Introduzione

Il lavoro svolto in questa tesi verte sullo sviluppo e l'integrazione del modello teorico conosciuto come *Biochemical Tuple Spaces for Self-Organizing Coordination*, in breve *BTSSOC*, in una piattaforma completa, chiamata *BTSSOC-Cellulat*, per la simulazione di sistemi biochinici, sviluppata utilizzando i linguaggi *Java , Prolog, TuCSoN e ReSpecT*.
La simulazione *in silico*, ovvero quella che si occupa di predizioni e modellazione di sistemi biologici in laboratori virtuali, è piuttosto utile e determinante per una miglior comprensione di quei problemi che risultano aperti o parzialmente irrisolti in questo ambito. Al fine di raggiungere questo obiettivo, il modello di questi sistemi deve essere organico, coerente e fedele alla realtà, inoltre per la natura intrinseca del problema, esso deve essere aperto e facilmente estendibile, al fine di aggiungere o modificare parti o comportamenti che via via devono essere integrati, a causa di quelli che sono i *feedback*, che vengono dalla ricerca in ambito biologico, in un processo di sviluppo a spirale.
L'idea alla base di questo percorso è da ricercarsi in un'area dell'informatica molto distante, quella dei sistemi distribuiti; i problemi nati da una nuova generazione di sistemi computazionali hanno come caratteristica principale la pervasività, dovuta ai numerosi e disomogenei i dispositivi tecnologici (e.g. telefonini, computer portatili, navigatori, tablet, etc...), che compongono sistemi complessi, in cui l'utente è anche e soprattutto produttore di informazione. In questo contesto, detto anche *nube computazionale* è impensabile lo sviluppo di un controllo, centralizzato e predefinito, per questo motivo sono stati teorizzati e progettati nuovi modelli di coordinazione, ispirati a sistemi naturali, i quali hanno caratteristiche importanti, come l'attitudine ad autogovernarsi, la topologia flessibile, la resistenza ai cambiamenti, la

correzione automatica degli errori e l'apertura ad evoluzioni future, senza che il sistema perda caratteristiche e proprietà.
L'idea dei *BTSSOC* è quella di utilizzare questi sistemi moderni di coordinazione per modellare i sistemi biochimici, al fine di sviluppare previsioni affidabili sugli stessi. Nello specifico le principali tecnologie utilizzate per l'implementazione *TuCSoN* e *ReSpecT*, come sarà spiegato in seguito in questo documento, offrono tutte le funzionalità necessarie, e le caratteristiche richieste dalla natura dei sistemi biologici, di cui i sistemi biochimici fanno parte.
Lo sviluppo di un sistema completo a partire da questo modello è il primo passo che permette l'attivazione del processo a spirale descritto in precedenza, da cui partiranno nuove idee e proposte per l'estensione del modello di partenza.
La bioinformatica, ormai non più nascente, ma assai consolidata branca dell'informatica, ha permesso un dettagliato studio e una miglior comprensione di quelli che sono i problemi biologici ancora aperti, forte anche delle caratteristiche peculiari dell'ambito simulativo, ossia il risparmio di tempo e di fondi necessari per la ricerca, che ormai troppo spesso sono criteri determinanti per determinare la bontà di quella che è la ricerca scientifica.
Questa trattazione, dopo un breve riassunto sugli applicativi presenti, tratterà i passi svolti per lo sviluppo della piattaforma *BTSSOC-Cellulat*, spiegando le motivazioni che hanno portato ad effettuare determinate scelte.

Capitolo 1

La modellazione e la simulazione di sistemi biologici

I problemi legati alla biologia dei sistemi sono stati affrontati per diversi anni a partire da osservazioni e da esperimenti su casi reali, come vuole il metodo scientifico. Tuttavia le proprietà di questa tipologia di sistema sono proprie dei sistemi complessi, rendendo molto difficile prevedere il loro comportamento. I sistemi complessi, infatti, sono caratterizzati da interazioni locali o comunque di breve raggio, che provocano cambiamenti nella struttura complessiva, che sono limitati solitamente da un attrattore. Questo fa sì che anche piccole perturbazioni o anomalie contribuiscano nei cambiamenti globali del sistema in maniera non deterministica. La cellula è un sistema complesso, infatti, come viene descritto in letteratura, piccole anomalie (e.g. uno stato patologico), possono propagarsi nel sistema determinando un'evoluzione del sistema, verso uno stato imprevisto.
Questo tipo di problematiche rendono particolarmente utile e necessario un approccio simulativo, al fine di poter prevedere e comprendere con più precisione il comportamento di tali sistemi, che ha il grande vantaggio di mostrare uno stato futuro, al seguito di una definizione di interazioni locali, spostando quindi l'attenzione sulla loro modellazione.
Negli ultimi anni questo modo di affrontare il problema ha creato una

nuova disciplina la *bioinformatica*, che si occupa appunto di modellare ed effettuare questo tipo di previsioni di sistemi biologici, dette comunemente *in-silico*.
La parte di modellazione e le tecnologie utilizzate nella gestione dei sistemi complessi, sono tuttavia ancora fonte di dibattito, questo ha prodotto numerosi approcci al problema, come vedremo in seguito.

1.1 Modelli di sistemi biologici

I modelli che si trovano in letteratura si possono essere classificati in due tipologie, la prima, meno importante, fornisce un modello statico, ovvero una fotografia della struttura del sistema in un determinato istante di tempo, la seconda, si occupa invece di fornire una modellazione dinamica al processo. Tra i modelli dinamici è possibile distinguere diversi approcci

- continui e discreti
- quantitativi e qualitativi
- deterministici e stocastici
- livello di dettaglio della descrizione del sistema

che come è possibile immaginare non sono tra loro mutuamente esclusivi e possono coesistere in un approccio ibrido.
I modelli discreti assumono che il tempo di simulazione scorra secondo prefissati intervalli temporali equidistanti tra loro. Un cambiamento o un evento è quindi associato ad un determinato intervallo temporale, e al di sotto della sua granularità due eventi sono tra loro contemporanei. In questo tipo di approccio spesso solo un numero limitato di azioni può essere svolto in un intervallo temporale.
Al contrario i modelli continui, fanno in modo che il sistema evolva continuamente nel tempo. Questo tipo di approccio spesso fa uso di equazioni differenziali, che rendono questa tipologia di modello computazionalmente più onerosa rispetto alla controparte. Un ruolo importante è inoltre svolto dall'integrazione numerica, che ben approssima il comportamento su macchine reali.

I modelli deterministici, determinano in modo certo l'evoluzione di un sistema a partire dalle condizioni iniziali, ma come abbiamo visto precedentemente questo non è mai verificato nei sistemi complessi, come quelli biologici, quindi il loro contributo è significativo, a patto che abbiano una componente probabilistica al loro interno.
I modelli stocastici, invece, sono molto più realistici perché ben modellano perturbazioni che il sistema subisce. Un esempio di questa tipologia di sistemi è dato da quelle che sono le *equazioni differenziali stocastiche*, che determinano la probabilità che una combinazione di molecole reagiscano tra loro in un determinato intervallo temporale.
I modelli quantitativi assumono che una determinata variabile sia distribuita uniformemente, al contrario i modelli qualitativi sostengono che le reazioni siano modellate da una costante cinetica, con cui esse occorrono nel tempo, che è invariante. Il livello di dettaglio, infine, definisce la visione che si ha del sistema, se a livello microscopico o macroscopico.

1.2 Modelli computazionali adottati nei sistemi biologici

Nel corso degli anni, sono stati sviluppati diversi modelli computazionali, per risolvere i problemi simulativi. Questi approcci sono nate a partire da visioni diverse del problema, scegliendo il sistema di comunicazione,di coordinazione e di processamento delle interazioni locali, come meglio si addice alle caratteristiche del sistema che si vuole enfatizzare. Gli approcci sono diversi su molti aspetti, uno dei più importanti è senza dubbio l'approccio parallelo sequenziale o distribuito, che influisce sia sulla durata della simulazione che sulla possibile distribuzione del calcolo su più sistemi.
La tabella 1.1 sintetizza le principali caratteristiche dei più significativi modelli computazionali, riportando: l'idea che ha ispirato l'approccio, la capacità cognitive che possono essere modellate, le tipologie di elaborazione delle informazioni supportate e la rete di segnalazione presa in considerazione.

Tabella 1.1: Sommario dei più significativi approcci computazionali al problema della segnalazione intracellulare

Approccio computazionale	**Idea ispiratrice**	**Capacità cognitive**	**Processing data supportati**	**Rete di segnalazione modellata**
Reti Booleane	La cellula può essere modellata come una rete di componenti dotati di due soli stati, che interagiscono tra loro. Lo stato di ciascun elemento dipende da una particolare funzione Booleana.	Computazione logica Booleana	Parallelo	Segnalazione intracellulare, reti genetiche
Sistemi esperti	Modellare le interazioni (e.g., attivazione, fosforilazione, etc.) tra elementi della rete di segnalazione attraverso production rules.	Inferenza e deduzione knowledge-based	Sequenziale, parallelo	Segnalazione intracellulare
Automi cellulari	Modellare le interazioni tra le cellule o le molecole come una matrice, in cui lo stato di ciascun elemento dipende dagli stati degli elementi vicini.	-	Parallelo	Segnalazione extracellulare, segnalazione intracellulare
Reti di Petri	La cellula e' vista come un grafo connesso con due tipi di nodi. Uno tipo rappresenta gli elementi (e.g., molecole segnale, proteine), l'altro rappresenta le transizioni (e.g., attivazione).	-	Sequenziale, concorrente	Segnalazione intracellulare
Reti neurali artificiali	Le proteine nella rete di segnalazione sono viste come neuroni artificiali in reti neurali. Ogni proteina riceve input pesati, produce un output e ha un valore di attivazione.	Memoria, learning, pattern recognition	Distribuito, parallelo, emergente	Reti di proteine, segnalazione intracellulare
Sistemi distribuiti (agenti)	La cellula e' vista come una società di agenti che lavorano in parallelo. Gli agenti comunicano tra loro tramite messaggi.	Memoria, learning, pattern recognition, gestione di fuzzy data, selezione adattativa	Distribuito, parallelo, emergente	Segnalazione intracellulare
Modelli continui	Il comportamento delle popolazioni di proteine dipende da variabili fisico-chimiche (i.e., affinità e concentrazione). La dinamica del modello e' data dalla variazione di queste variabili rispetto al tempo continuo, il tutto implementato da eq. differenziali.	-	Distribuito, parallelo, emergente	recettori EGF, NGF e della via MAPK. Sistemi Ca, PKC, PKA, CaM e CaMKII

1.3 Analisi di alcune piattaforme per la simulazione intracellulare esistenti

Questo capitolo è scritto prendendo come riferimento il lavoro di *Rui Alves, Fernando Antures e Armindo Salvator* su *Tools for kinetic modeling of biochemical networks*[20].
In questa sede saranno presentate brevemente alcune piattaforme standalone, presenti allo stato di fatto, che hanno lo stesso obiettivo della proposta di questa tesi, vedremo quindi analogie e differenze tra i vari approcci al problema.
Le applicazioni che vengono prese in esame in questa sede sono *CellDesigner*[4], *CellWare*[5], *COPASI*[6], *Dizzy*[8], *Dynetica*[10], *GEPASI*[11], *Pasadena Twain*[23], *PLAS*[18].
Di queste, le prime cinque, sono disponibili per i principali sistemi operativi, Windows, Linux e Mac OS X, le ultime invece sono disponibili solo per Windows. Di queste invece solo tre sono scritte in *Java*, *CellWare*, *Dizzy*, *Dynetica*.
Solo due di queste piattaforme sono invece OpenSource, Dizzy e Pasadena Twain, quindi modificabili per esigenze particolari. Sono stati studiati appositamente alcuni linguaggi di configurazione di sistemi biologici, in modo che i modelli di simulazioni creati siano il più possibile condivisibili tra le varie piattaforme, tra questi il più comune è senza dubbio *SBML*[13], che potrebbe essere considerato uno standard de facto. Infatti i modelli scritti in quel linguaggio non sono compatibili solo con tre dei programmi presi in considerazione ovvero Dynetica, Pasadena Twain, PLAS. Avere un linguaggio comune, facilmente comprensibile e modificabile, sia a livello software che a livello umano è particolarmente importante per l'integrazione di queste applicazioni che utilizzano tecniche e modelli diversi tra loro. Un limite di questo linguaggio è che non ha uno standard per reazioni sorgente (*souce*), ovvero quelle leggi che non hanno reagenti in ingresso, e recettore (*sink*), ovvero quelle che non hanno prodotti in uscita. Alcuni dei programmi descritti in questo capitolo utilizzano comunque questa tipologia di leggi, quindi i modelli prodotti potrebbero discostarsi leggermente da questo standard.
Parlando di interfacce utente ci sono due diverse metodologie di inserimento di dati, il primo è basato su finestre di dialogo, il secondo

su disegni di diagrammi. Il primo metodo richiede di specificare leggi compartimenti intracellulari, funzioni di vicinato e espressioni di rate tramite interfaccia grafica, il secondo invece sposta l'attenzione sul disegno di diagrammi, al posto di scrivere le reazioni chimiche per esteso, a parte per alcuni parametri, come il rate delle leggi che vengono comunque inseriti tramite finestra di dialogo. Dei programmi presi in considerazione BASIS, COPASI e GEPASI utilizzano il primo metodo di interazione con l'utente, gli altri hanno un approccio a diagrammi. CellDesigner permette la definizione di reazioni chimiche con rapporti stechiometrici non interi e ed l'unico che ha questa funzionalità totalmente funzionante. Anche se COPASI riesce a fare lo stesso per quanto riguarda la parte iniziale, ovvero la definizione della legge, non lascia definire espressioni di rate adeguate, quindi la sua gestione di questo aspetto non è perfetta.
Dynetica a differenza degli altri non permette la definizione di *metaboliti* la cui concentrazione è costante nel tempo. Anche Dizzy ha qualche problema in questo senso, in quanto non permette che questo processo sia attivato da una serie di reazioni chimiche.
Per quello che riguarda i motori di simulazione, ve ne sono di due tipi, stocastici e deterministici. I primi sono senza dubbio più precisi e realistici, perché considerano ogni molecola come una entità discreta nel sistema e soprattutto gestiscono le collisioni spaziali tra molecole come eventi, questo permette che una reazione diventi effettiva solo in caso di collisione. Tuttavia questi modelli richiedono uno sforzo maggiore all'utente come la definizione della massa delle molecole e la loro struttura o comunque la loro legge cinetica. Il più grande svantaggio di questo tipo di simulatori è che sono computazionalmente onerosi.
I motori deterministici invece assumono che le molecole siano in gran numero, disposte uniformemente all'interno del compartimento e quindi siano disponibili ovunque, in questi sistemi si parla appunto di concentrazione, al posto che di singole molecole. Altri problemi risultano quando le soluzioni sono ionizzate, questo tipo di modellazione infatti non è adatto a topologie ordinate o con uno psudo-ordine dettato da una legge elettrica. Dei programmi elencati solo VirtualCell tiene conto delle ionizzazioni e delle leggi elettriche.
Nessuna delle applicazioni qua citate fornisce comunque una buona simulazione con un motore stocastico che tenga conto delle variabili

spaziali.
Come possiamo vedere da questa breve relazione su questa tipologia di sistemi, molti sono i problemi aperti e ancora di più le soluzioni esplorate dagli applicativi disponibili. Nonostante una grande mole di idee sulla gestione della maggior parte degli aspetti fisico-chimici delle operazioni svolte all'interno della cellula, si hanno grossi problemi a uniformare queste proprietà in un unico applicativo. Quello che si vede ora sono infatti programmi validi per modellare certi aspetti, ma che ne tralasciano altri. Punto fondamentale a questo punto dello sviluppo è quello di favorire l'integrazione tra questi sistemi, e un buon passo in questa direzione è stato mosso dal linguaggio *SBML*[13].

Capitolo 2

Il modello BTS-based per la simulazione delle vie di segnalazione intracellulare

Come è stato introdotto nel capitolo precedente esistono varie applicazioni che hanno esplorato il campo della segnalazione intracellulare. Quello che spesso capita in questi sistemi è che ad un certo punto dello sviluppo, si arriva ad un livello di difficoltà crescente nell'integrazione di nuove funzionalità al modello, funzionalità che spesso necessitano di strumenti di coordinazione di alto livello. Quello che capita in natura è invece che la singola entità del nostro sistema si comporta secondo quelle che sono le leggi fisico-chimiche. In altre parole le nostre entità necessitano di un coordinatore (metafora per le leggi) che le organizzi, che allo stesso tempo risulti robusto, modificabile ed estendibile, per rispondere alle crescenti esigenze nel settore, infatti come possiamo vedere in figura 2.1, siamo di fronte innegabilmente ad un cosidetto sviluppo a *spirale*, dove i feedback ricevuti devono man mano integrare il modello e produrre nuovi problemi da risolvere.
Il modello *BTS-based* (Biochemical Tuple Spaces)[14], come suggerisce il nome, utilizza quelli che sono gli *spazi di tuple linda-like*, come modello di coordinazione, risolvendo nativamente i problemi dovuti all'interazione tra i vari componenti del sistema. Questo rappresenta un notevole vantaggio, in quanto lo sviluppo si limita a risolvere problemi a livello di requisiti di sistema, eliminando quelli che sono

problemi prettamente informatici, come quelli di coordinazione.

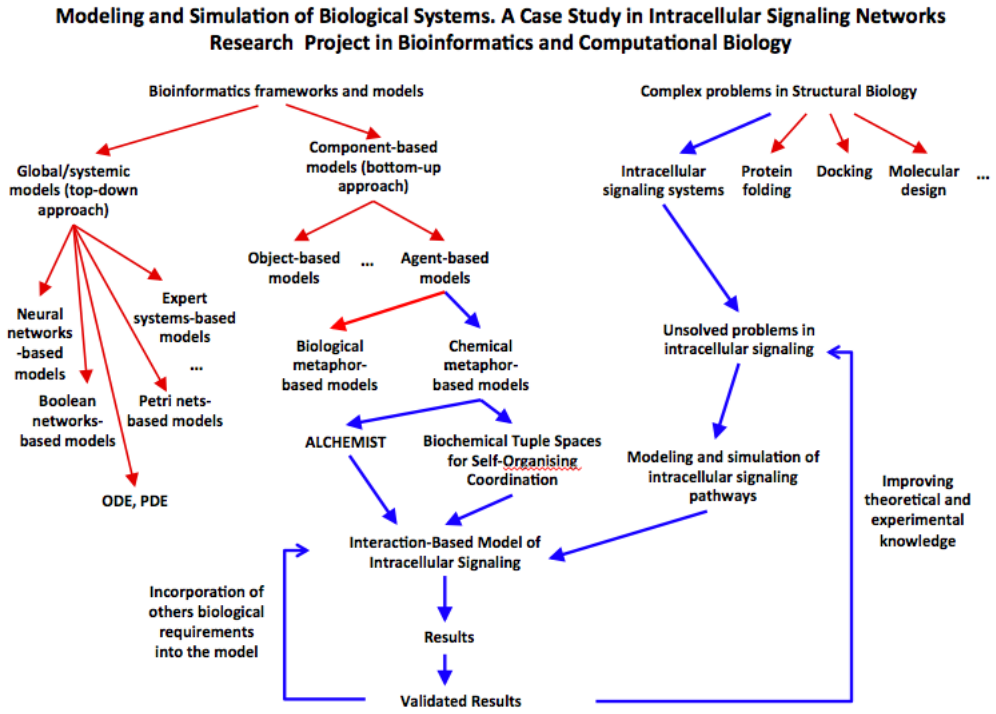

Figura 2.1: Stato attuale dello sviluppo dei sistemi biochimici

2.1 Analisi dei punti chiave del problema

La descrizione del problema che sarà fornita in questo capitolo, non vuole essere esaustiva nel descrivere il problema reale e le sue tante sfaccettature, ma vuole solo mettere in mostra quelli che sono i sottoproblemi più importanti da risolvere per raggiungere una buona approssimazione simulativa della realtà.
La cellula, al fine di svolgere le funzioni vitali, reagisce a stimoli esterni, adattandosi e rispondendo prontamente. Il ruolo chiave di questa attività è svolto dalle vie di simulazione intracellulare, che rispondendo

a stimoli esterni attiva o disattiva particolari recettori (ovvero alcuni tipi di proteine), rispondendo all'impulso. All'interno della cellula quello che avviene è più complesso, infatti essa è divisa in vari compartimenti cellulari, nello specifico *membrana, regione iuxta-membranale, cytosol e nucleo*, che comunicano tra loro mediante strade, attivate da catene di segnali, che sono appunto le vie di segnalazione.
I recettori sono presenti in ogni compartimento intracellulare, quando essi vengono attivati da una o più fosforilazioni, generano altre proteine di segnalazione, che a loro volta si legano ad altri recettori.
Il risultato di questa catena di segnali è quasi sempre l'inibizione (*feedback negativo*) o l'aumento (*feedback positivo*) di produzione di una proteina.
Da questa breve e semplificata descrizione è possibile notare che le entità in gioco nel nostro sistema sono compartimenti intracellulari, segnali e proteine. Il problema suppone inoltre che vi sia una comunicazione tra le proteine e i compartimenti cellulari mediante segnali, questo implica una coordinazione tra le entità in gioco. Ci serve, inoltre, per risolvere il problema un modello chimico valido per la gestione di reazioni all'interno di un compartimento intracellulare, che ricercheremo nel lavoro di Gillespie[9], che, come vedremo, modella le reazioni che avvengono all'interno di una soluzione.

2.2 Architettura del modello BTS-based

Nel paragrafo precedente è stata posta l'attenzione al problema in generale, ora mostreremo come il modello proposto si adatta naturalmente al problema.
Innanzitutto è stato illustrato come il carico della coordinazione sia legato al concetto di compartimento intracellulare, che sarà governato da leggi che modellano il suo comportamento. Viene quindi naturale un'associazione compartimento - spazio di tuple, in qualità del suo ruolo sia di località che di mezzo di coordinazione. La cellula in questa metafora sarà quindi composta da tanti spazi di tuple, quanti sono i suoi compartimenti, come si vede in figura 2.2.
Come inoltre descritto in [16], il problema illustrato è proprio dei

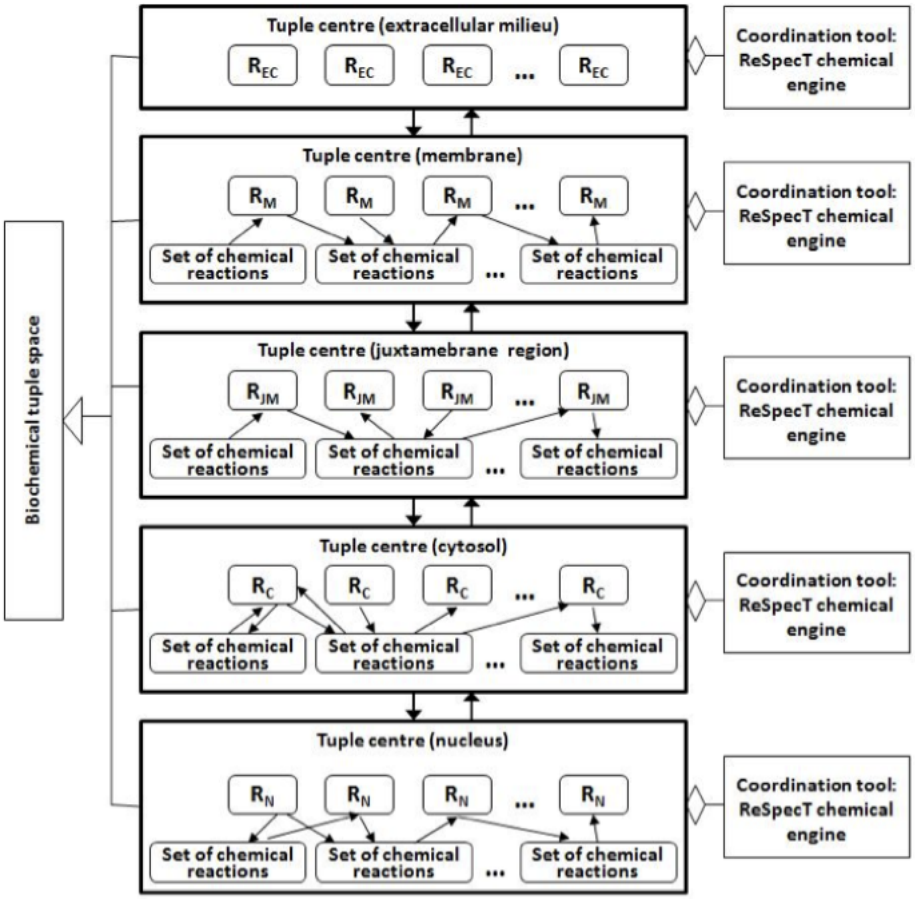

Figura 2.2: Architettura del modello BTS-Based

sistemi distribuiti i quali devono avere le proprietà di:

- *Situatedness*: la cellula deve rilevare la sua struttura, interpretare gli stimoli, attivare o disattivare proteine, che controllano le interazioni.
- *Adaptivity*: la cellula deve adattarsi dinamicamente ai cambiamenti esterni
- *Prosumption and Diversity*: ogni cellula ha la sua funzione e svolge particolari attività, ma esse possono cambiare nel corso

del tempo

- *Eternity*: deve essere tollerante ai cambiamenti, sia strutturali, che funzionali, che di composizione.

- *Topology*: i compartimenti intracellulari, hanno un contesto spaziale, e, in vista di future applicazioni, reti di cellule hanno anche esse contesti spaziali

- *Locality*: ogni cellula ha la visione del suo stato complessivo come aggregazione di transizioni di stato locali

- *Time*: in ottica simulativa, il concetto di tempo è, come vedremo, di vitale importanza

Queste proprietà, inoltre, rendono la parte informatica dell'applicativo scalabile, quindi completamente aperta a nuove funzionalità richieste per sviluppi futuri.
Le principali metafore alla base del modello saranno illustrate nei prossimi paragrafi.

2.3 I compartimenti e le strutture intracellulari come centri di tuple

Per definizione un *centro di tuple* è uno spazio di tuple programmabile. Nei centri di tuple al fronte dell'inserimento o rimozione o lettura di tuple dallo spazio, viene associato un comportamento, che in generale si traduce con la trasformazione di una o più tuple. L'implementazione specifica del linguaggio utilizzato per programmare i centri di tuple, utilizzata in questo lavoro, è *ReSpecT*[1]. Questo linguaggio permette al centro di tuple di *reagire* ad alcuni eventi, come l'inserimento, la rimozione, o la lettura di una tupla dallo spazio, o il raggiungimento di un dato istante temporale. E' possibile associare azioni a questi eventi scritte in linguaggio *Prolog* e/o azioni presenti nella specifica linda per le modifiche alle tuple del centro. In questo modo si migliora il modello di coordinazione dello spazio, dandogli un comportamento ben definito, rendendolo parte attiva del sistema (gli spazi di tuple

sono solo basi di conoscenza in cui sono presenti i dati, un'aggregazione tra i concetti di database e di monitor, quindi di fatto entità passive). ReSpecT estende inoltre la specifica *linda* aggiungendo tre nuove primitive, utili per una programmazione dinamica del centro di tuple:

- in_s : aggiunge una specifica ReSpecT al centro di tuple
- out_s : rimuove una specifica ReSpecT dal centro di tuple
- rd_s : legge senza rimuovere una specifica ReSpecT dal dentro di tuple

Con queste nuove primitive il set di istruzioni possibili è ben più ampio, ed è possibile modificare il comportamento del centro di tuple nel tempo.
Un'altra funzionalità dei centri di tuple implementati nell'infrastruttura *TuCSoN*[2] è data dalla presenza di primitive che abilitano la comunicazione tra i centri di tuple.
Dopo una breve, ma dovuta, introduzione su quelli che sono i centri di tuple e la loro importanza possiamo capire come questi possano aiutarci nel risolvere i problemi citati in precedenza. Il problema ci suggerisce che un compartimento intracellulare è di fatto un'entità guidata da leggi fisico-chimiche che eseguono azioni sugli oggetti presenti all'interno. L'analogia con i centri di tuple è quindi praticamente inevitabile, un compartimento intracellulare è un centro di tuple, con un compartimento guidato da leggi, al cui interno sono presenti le altre entità del problema, ovvero *proteine e segnali*. Inoltre la possibilità dei centri di tuple di comunicare tra loro risulta molto comoda nel propagare segnali a compartimenti vicini, abilitando la comunicazione tra i vari strati della cellula.
Un ulteriore vantaggio di questo tipo di modellazione è che né i compartimenti intracellulari, né la loro topologia viene definita a priori. Questo elemento è molto importante perché l'applicazione che verrà costruita sarà indipendente da ogni possibile struttura cellulare, lasciando aperto il campo della loro definizione alla ricerca.

2.4 Gli elementi di segnalazione come lista di reazioni chimiche

La trasduzione dei segnali intracellulari è dovuta come abbiamo visto in precedenza alla fosforilazione di un recettore, producendo una proteina di segnalazione. In generale quindi un segnale è attivato da una reazione chimica, che come abbiamo visto in precedenza dovranno essere eseguite dai compartimenti. I motivi di questa scelta sono dovuti al fatto che l'esecuzione di una legge chimica porta a trasformazioni all'interno del compartimento, trasformazioni che vengono subito recepite dallo stesso e attivano di conseguenza nuovi comportamenti. Questa proazione dei compartimenti modella quindi bene quella che è la rete di segnalazione, con un legame certo di causa-effetto. Le leggi chimiche inoltre possono essere eseguite solamente al fronte della trasduzione di certi segnali, quindi quelle che non sono attivate al momento non concorrono all'evolversi dello stato del sistema in quell'istante. Vi sono diverse tipologia di leggi chimiche, come illustrato in [16]:

- Attivazione collaborativa: si ha quando due o più proteine di segnalazione concorrono nell'attivarne una terza

$$[x]A* + [y]B* + [z]C \rightarrow [z]C*$$

- Attivazione alternativa: si ha quando la stessa proteina può essere attivata da due segnali diversi

$$[x]A* + [y]B \rightarrow [y]B*$$

$$[x]C* + [y]B \rightarrow [y]B*$$

- Formazione complessa: si ha quando vi è un'aggregazione tra più proteine, anche non attivate

$$[x]A* + [y]B* + [z]C \rightarrow ABC$$

dove i componenti delle reazioni seguiti da un asterisco sono proteine di segnalazione.

2.5 I segnali extra e intracellulari come reagenti

Ultima metafora del modello è data dalla corrispondenza tra segnale e reagente di una reazione, questo è corretto in virtù di quanto precedentemente detto, ovvero quando avviene una reazione di fosforilazione di un recettore, esso inizia la produzione di proteine di segnalazione, se quindi prendiamo in esame la metafora precedente capiamo che i segnali in ingresso, ovvero quelli che abilitano la fosforilazione, possono essere trattati come reagenti della reazione, mentre i segnali in uscita, possono essere appunto trattati come prodotti della stessa. Questi segnali popolano quindi il centro di tuple in qualità di reagenti con una determinata concentrazione, favorendo o bloccando quelle che sono le leggi che governano il comportamento del compartimento. In sintesi tutto il comportamento e l'evoluzione del sistema dipende dalla concentrazione e dalla presenza di questi segnali, come avviene appunto nei sistemi biologici.

Capitolo 3

La piattaforma BTSSOC-Cellulat

In questo capitolo viene fornita una descrizione di quella che è la piattaforma sviluppata in questo lavoro, *Biochemical Tuple Space for Self-Organizing Coordination - Cellulat*, da qui in avanti BTSSOC-Cellulat.

3.1 Architettura di sistema

Come mostrato in figura 3.1, a livello strutturale il sistema è composto da tanti centri di tuple, quanti sono i compartimenti cellulari o strutture cellulari definiti dall'utente, come scelto dal lavoro precedente di Marco Sbaraglia[21]. Nell'ottica di creare un sistema nuovo, e più orientato ai bisogni dell'utente, occorre integrare questa composizione con altri componenti, secondo il pattern *Model View Control*.
Per questo motivo è stato introdotto un *agente*, che si occupa di gestire l'interazione utente-centro di tuple. L'agente in questione si occupa di mediare la comunicazione tra utente e sistema, rendendola possibile, e limitandola nei punti critici, a cui l'utente non dovrebbe avere accesso.
Per la gestione degli eventi e dell'output del sistema, invece è stato predisposto un agente per ogni compartimento che restano in perenne attesa di queste informazioni, per elaborarle, oppure notificarle alla *view*.
Inoltre gli eventi sono collezionati in una struttura dati che funge da monitor, in modo tale da essere accessibili ai vari agenti che compon-

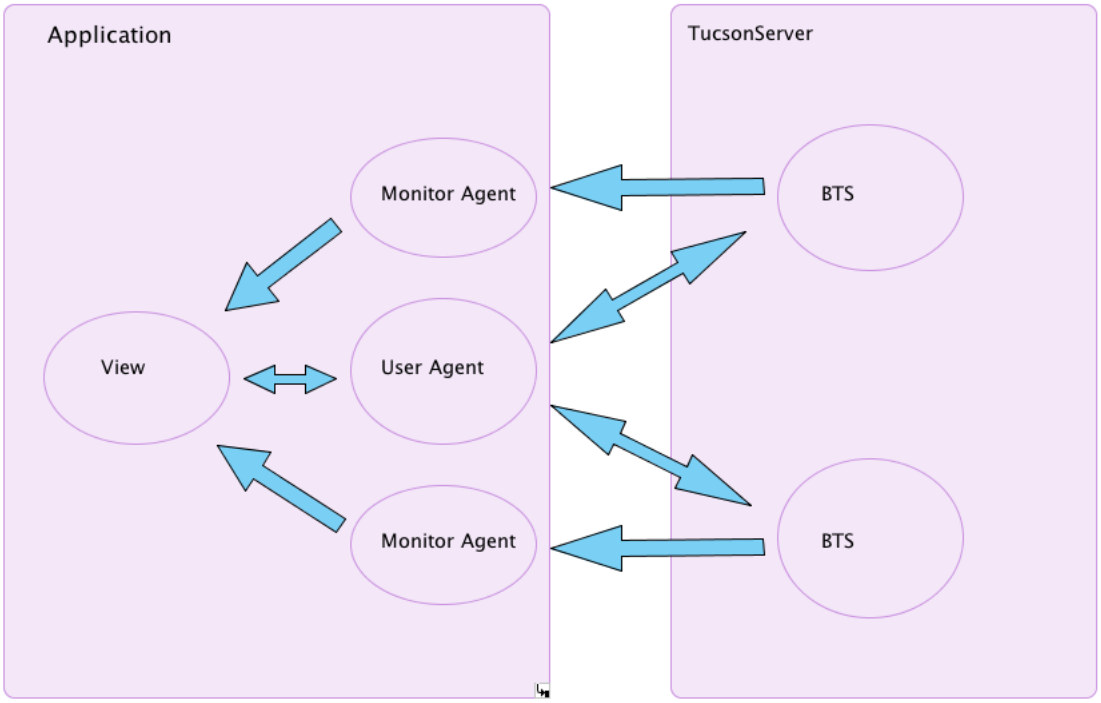

Figura 3.1: Macro architettura del sistema

gono il sistema, in questo modo si permette all'utente di richiedere informazioni sullo storico di quello che avviene nel sistema.

3.2 La specifica ReSpecT del motore chimico

3.2.1 Descrizione del modello di partenza

Il punto di partenza di questo progetto è il risultato del lavoro svolto da *Marco Sbaraglia*[21], che durante il suo percorso ha sviluppato una base funzionante per un generico motore biochimico, che, nel nostro caso, descrive le interazioni fra i componenti di segnalazione intracellulare, e la simulazione di una specifica via di trasduzione del segnale. Il modello, scritto in linguaggio ReSpecT[1], ha il grosso pregio di adattarsi alle diverse richieste in ambito simulativo e di essere sem-

plice, e di conseguenza facilmente modificabile, per dare risposta ai nuovi bisogni in ambito biologico.
L'idea alla base dell'astrazione è stata sviluppata nel software *Cellulat*[17], che propone di descrivere ogni compartimento cellulare come la porzione di una lavagna utilizzata a scopo di sincronizzazione, su cui pubblicare le molecole presenti, nei sistemi multi-agente questa soluzione ai problemi di sincronizzazione si dice appunto *backboard based*.
Questo approccio, però ha la grossa limitazione di spostare la parte di controllo dal mezzo di sincronizzazione, agli agenti che vi interagiscono, quindi servono agenti che eseguono reazioni chimiche, e agenti che si occupano di gestire e coordinare altri agenti.
Il lavoro precedente invece è stato quello di cambiare questa astrazione, e descrivere invece ogni compartimento all'interno di uno spazio di tuple. In questo modo il controllo di tali azioni si sposta all'interno del mezzo di sincronizzazione, semplificando notevolmente quello che è il modello, fornendogli inoltre una ben maggiore scalabilità.
Oltre ai reagenti, con questo approccio, si introducono nel centro di tuple anche le leggi biochimiche (ovvero reazioni) relative al compartimento che modella. In questo modo si sposta la parte di controllo delle reazioni all'interno, rendendo la gestione più facile e mirata.
E' possibile descrivere la topologia dei compartimenti mediante funzione di vicinato, ma questa astrazione è più orientata ad una topologia cellulare, piuttosto che intracellulare.
Le leggi devono essere scritte come termini *Prolog* con arità 3, in cui il primo argomento è la lista dei reagenti, il secondo è il rate della legge e il terzo è la lista dei prodotti. Il rate della legge descrive la probabilità con cui quella legge verrà selezionata, tra quelle eseguibili, secondo l'algoritmo di *Gillespie*[9].
Un tipo speciale di leggi sono le *leggi istantanee*, ovvero leggi a rate infinite, le quali hanno la precedenza su tutte le altre e vengono eseguite immediatamente.
Una terza tipologia sono invece le leggi temporali, ovvero leggi che vengono eseguite dopo un intervallo di tempo definito.
Sia le leggi temporali che quelle istantanee non concorrono all'algoritmo di *Gillespie*, quindi la loro scelta avviene ogni volta che si verificano le condizioni di esecuzione, quindi le leggi istantanee vengono eseguite ogni volta che nel compartimento sono presenti tutti i reagenti ne-

cessari, le leggi temporali, invece, ogni volta che trascorre l'intervallo temporale definito.
Un'altra funzionalità importante del motore, è quella di pubblicare i prodotti in un compartimento vicino, mediante le cosidette *firing rules*, nel modello di partenza, la scelta è equiprobabilistica, ma in futuro potrebbe essere comodo modificare le *firing rules* per pubblicare i prodotti su un vicino specifico.
Se non vi sono leggi attivabili, il motore chimico si sospende, in attesa di nuovi reagenti, quando questi vengono pubblicati nel centro di tuple, infatti viene risvegliato e vengono rivalutate le reazioni da eseguire. Questo è possibile grazie a delle tuple speciali *engine_suspended* e *trigger_engine/2*. La prima indica che il motore è fermo per mancanza di reagenti, la seconda invece si occupa di risvegliare il motore dopo un lasso di tempo casuale, questo per simulare la quantità di tempo trascorsa tra due reazioni nella realtà.

L'algoritmo di Gillespie

Il cuore del sistema è l'algoritmo di Gillespie[9], che secondo la ricerca, ben modella i sistemi reali. Secondo il suo lavoro una soluzione in cui avvengono reazioni chimiche può essere simulata mediante catene di Markov tempo-continue. La versione implementata dall'algoritmo di Gillespie in questo lavoro è una variazione delle catene di Markov tempo-discrete, dove le transazioni non sono attivate dal *rate* degli archi, ma da una probabilità. Se una transizione modella l'esecuzione di una legge, sul rispettivo arco avremo quindi la probabilità che la legge venga scelta. Le transazioni inoltre in questo modo non hanno bisogno di un tempo continuo per essere attivate a patto che le probabilità uscenti da un nodo (stato della soluzione) sia unitario.
Il rate di una legge rappresenta quindi il valor medio della frequenza di esecuzione di una legge, o se preferiamo l'inverso del periodo medio di attesa di attivazione.
In seguito sono elencati i passi di questo algoritmo per una migliore comprensione del sistema.

1. Viene calcolato il rate di ogni singola legge attivabile secondo la

formula :

$$Rate = ConstantRate * \prod_{i=1}^{num\_reagents} \binom{Mol_i}{RequiredMol_i} \quad (3.1)$$

Il rate con cui verrà scelta la legge è pari al della legge moltiplicato per la produttoria dei coefficienti binomiali delle moli di ogni reagente nel compartimento sul numero di moli richiesto dello stesso dalla legge. Se la legge non è attivabile per mancanza di un reagente avrà rate nullo.

2. Viene fatta la somma totale dei rate delle leggi attivabili, chiamato RTot.

3. Vengono ordinate le leggi per valori di rate decrescenti

4. Viene scelto un numero casuale tra 0 e 1 che chiamiamo ψ.

5. Tra le n leggi attivabili dal punto 1 viene scelta la i-esima se:

$$\psi <= \frac{\sum_{i=1}^{n} Rate_i}{RTot} \quad (3.2)$$

Per l'ultima legge il valore della sommatoria è unitario, quindi, se ci sono leggi attivabili, una di esse è sempre scelta.

6. Scelto un ulteriore numero random tra 0 e 1, chiamato τ si blocca l'esecuzione delle leggi per un tempo pari a

$$waste\_time = \frac{-ln(\tau)}{RTot} \quad (3.3)$$

3.2.2 Problemi noti e limitazioni trovate

Dopo una breve descrizione del modello iniziale, questa sezione si occupa di analizzare il funzionamento del motore di partenza.
Innanzitutto erano presenti dei piccoli problemi nella specifica, che non erano emersi in un caso di studio statico, ma solamente dopo un utilizzo un po' più complesso.
Introducendo i reagenti e le leggi velocemente via software, ad esempio,

il motore chimico veniva sospeso, anche se vi erano reazioni applicabili. Questo problema era dato da un errato calcolo del prossimo istante di tempo in cui il motore chimico doveva essere *triggerato*.
Infatti, giustamente, ogni qualvolta un vicino, pubblica un reagente sullo spazio di tuple che stiamo osservando, il motore deve rivalutare quelle che sono le leggi applicabili e quindi decidere nuovamente quale legge applicare dopo un certo lasso di tempo. Un errore appunto nel calcolo del nuovo lasso di tempo (di sei ordini di grandezza) generava appunto il problema.
Altri problemi sono stati invece riscontrati nella gestione delle *firing rules*, infatti i reagenti venivano pubblicati, sullo stesso spazio di tuple e non su uno dei vicini, questo problema ha reso necessari diversi test (anche sullo stesso TuCSon), per verificare il funzionamento corretto di predicati quali @ e *?* che per la loro semantica si occupano della comunicazione tra centri di tuple. Tuttavia il problema era da ricercare altrove, infatti era un problema di inizializzazione del sistema, che pubblicava nel centro di tuple due copie della stessa tupla, che modellava le relazioni di vicinato, per cui nello spazio era sempre presente una tupla che indicava la mancanza di vicini.
Un terzo problema era invece relativo alla gestione dei reagenti pubblicati mediante *firing rules*. Nel caso in cui nel centro di tuple non sia mai stato pubblicato un reagente da interfaccia, infatti, il reagente pubblicato dalla *firing rules*, non veniva correttamente processato. Questo problema era presente perché la reazione *ReSpeCT* relativa a questa gestione, falliva nel tentativo di prelevare la lista dei reagenti, la quale in questo scenario non era appunto presente.
Tutti questi problemi di specifica sono stati risolti.
Parlando invece di limitazioni, bisogna innanzitutto dire che il sistema non è usabile da chi non conosce discretamente *Prolog* oppure *TuCSon*[2], quindi la parte di set-up e anche di comprensione della simulazione e di quello che sta accadendo è preclusa ai cosidetti *addetti ai lavori*.
Un esempio pratico di questa limitazione è la pubblicazione di una legge nel sistema, per chi si occupa di ricerca in ambito biologico si aspetterebbe di scrivere una reazione in questo modo:

$$NaOH + HCl \rightarrow NaCl + H_2O \tag{3.4}$$

Ben diversa è invece la realtà infatti la stessa legge, per essere compresa deve essere scritta:

```
law(['NaOH','HCL'],0.0010,['NaCL','H2O']).
```

Oltre quindi all'ovvio problema del sistema dell'approcciarsi a ricercatori umani e ai loro bisogni, si possono ritrovare anche altre limitazioni del modello, guardando i risultati di quella che è la ricerca in questo ambito, in questo modo il modello viene aggiornato sia con un approccio *top-down*, da contributi teorici, che *bottom-up*, da contributi sperimentali.
Il modello non tiene ad esempio in considerazione che le reti di segnalazione non sono completamente unidirezionali, il sistema di vicinato introdotto è invece unidirezionale, questo causa problemi nella simulazione di vie di segnalazione più complesse, dove non è così ben definito il contorno della funzione di vicinato.
Potrebbe quindi essere opportuno estendere il motore, per far sì che ci siano delle corsie preferenziali sulla pubblicazione dei reagenti nei compartimenti vicini.
Un'altra limitazione è ancora da ritrovarsi nella specifica delle leggi, esse infatti permettono solo in parte la stesura di leggi con reagenti il cui *rapporto stechiometrico* non è unitario. Ad esempio si vorrebbe anche poter scrivere una reazione di questo tipo:

$$C_6H_{12}O_6 + 6O_2 \rightarrow 6CO_2 + 6H_2O + E$$

allo stato attuale delle cose, infatti per poter gestire una reazione del genere, bisognerebbe dover replicare i reagenti e i prodotti il numero di volte necessario per raggiungere la giusta quantità, il che oltre ad essere umanamente complicato è la sua gestione è anche computazionalmente più complessa e onerosa.
Un altro punto da evidenziare è la mancanza di leggi temporali. Queste leggi sono attivate ciclicamente allo scorrere del tempo, esse possono inoltre simulare comportamenti di interazioni con il medio extracellulari, ad esempio producendo molecole segnale o ritirando molecole dai compartimenti intracellulari. E' opportuno definire per queste leggi un intervallo di tempo nel quale dovrebbero attivarsi per migliorare la simulazione di questo comportamento.
Un'altra carenza nella modellazione è la mancanza di eventi all'interno

della simulazione. Per fare un esempio potrebbe esservi una legge che se attivata, porti alla *morte cellulare*. E' bene tener conto di questi fenomeni, notificarli all'utilizzatore e possibilmente definire quali azioni da compiere al fronte di questo evento.
Esempi di possibili azioni potrebbero essere l'inibizione di recettori, la rimozione di leggi all'interno di compartimenti, l'aggiunta di leggi all'interno di compartimenti, e tutto quello che in futuro potrà essere ritenuto valido, al fronte di ulteriori ricerche.
Non esiste inoltre una definizione valida di cosa sia il tempo di simulazione, assimilando il tempo in cui occorre un evento con il tempo di sistema. In generale è un approccio sbagliato perché potrebbe non essere garantito l'ordine temporale degli eventi, e infatti esso non è garantito, ma non c'è nessuna altra relazione tra tempo di simulazione e tempo di sistema, ad esempio se il motore chimico è in fase di stop, il tempo di sistema continua a scorrere, viziando il tempo di cadenza di un evento.
Totalmente assenti sono i controlli sull'utilizzatore, niente vieta a chi usa la specifica di utilizzarla in maniera anomala, danneggiando anche il sistema.
Potrebbe avvenire per esempio che il processo di *Stop* sia parziale o comunque l'utente non sappia bene come attivarlo (ci sono due tuple di stop, una delle quali è solo un messaggio interno). Quindi il risultato potrebbe essere quello di rompere alcuni meccanismi che sono utilizzati internamente per il corretto funzionamento del sistema. L'accesso al centro di tuple deve essere quindi limitato dalle interazioni utente-sistema ammissibili.
Durante il lavoro è stato trovato un baco nella piattaforma *TuCSoN*, che non permetteva la corretta esecuzione di simulazioni particolarmente lunghe. Infatti i centri di tuple con molteplici e veloci interazioni esterne, interrompevano bruscamente la comunicazione con gli agenti esterni, inibendo quelli che sono i comandi d'interazione con l'utente, ma anche gli agenti di monitoring, che devono comunque colloquiare con loro.
Questo è dovuto al fatto che in TuCSoN vengono create ed accettate continuamente connessioni con gli agenti, se queste sono di varia natura, quindi risulta necessario un sistema di *caching*, delle *socket*. Tuttavia, la soluzione adottata in questo lavoro, che è di qualità peg-

giore ed è più un *workaroud* del problema che potrebbe rimanifestarsi in altre forme, magari in altre applicazioni, aggiunge sia una funzione interna delle socket per riusare le connessioni attive, la *setReuseAddress(boolean b)*, che la gestione di alcune eccezioni che facevano terminare l'esecuzione di uno dei thread che compongono un agente TuCSoN, senza notificarlo all'utilizzatore.

3.2.3 Le soluzioni proposte

E' chiaro che una semplice specifica ReSpecT, per quanto potente ed elaborata, non basta per risolvere tutti i problemi sopracitati, per i quali bisogna creare un sistema più completo ed omnicomprensivo.
Permettere la scrittura delle leggi chimiche come da (3.4), ad esempio semplifica parecchio la lettura e la comprensione di quelli che sono i processi interni ai vari compartimenti cellulari.
Per fare questo si è scelto di implementare un *parser*, il cui *AST* prodotto è proprio la sintassi Prolog delle leggi. In questo modo si risolve non solo il problema sopracitato, ma si verifica anche che la legge in ingresso sia valida, il tutto lasciando invariata la sintassi precedente delle leggi. Un altro punto chiave è quello delle estensioni del motore chimico, per renderlo il più possibile preciso e aderente alle specifiche dettate dalla letteratura e dalla ricerca.
E' opportuno quindi estendere le potenzialità nella definizione delle leggi, permettendo di specificare dei pesi sia ai reagenti che ai prodotti.
Questa modifica è decisamente più sostanziosa della precedente, infatti i reagenti e i prodotti sono trattati in liste Prolog, come visto in precedenza, quindi la modifica intacca una buona parte del lavoro precedente.
La gestione dei vicini è invece da rifare, la scelta equiprobabilistica non sembra rispecchiare i dati sperimentali sulle pubblicazioni dei reagenti nei vari compartimenti, per i quali è migliore una scelta deterministica, per fare questo bisogna dapprima estendere la sintassi delle leggi:

$$Ca + 2OH \rightarrow 2CaOH@membrana \tag{3.5}$$

dove membrana è il compartimento di destinazione del reagente.
Oltre alle già presenti e funzionanti leggi e leggi istantanee, bisogna rimodellare e creare un nuovo tipo di leggi, quelle temporali, che per definizione sono leggi che si attivano con cadenza periodica. Per queste leggi si sceglie di far specificare all'utente un intervallo temporale in cui esse verranno attivate, ciclicamente. Per fare in modo che questo tempo sia certo, basterà quindi eguagliare il massimo e il minimo di questo intervallo.
Infine bisogna aggiungere una nuova *firing rule* da affiancare eventualmente al tipo esistente per retrocompatibilità, in cui specificare direttamente il vicino. Per la sintassi di vicinato si accetta quella precedente, non sembrano volerci nuove estensioni.
La qualità di implementazione delle tuple di debug non è ottimale.
Nella specifica originale si produce debug solo per un incremento o una riduzione di una concentrazione di un reagente, in un determinato compartimento, mentre a mio avviso si dovrebbe estendere la pubblicazione del debug anche ad altri effetti degni di nota, quindi il debug deve essere qualcosa di completamente slegato da un semplice fatto, un esempio potrebbe essere l'esecuzione di una legge, l'interazione con un compartimento vicino o quant'altro. In sintesi possiamo permetterci di ottenere informazioni molto migliori e complete, con poco sforzo. Per raggiungere questo obiettivo descriverei queste tuple di debug con il tempo in cui occorre e la descrizione di un fatto accaduto.
Notare che questa definizione non è limitativa, anzi la descrizione può e deve contenere tutte le informazioni che possono essere utili per classificare ciò che è accaduto.
L'utente potrebbe inoltre richiedere che alcune azioni che avvengono nel compartimento siano a lui notificate, quali eventi importanti.
Per questo è opportuno estendere la specifica e permettere all'utente di definire un evento.
Quello che vogliamo ottenere è un evento legato ad una azione che avviene all'interno di un compartimento che può essere ad esempio l'esecuzione di una legge o un reagente che raggiunge una certa concentrazione. Deve essere inoltre possibile definire una serie di azioni che vengono eseguite all'avverarsi dell'evento al fine di inibire o di migliorare la produzione di reagenti.

In questo modo l'utente ha un metodo molto flessibile e facilmente estendibile che permette loro di definire eventi, azioni da compiere al fronte dell'evento e attivazione dell'evento.
Un altro passo di notevole importanza è quello di giungere ad un concetto di *tempo di simulazione*, questo concetto è totalmente assente, e si assimila il tempo di simulazione congruente al tempo di sistema. Questa assunzione è ovviamente errata e fuorviante, perché non si gestisce a modo lo *Stop* dell'engine, ma nemmeno di eventuali altri eventi che dovrebbero influenzare il corso del tempo.
Non è corretto assumere che una simulazione che ha bisogno di operatori umani parta istantaneamente dopo la fase di inizializzazione del sistema, e tra l'altro questo non è possibile nemmeno fisicamente a causa dei tempi (non nulli) di esecuzione.
Serve andare quindi verso un concetto di tempo più esteso, che tenga conto di queste problematiche, o che almeno garantisca, in un primo momento, la giusta sequenza temporale degli eventi, e che reagisca correttamente ai segnali di start e di stop.

3.3 Le funzionalità aggiunte

3.3.1 Estensioni sintattiche e semantiche alle leggi

E' stata estesa la sintassi delle leggi, in particolare ora è possibile specificare le concentrazioni delle molecole presenti nelle leggi, la sintassi scelta è aderente alla specifica dei reagenti in un compartimento, ovvero:

```
reactant(NAME,CONCENTRATION)
```

per cui la nuova sintassi per le leggi diviene:

```
law(RL,RATE,PL)
```

dove *RL* e *PL* sono rispettivamente la lista dei reagenti e dei prodotti al cui interno sono contenute molecole specificate come scritto in precedenza.
Per la modifica alle *firing rules* la sintassi delle leggi è stata estesa in modo da specificare, univocamente, il compartimento di destinazione del prodotto.

```
firing(P,neighbour(NAME,ADDR))
```

dove P è la molecola prodotta, scritta come visto in precedenza, mentre $NAME$ e $ADDR$ sono il nome e l'indirizzo di rete in cui si trova il compartimento di destinazione.

3.3.2 Parsing delle leggi

Come si può notare le sintassi Prolog utilizzate dalla nuova specifica, seppur decisamente più potenti dal punto di vista espressivo, sono anche peggiori se lette da un punto di vista umano. Per questo motivo la traduzione Legge chimica-Prolog e viceversa è stata affidata ad un parser.
Innanzitutto sono stati definiti i *token* del linguaggio delle leggi chimiche, con alcune aggiunte che commenterò:

- *number* : numeri interi, per riconoscere le giuste dosi dei reagenti in una reazione.
- *literal* : parole che iniziano per una lettera, ma che comprendono anche altri simboli all'interno, non sono solo ammessi altri token del linguaggio come *+,-,.,@*, servono a riconoscere sia le molecole in gioco, che il nome dei vicini.
- *rate* : espresso dalla sintassi concreta *rate=float* specifica il rate della reazione.
- *instant* : espresso dalla sintassi concreata *rate=inf* specifica una legge istantanea.
- *+ e -¿* : riconoscere gli operatori corrispettivi nelle leggi chimiche.
- *@* : sintassi speciale per le nuove firing rules.
- *.* : zucchero sintattico aggiunto per determinare la fine di una legge.

Per concludere la definizione della sintassi delle leggi va fornita la specifica *BNF* del linguaggio, i simboli terminali sono tra parentesi quadre:

```
laws ::= [] | law,laws.
law ::= reagents,['->'],products,['.'],rate.
reagents ::= num_mol_opt,[literal(_)],reagents_rest.
reagents_rest ::= [] | ['+'],reagents.
products ::= num_mol_opt,[literal(_)],firing_opt,products_rest.
products_rest ::= [] | ['+'],products.
firing_opt ::= [] | ['@'],[literal(_)].
rate::= [rate(_)] | [].
num_mol_opt ::= [] | [num(_)].
```

Per quello che riguarda le leggi istantanee la sintassi non è molto diversa se non per il fatto di avere un rate infinito, introdotto come espediente di distinzione:

```
instant_law ::= reagents,['->'],products,['.'],[instant].
```

Le leggi temporali hanno invece bisogno di un intervallo di tempo nella loro definizione come descritto in precedenza. E' stata studiata inoltre un'altra estensione nella loro definizione permettendo di lasciare la parte destra o quella sinistra della reazione vuota. In questo modo è possibile simulare la pubblicazioni di reagenti da compartimenti vicini, anche se non modellati, oppure simulare una pubblicazione ad un compartimento vicino non modellato.
Con questa estensione è possibile allungare inoltre la durata delle simulazioni, simulando quella che è la vita di una cellula e le interazioni esterne.
La sintassi BNF quindi va estesa con le seguenti regole:

```
timed_law ::= reagents_opt,['->'],products_opt,['.'],
        [timed],[num(_)],[','],[num(_)].
reagents_opt ::= reagents | [].
products_opt ::= products | [].
```

Una volta riconosciuta la grammatica del linguaggio chimico la costruzione dell'*Abstract Syntax Tree*, come voluto in ingresso dal motore

chimico diventa poco più che un semplice esercizio.
La traduzione inversa è invece molto più agevole per il sistema, e di più facile implementazione, basta specificare poche regole Prolog, che si occupano di concatenare le stringhe nel giusto ordine.

3.3.3 Le leggi temporali

Le leggi temporali sono state progettate per essere attivate ciclicamente, in un intervallo scelto dall'utente è stata quindi scelta una sintassi opportuna per raggiungere tale scopo, ovvero:

```
timed_law(RL,[MIN,MAX],PL).
```

Dove oltre ai normali *PL* e *RL* che rappresentano come al solito la lista dei prodotti e dei reagenti, possiamo definire due interi *MIN,MAX* che rappresentano il minimo e il massimo intervallo che può trascorrere prima che la legge sia attivata. Questo tipo di leggi è salvato in una struttura dati per il reperimento da parte dell'utente, una volta calcolate viene generato un numero casuale compreso nell'intervallo specificato e viene pubblicata una nuova reazione nella specifica che viene chiamata quando il sistema giunge al nuovo intervallo di tempo calcolato ripubblicando la legge, in seguito vi è un tentativo di esecuzione della legge, se il tentativo va a buon fine, si decrementano i reagenti e si aggiungono i prodotti, come per l'altro tipo di leggi, se invece fallisce, la reazione va comunque a buon fine, per permettere di eliminare le tuple residue che compongono la segnalazione interna.
Sono state inoltre fornite le funzionalità base per definire queste leggi dall'esterno, quando un utente inserisce una legge temporale, viene rimossa dal centro di tuple e viene aggiornata la lista delle leggi temporali presenti, inoltre se il motore chimico è attivo la legge temporale viene immediatamente mandata in esecuzione attivando quindi il suo comportamento ciclico descritto in precedenza. La fase invece di rimozione è più complessa e deve essere eseguita prima della richiesta dell'utente, bisogna prima di tutto verificare la presenza della legge all'interno della lista, e rimuovere la legge, pubblicare la legge richiesta nel centro di tuple per garantire il corretto funzionamento della

rimozione da parte dell'utente, e infine rimuovere il *Trigger*, se presente, della legge, al fine di terminare il suo comportamento ciclico. Per rispondere infine alle richieste di *read* bisogna dividere il carico su due reazioni; la prima che viene servita prima della richiesta utente si occupa di verificare la presenza della legge all'interno della lista, se la verifica non va a buon fine la reazione fallisce, e di conseguenza anche la richiesta utente, se la verifica invece produce un risultato viene pubblicata una tupla con la legge all'interno del centro di tuple, per rispondere alla richiesta utente, nella seconda fase, dopo la richiesta, questa tupla deve essere rimossa, in quanto la read la lascerebbe all'interno del centro.

3.3.4 Gli eventi

Come descritto in precedenza gli eventi definiti dall'utente sono caratterizzati da tre componenti

- nome dell'evento
- corpo dell'evento, ovvero una collezione di azioni da eseguire quando l'evento viene eseguito
- *links* dell'evento, ovvero al fronte di quali cambiamenti o azioni viene eseguito l'evento.

Questa descrizione è molto valida e generale, in quanto non dipendente da singoli cambiamenti o non limitato a solo poche azioni, quindi, anche se si pongono limitazioni all'atto pratico nella loro definizione esse sono facilmente aggirabili o estendibili.
L'utente per definire un evento deve fornire al centro di tuple una tupla del tipo

```
event(NAME,BODY,LINKS)
```

il centro di tuple risponde all'inserimento di questa tupla eliminandola e pubblicando al suo posto una tupla di *link* per ogni *link* richiesto, questa tupla è del tipo

```
event_link(LINK,EV_NAME)
```

dove *LINK* è l'oggetto che lancia l'evento di nome *EV_NAME*.
Inoltre viene pubblicata un'altra tupla del tipo

```
event_define(EV_NAME,BODY)
```

con le istruzioni che formano il corpo dell'evento. Questa divisione viene fatta semplicemente allo scopo di migliorare le prestazioni del motore e non è strettamente necessaria, infatti si potrebbe raggiungere lo stesso obiettivo utilizzando la tupla di partenza.
Il processo stop è stato aggiornato per rimuovere i *Trigger* delle leggi temporali, mentre il processo di start è stato aggiornato per rieseguire ogni legge temporale, ripristinandone il comportamento ciclico.
Come scritto in precedenza ora gli eventi sono caratterizzati da un tempo di occorrenza e da una descrizione, questo permette di tracciare ulteriori eventi importanti, come l'esecuzione di una legge, l'aggiunta di un reagente e il firing di un reagente, in questo modo otteniamo una informazione molto più utile e completa di ciò che accade nei compartimenti, in vista di successive elaborazioni di questi dati.

3.3.5 Avvio e blocco della simulazione

Lo start e lo stop della simulazione ora si occupano anche di riprendere il corso del tempo di simulazione e di bloccarlo, in questo modo i dati temporali sono già molto più realistici.
All'interno del sistema è stato associato ad ogni compartimento un agente che si occupa di gestire gli eventi ed elaborarne le informazioni, con questo espediente si toglie l'onere computazionale di una elaborazione al centro di tuple, inoltre un agente è in grado di comunicare i risultati di tale elaborazione in tempo reale, e non su richiesta, come farebbe il centro di tuple, migliorando nettamente l'interazione con l'utente. Inoltre un agente è facilmente modificabile, e può gestire anche descrizioni dell'evento molto complesse e strutturate.

3.3.6 Salvataggio e caricamento di una simulazione

E' comodo per un utente inoltre poter salvare e ricaricare una simulazione, senza doverla ripreparare ogni volta da zero, per questo motivo sono state pensate e aggiunte funzionalità atte a svolgere questo compito. E' importante scegliere una rappresentazione valida dei file di salvataggio per fornire un valido supporto anche ad estensioni future. Per questo motivo è stato pensato di salvare le rappresentazioni prolog, che sono molto espressive, in stringhe JSon, che anche se meno espressive sono uno standard che ha implementazioni molto valide e performanti per il reperimento delle informazioni, e oltretutto il potere espressivo di prolog ci torna molto comodo per la rappresentazione degli oggetti di simulazione, mentre per aggregare i vari oggetti, ci basta appunto il concetto di collezione estendibile e quello di lista, che sono presenti in JSon.
Per il file di salvataggio viene creato un Array JSon, contenente oggetti di tipo compartiemnto. Ogni compartimento presenta vari valori, che listiamo in seguito:

- *compartment* che contiene il nome del compartimento cellulare
- *address* che contine l'indirizzo del compartimento cellulare
- *laws* un array JSon che contiene la rappresentazione prolog delle leggi ordinarie
- *instant laws* un array JSon che contiene la rappresentazione prolog delle leggi istantanee
- *timed laws* un array JSon che contiene la rappresentazione prolog delle leggi temporali
- *neighbours* un array JSon che contiene la rappresentazione prolog dei vicini
- *reactants* un array JSon che contiene la rappresentazione prolog dei reagenti
- *events* un array JSon che contiene la rappresentazione prolog degli eventi definiti dall'utente

3.4 Descrizione del modello informatico

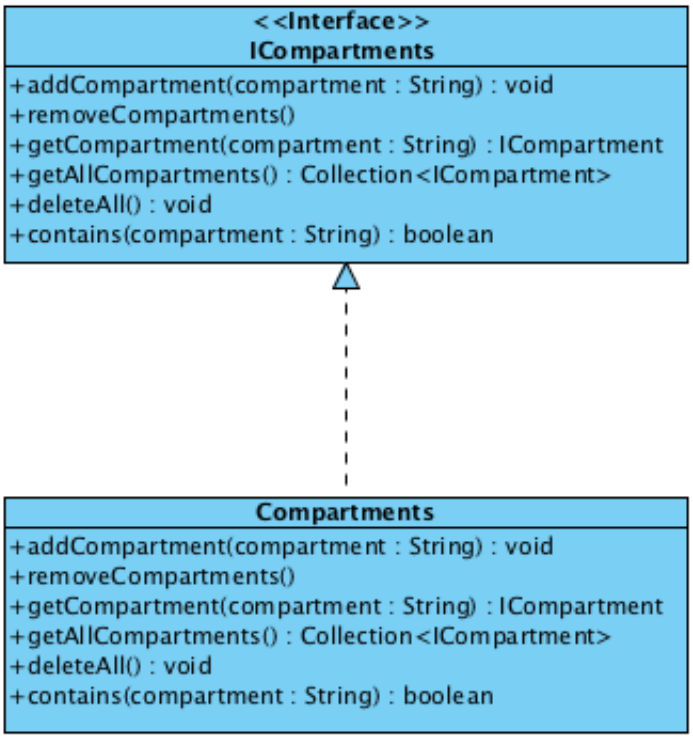

Figura 3.2: Modello compartimenti cellulari

Per andare verso la realizzazione del sistema sono state innanzitutto analizzate le entità che compongono il modello. Prima tra tutte l'astrazione di quello che è un compartimento cellulare, il cui modello UML è mostrato in 3.3, con le sue funzionalità base, descritte in precedenza. Ad una prima analisi la classe dovrebbe avere queste funzionalità:

- Aggiungere una certa quantità di un reagente
- Togliere una certa quantità di un reagente
- Aggiungere una legge (ordinaria,istantanea o temporale)

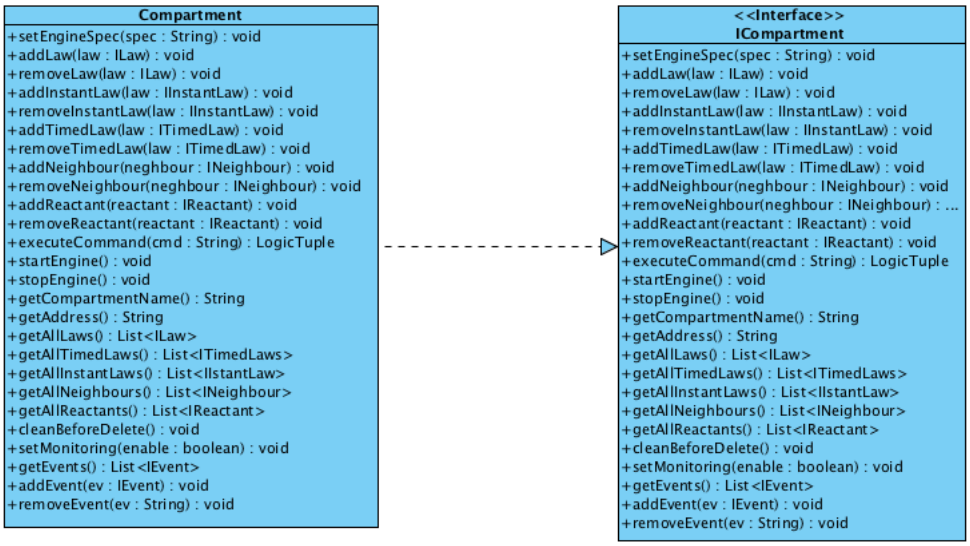

Figura 3.3: Modello di un compartimento cellulare

- Togliere una legge (ordinaria,istantanea o temporale)
- Aggiungere un vicino
- Togliere un vicino
- Restituire le leggi presenti
- Restituire le leggi istantanee presenti
- Restituire le leggi temporali presenti
- Restituire i reagenti presenti e la loro quantità
- Restituire i compartimenti vicini
- Avviare il motore chimico
- Fermare il motore chimico

- Avviare/Fermare il processo di *monitoring*, ovvero quello che si occupa di tener traccia degli eventi che occorrono e del loro ordine temporale

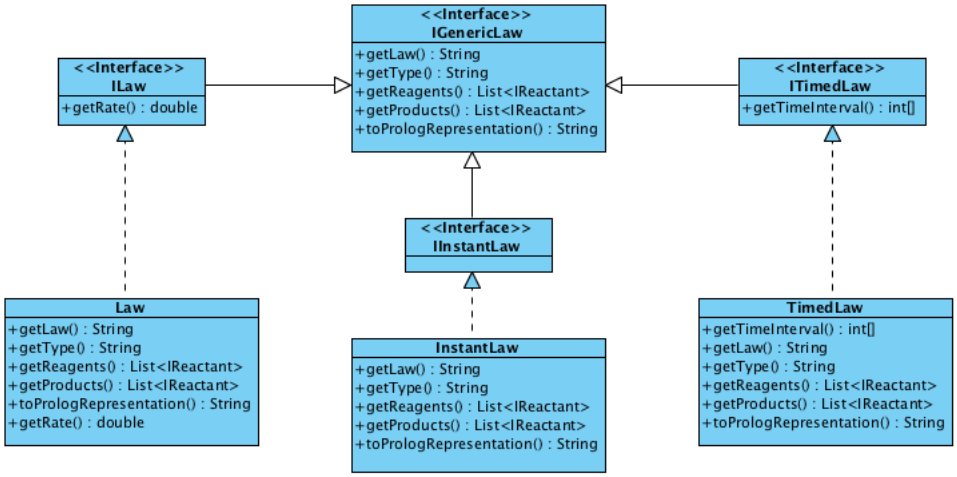

Figura 3.4: Modello delle leggi

Viene quindi naturale completare lo studio delle entità con leggi, vicini e reagenti.
Per quanto riguarda le leggi va considerato il fatto che è utile tener traccia sia della loro stesura in termine prolog, sia la stesura in forma umanamente comprensibile, per cui la sua interfaccia avrà metodi per:

- Restituire la legge sotto forma di stringa
- Restituire la legge sotto forma di termine prolog
- Restituire l'elenco dei compartimenti esterni in cui la legge pubblica almeno un prodotto

Le leggi istantanee non necessitano di altre funzionalità, mentre le leggi ordinarie necessitano di un ulteriore metodo che restituisce il rate della stessa, mentre per quello che riguarda le leggi temporali l'interfaccia va estesa con un metodo che restituisce l'intervallo di tempo nel quale

può avvenire la sua attivazione.
L'UML in figura 3.4 mostra il diagramma delle classi relative alle leggi. Anche i reagenti, il cui modello UML è mostrato in 3.5, sono entità

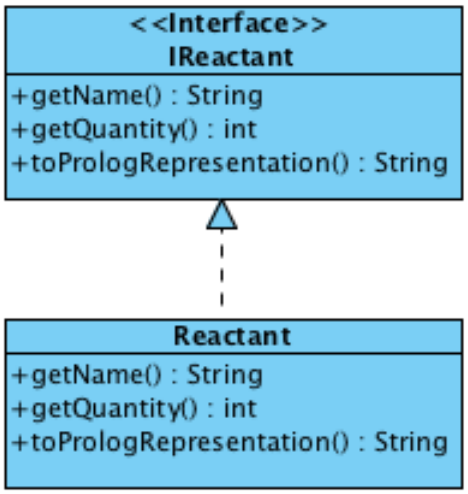

Figura 3.5: Modello di un reagente

statiche e non modificabili come le leggi, in quanto la parte di controllo e di modifica degli stessi è delegata alla specifica del motore chimico, quindi le loro funzionalità si riducono a:

- Restituire il nome del reagente
- Restituire la quantità del reagente
- Restituire il termine prolog del reagente

I vicini sono invece deprecati nel loro uso, ma sono stati lasciati per retrocompatibilità. Il loro modello (figura 3.6) offre quindi le poche funzionalità che erano state previste nel modello originale, ovvero:

- Restituire il nome del vicino
- Restituire l'indirizzo fisico del vicino
- Restituire la rappresentazione in prolog del vicino

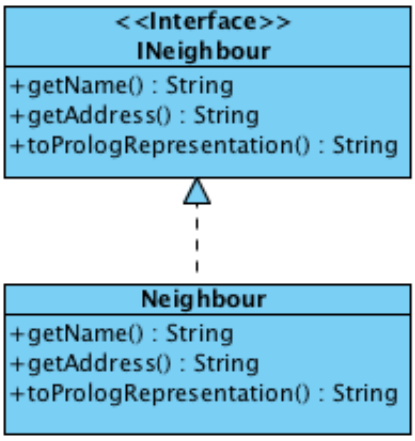

Figura 3.6: Modello di un vicino

Infine il modello comprende gli eventi definiti dall'utente, mostrato in figura 3.7. Essi sono composti da una parte di definizione, in cui si descrivono le azioni da eseguire al fronte dell'evento, e una parte in cui si associano alcune leggi all'evento, in questo modo l'evento viene attivato in seguito all'esecuzione di ogni legge associata. Quindi la descrizione dell'interfaccia di evento è quella leggermente più complicata rispetto alle altre interfacce di base e comprende di:

- Restituire il nome dell'evento
- Restituire la lista di azioni che compongono il corpo dell'evento
- Restituire la lista di associazioni evento-legge
- Aggiungere una legge alla lista delle associazioni legge-evento
- Aggiungere un'azione di rimozione / aggiunta di un reagente
- Aggiungere un'azione di rimozione / aggiunta di una legge
- Eliminare un'azione dall'evento
- Rimuovere un'associazione legge-evento

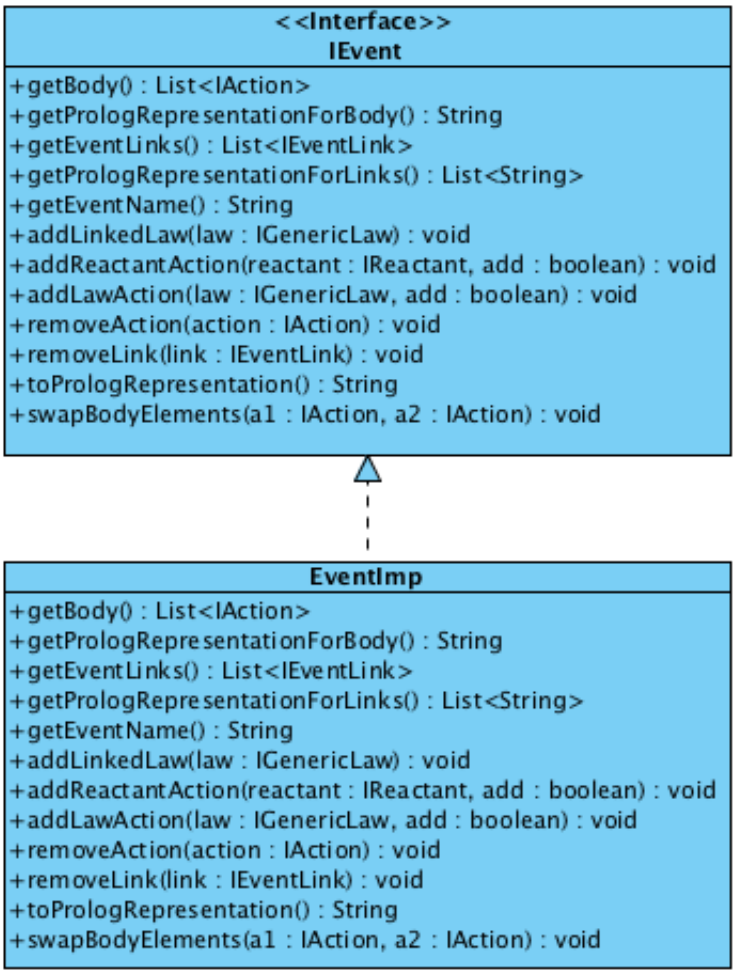

Figura 3.7: Modello di un evento

- Intercambiare la posizione di due azioni all'interno del corpo
- Restituire la rappresentazione in prolog dell'evento

Differentemente da tutte le altre interfacce, questa propone alcuni metodi per modificarne la struttura interna, questo è dovuto al fatto che rispetto alle altre parti di modello ha una rappresentazione più com-

plicata e ricrearla più volte potrebbe essere dannoso per le prestazioni globali del sistema.

Questa interfaccia inoltre, racchiude implicitamente la definizione di

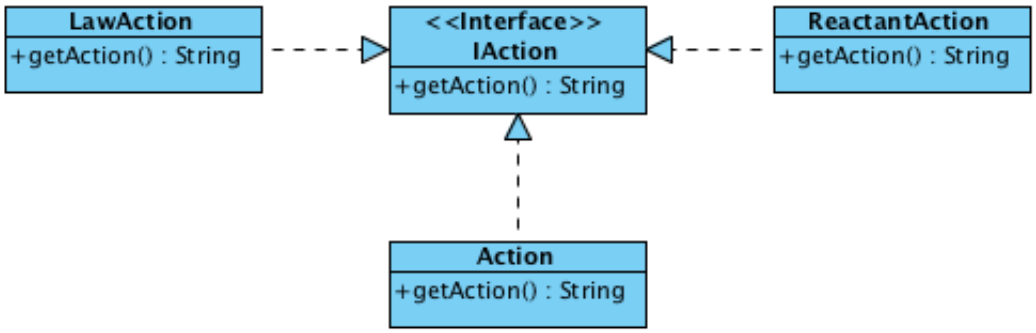

Figura 3.8: Modello di un'azione

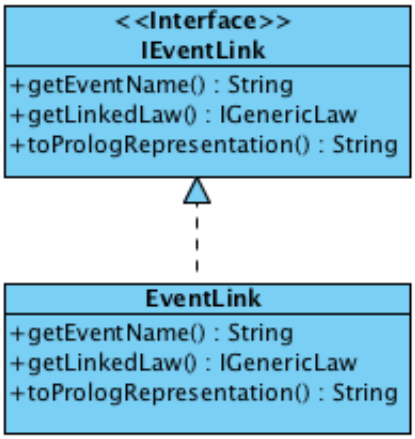

Figura 3.9: Modello di un'associazione legge-evento

Action e di *EventLink*, quindi sono necessarie altre due interfacce che

modellano questi aspetti, anche se non utili a livello di utente, sono necessari per completezza e per rendere il modello di evento più comprensibile.
L'interfaccia di azione, mostrata in figura 3.8, è molto semplice, infatti deve solo restituire l'azione che modella e la sua descrizione.
L'interfaccia di *link* (figura 3.9) ha invece più funzionalità, infatti deve restituire il nome dell'evento e la legge associata all'evento, che sono i due aspetti fondamentali del link. Un ulteriore metodo invece restituisce per completezza, la rappresentazione in prolog dell'oggetto.
Un'ultima parte importante è l'utilizzo del pattern *Factory* per quanto riguarda la costruzione delle entità in gioco, in questo modo è possibile implementarle in altri modi e intercambiarli agilmente.

3.5 Gli agenti del sistema

Gli agenti all'interno del sistema sono, come visto in precedenza di due tipi, il primo, chiamato nell'implementazione *Master* si occupa della interazione utente-BTS, è l'agente che crea il modello a partire da rappresentazioni prolog, prelevate dai centri di tuple, o da rappresentazioni umane, definite dall'utente. Questa parte di sistema si occupa di gestire una serie di eventi, causati dall'interazione con l'utente. Gli eventi disponibili sono di varia natura e rispecchiano le richieste dell'utente.
Tra i principali vi sono quelli che richiedono le funzionalità descritte in precedenza, come il salvataggio e il caricamento di una simulazione, altre si occupano di notificare la volontà dell'utente di creare parti del modello, o di gestirle.
I *MonitoringAgent*, invece, si occupano di notificare, in varie forme, tutto ciò che concerne l'evoluzione del sistema, come ad esempio l'esecuzione di una legge o l'attivazione di un evento. Questi agenti sono associati ad un BTS specifico, da cui devono trarre informazioni, trasformarle, e notificarle ad altri agenti oppure all'utente.
Entrambi i tipi di agente sono necessari ai fini di una buona simulazione, il primo per garantire una completa interazione utente-sistema, i secondi per la duale, notificando le evoluzioni del sistema in tempo

reale.

3.6 Un framework per la definizione del tempo di simulazione

Uno degli ultimi problemi affrontati in questo lavoro, tratta di una argomento che è da sempre stato spinoso, nelle architetture distribuite. I BTS che compongono il sistema, per la loro natura, contengono informazioni e in diverso numero, questo produce un effetto non trascurabile di desincronizzazione del tempo, in quanto una reazione impiega più o meno tempo ad essere eseguita, in base al carico di lavoro che deve supportare il motore chimico.
Per risolvere questo problema è stato pensato un ulteriore centro di tuple, a cui i vari BTS si devono registrare, che si occupa di far da barriera, per quel che riguarda lo scorrere del tempo. I segnali di start e di stop della simulazione, in questo modo al posto di essere dati direttamente a tutti i BTS, saranno dati solamente a questa entità, che rappresenterà un *medium temporale*.
Scopo di questo è infatti quello di notificare il nuovo istante temporale ai BTS presenti nel sistema, per poi rimanere in ascolto di segnali, restituiti dagli stessi, che comunicano il termine del lavoro in quell'istante temporale. Una volta che tutti i BTS hanno finito il loro lavoro, cade la barriera alzata dal *medium temporale*, che incrementa e notifica l'incremento del tempo di simulazione.
In questa maniera si risolvono completamente i problemi pendenti, riguardanti la precisione e l'affidabilità della simulazione, ad un costo accettabile. Questo tipo di approccio inoltre, elimina completamente buona parte del lavoro svolto dai BTS per gestire il loro tempo locale come meglio potevano, semplificando la loro specifica.
Tuttavia questa implementazione non è resa possibile allo stato attuale, dalla piattaforma TuCSoN, che necessita di qualche correzione, per poter gestire un grande numero di messaggi di interazione tra le varie componenti del sistema. In questa proposta di soluzione infatti il numero di messaggi scambiati, tra i centri di tuple è tanto più alto quanti sono i componenti del sistema. Si potrebbe quindi provare ad esplorare in futuro nuove strade, ad esempio la gestione del tempo da

parte di un agente esterno, o una possibile evoluzione del linguaggio ReSpecT che permetta una sincronizzazione tra più di due entità.

Capitolo 4

Proposta di una interfaccia grafica per il modello BTS-based

Un ultimo passo del progetto è quello di definire una interfaccia grafica che sia in grado di gestire tutta l'interazione con il sistema richiesta dall'utente, tanto a livello di strumenti di controllo come di visualizzazione. In primo luogo si richiamano le funzionalità di base che il sistema offre:

- Creare/Eliminare un compartimento cellulare
- Inserire/Rimuovere una legge istantanea in un determinato compartimento cellulare
- Inserire/Rimuovere una legga ordinaria in un determinato compartimento cellulare
- Inserire/Rimuovere una legga temporale in un determinato compartimento cellulare
- Inserire/Rimuovere un reagente in un determinato compartimento cellulare
- Inserire/Rimuovere un vicino di un determinato compartimento cellulare

- Restituire la durata della simulazione, sia per un compartimento, che per l'intero sistema
- Restituire lo stato della concentrazione/molarità dei reagenti in un determinato istante temporale, sia per un compartimento, che per l'intero sistema
- Mostrare lo stato corrente dei reagenti in un compartimento cellulare
- Mostrare le leggi ordinarie/istantanee/temporali presenti un compartimento cellulare
- Mostrare i vicini di un compartimento cellulare
- Aggiornare/Inserire/Rimuovere un evento custom.
- Salvare/Caricare una simulazione
- Avviare/Fermare la simulazione
- Eseguire comandi tipici di un agente TuCSon (introdotto per debug)

Come suggerisce il pattern *Model View Control (MVC)* L'interfaccia grafica deve interagire con la parte di controllo per richiedere uno o più di questi servizi, e al contempo deve fornire un'interfaccia che permetta alla parte di controllo, di eseguire trasformazioni sulla visualizzazione, o un aggiornamento di dati di modello. Nello specifico la parte di View dovrebbe permettere queste operazioni:

- Notificare un messaggio, o un errore, o un *warning* all'utente.
- Notificare all'utente che è stato creato un nuovo compartimento
- Notificare all'utente che tutti i compartimenti sono stati eliminati
- Visualizzare le leggi ordinarie/istantanee/temporali presenti attualmente in un compartimento

- Visualizzare i reagenti presenti attualmente in un compartimento, o in un tempo passato
- Visualizzare gli eventi presenti in un compartimento
- Visualizzare i vicini di un compartimento
- Aggiungere un evento al log degli eventi
- Notificare il cambiamento della concentrazione/molarità dei reagenti in un dato istante di tempo

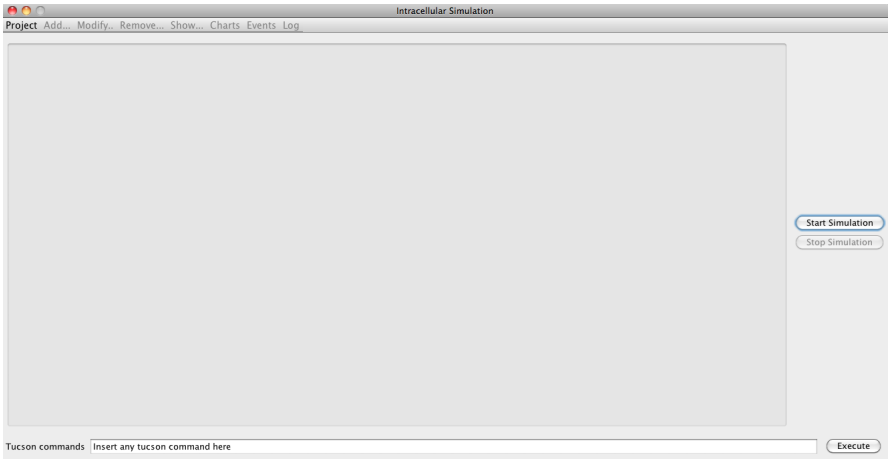

Figura 4.1: Interfaccia principale dell'applicazione

La definizione di queste interazioni è molto importante per il lavoro svolto, sia per eventuali lavori futuri, nei prossimi paragrafi si andrà a descrivere la mia proposta di rappresentazione dei dati forniti da questa interazione, che non è sicuramente unica, né ha la pretesa di essere la migliore realizzabile, per questo si è cercato di rendere questi messaggi d'interazione il più indipendenti possibili dagli specifici problemi grafici o dalla attuale soluzione agli stessi. D'altro canto questo set di interazioni spazia completamente le funzionalità che il sistema offre senza limitarle ulteriormente.

4.1 Strumenti di controllo

In questa sede mostriamo le scelte che sono state fatte, per permettere all'utente di interagire con il sistema.

4.1.1 Menù utente

In primo luogo è stato introdotto un menù nella parte superiore dell'interfaccia che presenta le voci

- Project
- Add...
- Modify...
- Remove...
- Show...
- Charts
- Events
- Log

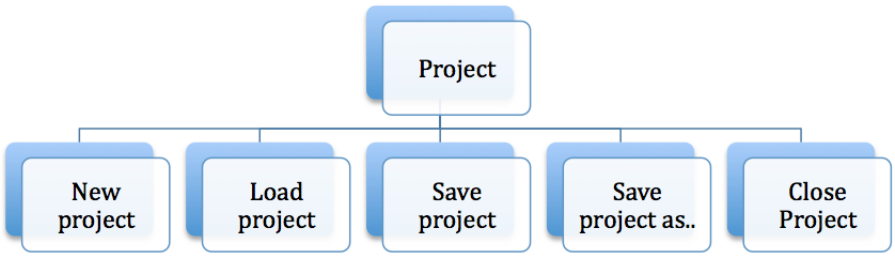

Figura 4.2: Gerarchia menù project

Come si vede dalla figura 4.2 il menù di progetto presenta varie opzioni, la prima delle quali inizializza il sistema creando un nuovo progetto

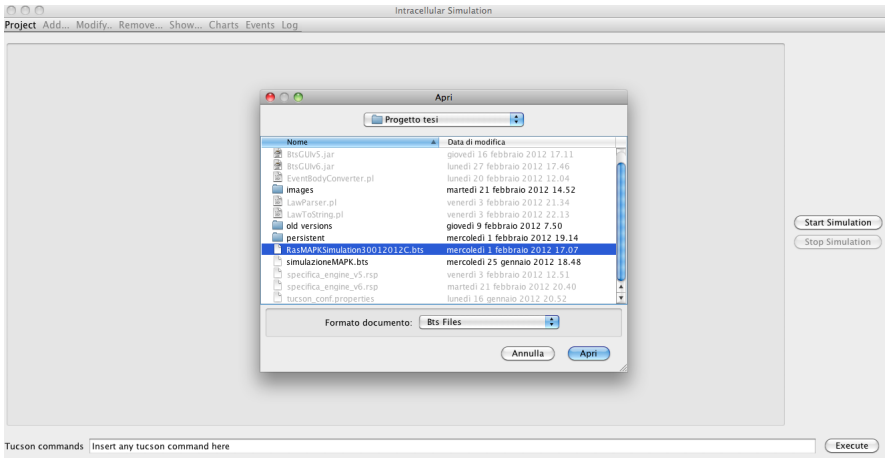

Figura 4.3: Caricamento di un file di simulazione

vuoto. In questo si è pronti per iniziare a lavorare su una nuova simulazione.

Il salvataggio e il caricamento di una simulazione sono sue utilità che permettono di caricare lo stato attuale e ricaricarlo in un secondo momento, vengono salvati o ripristinati per ogni compartimento: i reagenti presenti e la loro concentrazione/molarità, i vicini, le leggi, che siano ordinarie temporali o istantanee e gli eventi definiti dall'utente. E' possibile salvare la simulazione in un qualsiasi momento, anche dopo averla avviata. Per quanto riguarda il caricamento (mostrato in figura 4.3), prima di eseguirlo viene creato un nuovo progetto, quindi le vecchie modifiche, se non salvate, andranno perse.
Per finire l'ultima opzione selezionabile, chiude ed elimina il progetto corrente, facendo ritornare il sistema allo stato iniziale.

Il secondo menù si occupa di aggiungere elementi alla simulazione corrente, le varie opzioni disponibili sono mostrate in figura 4.4. All'aggiunta di un nuovo compartimento viene creato un nuovo *frame* (come si vede in figura 4.5), in cui è possibile immettere il nome che si vuol dare al compartimento o alla struttura cellulare, che si andrà ad aggiungere alla simulazione.

Tutte le altre opzioni di questo menù necessitano di avere un compartimento cellulare selezionato, quindi ne deve essere creato e scelto almeno uno, altrimenti sarà lanciato e visualizzato un errore.

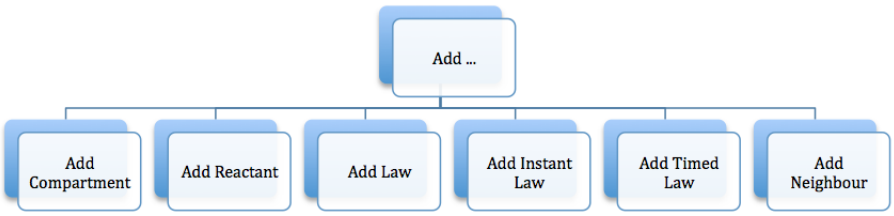

Figura 4.4: Gerarchia menù add

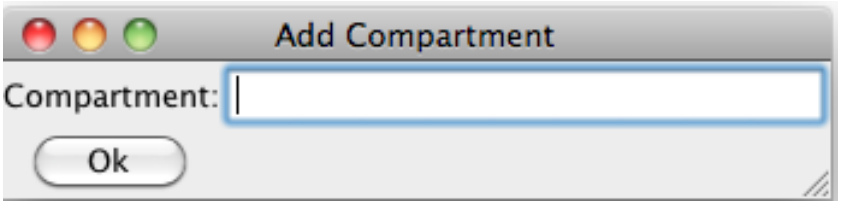

Figura 4.5: Finestra di aggiunta di un compartimento intracellulare

La seconda voce si occupa di inserire un reagente in un compartimento esistente, selezionandola viene creata una nuova finestra (mostrata in figura 4.6), in cui è possibile immettere il nome del reagente e la concentrazione/molarità desiderata.

Anche per aggiungere leggi viene creata una nuova finestra (si vedano le figure 4.7 e 4.8), ma essa è leggermente diversa a seconda della tipologia scelta, infatti, le leggi ordinarie avranno un capo rate compilabile, in cui viene specificato il rate della reazione, e per le leggi

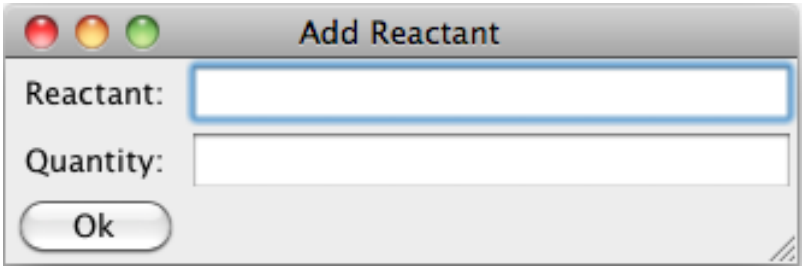

Figura 4.6: Finestra di aggiunta di un reagente

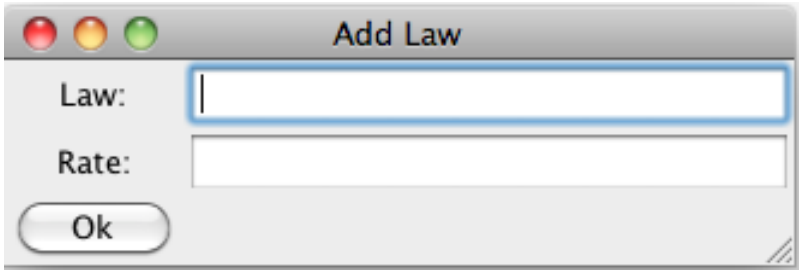

Figura 4.7: Finestra di aggiunta di una legge ordinaria

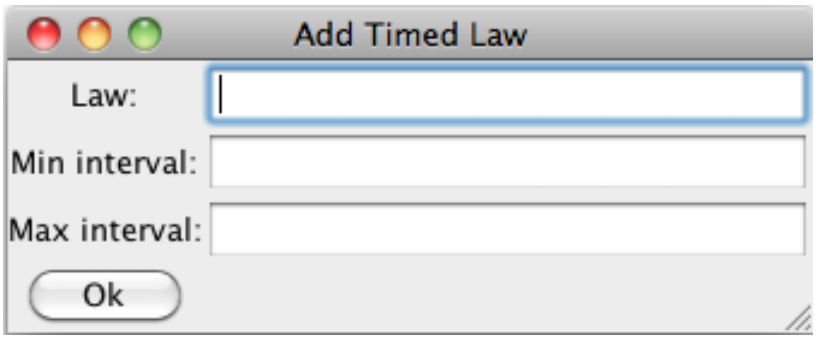

Figura 4.8: Finestra di aggiunta di una legge temporale

temporali possono venire specificati due intervalli temporali, in cui la legge occorre, l'unità di misura di questi intervalli temporali è il millisecondo.
Quello che accomuna le varie tipologie di leggi è invece la loro definizione, che come descritto in precedenza avviene secondo la sintassi di una legge chimica, quindi per definire una legge che prende in ingresso 2 molecole di *'A'* e una di *'B'* per produrre 3 molecole di *'C'*, può venire inserita scrivendo

```
2A + B -> 3C
```

Per definire delle firing rules, bisogna mettere accanto al prodotto una opportuna aggiunta per specificare il compartimento di destinazione, che nel prossimo esempio chiamerò *target*:

```
2A + B -> 3C @ target + D
```

In questo modo il prodotto della reazione viene pubblicato in parte nel compartimento attuale, in parte in quello destinazione, per essere precisi le tre molecole di *C* verranno pubblicate nel compartimento denominato *target* e la molecola di D invece verrà pubblicata nel compartimento attuale.
Infine è possibile aggiungere un vicino, selezionando l'opportuna voce si apre una nuova finestra, in cui è possibile specificare il nome del compartimento che si vuole indicare come vicino e il suo indirizzo di rete (che di default è *localhost*).
Il menù *Modify...*, mostrato in figura 4.9, permette, come suggerisce la parola, di modificare le informazioni immesse in precedenza, tramite il menù *Add..*, è possibile modificare ogni campo immesso in precedenza, anche i nomi dei reagenti o la definizione di una legge, questo per sopperire ad eventuali errori di battitura. In generale, si apre una nuova finestra, che comprende una *combo box* e due pulsanti, uno per rimuovere, l'altro per modificare. Nella combo box è possibile selezionare la descrizione dell'elemento che si vuole modificare. Premendo invece il bottone *modify* la finestra si espande, e vengono visualizzati gli stessi campi descritti in precedenza, e un ulteriore bottone *ok*, premendo il quale si rendono effettive le modifiche effettuate.
Per quello che riguarda i reagenti (mostrato in figura 4.10), nella *combo box* è possibile selezionare il nome del reagente, i due campi sottostanti

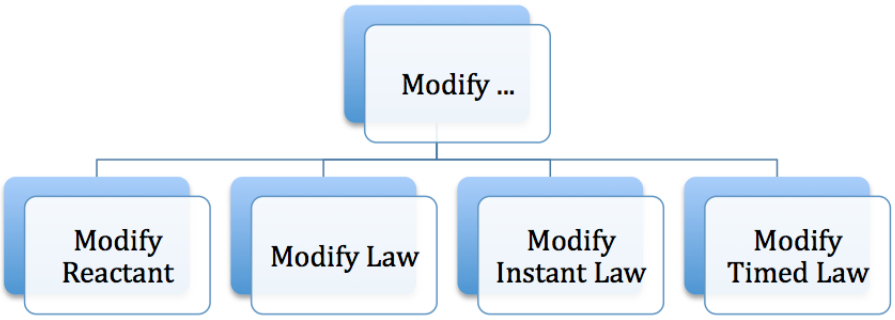

Figura 4.9: Gerarchia menù modify

Reactant e *Quantity*, sono modificabili e quindi è possibile ridefinire, anche completamente, i valori precedentemente inseriti. Scegliendo il bottone *ok* si rendono quindi effettive le modifiche.

Allo stesso modo per quello che riguarda le leggi, si faccia riferimento alle figure 4.11, 4.12 e 4.13, nella *combo box* è possibile scegliere la definizione della legge che si vuole modificare, nei campi sottostanti, si modificano le specifiche della legge, come nel precedente menù. E' possibile modificare anche radicalmente la legge.

Il menù *Remove...* è più strutturato e riprende tutte le voci del menù *Add..*, come si vede in figura 4.14.

In generale si apre sempre una nuova finestra, in cui è presente una *combo box* che permette di selezionare l'oggetto da rimuovere, un bottone *Remove* che si occupa della rimozione e un ulteriore bottone *Ok* che chiude la finestra.

Per quello che riguarda la rimozione di un compartimento cellulare, le voci nella *combo box* sono i nomi dei compartimenti presenti nell'attuale simulazione. Rimuovendo un compartimento, si ha, come *side effect*, la rimozione di tutti gli oggetti presenti nello stesso.

Per rimuovere un reagente (come si vede in figura 4.15), invece, va selezionato il nome come in precedenza, e selezionata la quantità da rimuovere nel campo apposito.

Per quello che riguarda le leggi (figura 4.16), invece, la scelta è limitata alla selezione della legge, i campi sottostanti, servono solamente da richiamo per ricordare il rate della legge, oppure altre informazioni su

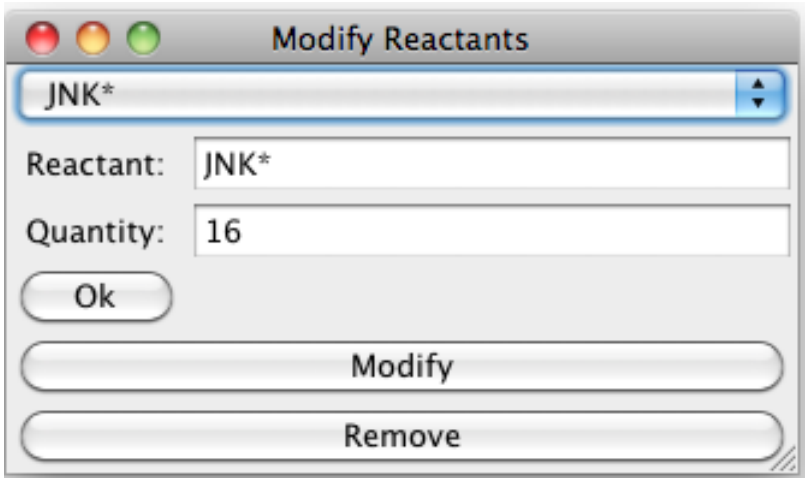

Figura 4.10: Finestra di modifica reagenti

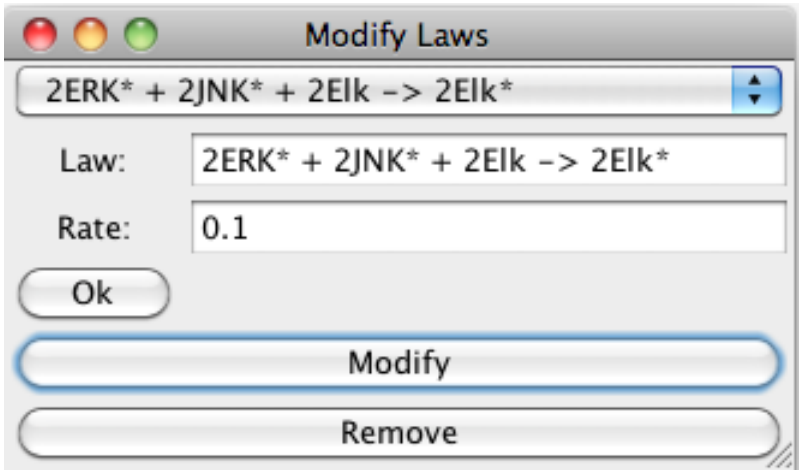

Figura 4.11: Finestra di modifica leggi ordinarie

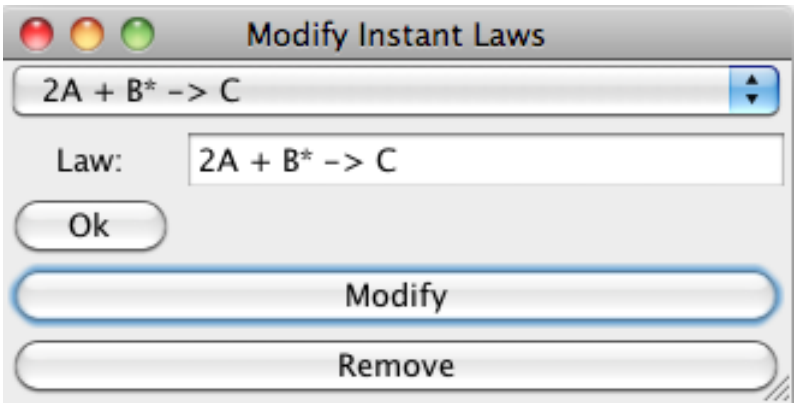

Figura 4.12: Finestra di modifica leggi istantanee

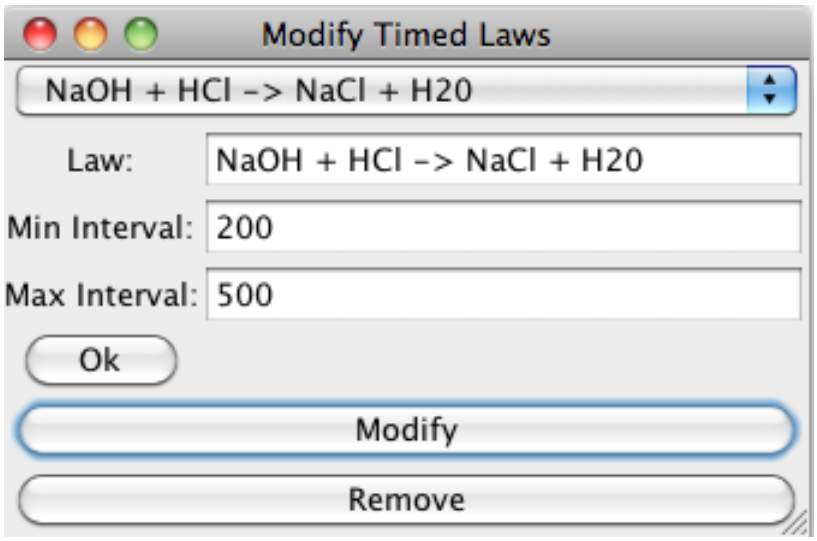

Figura 4.13: Finestra di modifica leggi temporali

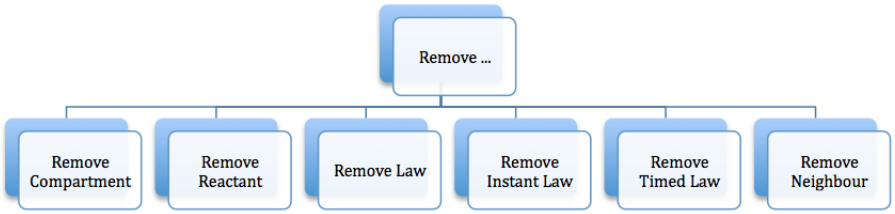

Figura 4.14: Gerarchia menù remove

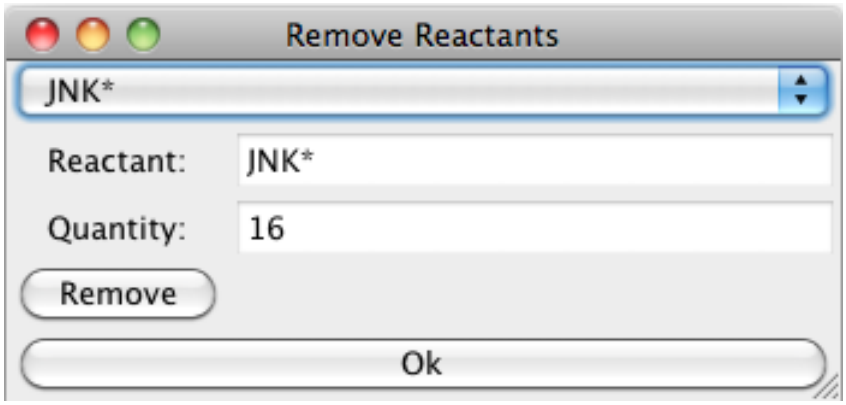

Figura 4.15: Finestra di rimozione di un reagente

di essa.

Il menù *Show...* si occupa invece di chiedere la visualizzazione dello stato attuale del sistema, possono essere visualizzati i reagenti, le leggi sia ordinarie, che istantanee, che temporali, sia singolarmente, che tutte e tre le opzioni contemporaneamente, come mostrato nella figura 4.17. La trattazione dell'output di queste richieste sarà fornita nel prossimo capitolo.

Il menù *Charts* si occupa invece di fornire grafici, o strumenti per visualizzare l'andamento della simulazione. E' possibile attraverso questo menù richiedere il grafico che mostra il cambiamento dei reagenti nel tempo, sia per il compartimento selezionato che del sistema globale, come si nota in figura 4.18.

Un'altra scelta invece è quella di mostrare la concentrazione/molarità dei reagenti in ogni istante di tempo.

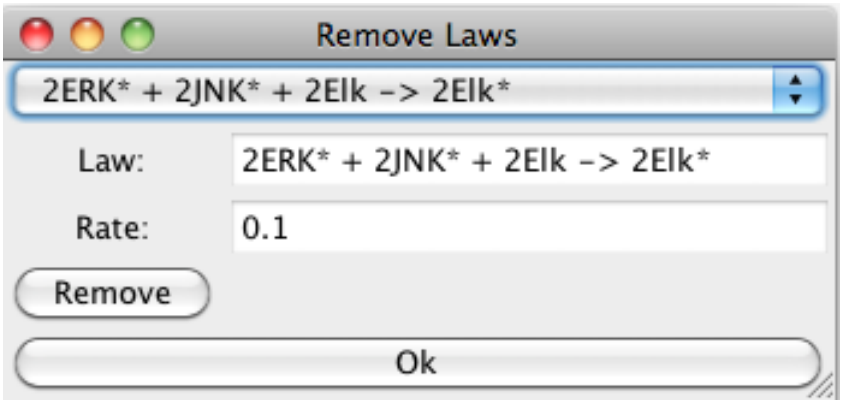

Figura 4.16: Finestra di rimozione delle leggi ordinarie

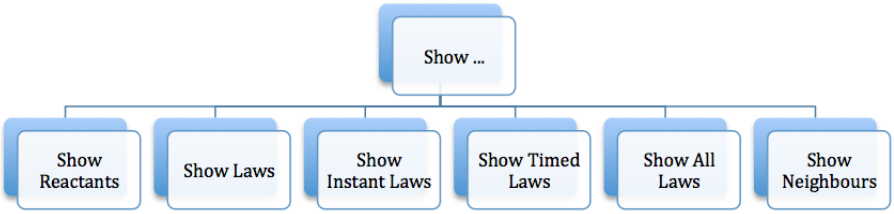

Figura 4.17: Gerarchia menù show

Entrambe queste opzioni saranno spiegate meglio nel prossimo capitolo.
Il menù *Events*, le cui voci sono mostrate nella figura 4.19, si occupa della gestione e della visualizzazione degli eventi definiti dall'utente presenti nel sistema. Il pannello relativo al compartimento, come mostra la figura 4.20, viene diviso verticalmente in tre porzioni, la prima che si occupa, di mostrare i nomi degli eventi, la seconda mostra le azioni associate a quell'evento e l'ultima parte l'elenco delle leggi che attivano l'evento.

La parte a sinistra è divisa in una parte soprastante che contiene una lista, in cui compaiono i nomi degli eventi del compartimento e in una sottostante dove sono presenti due bottoni, uno per aggiungere un evento l'altro per rimuovere l'evento selezionato. Selezionando un evento dalla lista si attivano le altre due parti, mostrando quelle che

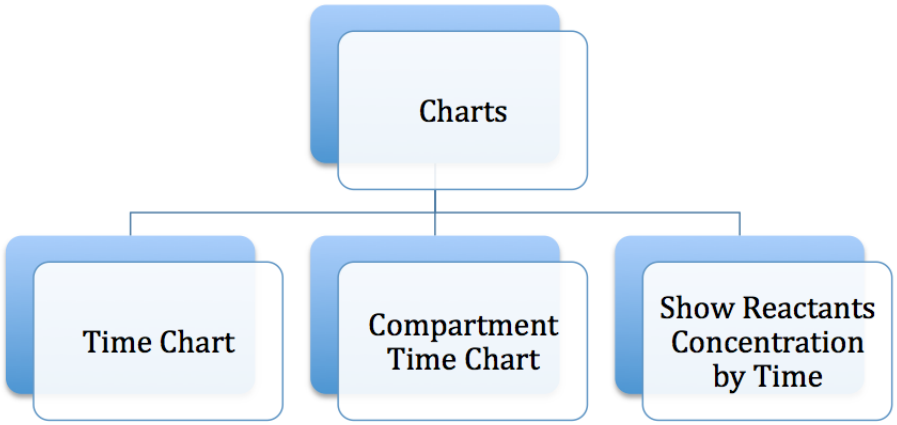

Figura 4.18: Gerarchia menù charts

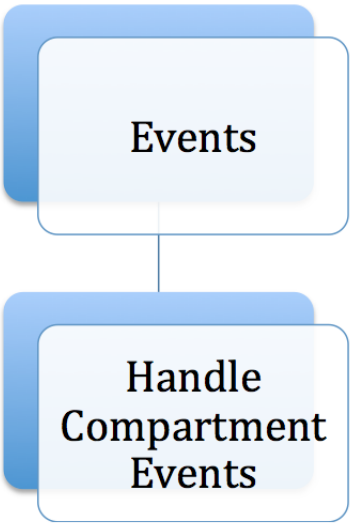

Figura 4.19: Gerarchia menù events

sono le azioni dell'evento e le leggi che attivano l'evento, permettendo inoltre di fare trasformazioni sull'evento.
La parte centrale, si occupa della definizioni delle azioni da eseguire

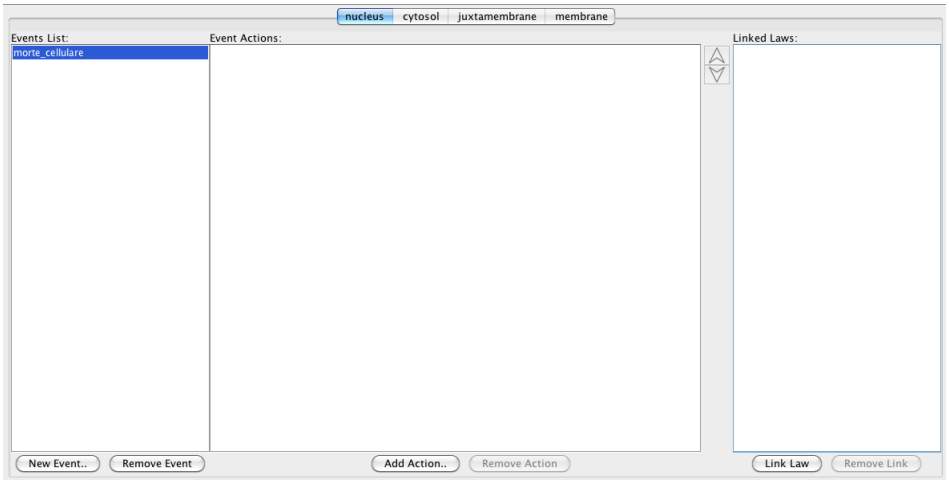

Figura 4.20: Pannello di gestione degli eventi

una volta attivato l'evento. Anche questa parte presenta in alto una lista in cui sono presenti le azioni dell'evento e una parte sottostante con due bottoni, una per aggiungere, l'altra per rimuovere una azione. Premendo il bottone per aggiungere un'azione compare un *popoup menù* con la lista delle possibili azioni:

- aggiungere / rimuovere un reagente
- aggiungere / rimuovere una legge
- aggiungere / rimuovere una legge temporale
- aggiungere / rimuovere una legge istantanea

Alla selezione di ognuna di queste operazioni compare una nuova finestra, identica a quella rispettiva del menù *Add..* descritta in precedenza, in cui è possibile finire la definizione dell'azione. Una volta inserita l'azione essa viene messa in coda alla lista di azioni nella parte superiore del pannello centrale. La parte destra del pannello presenta altri due pulsanti, aventi come icona il primo una freccia rivolta verso l'alto, l'altro con una freccia rivolta verso il basso, con i quali è possibile modificare l'ordine di esecuzione delle azioni legate all'evento.
Il pannello di destra, infine si occupa di definire quale è l'attivazione dell'evento. Anche questo pannello presenta una parte superiore,

contenente la lista di leggi che attivano l'evento e una parte inferiore che contiene due pulsanti una per aggiungere una legge a quelle che attivano l'evento, l'altra per rimuoverla. E' possibile associare l'evento sia all'esecuzione di leggi ordinarie, che istantanee che temporali.

L'ultimo menù presente, *Log*, la cui gerarchia è mostrata in figura

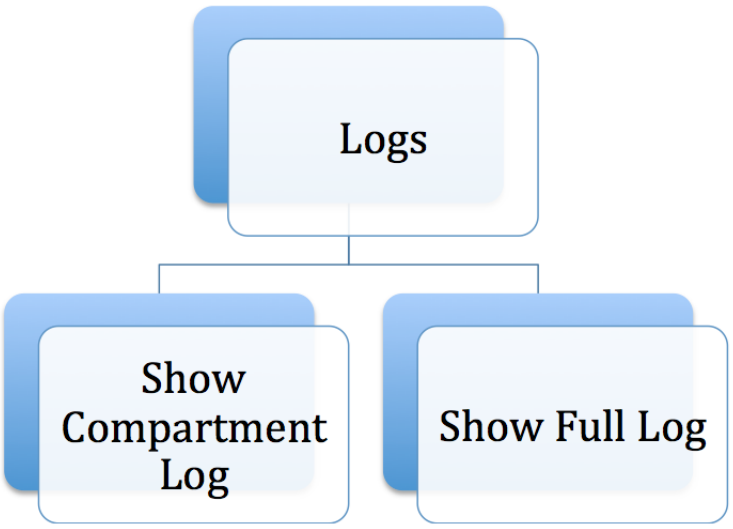

Figura 4.21: Gerarchia menù logs

4.21, è per il momento strettamente di debug, mostra quello che accade all'interno del compartimento oppure globalmente in una nuova finestra.

4.1.2 I bottoni di start e stop simulation

Nella parte destra dell'interfaccia principale, sono disposti due bottoni, *Start simulation* e *Stop simulation*, con i quali l'utente può avviare o fermare i processi biochimici attivabili all'interno dei vari compartimenti. Lo stato dell'applicazione è visibile graficamente, infatti, se la simulazione è ferma, solo il bottone di *Start* sarà selezionabile e viceversa. Una volta selezionato il pulsante di start, il tempo di simulazione inizierà a scorrere, mentre una volta premuto il pulsante stop

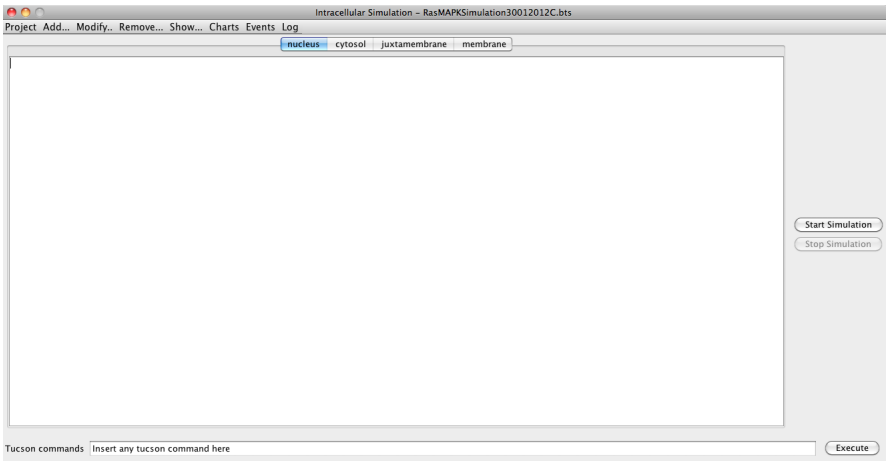

Figura 4.22: Vista dei compartimenti intracellulari

esso sarà bloccato, per poi riprendere dall'ultimo istante trascorso nella simulazione.
Lo stato iniziale della simulazione è *Stop*.

4.1.3 La console di comandi TuCSon

Nella parte inferiore della finestra principale è stato inserito un campo di testo in cui è possibile dialogare direttamente con il compartimento corrente, secondo una specifica *linda-like*, implementata in TuCSon, questa parte dell'applicativo è presente per permettere ad utenti esperti di avere il controllo su quello che accade a basso livello nei centri di tuple, che modellano un compartimento.
Con questo linguaggio è anche possibile fare cambiamenti *runtime* a quella che è la specifica dei compartimenti, per provarne direttamente gli effetti, senza riavviare l'applicazione.

4.2 Strumenti di visualizzazione

Una volta definiti, i compartimenti sono visualizzati in un *tabbed pane*, al centro della finestra principale dell'applicazione, come mostrato in figura 4.22, quindi la struttura del sistema che si vuole simulare è

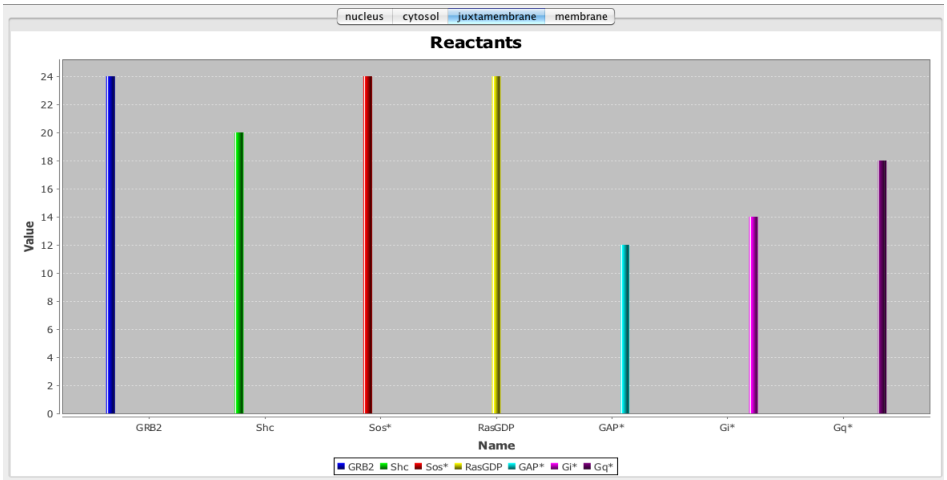

Figura 4.23: Grafico dei reagenti per compartimento

sempre visibile in ogni momento all'utente. Per cambiare compartimento, inoltre, è sufficiente selezionare la linguetta avente il nome del compartimento desiderato. Tutte le operazioni di visualizzazione, se non specificato, avvengono nel compartimento corrente, quindi quello attualmente selezionato nel *tabbed pane*.

4.2.1 Visualizzazione dei reagenti

Per mostrare i reagenti, si è scelto di utilizzare una rappresentazione mediante un grafico a colonne, simile a quello mostrato in figura 4.23. Sull'asse delle ascisse, sono elencati i nomi dei reagenti, sull'asse delle ordinate i valori della loro concentrazione/molarità. Come output si ottiene quindi un grafico, con tante colonne quanti sono i reagenti, di altezze variabili a seconda della loro concentrazione/molarità.
E' possibile esportare il grafico, mediante click destro sullo stesso come immagine, per poi utilizzarlo in altre applicazioni.

4.2.2 Visualizzazione delle leggi

Il pannello che si occupa della visualizzazione delle leggi è composto unicamente da una tabella che mostra la legge, secondo la sintassi chimica, ed eventualmente il rate o l'intervallo di tempo per quello

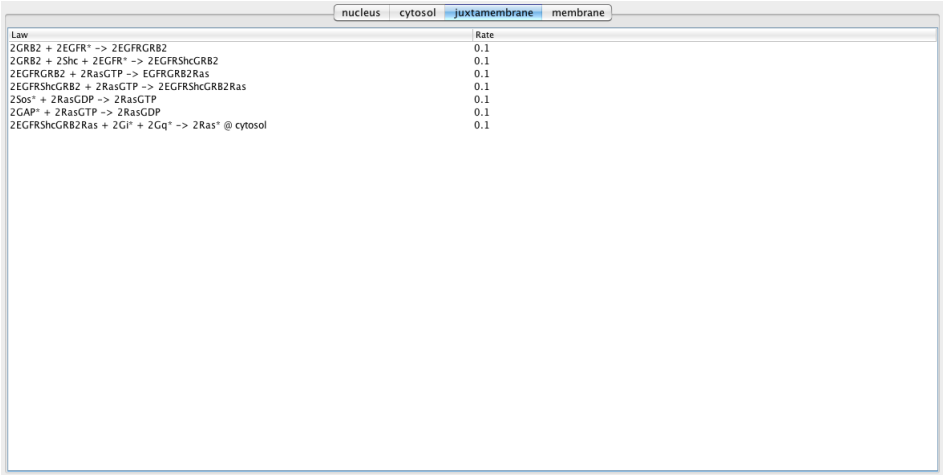

Figura 4.24: Visualizzazione delle leggi ordinarie

che riguarda le leggi ordinarie o quelle temporali, come mostrato nelle figure 4.24 e 4.25.
La tabella non è modificabile, in quanto si occupa solamente di mostrare all'utente le leggi presenti all'interno del compartimento.

4.2.3 Visualizzazione della concentrazione/molarità dei reagenti nel tempo

Alla richiesta di questa opzione, viene creata una nuova finestra e non è necessario avere un compartimento selezionato per attivarla.
Come si vede in figura 4.26, nella parte superiore abbiamo una *combo box* in cui è possibile selezionare il compartimento in cui si vuole visualizzare la concentrazione/molarità di un dato reagente, oppure un'ultima azione *All Compartments*, che contiene le informazioni su tutti i compartimenti.
Nella parte centrale della finestra, vi è una tabella in cui sono elencati tutti i reagenti presenti attualmente nel sistema. Essa ha tre colonne ed un numero di righe pari alla somma dei reagenti presenti nei vari compartimenti, o nel compartimento selezionato dalla *combo box*.
La prima colonna è riempita col nome del compartimento cellulare in cui si trova il reagente, la seconda con il nome del reagente e l'ultima con la quantità del reagente presente nel momento selezionato.

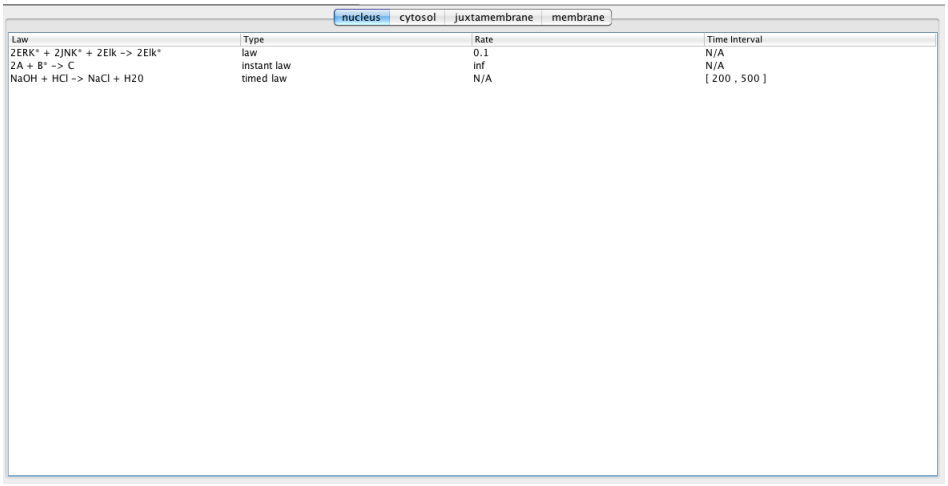

Figura 4.25: Visualizzazione di tutte le leggi di un compartimento

La parte destra della finestra, invece presenta un campo, in cui è possibile specificare il tempo della simulazione in cui si vogliono ottenere le informazioni sulla concentrazione/molarità dei reagenti. Un altro modo per specificare il tempo è far scorrere lo *slider* superiore.

4.2.4 Visualizzazione del grafico della concentrazione/molarità dei reagenti nel tempo

La funzionalità è la stessa descritta in precedenza, ma da un punto di vista differente. Infatti se prima l'informazione immediatamente reperibile era la fotografia delle concentrazioni/molarità in un determinato istante di tempo, ora si vuole descrivere l'andamento della concentrazione/molarità dei reagenti nel tempo.
Questo grafico, mostrato nelle figure 4.27 e 4.28, mostra in ascissa il tempo di simulazione e in ordinata i valori discreti della concentrazione/molarità di un reagente. Seppur le funzioni graficate sono a valori continui nelle ordinate, per questioni estetiche, esse hanno un senso solo nei valori misurati, che sono disegnati con pallini o altre forme geometriche simili.

Nel grafico sono presenti tutti i reagenti presenti nel sistema (o nel

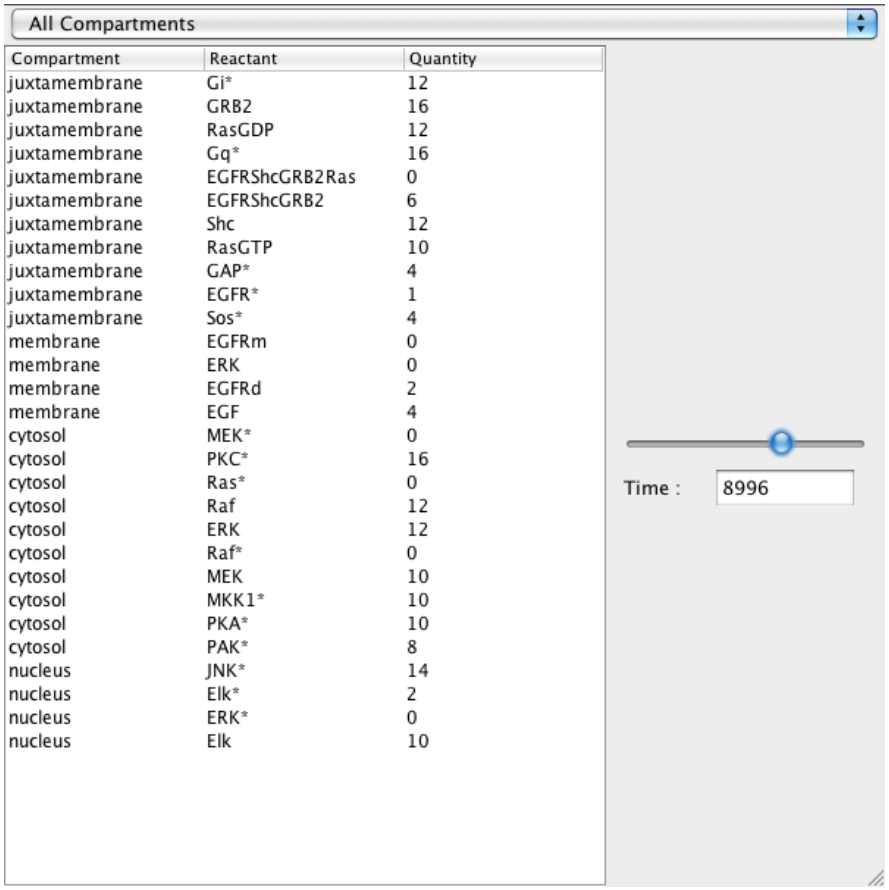

Figura 4.26: Tabella della concentrazione/molarità dei reagenti nel tempo

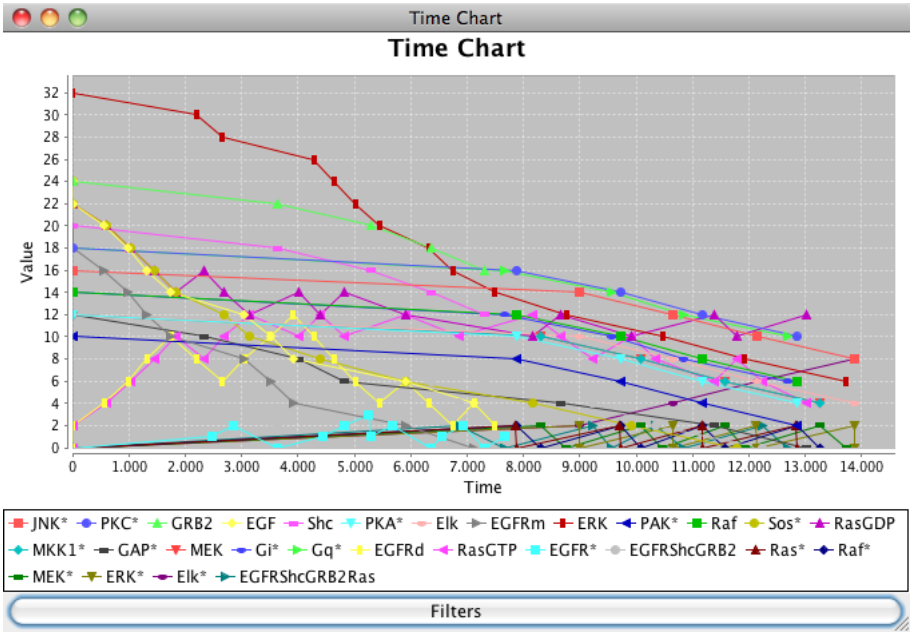

Figura 4.27: Grafico della concentrazione/molarità dei reagenti nel tempo

compartimento se si è scelto di visualizzare il grafico per un solo compartimento), ma è possibile selezionarne solo alcuni, tramite un opportuno bottone *filters*. Questo bottone attiva un pannello in cui è possibile selezionare o deselezionare tramite *check boxes* i reagenti che si vogliono visualizzare nel grafico, come mostrato in figura 4.29. In questo modo è possibile far scegliere all'utente le funzioni da visualizzare nello stesso grafico, anche in ottica di paragonare gli andamenti tra loro.

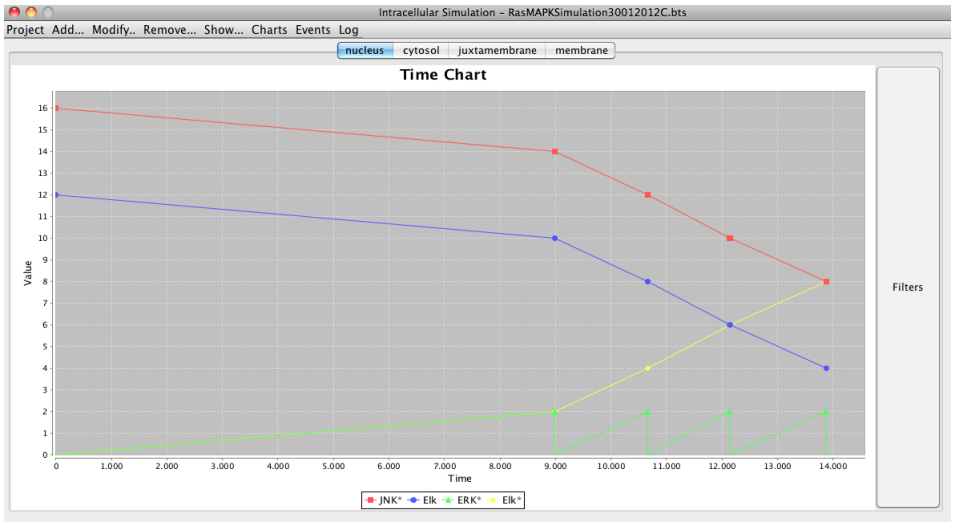

Figura 4.28: Grafico della concentrazione/molarità dei reagenti nel tempo per compartimento

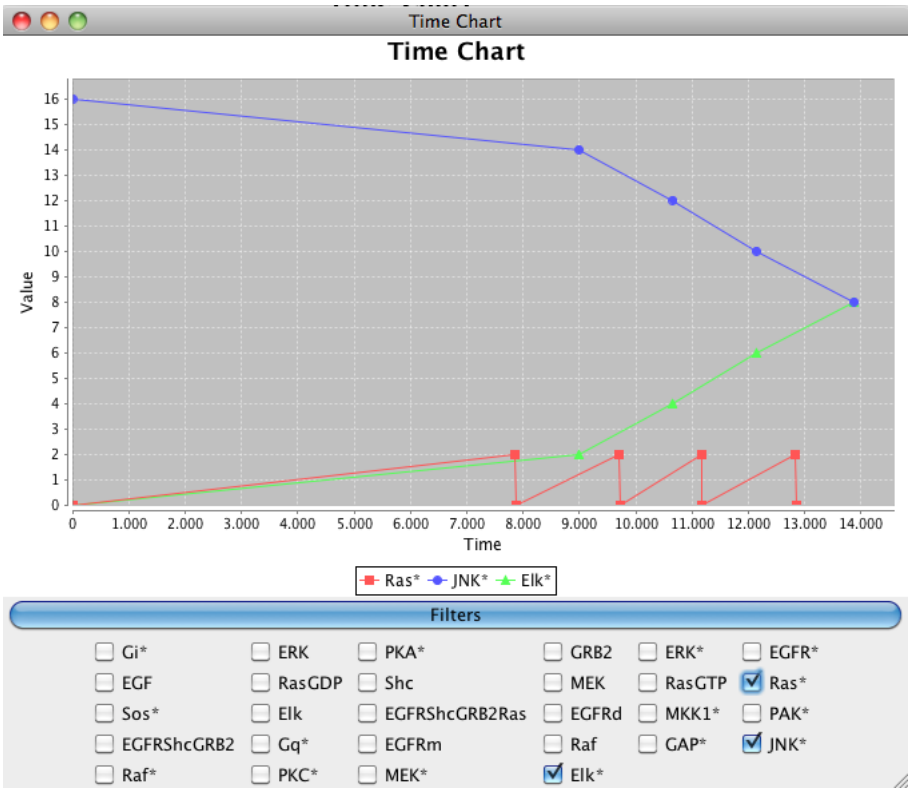

Figura 4.29: Grafico della concentrazione/molarità dei reagenti nel tempo visualizzazione filtri

Capitolo 5

Modellazione e simulazione della rete di segnalazione PI3K/AKT

In questo capitolo si affronta la modellazione e la simulazione di un caso di studio, che tratta di una via di segnalazione coinvolta nella induzione di *apoptosi* o morte cellulare programmata.
Si procede con la descrizione di tale via di segnalazione, denominata *PI3K/AKT*, a cui fa seguito la modellazione vera e propria della stessa, per poi illustrare i passi più importanti che hanno portato alla sua implementazione, simulazione e sperimentazione sulla piattaforma BTSSOC-Cellulat.

5.1 La via di segnalazione intracellulare Phosphatidylinositol 3-kinases (PI3K)/AKT (Protein Kinase B)

Le *phosphatidylinositol 3-kinases (PI3K)* costituiscono una famiglia conservatrice di *lipid kinases*, che catalizzano la fosforilazione dei *phosphatidylinositol (Ptdlns) membranali*, nella posizione D3 dell'anello dell'inositol, dove il *phosphatidylinositol-(3,4,5)-triphosphate* è il più

diffuso [3]. Queste proteine kinase sono *eterodimeri* formati da una sottounità catalitica, *p110*, e da una sottounità regolatrice, in cui *p85* è la più comune.

Le PI3Ks vengono classificate a seconda della selettività del sostrato e si raggruppano in cinque classi (IA, IB, II, III e IV), anche se solo la PI3K appartiene alla classe IA, la cui funzione viene associata alla trasduzione dei segnali provenienti dai recettori *tirosin-kinase*, è stata relazionata con il cancro in esseri umani [19]. D'altro canto, bisogna tener conto anche della classe IB, che partecipa alla trasduzione dei segnali provenienti dai recettori di membrana accoppiati alla proteina G, e può attivare la PI3K-IA attraverso il reclutamento di *tirosin-kinase* intracellulari [19].

La PI3K controlla parecchie funzioni chiave correlate alla biologia del cancro, che includono la proliferazione, la sopravvivenza cellulare, la migrazione e l'*angiogenesi* [19] [3]. L'attività della PI3K viene regolata da due *phosphatase*:

1. *PTEN (phosphatase and tensin homolog)*, il quale è un soppressore tumorale, dal momento che idroliza il *PIP3* localizzato nella posizione 3', generando la sua inattivazione e l'inibizione della via di segnalazione [3] [22]

2. *SHIP*, phosphatase che agisce sul PIP3 localizzato nella posizione 5' generando il *phosphatidylinositol-(3,4)-diphosphate (PIP2)*, il quale è un messaggero secondario [3] [22] (figure 5.1).

La *serine/threonine protein kinase AKT*, anche conosciuta come *protein kinase B(PKB)*, costituisce uno dei principali *target* di PIP3. Negli esseri umani sono state individuate tre isoforme diverse (AKT1, AKT2 e AKT3). L'unione di AKT a PIP3 genera un cambiamento conformazionale in AKT che porta alla sua fosforilazione e, dunque, viene attivata dalle *phosphoinositide-dependent kinase (PDKs)* [15].

Fra i target di AKT si trovano molteplici fattori di trascrizione e proteine di segnalazione, dal momento in cui l'attivazione di AKT porta sia alla diminuzione dei livelli d'espressione o attività di parecchie

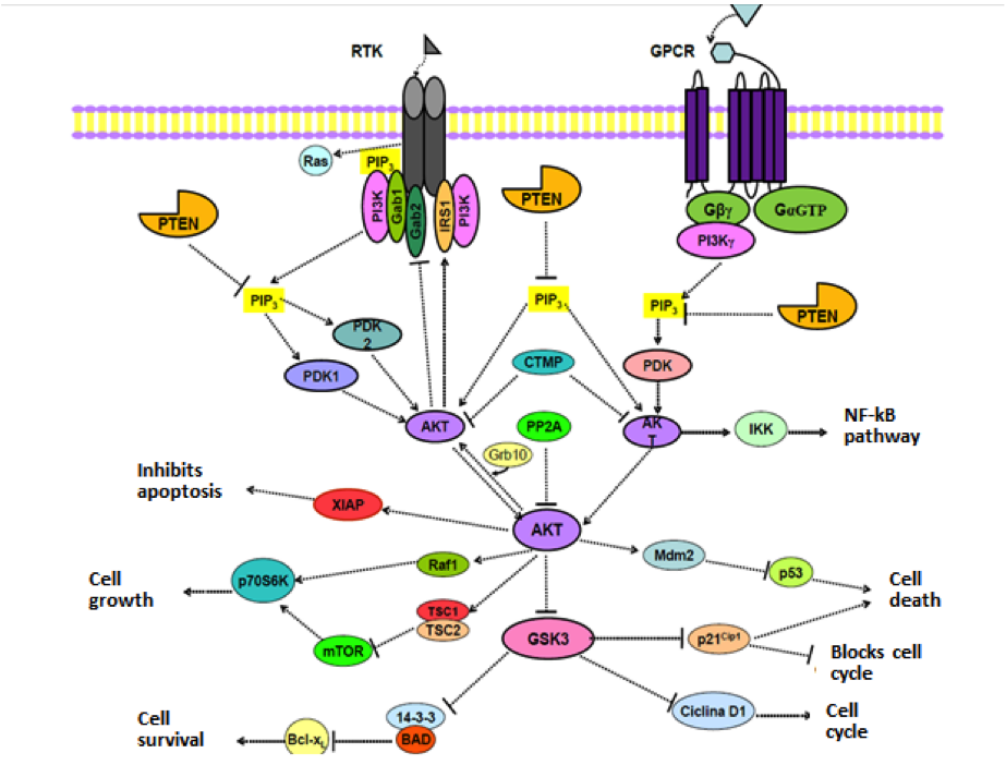

Figura 5.1: Via di segnalazione PI3K/AKT

proteine proapoptotiche, come *FasL* e *Bim* [19] [15] [7], che all'aumento delle proteine *antiapoptotiche Bcl-2, Bcl-xL e XIAPs* [19] [15] , e delle proteine di sopravvivenza come la stessa AKT [19] [15]. La proteina AKT promuove, inoltre, il ciclo cellulare attraverso l'inibizione della regolazione dei ponti di controllo, come la fosforilazione e inibizione degli inibitori delle *cyclin-dependent kinases p21Cip/WAF1 e p27Kip1* [19] [15] e *glycogen synthase (GSK3-α e β)*, aumentando la glicolisi nella cellula e, dunque, lo sviluppo cellulare [19].

Per quanto riguarda il ruolo di AKT nel cancro, è stato ben stabilito e descritto che l'attivazione di AKT costribuisce direttamente alla sopravvivenza e proliferazione delle cellule cancerogene, così come all'aumento della misura della loro dimensione fisica [12] [15].

5.2 Processo incrementale per la definizione del modello delle vie di segnalazione

Si è scelto di scalare la sua complessità intrinseca in maniera incrementale come vediamo in figura 5.2.
Si sceglie, inizialmente, quindi di prendere in esame solo una via di segnalazione, delle tante presenti in PI3K/ATK, per poi arrivare a comprenderne altre e rendere la simulazione più completa.
La seconda dimensione in cui si semplifica il modello è quella delle leggi, innanzitutto vengono prese in considerazione leggi molto semplici, in cui i componenti compaiono con molarità unitaria, che occorrono con un rate uguale. Non vi è nessuna restrizione sui tipi di reazioni, mostrati nel capitolo 2, possono essere infatti utilizzare reazioni collaborative, come

$$[x]A* + [y]B* + [z]C \rightarrow [z]C*$$

che alternative, ad esempio

$$[x]A* + [y]B \rightarrow [y]B*$$

$$[x]C* + [y]B \rightarrow [y]B*$$

che complesse:

$$[x]A* + [y]B* + [z]C \rightarrow ABC$$

Infine, i componenti di segnalazione in esame avranno solo due stati, attivo e non attivo. In futuro saranno presi in esame casi in cui l'attivazione avviene per gradi, fino ad arrivare ai livelli di dettaglio richiesti (e.g. concentrazione monomera, concentrazione dimera, stato di fosforilazione, etc.).

Per concludere si assume che la quantità di soluzione nei compartimenti cellulari sia sempre di *1 litro*, rendendo la concentrazione pari al numero di moli, come vediamo dalla sua definizione

$$concentrazione = \frac{num.moli}{num.litri}$$

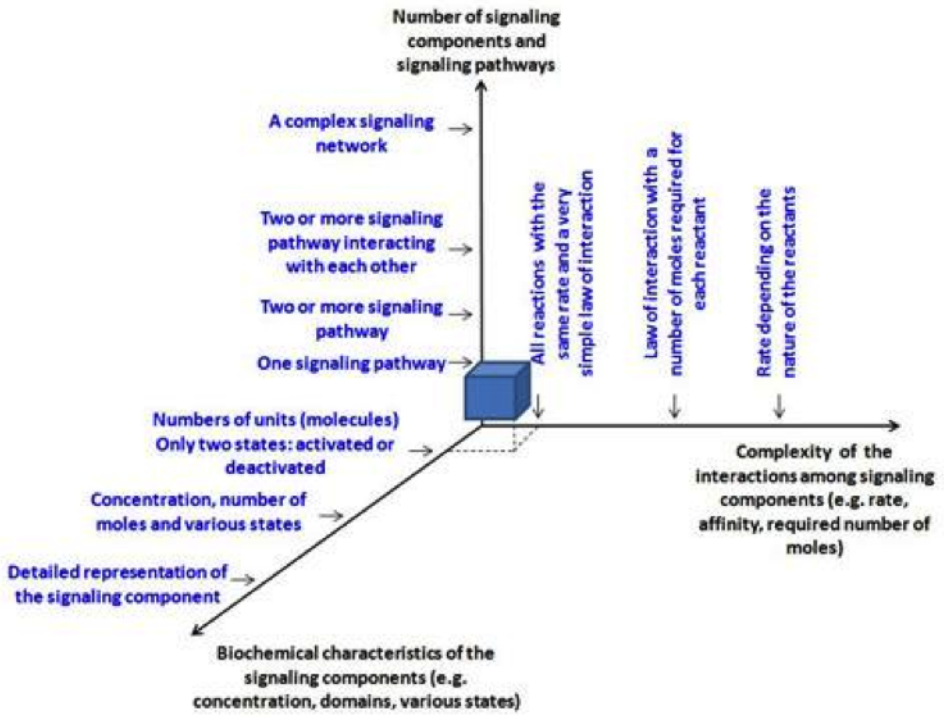

Figura 5.2: Processo di modellazione di tipo incrementale delle vie di segnalazione

5.3 Modellazione della via di segnalazione AKT/PI3K

In questa sezione sarà mostrato solamente il livello più semplice, scopo di questa tesi è infatti presentare la piattaforma *BTSSOC-Cellulat*, di cui si mostrerà il funzionamento con un caso di studio reale.
Sono stati innanzitutto individuati gli elementi di segnalazione in gioco, ovvero quelli che che sono coinvolti nella via di segnalazione, che sono mostrati in tabella 5.1.

Per completare la modellazione, come richiesto dal modello *BTS-based*, sono state quindi descritte reazioni, che indicano le interazioni tra i vari componenti di segnalazione, descritti in precedenza, e il comartimento ai quali appartengono. A questo scopo si faccia

Tabella 5.1: Elementi di segnalazione coinvolti nella via di segnalazione PI3K/ATK

Types of signaling elements	**Acronym**	**Full name**
Receptor	GPCR	G-protein Coupled Receptors
Receptor	RTK	Receptor Tyrosine Kinases
Ligands. Growth factor	EGF, PDGF, TGF-beta, TGF-a IGF-I	Epidermal, Platelet Derivated, Tumor, Insulin Growth Factor
Subunit	Gbeta	G protein Beta subunit
Subunit	Ggamma	G protein Gamma subunit
Kinase protein	PI3K	Phospatidil Inositol 3 Kinase
Addapter protein	Gab1	GRB2-Associated-Binding Protein 1
Adapter protein	Gab2	GRB2-Associated-Binding Protein 2
Receptor	IRS1	Insulin Receptor Substrate 1
Signaling component	PIP3	Fosfatidilinositol-(3,4,5)-Trifosfato
Signaling component	PIP2	Fosfatidilinositol-(3,4)-Difosfato
Kinase protein	PDK	Phosphoinositide-dependent protein kinase
Kinase protein	AKT/PKB	serine/threonine kinase (protein kinase B)
Negative regulator	Mdm2	Murine Double Minute 2
Kinase protein	IKK	Inhibitor of Nuclear Factor Kappa-B Kinase Subunit Beta
Inhibitor of apoptosis	XIAP (IAP3)	X-linked inhibitor of apoptosis protein
Kinase protein	Raf 1	a mitogen-stimulated serine-threonine protein kinase
Peripheral membrane protein Vesicular transport	TSC1 Hamartin	Tuberous Sclerosis protein 1
Peripheral membrane protein Vesicular transport	TSC2 Tuberin	Tuberous Sclerosis protein 2
Protein that binds DNA	p53	Protein 53

riferimento alle tabelle 5.2, 5.3 e 5.4.

Tabella 5.2: Leggi che regolano la segnalazione nella via PI3K / ATK

Cellular compartment	**Reactions**
Extracellular space and Membrane	$GPCR + GF \rightarrow GPCR*$ $RTK + GF \rightarrow RTK*$
Cytosol	$GPCR*+PG+AC \rightarrow GPCR*GBetaGGamma+ACGAlfa$ $ACGAlfa + ADP \rightarrow cAMP$ $cAMP + PI3K \rightarrow PI3K*$ $cAMP + Gab1 \rightarrow Gab1*$ $cAMP + Gab2 \rightarrow Gab2*$ $cAMP + IRS1 \rightarrow IRS1*$
Cytosol	$GPCR * GBetaGGamma + PI3K * + PIP3 \rightarrow PIP3*$ $RTK*+PI3K*+Gab1*+Gab2*+IRS1*+PIP3 \rightarrow PIP3*$
Cytosol	$PIP3 * + PDK \rightarrow PDK*$ $PIP3 * + PDK1 \rightarrow PDK1*$ $PIP3 * + PDK2 \rightarrow PDK2*$ $PIP3 * + AKT \rightarrow AKT*$
Cytosol	$PDK * + AKT \rightarrow AKT*$ $PDK1 * + AKT \rightarrow AKT*$ $PDK2 * + AKT \rightarrow AKT*$

Tabella 5.3: Leggi che regolano la segnalazione nella via PI3K / ATK

Cellular compartment	**Reactions**
Cytosol	$AKT* + Mdm2 \rightarrow Mdm2*$ $AKT* + IKK \rightarrow IKK*$ $AKT* + AKT \rightarrow AKT*$ $AKT* + XIAP \rightarrow XIAP*$ $AKT* + Raf1 \rightarrow Raf1*$ $AKT* + IRS1 \rightarrow IRS1*$ $AKT* \rightarrow ATK* @nucleus$
Nucleus	$AKT* + TSC1TSC2 \rightarrow TSC1TSC2*$
Cytosol	$Mdm2* + p53* \rightarrow p53$ $AKT* + GSK3* \rightarrow GSK3$ $4p53* \rightarrow 2p53* @nucleus$
Nucleus	$TSC1TSC2* + mTOR* \rightarrow mTOR$
Nucleus	$Raf1* + p70S6K \rightarrow p70S6K*$ $XIAP* \rightarrow event(''Inhibitsapoptosis'')$
Mitochondria	$4GSK3* \rightarrow 2GSK3* @nucleus + 2GSK3* @cytosol$
Nucleus	$p53* \rightarrow event(''Celldeath'')$ $GSK3* + p21Cip1* \rightarrow p21Cip1$ $GSK3* + cyclinDI* \rightarrow cyclinDI$

Tabella 5.4: Leggi che regolano la segnalazione nella via PI3K / ATK

Cellular compartment	**Reactions**
Cytosol	$GSK3* + BAD/1433* \rightarrow BAD/1433$
Nucleus	$mTOR* + P70S6K \rightarrow P70S6K*$
Nucleus	$P70S6K* \rightarrow event(''Proliferation'')$
Mitochondria	$4BclXL* \rightarrow 2BclXL* @cytosol$
Cytosol	$BAD/1433* + BclXL* \rightarrow BclXL@nucleus$
Nucleus	$p21Cip* \rightarrow event(''BlocksCellCycle'')$ $cyclinDI* \rightarrow event(''CellCycle'')$ $BclXL \rightarrow event(''Survival'')$

5.4 Implementazione, simulazione e sperimentazione della via di segnalazione AKT/PI3K sulla piattaforma BTSSOC-Cellulat

A partire dal modello precedente, sarà ora illustrato come implementare la simulazione nella piattaforma *BTSSOC-Cellulat*.
Come descritto in precedenza questa simulazione farà riferimento ad un modello minimalista, che considera solamente le caratteristiche principali proprie della via di segnalazione AKT/PI3K, ovvero quel-

le strettamente necessarie per mostrare l'utilizzo della piattaforma e raggiungere un primo prototipo che sia significativo e funzionante.
In questa sede, si assume che le quantità di reagenti introdotte nei vari compartimenti siano tutte pari a 60unità, non facendo quindi riferimento a misure chimico-fisiche realistiche, quali le moli.
Inoltre tutti i rate delle reazioni saranno eguali, come scritto in precedenza ad una costante, posta a 0.001.
Queste assunzioni, seppur non corrispondenti alla realtà, ci permettono di verificare il funzionamento e la giusta interazione tra le parti che compongono il sistema.
Si passa ora alla descrizione delle parti più salienti dell'implementazione della simulazione:

1. *Creazione del progetto*: avviata l'applicazione bisogna innanzitutto creare un nuovo progetto, scegliendo l'opportuna voce del menù project, *New Project..*, attivando in questo modo tutti i menù e inizializzando il sistema, per renderlo pronto ad una nuova simulazione.

2. *Creazione dei compartimenti*: a questo punto si creano i compartimenti, in gioco nella via di segnalazione che si vuole simulare (in questo caso di studio sono quattro: *membrana*, *cytosol*, *mitochondria* e *nucleus*, quelli che avevamo analizzato in fase di modello); scegliendo la voce *Add Compartment* del menù *Add...*, compare una nuova finestra (fig. 4.5), in cui è possibile inserire il nome del compartimento cellulare da creare dove è possibile premere il bottone *ok* per rendere effettiva la creazione. Una volta terminata questa fase, verrano visualizzati i compartimenti cellulari nel pannello centrale come mostrato in figura 5.3.

3. *Introduzione dei reagenti*: in questo passo si prendono i reagenti mostrati in tabella 5.1 e si introducono negli opportuni compartimenti cellulari; per prima cosa bisogna attivare il pannello relativo al compartimento intracellulare in cui si vuole introdurre il reagente, selezionando la relativa *label*, per poi scegliere la voce *Add Reactant*, sotto il menù *Add...*, viene creata una nuova finestra (fig. 4.6), in cui è possibile introdurre il nome del reagente e la quantità desiderata, in questo caso 60, come descritto

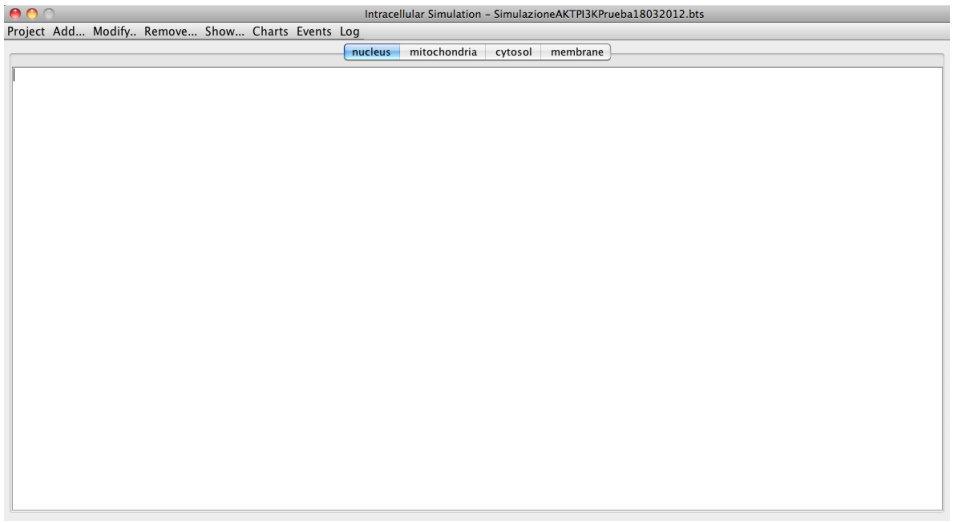

Figura 5.3: Vista dei compartimenti della simulazione

in precedenza.
Riassumendo nella simulazione implementata, all'interno della *membrana* sono presenti: GPCR, RTK e GF; nel *cytosol*: XIAP, BAD1433*, P53*, AC, Raf1, IRS1, PDK2, ADP, PG, Gab2, Gab1, PI3K, PIP3, PDK, PDK1, AKT, Mdm2, IKK, PIP3*; nella *mitochondria*: BclXL*, GSK3*; e infine nel nucleo: CyclinD1*, MTOR*, TSC1TSC2, AKT*, p70S6K, P21Cip1*. E' possibile verificare l'esattezza dei reagenti introdotti scegliendo la voce *Show Reactants* nel menù *Show..*, in questo modo il pannello mostrerà un grafico a colonne (una per ogni reagente), la cui altezza è pari al numero di unità di quel reagente presente nel compartimento intracellulare.

4. *Introduzione delle reazioni*: prendendo in esame le reazioni all'interno delle tabelle 5.2, 5.3 e 5.4, si introducono negli appositi compartimenti intracellulari le reazioni chimiche: scegliendo la voce *Add Reactant* sotto il menù *Add..*, appare una nuova finestra (fig. 4.7), in cui va inserita la legge, scritta nel linguaggio delle reazioni chimiche, come descritto nel capitolo 3, e il suo rate che in questo caso sarà pari a 10^{-3}, inserendo quindi il valore *1e-3*. Come per i reagenti è possibile verificare la correttezza

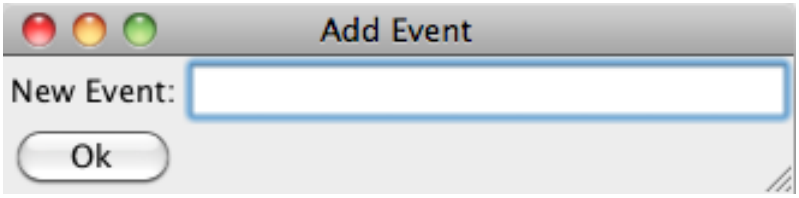

Figura 5.4: Finestra di aggiunta di un evento

delle leggi inserite, per poi eventualmente modificarle, selezionando *Show Laws* all'interno del menù *Show...*, facendo sì che nel pannello centrale compaia una tabella contenete tutte le leggi all'interno del compartimento intracellulare selezionato.

5. *Definizione degli eventi*: l'ultimo passo è quello di definire gli eventi illustrati nelle tabelle 5.2, 5.3 e *lawsatk3*, che avvengono nel nucleo. A questo scopo, dopo aver selezionato il compartimento *nucleus*, si sceglie la voce *Handle compartment events*, del menù *Events*. Nel pannello centrale compaiono, a questo punto, tre ulteriori pannelli che contengono rispettivamente: la lista degli venti, la lista delle azioni e la lista delle leggi che attivano l'evento (fig. 4.20).
   Alla pressione del bottone *Add event*, si apre una nuova finestra in cui è possibile introdurre il nome dell'evento (fig. 5.4), selezionando il tasto *ok*, viene creato e aggiunto nella lista a sinistra l'evento appena creato. Scegliendo quella voce nella lista, si sbloccano gli altri due pannelli, che saranno vuoti, in cui è possibile aggiungere azioni relative all'evento, che in questo caso non sono presenti, in quanto l'obiettivo finale è solamente la notifica, dell'esecuzione dell'evento, e il pannello relativo alle leggi che attivano l'evento.
   Premendo, quindi il bottome *Link Law*, viene mostrato un sottomenù (fig. 5.5), in cui è possibile scegliere il tipo di legge che scatenerà l'evento, in qusto caso, avendo solo leggi ordinarie, *Link a Law*. Infine immettiamo la legge che scatenerà l'evento nella finestra che compare, analogamente al caso di immissione precedente delle leggi.

6. *Salvataggio della simulazione*: per concludere si salva la simu-

Figura 5.5: Menù di link delle leggi

lazione in un nuovo file, sfruttando la voce *Save project as..*, all'interno del menù *Project*. In questo modo per ripetere la simulazione non è necessario ripetere tutti i passi precedenti, ma solamente ricaricare il progetto dallo stesso menù.

Terminata la fase di inizializzazione, si passa all'esecuzione della simulazione, premendo il bottome *Start Simulation*, si stimolano i *BTS*, che iniziano il loro comportamento iterativo.
Al termine della simulazione si ottiene un risultato simile, essendo l'algoritmo di Gillespie [9], un algoritmo stocastico, a quello mostrato in figura 5.6, che è un grafico che mostra i cambiamenti dei reagenti nell'intervallo di tempo.
Analizzando meglio alcune parti del grafico, ad esempio la membrana, in particolare guardando i reagenti e le leggi che sono state inserite (tabella 5.2), notiamo che all'interno della membrana saranno eseguite esattamente sessanta reazioni, pari alle unità di *GF* introdotte, quindi come risultato finale si otterrà, all'interno della membrana, che non vi sarà nessuna unità di *GF* e nel contempo saranno consumate altre sessanta unità, tra *GPCR* e *RTK*, lasciandone ulteriori sessanta unità all'interno del compartimento. La figura 5.7 mostra come questo risultato atteso sia stato raggiunto, infatti all'interno della membrana sono presenti ventinove unità di *RTK* e trentuno unità di *GPCR*, come previsto.

I prodotti di queste leggi, GBRC* e RTK*, devono essere entrambi pubblicati nel compartimento *cytosol*, e come possiamo vedere dal grafico temporale del compartimento in questione, questo avviene, infatti vi è un aumento di questi nel tempo, al suo interno come mostrato in figura 5.8 La simulazione effettuata in questa prova, ha portato all'attivazione dell'evento *proliferation*, quando questo avviene viene

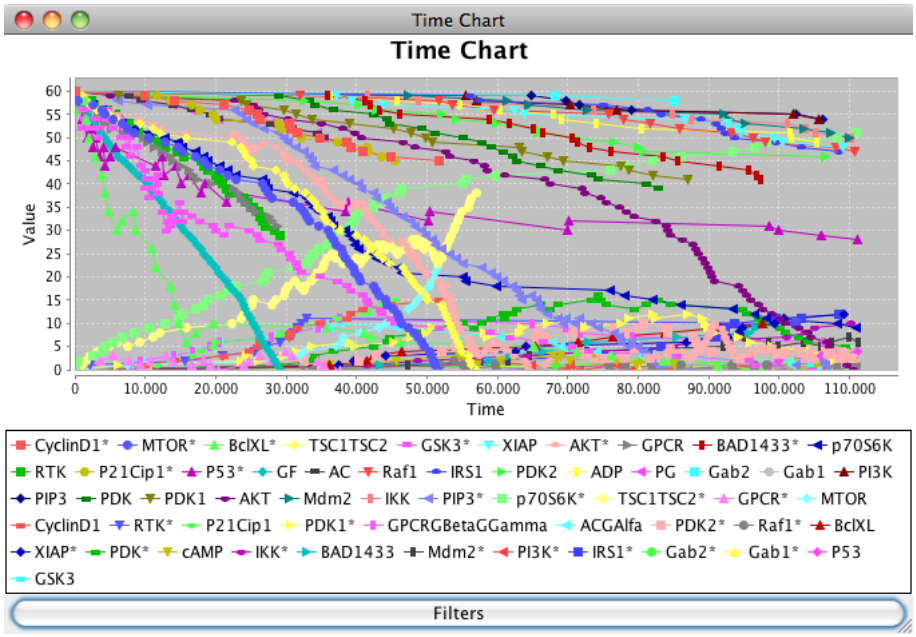

Figura 5.6: Grafico temporale della simulazione PI3K/ATK

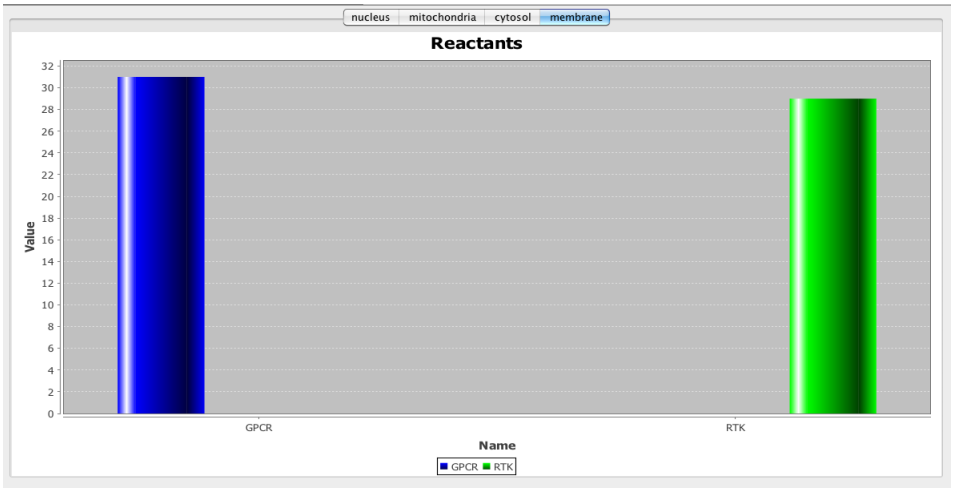

Figura 5.7: Reagenti residui nella membrana al termine della simulazione PI3K/ATK

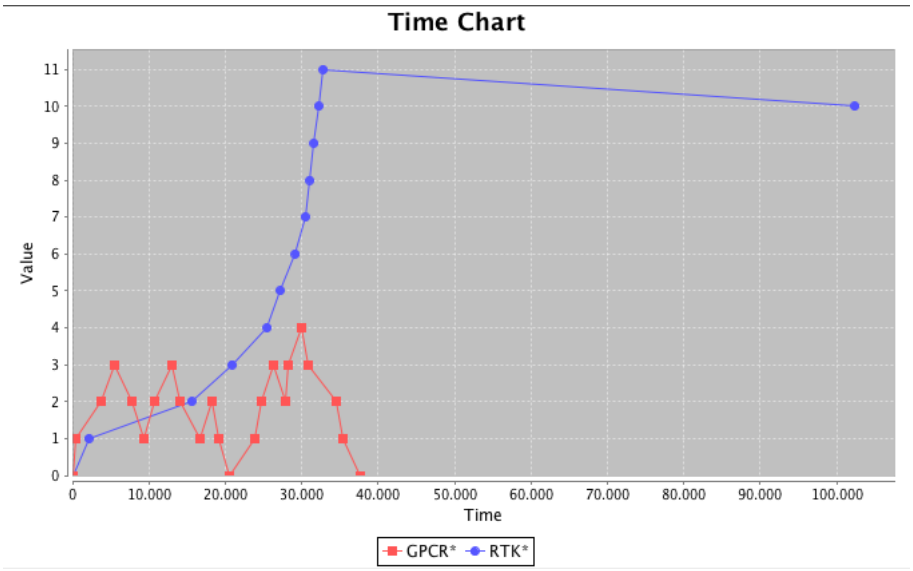

Figura 5.8: Grafico temporale dei reagenti GBRC* e RTK* all'interno del cytosol

visualizzato un warning sulla piattaforma, come possiamo notare in figura 5.9, contenente il tempo di attivazione, in millisecondi.

Scopo di questa simulazione della via di segnalazione PI3K/ATK è quello di mostrare l'attivazione di eventi quali *morte cellulare* e *proliferazione*. Facendo riferimento alla tabella 5.4, vengono mostrati in figura 5.10, l'evoluzione nel tempo dei reagenti che giocano il ruolo principale in questi processi. Dal grafico possiamo notare infatti che p60S6K*, responsabile della *proliferazione* è il primo a manifestarsi, e anche il più veloce a riprodursi.

Sebbene le condizioni iniziali e i rate delle reazioni siano descritti in maniera semplificata e lontana dalla realtà biologica, esse potranno e dovranno essere raffinate nel tempo, per cercare di validare i risultati raggiunti anche su dati sperimentali, e non solo qualitativamente a livello di comportamento, come descritto in questa sede. I feedback della biologia rimangono sempre di grandissima importanza nel processo di raffinamento della piattaforma e per possibili estensioni, in un'ottica di sviluppo a spirale.

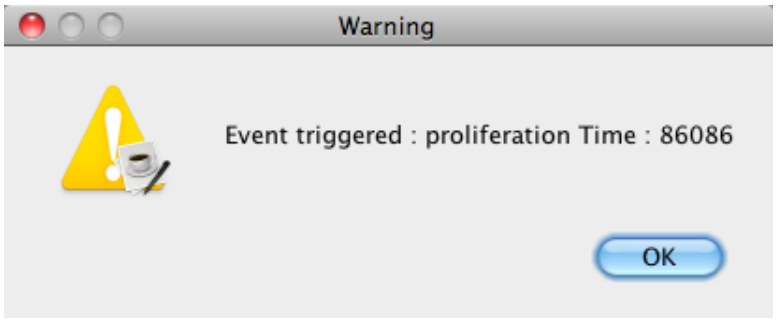

Figura 5.9: Messaggio di avviso per l'evento *proliferation*

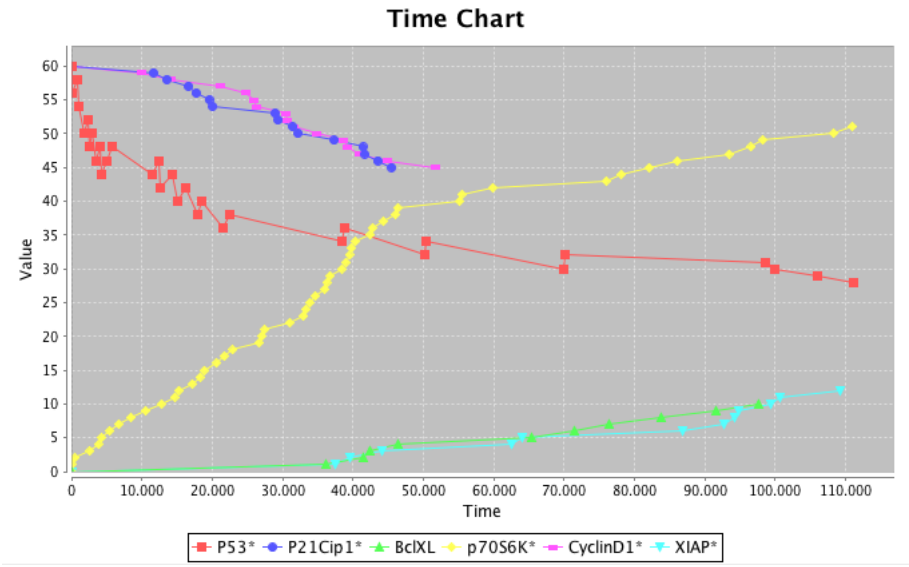

Figura 5.10: Grafico temporale di alcuni reagenti nel nucleo, responsabili di morte cellulare o proliferazione

Capitolo 6

Conclusioni e possibili sviluppi

Lo sviluppo della piattaforma BTSSOC-Cellulat è giunto, con questo lavoro, in una fase decisiva. In questo documento viene mostrato infatti come essa possa già essere utilizzata per simulare processi biologici non semplici.
BTSSOC-Cellulat è attualmente in uso per la predizione *in silico* del meccanismo di azione dei *polifenoli* anticancro degli alimenti; la via di segnalazione intracellulare PI3K/ATK, presa in esame, infatti, è una delle tre che inducono la cellula in *apoptosis*, assieme a MAPK e PKC.
Gli esperimenti finora condotti, con questa piattaforma, dimostrano il funzionamento di tutte le meccaniche introdotte nel modello, sia con simulazioni complete, ma anche con simulazioni volte a testare solamente alcuni aspetti.
Tuttavia, sono ancora necessari dati sperimentali dalla ricerca, per poter verificare ulteriormente il corretto funzionamento.
Il risultato raggiunto è lo sviluppo di una nuova piattaforma per la simulazione intracellulare secondo il modello *BTS-Based*, la quale è pronta per essere utilizzata, grazie all'interfaccia grafica sviluppata, anche da utenti non esperti di informatica, quindi utile per la ricerca in campo biologico.
Il modello di base è oltretutto notevolmente semplice e intuitivo, e si basa su poche astrazioni, ovvero quella di compartimento intracellulare come centro di tuple, di reagente come segnale intra ed extra-cellulare,

e di reazione chimica come elemento di segnalazione.
La piattaforma offre allo stato attuale diverse funzionalità, come il controllo sui reagenti, sulla definizione delle leggi secondo il linguaggio chimico, la definizione di eventi e il salvataggio e il caricamento della simulazione, che si sono rivelate utili nella modellazione e simulazione del caso di studio.
L'ottica secondo la quale è stata sviluppata, è quella di un processo di sviluppo a spirale, quindi è aperta alla richiesta di nuove funzionalità, che emergeranno con l'utilizzo.
La specifica del motore chimico per raggiungere ad un effettivo tempo di simulazione, seppur completa, presenta qualche problema a livello di infrastruttura *TuCSoN*, sulla quale è stato sviluppato, vengono infatti perduti alcuni messaggi, facenti parte del protocollo di sincronizzazione implementato in questo lavoro, invalidandolo. Con la soluzione a questi problemi l'applicazione avrà questa nuova funzionalità, grazie al lavoro sinora svolto, cambiando semplicemente la configurazione iniziale.
Saranno inoltre sviluppati ulteriori grafici, atti a catturare altre proprietà della simulazione svolta, facendone una fotografia altri punti di vista utili.
Un ulteriore cambiamento sarebbe la modifica del concetto di *rate*, con una misurazione biologica precisa, in quanto con questa astrazione i ricercatori sono costretti ad elaborare i dati raccolti, al fine di determinare la costante frequenza di una reazione.
Per quanto riguarda gli eventi, sarebbe inoltre opportuno estendere le azioni possibili, con altre che permettano di modellare nel migliore dei modi lo stato raggiunto (i.e. proliferazione apoptosis, divisione nucleare ecc..).

Elenco delle figure

Elenco delle tabelle

Bibliografia

[1] aliCE. Respect site. http://respect.alice.unibo.it/, 2008.

[2] aliCE. Tucson site, 2008.

[3] Cantley. L.c. the phosphoinositide 3-kinase pathway. *Science 296*, 2002.

[4] CellDesigner. Celldesigner site. http://www.celldesigner.org/.

[5] CellWare. Cellware site. http://www.bii.a star.edu.sg/achievements/applications/10cellware/ index.aspapplications/10cellware/index.asp.

[6] COPASI. Copasi site. http://www.copasi.org/.

[7] Dijkers. Expression of the pro-apoptotic bcl-2 family member bim is regulated by the forkhead transcription factor fkhr-l1. *Department of Pulmonary Diseases, University Medical Center Utrecht*, 2000.

[8] Dizzy. Dizzy site. http://magnet.systemsbiology.net/software/Dizzy/.

[9] D.T.Gillespie. Exact stochastic simulation of coupled chemical reactions. *The journal of physical chemistry*, 1977.

[10] Dynetica. Dynetica site. http://www.duke.edu/ you/Dynetica_page.htm.

[11] GEPASI. Gepasi site. http://www.gepasi.org.

[12] Hanahan. The hallmarks of cancer. *Cell*, 2000.

[13] SBML Language. Sbml site. http://sbml.org/.

[14] Matteo Casadei Mirko Viroli. Biochemical tuple spaces for self-organising coordination. *Università di Bologna*, 2009.

[15] Nicholson. The protein kinase b/akt signalling pathway in human malignancy. *Cell Signal 14*, 2002.

[16] Marco Sbaraglia Pedro Pablo Gonzalez Perez, Andrea Omicini. A biochemically-inspired coordination-based model for simulating intracellular signalling pathways, 2012.

[17] Pedro Pablo Gonzalez Perez. Cellulat. *Proceedings of the eighth international conference on Artificial life.*, 2003.

[18] PLAS. Plas site. http://www.dqb.fc.ul.pt/docentes/aferreira/plas.html.

[19] Ptasznik. Crosstalk between bcr/abl oncoprotein and cxcr4 signaling through a src family kinase in human leukemia cells. *J Exp Med*, 2002.

[20] Armindo Salvador Rui Alves, Fernando Antunes. Tools for kinetic modeling of biochemical networks. *Nature Publishing Group*, 2006.

[21] Marco Sbaraglia. *Coordinazione space-based di ispirazione biochimica per la piattaforma bioinformatica Cellulat.* PhD thesis, Bologna University, 2009.

[22] Stambolic. Negative regulation of pkb/akt-dependent cell survival by the tumor suppressor pten. *Cell 95*, 1998.

[23] Pasadena Twain. Pasadena twain site. http://sbw.sourceforge.net/sbw/software/index.shtml.