ALMA MATER STUDIORUM - UNIVERSITA' DI BOLOGNA
SEDE DI CESENA
CORSO DI LAUREA IN INGEGNERIA E SCIENZE INFORMATICHE

Nicolò Sordoni, Davide Solazzi

Utilizzo di MAS per la realizzazione di sistemi biologici: il compartimento cellulare

Sistemi distribuiti

Sommario

1. I Sistemi biologici

L'utilizzo dell'informatica nell'ambito della biologia risale agli anni ottanta, quando venne introdotta la bioinformatica. L'obiettivo di tale disciplina consiste nello sfruttare tecniche informatiche, ma anche nozioni di statistica, matematica e chimica per risolvere problemi biologici. I compiti principali della bioinformatica sono i seguenti:

- Sviluppare modelli statistici che si occupino di interpretare dati provenienti da esperimenti chimici e molecolari al fine di identificare leggi numeriche;
- Costruire strumenti matematici in grado di analizzare sequenze di DNA, RNA e proteine;
- Organizzare in basi di dati le conoscenze acquisite, nel corso degli anni, su genoma e proteoma in modo da rendere i dati accessibili a tutti;

I settori dell'informatica maggiormente coinvolti sono quindi gli algoritmi e le basi di dati, mentre le tecniche principalmente utilizzate consistono nello sfruttare modelli statistici e matematici per descrivere i risultati dei fenomeni biologici, trascurando maggiormente le loro possibili evoluzioni temporali e funzionali. Questo tipo di bioinformatica, studiata nei primi suoi anni di vita, è chiamata appositamente statica, perché si concentra in modo maggiore sul contenuto informativo, piuttosto che sul flusso informativo (bioinformatica dinamica). Il motivo principale per il quale la bioinformatica statica si è trovata in una posizione privilegiata è da individuare nella grande evoluzione che la

biologia ha ricevuto ultimamente. L'analisi e lo studio del genoma umano ha portato alla scoperta di una enorme quantità di dati. Conoscendo in maniera dettagliata il comportamento singolo dei componenti del DNA nasce una nuova sfida, determinare come essi interagiscono tra loro. Da quì la forte necessità di sviluppo di nuove tecniche in grado di modellare il comportamento dei sistemi biologici e non solamente la loro struttura. Questi sistemi risultano però essere molto complessi ed il loro studio richiede l'utilizzo di tecniche non ambigue di modellazione e di analisi. Per questo motivo viene introdotto un nuovo ramo della biologia che prende il nome di biologia dei sistemi che utilizza la teoria dei sistemi complessi per modellare quelli biologici. Questa area della biologia risulta essere oggi fortemente attiva e si sta cercando di sviluppare un progetto simile a quanto avvenuto nel caso del genoma umano ma che sia maggiormente concentrato sulle interazioni e funzionalità dei componenti che lo costituiscono. A tale scopo risultano essere particolarmente adatti i sistemi multiagente (MASs). Un sistema di questo tipo non è altro che un insieme di agenti, oggetti ad entità autonoma, situati in un ambiente e interagenti tra loro secondo una opportuna organizzazione. I MASs consentono di modellare efficientemente i sistemi biologici, in cui l'interazione all'interno del sistema, e dello stesso con l'ambiente che lo circonda, risulta essere fondamentale per dedurre il comportamento finale. Gli approcci per una loro realizzazione sono essenzialmente due:

- Sistemi costituiti da agenti complessi interagenti fra loro e privi di conoscenza relativa all'ambiente che li circonda;
- Sistemi costituiti da agenti semplici interagenti con l'ambiente e consci dell'ambiente circostante;

Nel nostro caso viene richiesta una soluzione ibrida, con agenti complessi ma consapevoli dell'ambiente che li circonda. Nei capitoli successivi verranno descritte le tecnologie ed i dettagli implementativi relativi alla realizzazione del sistema multiagente.

1.1 Concetti di base

I sistemi biologici complessi sono considerati come l'oggetto di studio della biologia dei sistemi. Essa ha come scopo lo studio del comportamento degli organismi viventi come sistemi che si evolvono nel tempo. Integrando informazioni di varia natura e studiando la rete di connessioni dinamiche esistenti tra geni, proteine ed altre molecole, essa si propone di simulare, attraverso modelli computazionali, le proprietà di una cellula ed in generale di tutto l'organismo. La biologia dei sistemi parte quindi dalla conoscenza dettagliata dell'assetto molecolare per determinare i cambiamenti dinamici derivanti da perturbazioni del sistema. I dati sperimentali ottenuti vengono poi elaborati, utilizzando gli approcci della teoria dei sistemi, della bioinformatica e della matematica, con l'obiettivo di creare un modello predittivo del funzionamento dei sistemi e di individuarne le proprietà emergenti. Per poterli studiare è necessaria la creazione di un modello informatico dell'intero sistema. Questi modelli possono essere realizzati secondo due tecniche principali:

- Approccio Top-down;
- Approccio Bottom-up;

Nel primo caso vengono utilizzati modelli matematici creati a partire dai dati raccolti. Ognuno di essi è dotato di alcune proprietà principali, che definiscono il comportamento del sistema, e da equazioni che descrivono come tali proprietà cambiano nel dominio spazio, tempo. Questi tipi di approcci richiedono che siano soddisfatte condizioni complesse di continuità, per questo motivo vengono poco utilizzati. L'approccio Bottom-up risulta invece essere particolarmente adatto alla modellazione dei sistemi biologici complessi, in quanto si parte dalla conoscenza delle singole componenti, delle loro interazioni e del loro comportamento. In questo modo il sistema può essere descritto in funzione delle leggi che caratterizzano l'interazione locale delle componenti.

I sistemi biologici possono essere visti come una gerarchia, in cui ogni strato coinvolge entità che interagiscono fra loro:

- Interazioni tra molecole all'interno del compartimento;
- Interazioni fra i compartimenti all'interno della cellula;
- Interazioni tra le cellule in apparato;
- Interazioni tra gli apparati;

La parte che verrà da noi modellata corrisponde al primo strato, ovvero al compartimento, luogo in cui avvengono i processi intracellulari che possono essere suddivisi in:

- Percorsi metabolici: sono percorsi che contengono delle catene di reazioni biochimiche, dove ognuna di esse funge da sottostrato per la successiva. Questi percorsi controllano la produzione ed il consumo dell'energia all'interno della cellula e risultano essere autosufficienti;
- Percorsi di segnalazione: percorsi che si occupano di regolare e propagare il flusso delle informazioni all'interno ed all'esterno della cellula. Grazie a questi percorsi la cellula è in grado di adattarsi ad un ambiente in continuo mutamento.

1.2 Compartimento

All'interno di una cellula avvengono numerosi tipi di reazioni. Alcune di queste sono incompatibili tra loro, cioè se entrassero in contatto potrebbero produrre sostanze chimiche nocive per la cellula stessa. Affinché questo non avvenga al suo interno è presente una particolare membrana che funge da separatore, suddividendola in più scomparti e garantendo la sicurezza delle reazioni che avvengono. Ognuno di questi scomparti prende il nome di compartimento cellulare ed è il sistema che si intende modellare. Le simulazioni si concentreranno in particolare sull'analisi dei percorsi metabolici contenuti all'interno del compartimento.

1.3 Biocomponenti

Sono i responsabili delle reazioni che avvengono all'interno del compartimento. Essi vengono anche detti biocomponenti attivi in quanto svolgono azioni che consumano e producono molecole, modificando quindi anche lo stato interno del compartimento stesso. Ne esistono di numerosi tipi, ognuno dei quali scatena reazioni differenti. Per poterle scatenare essi necessitano di particolari molecole contenute all'interno del compartimento. L'accoppiamento viene realizzato grazie a particolari unità dette siti di legame, che possono legarsi ad un unico tipo di molecola. Queste unità costituiscono parte di una molecola e si occupano di formare legami con i reagenti. A seconda dei siti che riescono ad effettuare l'accoppiamento con le rispettive molecole , il biocomponente può trovarsi in stati differenti:

- Pronto: rappresenta lo stato di default;
- Inibito: il componente non può eseguire alcuna reazione fino a quando non riesce ad ottenere le molecole necessarie al risveglio;
- Attivo: vengono rafforzati i siti di legame aumentando la probabilità di ottenere le molecole richieste;
- Reattivo: stato in cui avviene la reazione del componente. Al termine le molecole prodotte vengono rilasciate all'interno del compartimento;

I siti vengono divisi anche per ruolo; se tutti i siti di un determinato ruolo riescono a legarsi con le rispettive molecole, viene eseguita una transizione di stato. Le possibili transizioni sono le seguenti:

- Inibitori: pronto → inibito;
- Attivatori: pronto → attivo; inibito → pronto;
- Reattori: pronto, attivo → reattivo;

Di seguito viene riportato un diagramma UML degli stati che mostra quanto descritto in precedenza:

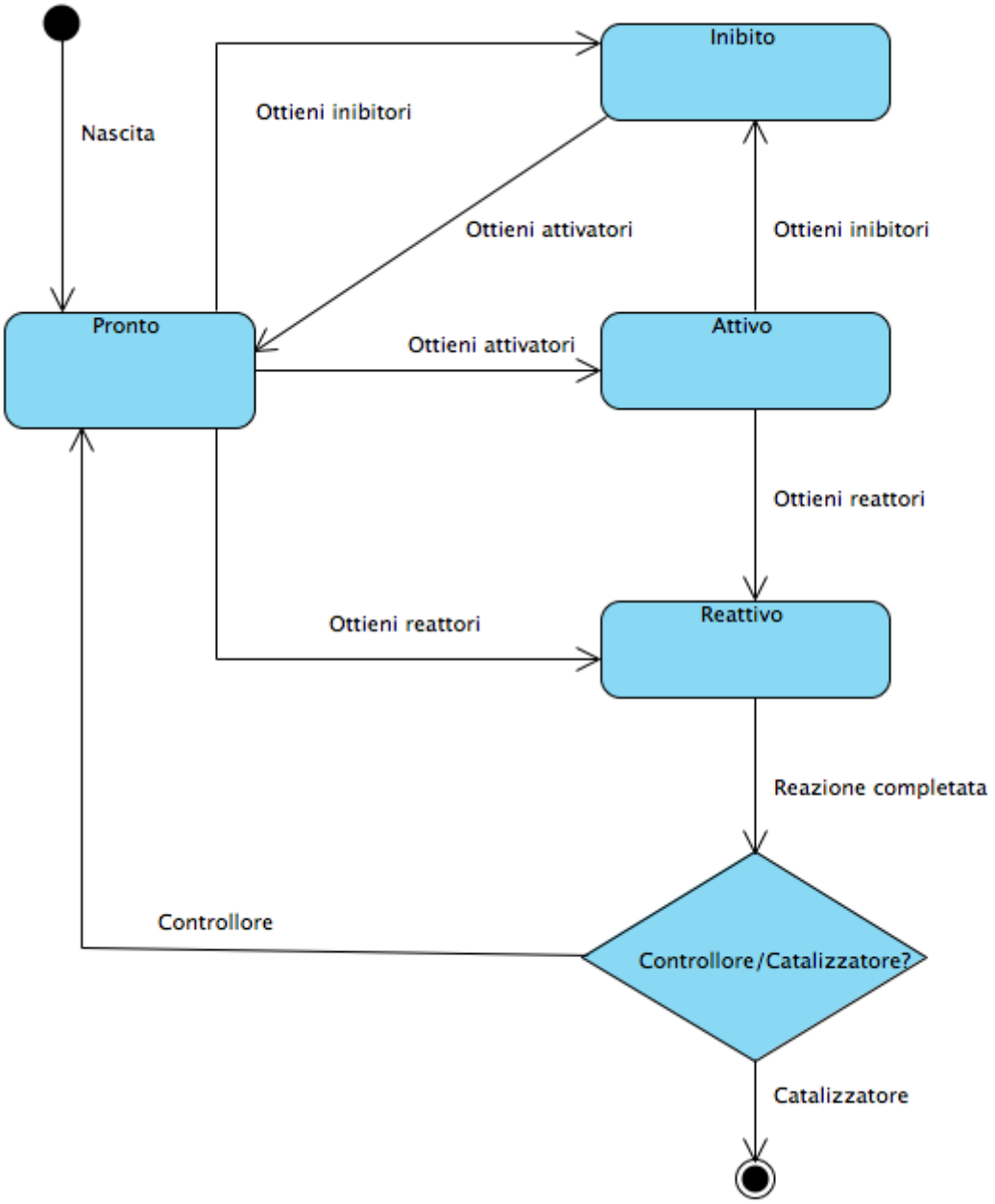

Figura 1.1 Il diagramma degli stati dei biocomponenti

Per poter ricevere le molecole necessarie ad effettuare la reazione ogni componente deve effettuare una richiesta al compartimento. Il numero di richieste che arrivano nel comparto risulta essere elevato a causa del grande numero di componenti presenti all'interno della cellula; inoltre si potrebbe verificare il caso in cui delle molecole, che vengono richieste da un determinato componente per effettuare la reazione, non siano presenti nel compartimento. Per

questo motivo per poter schedulare in maniera corretta una richiesta si è deciso di assegnarle uno stato. Le situazioni possibili sono le seguenti:

- Richiesta da effettuare: rappresenta lo stato di default;
- Richiesta in corso: la richiesta del biocomponente è stata inoltrata al compartimento ma deve ancora essere soddisfatta;
- Richiesta terminata: il compartimento ha letto la richiesta ed ha assegnato al componente le molecole richieste;

In uno stesso momento potrebbero essere effettuate richieste da parte di componenti diversi per uno stesso tipo di molecole. Per stabilire la priorità con cui assegnarle si introduce il concetto di affinità; le richieste che hanno un maggior grado di affinità per un certo tipo di molecole saranno quelle che verranno soddisfatte per prime. Tali affinità variano in base allo stato in cui si trova il componente. Questo perché a seconda della situazione in cui si trova il biocomponente si avrà una maggior o minor predisposizione verso la molecola a cui il sito si deve legare.

Come già detto in precedenza esistono numerosi tipi di biocomponenti; quelli dello stesso tipo scatenano sempre la medesima reazione, cioè le molecole richieste e quelle prodotte non variano mai, così come le caratteristiche dei siti di legame. Un'altra classificazione, indipendente dalla precedente, prevede la distinzione tra quei biocomponenti che una volta terminata la reazione cessano di esistere e quelli che passano ciclicamente dallo stato Reattivo a quello Pronto fino a quando non interviene una entità esterna ad interromperlo. Questa differenza verrà descritta in maniera più dettagliata nel paragrafo seguente.

1.4 Percorsi metabolici

Come già detto in precedenza quello che si intende modellare non riguarda tanto le singole reazioni, ma i cosiddetti percorsi metabolici. Con questo termine si indica un insieme di reazioni che devono avvenire in sequenza, in quanto il prodotto della prima reazione è reagente della seconda, il cui prodotto è reagente

della terza e così via. Il percorso è quindi costituito da una serie di sotto percorsi detti stages, ognuno dei quali è costituito da una serie di reazioni e fa da substrato per il successivo. Ogni stage inizia al termine del precedente, eccetto il primo, e termina nel momento in cui tutte le reazioni previste in quello stage sono state concluse.

1 Oltre alle reazioni che avvengono all'interno dei singoli stage, in ogni percorso metabolico possono essere definite delle reazioni che vengono eseguite ciclicamente dall'inizio del primo alla fine dell'ultimo stage. Questa classificazione consente di comprendere meglio la classificazione citata nel paragrafo precedente tra i biocomponenti che eseguono la propria reazione una volta sola e quelli che la eseguono ciclicamente fino a quando non vengono interrotti. I primi sono chiamati catalizzatori ed il loro ciclo di vita corrisponde a quello di un singolo stage, non appena viene completata la reazione le molecole prodotte vengono inserite all'interno del compartimento ed il biocomponente che ha reagito viene rimosso. Lo stage termina la sua esecuzione e passa il controllo allo stage successivo solamente quando tutti i catalizzatori al suo interno avranno effettuato le loro reazioni. Il principale compito degli agenti catalitici consiste nel far sì che processi che avverrebbero molto lentamente siano eseguiti e si concludano in tempi relativamente brevi. La maggiore velocità è resa possibile grazie alla variazione del meccanismo di reazione che quindi comporta un diverso livello di energia richiesta per far sì che i reagenti si trasformino nel prodotto. Grazie a questa caratteristica è possibile eseguire reazioni che in condizioni normali non potrebbero avvenire. Il ciclo di vita dei biocomponenti del secondo tipo ha la durata di un intero percorso metabolico. Tali componenti sono denominati controllori, dato che regolano l'esecuzione dell'intero processo garantendo il giusto equilibrio fra le molecole presenti nel compartimento. Oltre a questi due tipi di reazioni vi è un terzo tipo che

avviene direttamente all'interno del compartimento. I biocomponenti che se ne occupano hanno un comportamento simile a quello dei controllori, in quanto essi sono costantemente in esecuzione all'interno del compartimento con lo scopo di garantire il giusto equilibrio fra le molecole presenti.

2. Strumenti utilizzati

La realizzazione del compartimento è stata effettuata utilizzando i sistemi multiagente (MAS) ed in particolar modo l'astrazione Agents&Artifacts. Tali sistemi infatti si prestano in maniera ottimale alla modellazione distribuita dei sistemi biologici. Per implementare l'astrazione specificata è stato utilizzato il framework TuCSoN (Tuple Centre Spread Over the Network) che sfrutta i centri di tuple. Nel prossimo paragrafo verrà fornita una breve introduzione ai sistemi multiagente, in cui verrà analizzata l'astrazione A&A, per poi passare ad una descrizione di TuCSoN ed al linguaggio utilizzato per programmare i centri di tuple che prende il nome di ReSpecT.

2.1 I sistemi multiagente

Sono sistemi costituiti da un insieme di agenti intelligenti situati in un certo ambiente ed in grado di interagire tra loro. Un agente può essere considerato come una entità in grado di capire l'ambiente che lo circonda tramite dei sensori e di eseguire determinate azioni attraverso degli attuatori. Gli aspetti più importanti di questo tipo di sistemi sono i seguenti:

- Sono composti da un numero elevato di entità omogenee o eterogenee;
- Ogni entità che li compone è dotata di risorse limitate in termini di calcolo e comunicazione;
- Devono poter operare in ambienti dinamici ed impredicibili;

- A causa della loro complessità devono poter funzionare con un limitato supporto di un coordinamento centralizzato;
- Le entità possiedono una conoscenza limitata dell'ambiente nel quale operano;

La parte fondamentale di tali sistemi sono proprio le entità che li costituiscono, in quanto il loro comportamento influenza quello dell'intero sistema.

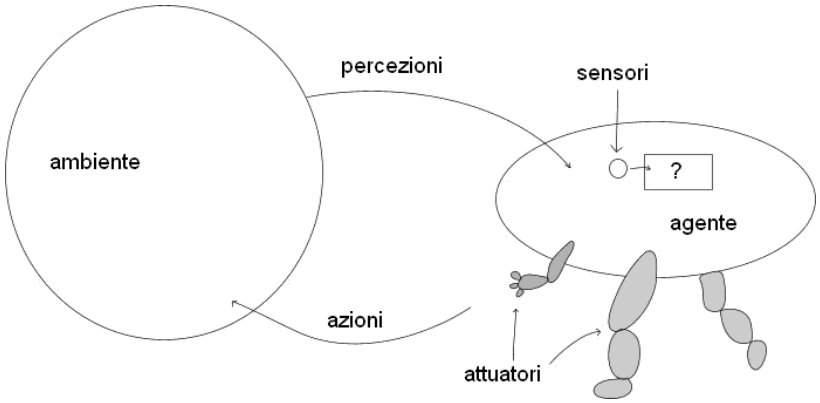

Figura 2.1 Interazione di un agente con l'ambiente esterno

Le caratteristiche principali di un agente sono le seguenti:

- Autonomo;
- Capacità di interagire con l'ambiente circostante;
- Capacità di comprendere l'ambiente circostante;
- Capacità di riprodursi;

Nella maggior parte dei casi un agente non ha completa conoscenza e controllo dell'ambiente che lo circonda. L'ambiente è in grado di evolvere in maniera dinamica ed indipendentemente dall'agente e le azioni di quest'ultimo possono anche fallire. Il problema principale per un agente è quello di stabilire quale delle sue azioni dovrà eseguire per raggiungere gli obiettivi preposti. Esso infatti deve essere dotato di un comportamento flessibile, ovvero con le seguenti caratteristiche:

- Reattivo: deve essere in grado di percepire il suo ambiente e di rispondere in maniera tempestiva ai cambiamenti che si verificano in esso;
- Pro-attivo: deve essere capace di esporre un comportamento finalizzato all'obiettivo prendendo l'iniziativa;
- Sociale: interazione con altri agenti;

Tali agenti possono avere anche altre proprietà, come la mobilità, ovvero la capacità di spostarsi tra i nodi della rete, o l'apprendimento per migliorare le proprie prestazioni. In questo tipo di sistema gli agenti possono avere due comportamenti distinti:

- Comportamento interno: rappresenta la componente computazionale interna, governata dall'agente stesso;
- Comportamento esterno: rappresenta l'aspetto osservabile dell'interazione tra le componenti computazionali degli agenti;

Analizzando questi due tipi di comportamenti si possono notare che essi operano a due livelli differenti, di conseguenza è necessario distinguere due tipi di linguaggi:

- Linguaggi intra-agenti: linguaggi che definiscono il comportamento computazionale dell'agente;
- Linguaggi inter-agenti: linguaggi che definiscono il comportamento interattivo degli agenti;

Delle due categorie citate risulta di particolare interesse la seconda, in quanto in un sistema multiagente la comunicazione tra gli agenti è fondamentale per l'interazione. Il meccanismo solitamente utilizzato consiste nello scambio di messaggi. L'invio o la ricezione di un messaggio può essere visto come un'azione eseguita dall'agente. I linguaggi di comunicazione tra agenti (ACL) sono l'esempio più semplice di linguaggi inter-agenti, essi specificano come gli agenti devono comunicare con gli altri, e potrebbero imporre restrizioni sull'architettura interna degli stessi. Gli ACL maggiormente diffusi sono:

- FIPA-ACL (Foundation for Intelligent Physical Agents-ACL): devono il nome alla fondazione che si è occupata di sviluppare uno standard per agenti eterogenei ed interattivi su sistemi basati ad agente. L'obiettivo dell'organizzazione era la definizione di un insieme di standard per implementare sistemi in cui gli agenti potessero essere messi in esecuzione e definire il modo con cui comunicare ed interagire;
- KQML (Knowledge Query and Manipulation Language): questo linguaggio di comunicazione per sistemi ad agente è stato creato da una organizzazione che inizialmente aveva come obiettivo lo sviluppo di tecniche per la costruzione di basi di conoscenza su larga scala condivisibili e riutilizzabili. Inizialmente veniva utilizzato come interfaccia per questo tipo di sistemi, ma poi è stato riproposto come linguaggio di comunicazione tra agenti;

La programmazione ad agenti presenta molte similarità con quella ad oggetti. Le due entità che lo compongono, gli agenti e gli oggetti, presentano infatti numerose similarità: entrambi incapsulano uno stato, sono in grado di eseguire delle azioni e di comunicare tramite scambio di messaggi. Esistono, però, anche delle differenze, la più marcata ed importante riguarda l'autonomia. Se un oggetto ha un metodo pubblico qualsiasi altro oggetto è in grado di invocarlo, e quando gli viene richiesto deve eseguirlo. Nel caso degli agenti questo non viene garantito perché essi sono entità autonome.

La programmazione ad agenti viene vista come un nuovo paradigma, evoluzione di altri paradigmi, ed in modo particolare di quello orientato agli oggetti. Il suo grande successo è dovuto a numerosi fattori. Sistemi realizzati in questo modo risultano avere una elevata robustezza rispetto a cambiamenti esterni, inoltre presentano un elevato grado di adattabilità e di autonomia, dovuti al fatto che sia il funzionamento locale delle entità e della rete che descrive l'interazione sono spesso generate in modo spontaneo come reazioni alle condizioni esterne e ad improvvisi cambiamenti delle stesse.

2.2 Il meta-modello A&A

Per la realizzazione del compartimento cellulare è stata utilizzato il meta-modello Agents & Artifacts. Esso è costituito da tre astrazioni di base:

- Agenti: sono i componenti pro-attivi del sistema. Essi possiedono un comportamento autonomo che viene eseguito all'interno di un qualche ambiente;
- Artifatti: sono i componenti passivi del sistema. Tali entità vengono utilizzate per regolare e supportare l'interazione tra gli agenti coinvolti;
- Sistemi multiagente: sono sistemi in cui l'ambiente è organizzato in workspaces. Essi sono contenitori di agenti e artifatti che possono essere utilizzati per strutturare l'insieme delle entità definendo una topologia dell'ambiente e fornendo una metodologia per regolare le interazioni al suo interno;

In questo tipo di sistemi gli agenti usano e condividono gli artefatti per interagire tra loro e per svolgere le proprie attività. Di seguito analizzeremo le caratteristiche delle tre astrazioni principali.

2.2.1 Agenti A&A

Gli agenti vengono definiti come entità computazionali autonome che incapsulano il flusso di controllo e non possono essere controllati né invocati. Essendo un'entità pro-attiva esegue un qualche tipo di attività, individuale o collettiva, finalizzata al raggiungimento di un obiettivo. Le attività che vengono eseguite sono considerate come una sequenza di azioni, che in base al modello A&A possono essere classificate come:

- Azioni interne: sono le computazioni che avvengono all'interno dell'entità stessa;

- Azioni di comunicazione: specificano comunicazioni dirette con uno o più agenti tramite un qualche tipo di ACL. Queste azioni possono provocare un cambiamento di stato degli agenti coinvolti;
- Azioni pragmatiche: vengono svolte all'interno del working environments e riguardano la costruzione e condivisione degli artefatti, con modifica dello stato associato;

Per potere eseguire azioni su un artefatto l'agente deve essere situato in un working environment, di conseguenza ogni agente è strettamente collegato all'ambiente in cui vive e con cui interagisce.

2.2.2 Artefatti A&A

Come già descritto in precedenza, a differenza degli agenti, gli artefatti sono entità computazionali passive e con stato, progettate per fornire funzionalità agli agenti che li utilizzano. Quest'ultimi li sfruttano come strumento, di conseguenza gli artefatti non sono autonomi ma vengono governati dagli agenti stessi ed il loro comportamento emerge solamente quando necessario. La funzionalità di un artefatto è definita dalle operazioni che lo stesso è in grado di offrire, la cui esecuzione può essere effettuata sfruttandone l'interfaccia che definisce l'insieme delle operazioni fornite. Tale interfaccia può cambiare in maniera dinamica in base allo stato dell'artefatto, è quindi possibile progettare artefatti che mostrano interfacce differenti in base allo stato di funzionamento. Oltre alle operazioni l'interfaccia può mostrare delle proprietà osservabili, ovvero proprietà i cui valori possono essere osservati dagli agenti senza dover necessariamente interagire con l'artefatto. L'esecuzione di una operazione può portare sia a cambiamenti dello stato interno dell'artefatto, che alla generazione di nuove proprietà osservabili dall'agente.

2.2.3 Sistemi multiagente A&A

Nel meta-modello A&A i MAS sono sistemi computazionali costituiti da agenti e artifatti. Essi sono sempre situati all'interno di un ambiente e progettati in maniera indipendente. Il comportamento di tali sistemi è dato dall'interazione tra le entità autonome, ovvero gli agenti, e le entità funzionali, gli artefatti. Le interazioni possibili all'interno di un MAS tra agenti e artefatti sono di quattro tipi:

- Comunicazione: agenti che comunicano tra loro;
- Operazione: agenti che sfruttano artefatti per raggiungere i propri obiettivi;
- Composizione: agenti che si collegano ad altri artefatti;
- Presentazione: Artefatti che si mostrano agli agenti;

Il MAS è in grado di interagire con l'ambiente esterno. Queste interazioni possono essere attribuite ad agenti, artefatti o allo stesso MAS a seconda del livello di astrazione richiesto.

2.3 Il framework TuCSoN

TuCSoN (Tuple Centres Spread over the Network) è un modello di coordinazione per sistemi distribuiti di tipo tuple based. Tale modello ha il compito di governare tutte le interazioni tra le entità che avvengono all'interno del sistema. I componenti fondamentali che costituiscono un modello tuple-based sono i seguenti:

- Coordinabili: le entità che agiscono all'interno del sistema e che dovranno essere coordinate;
- Medium di coordinazione: spazio in cui vivono ed interagiscono le entità. Visto che si sta lavorando con un modello tuple based in questo caso avremo uno spazio di tuple, ovvero un insieme di elementi eterogenei detti

tuple. A causa di alcune sue limitazioni che vedremo a breve tale spazio si è evoluto in centro di tuple;

- Leggi di coordinazione: stabiliscono il comportamento del medium in risposta all'interazione. Queste regole vengono definite in termini di linguaggio di comunicazione, rappresentato dalle tuple, e di linguaggio di coordinazione, costituito da primitive che vengono utilizzate per inserire o recuperare tuple dallo spazio;

Le tuple presentano una struttura a record e sono particolarmente adatte per aggregare conoscenza. Ogni tupla è indipendente dal coordinabile che l'ha generata, di conseguenza è ugualmente accessibile da tutti i coordinabili. L'accesso ad una tupla situata all'interno dello spazio è basato sul tuple matching analizzando il contenuto e la struttura della tupla stessa.

Utilizzando uno spazio di tuple la coordinazione avviene leggendo o scrivendo una tupla alla volta, il comportamento del medium è quindi fissato e solo certi problemi possono essere risolti in modo efficiente. Una soluzione a tale problematica consiste nell'inserire all'interno dello spazio capacità computazionale, in questo modo è possibile variarne il comportamento in base al problema di coordinazione da affrontare. Lo spazio di tuple viene quindi espanso inserendo un comportamento e trasformandosi in centro di tuple. Con tale termine si indica semplicemente uno spazio di tuple con l'aggiunta di una specifica di comportamento per gestire le interazioni tra le entità. Il linguaggio utilizzato per la specifica è espresso come reaction specification language, ovvero un insieme di attività dette reazioni.

2.2.1 Il centro di tuple TuCSoN

Come già descritto in precedenza TuCSoN è un modello tuple based per la coordinazione di sistemi distribuiti, di conseguenza risulta particolarmente adatto ad essere utilizzato come framework per la coordinazione nei sistemi multiagente. TuCSoN utilizza come mezzo di coordinazione il centro di tuple,

ovvero uno spazio di tuple con in più l'aggiunta del comportamento necessario per gestire le interazioni tra le entità che lo costituiscono. Il tuple center presenta la stessa interfaccia dello spazio, gli agenti sono quindi in grado di utilizzarlo allo stesso modo in cui utilizzavano lo spazio, ovvero tramite un insieme di primitive di coordinazione. Le operazioni TuCSoN presentano due fasi:

- Invocation: rappresenta la richiesta inviata dall'agente al centro di tuple contenente tutte le informazioni necessarie;
- Completion: rappresenta la risposta inviata dal centro di tuple all'agente contenente tutte le informazioni riguardo l'esecuzione dell'operazione da parte del centro;

Mentre le primitive che è in grado di eseguire un agente per interagire con il centro sono le seguenti:

- Out(Tupla): operazione con il quale un agente scrive una tupla all'interno del centro;
- In(Template): si ricerca una tupla che faccia matching con il Template all'interno del centro. Se questa viene trovata l'esecuzione ha successo e viene rimossa e restituita la tupla, altrimenti l'esecuzione è sospesa fino a che non viene individuata una tupla corrispondente nel centro di tuple;
- Rd(Template): si ricerca una tupla che faccia matching con il Template all'interno del centro. Se questa viene trovata l'esecuzione ha successo e viene restituita la tupla, altrimenti l'esecuzione è sospesa fino a che non viene individuata una tupla corrispondente nel centro di tuple;
- Inp(Template): si ricerca una tupla che faccia matching con il Template all'interno del centro. Se questa viene trovata l'esecuzione ha successo e viene rimossa e restituita la tupla, altrimenti l'esecuzione fallisce ed il Template viene restituito;
- Rdp(Template): si ricerca una tupla che faccia matching con il Template all'interno del centro. Se questa viene trovata l'esecuzione ha successo e viene restituita la tupla, altrimenti l'esecuzione fallisce ed il Template viene restituito;

- No(Template): si ricerca una tupla che faccia matching con il Template all'interno del centro. Se questa non viene trovata l'esecuzione ha successo e viene restituito il Template, altrimenti l'esecuzione è sospesa fino a che non viene individuata una tupla corrispondente nel centro di tuple;
- Nop(Template): si ricerca una tupla che faccia matching con il Template all'interno del centro. Se questa non viene trovata l'esecuzione ha successo e viene restituito il Template, altrimenti l'esecuzione fallisce e la tupla viene restituita;
- Get: restituisce il contenuto del centro di tuple;
- Set(Tuple): sovrascrive il contenuto del centro con le tuple specificate come argomento;

Grazie alle caratteristiche di TuCSoN è possibile utilizzare i centri di tuple attraverso la rete. Lo spazio di coordinazione è costituito da un insieme di nodi in grado di interagire attraverso la rete. Ogni nodo può contenere più centri di tuple, identificati univocamente da un nome logico. Gli agenti possono utilizzare un nome globale, costituito dal nome logico più l'indirizzo del nodo che ospita il centro di tuple, o locale, in modo da poter usare rispettivamente lo spazio di coordinazione globale o locale.

Come già anticipato in precedenza il centro di tuple TuCSoN presenta al suo interno un comportamento che gli permette di adattarsi ai bisogni del sistema, definendo un insieme di reazioni che determinano come il centro deve reagire a determinati eventi. Il centro di tuple può quindi essere considerato programmabile. Il linguaggio utilizzato per la specifica delle reazioni prende il nome di ReSpecT.

2.2.2 Il linguaggio ReSpecT

Tale linguaggio, utilizzato all'interno dell'infrastruttura TuCSoN per specificare il comportamento di un centro di tuple, è di tipo logic based. Questo significa che viene utilizzata la logica del primo ordine per rappresentare ed

elaborare l'informazione. Sia le tuple che i template sono quindi tuple logiche, ovvero fatti prolog, e come tecnica di matching si sfrutta l'unificazione. Essendo un linguaggio esso consente la definizione di reazioni all'interno del centro di tuple, così come l'associazione di reazioni ad eventi che avvengono al suo interno. Le reazioni sono definite come tuple logiche del tipo reaction(E,G,R), dove E rappresenta l'evento che si è verificato, G è la condizione di guardia, mentre R è la reazione che viene eseguita in risposta all'evento E se la guardia G è verificata. Una reazione viene definita come sequenze di goal di reazione logici ed ha successo solamente se lo hanno tutti gli obiettivi. In caso di fallimento non si ha alcuna ripercussione sullo stato del centro di tuple.

Come nel caso di TuCSoN tutte le operazioni di meta-coordinazione hanno due fasi, ovvero quelle di invocation e di completion. Inoltre le primitive di meta-coordinazione utilizzabili dagli agenti coincidono con quelle di coordinazione, tranne per il fatto che in questo caso si lavora in un centro contenente tuple codificate in linguaggio ReSpecT.

3. Analisi del problema e progettazione del sistema

3.1 Analisi del problema

L'obiettivo finale del nostro progetto è la realizzazione di un framework che consenta l'esecuzione di simulazioni distribuite del comportamento di un compartimento cellulare.

Il sistema che si intende realizzare potrebbe essere implementato in maniera che i principali sottosistemi, cioè i biocomponenti ed il compartimento a cui fanno riferimento, risiedano all'interno di un unico terminale; per garantire una maggiore flessibilità ed un maggior livello di scalabilità però, si è scelto di realizzare il tutto in modo che le singole parti possano trovarsi all'interno di dispositivi distinti ed eventualmente eterogenei (ad esempio un dispositivo Android, un PC, ecc). L'unica condizione necessaria è la presenza, all'interno della macchina, di una JVM. A tale scopo Tucson si è rivelato estremamente utile, in quanto permette nativamente la separazione fra i singoli agenti (nel nostro caso i biocomponenti) ed i centri di tuple con cui interagiscono (nel nostro sistema ogni agente fa capo ad un unico tuple centre, che rappresenta il compartimento in cui il biocomponente vive).

La soluzione migliore dal punto di vista concettuale, sarebbe stata quella di concepire ogni singolo biocomponente attivo come un agente, dato che, come dice il nome stesso, rappresentano le entità attive all'interno del sistema, senza le

quali non sarebbe attuabile alcuna reazione. Tale soluzione incontra però un problema in fase di esecuzione: dato che, in TuCSoN, ad ogni agente è assegnato un thread per la propria esecuzione ed un secondo thread nel momento in cui interagisce con il centro di tuple, il numero complessivo di thread in esecuzione nel sistema sarebbe dell'ordine di O(2 * n° biocomponenti attivi). Vanno inoltre aggiunti eventuali altri thread utilizzati per differenti scopi. Per la realizzazione di sistemi piccoli, con un numero ridotto di entità attive nello stesso istante, tale soluzione potrebbe non presentare eccessivi problemi, ma, dato che buona parte delle reazioni intracellulari coinvolgono una quantità elevata di enzimi o proteine, il problema dell'esplosione dei thread potrebbe degradare le prestazioni dell'intero sistema, riducendo notevolmente la scalabilità. Pertanto si è resa necessaria l'individuazione di una strada alternativa, che permetta sia di simulare l'esecuzione concorrente delle singole reazioni, che di garantire un livello di scalabilità accettabile. A tale scopo si è deciso di introdurre due nuovi concetti: il controllore, che si occupa di gestire tutti quei biocomponenti il cui ciclo di vita corrisponde a quello dell'intero percorso metabolico ed il catalizzatore, che invece agisce all'interno di un singolo stage. Pertanto il primo gestirà delle entità che eseguiranno ciclicamente la propria reazione, mentre quel le del secondo cesseranno di esistere una volta completato il proprio scopo. Per mantenere comunque elevato il livello di concorrenza, si è scelto di vincolare entrambi i concetti appena espressi ad un singolo tipo di biocomponenti: pertanto se uno stage è costituito da 30 enzimi di tipo PFK e 20 proteine di tipo PP1, si avranno due istanze di tipo catalizzatore, che agiranno in parallelo.

Si intende modellare il sistema in modo da garantire la maggior flessibilità possibile, prevedendo anticipatamente eventuali modifiche strutturali che, se ritenuto necessario, potranno essere attuate con semplicità. Una di queste è la possibilità di modificare il comportamento appena descritto per permettere ad un agente di gestire in contemporanea biocomponenti di differente tipo. Tale assunzione permette di giustificare alcune delle decisioni che verranno affrontate nel seguito del paper.

Per poter rappresentare graficamente la struttura logica del sistema, si è scelto di realizzare il diagramma delle classi UML mostrato in figura 3.1. In tale schema possono essere individuati tutti i concetti descritti finora, più alcuni che sono stati aggiunti ai fini dell'implementazione. Uno di questi è rappresentato dalla classe AgenteDiCompartimento. Dato che secondo le specifiche, un compartimento può eseguire reazioni e produrre molecole spontaneamente, si è scelto di modellare tale comportamento, anzichè all'interno della classe Compartimento, che ha il solo scopo di fungere da interfaccia per la comunicazione con il Tuple Centre, all'interno di un'apposita classe. Pertanto ad ogni compartimento darà associato uno specifico agente di compartimento che, simulando il comportamento di un biocomponente, effettuerà le reazioni "a nome" del compartimento stesso.

Una scelta importante riguarda la relazione fra il biocomponente ed il proprio tipo. Dato che i biocomponenti dello stesso tipo eseguono le stesse reazioni e sono identici anche per quanto riguarda il numero di siti ed il tipo di molecola con cui ognuno di essi può legarsi, in una prima stesura si era ipotizzato di concepire proteine dello stesso tipo come istanze della medesima sottoclasse di Proteina (discorso analogo per gli enzimi). Una riflessione più attenta ci ha permesso, però, di capire che dal punto di vista sia del comportamento che dell'interfaccia esposta, non vi sarebbe stata alcuna differenza fra le varie sottoclassi. Inoltre va considerato il fatto che nuovi tipi di biocomponenti potrebbero essere definiti anche a runtime, e per permettere ciò dal punto di vista implementativo sarebbe stato richiesto uno sforzo tale da non giustificare il risultato che si sarebbe ottenuto. Alla luce di tali conclusioni si è optato per una soluzione differente: realizzare il tipo come una classe a parte a cui fanno riferimento le istanze di biocomponente. Ogni istanza di tipo specifica tutti quei parametri che permettono di distinguere una categoria dall'altra e cioè, nello specifico:

- Nome del tipo
- Siti di legame con le rispettive tipologie di molecole

- Indicazione delle tipologie dei siti di legame (quali siti sono attivatori, quali reattori e quali invece inibitori)
- Affinità di ogni sito in corrispondenza di ognuno degli stati in cui il biocomponente può trovarsi.
- Velocità di esecuzione della reazione
- Tipo e quantità di molecole prodotte al termine della reazione

In questo modo, oltre ad avere un'architettura logica meno complesso e meglio organizzata, è estremamente più semplice caricare a tempo di esecuzione nuove tipologie di biocomponenti in quanto, anzichè definire a runtime una nuova classe, è sufficiente istanziare un semplice oggetto. Un'altra considerazione che è stata effettuata, sempre riguardante la tematica appena citata, riguarda l'archiviazione delle varie tipologie. Oltre che dare la possibilità agli utenti di definire in maniera autonoma le varie tipologie di biocomponenti, è indispensabile garantire la loro persistenza; pertanto è stato necessario decidere il metodo di archiviazione più adatto alle nostre esigenze. Dato che il framework che si sta realizzando potrà essere utilizzato da gruppi di lavoro di dimensione variabile, è necessario che tutti gli utenti abbiano una visione unificata e coerente della conoscenza, e che, a fronte delle varie modifiche apportate dagli utenti autorizzati a farlo , rimanga in uno stato coerente. A tal proposito si è scelto di centralizzare tutte le informazioni di dominio comune in un'unica base dati relazionale. Tale scelta è giustificata dal fatto che i RDBMS disponibili forniscono già funzionalità di supporto alle nostre esigenze, senza che ci sia quindi il bisogno di implementarle nuovamente (il DB in questione è argomento del successivo paragrafo). A partire da tale riflessione siamo riusciti ad individuare ulteriori informazioni che abbiamo ritenuti utili centralizzare. In particolare, la memorizzazione dei percorsi è stata ritenuta critica allo scopo di garantire al sistema consistenza e riusabilità e quindi indirettamente una maggiore scalabilità. Ciò è dovuto al fatto che, un percorso metabolico definito all'interno del gruppo di lavoro, dovrà essere il medesimo per tutti i componenti

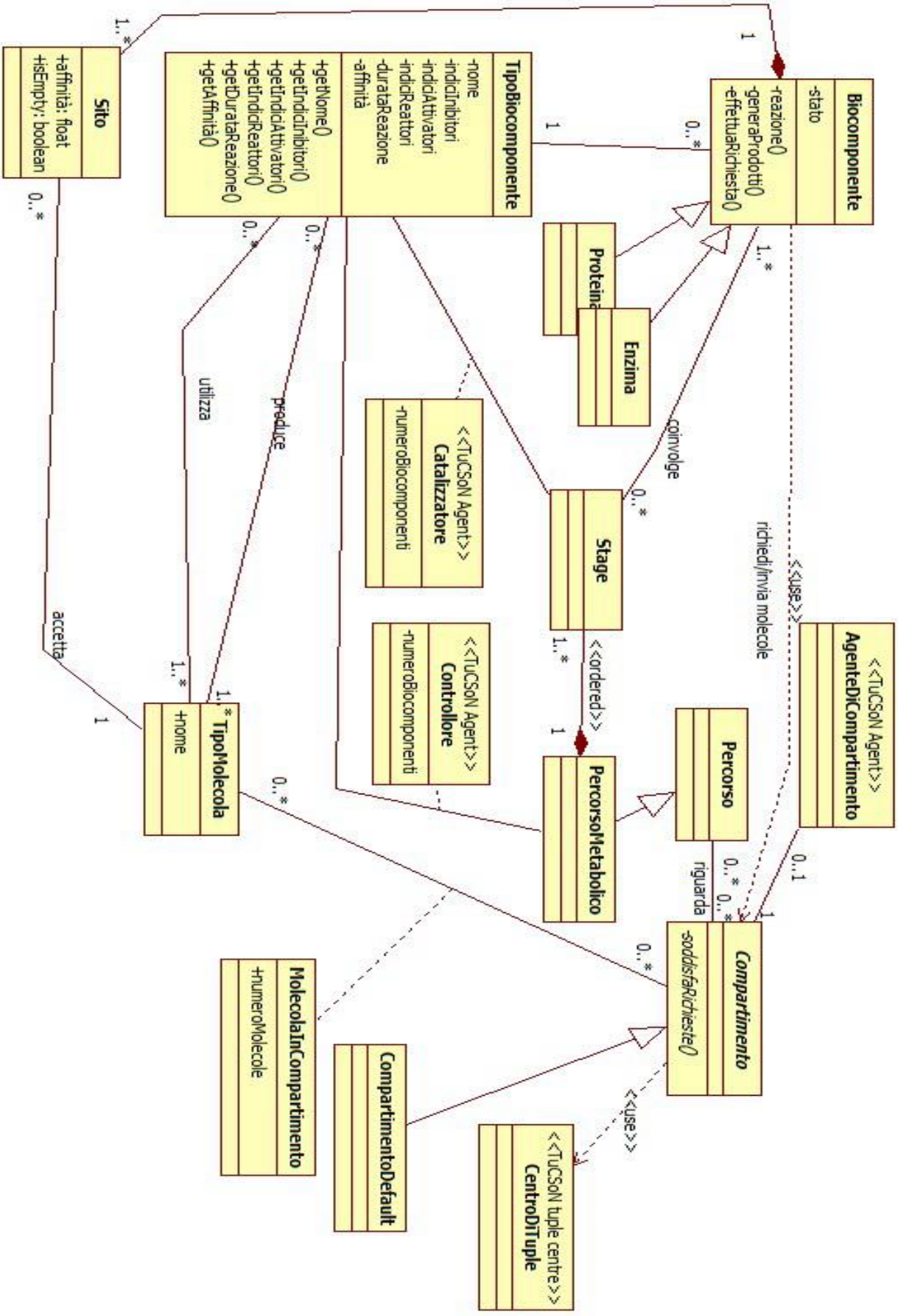

Figura 3.1 – Il diagramma delle classi del sistema

del gruppo. Impedire di centralizzare tali informazioni, implicherebbe implementarlo localmente sulla propria macchina da parte di ognuno. Questa soluzione condurrebbe, con buona probabilità, ad errori o inconsistenze fra le varie implementazioni della medesima catena di reazioni, producendo risultati incompatibili ed instabili. Alla luce di tali considerazioni ci si è resi conto che memorizzare i percorsi in maniera centralizzata non era solamente consigliabile, ma piuttosto necessario, in quanto altrimenti si sarebbe compromesso il funzionamento dell'intera architettura.

L'unica eccezione a tale caso, è costituito da quei contesti in cui, anzichè un gruppo di lavoro, il sistema debba essere utilizzato esclusivamente da un singolo utente. Per comprendere tali casi è stata predisposta la possibilità di utilizzare una singola macchina escludendo il database ed archiviando tutte le informazioni in un unico file di testo, dato che in tale contesto non sarebbero più necessarie le varie facilitazioni offerte dai DBMS. Per evitare però che queste situazioni richiedano modifiche eccessivamente corposo all'intero sistema, si è scelto di standardizzare la struttura dell'input del sistema. Si è scelto pertanto di utilizzare lo stesso formato sia per l'invio di dati ai client da parte del server, in ambiente multiutente, sia per memorizzazione di dati su file in ambiente monoutente. Tale struttura è descritta dettagliatamente nel seguente capitolo.

3.2 Struttura dei dati in input

Come accennato nel capitolo precedente, il sistema potrebbe avere una struttura differente in base al numero di utenti. Per un sistema multiutente è necessario realizzare un'architettura client-server, altrimenti è sufficiente una singola macchina. Nel primo caso, l'input di ogni client è costitutito, oltre che dall'interazione con l'utente, dalle informazioni che provengono dal server le quali descrivono i tipi di biocomponente ed i percorsi metabolici condivisi; nel secondo caso, invece, tali informazioni saranno memorizzate in un file di testo.

In entrambi i casi, comunque, è necessario unificare la struttura di tali informazioni; a tale scopo si è scelto il seguente formato:

```
tipo
{
    nome:       pfk;
    durata:     50;
    sito:       f6p,    R,      0.5,    1.0,    0;
    sito:       atp,    R,      0.5,    1.0,    0;
    sito:       atp,    I,      0.1,    0.1,    0;
    sito:       adp,    A,      1.0,    0,      1.0;
    sito:       adp,    A,      1.0,    0,      1.0;
    prodotto:   adp,    1;
    prodotto:   fbp,    1;
}

percorso
{
    controllore:                pfk,    1;
    stage
    {
        catalizzatore:          pfk,    1;
    }
    stage
    {
        catalizzatore:          pfk,    1;
    }
}
```

In primo luogo vengono descritti i tipi di biocomponenti, che richiedono la memorizzazione dei seguenti dati:

- Nome della tipologia.
- Siti di legame, per ognuno dei quali vanno memorizzate le seguenti informazioni: tipo di molecola, tipo del sito di legame (attivatore, inibitore o reattore), affinità per ognuno degli stati in cui può trovarsi il biocomponente (pronto, attivo o inibito; nello stato reattivo non è necessario in quanto i siti di legame sono inattivi).
- Tipi e quantità di molecole rilasciate al termine della reazione.
- Durata della reazione (in ms).

Dopo i tipi è necessario memorizzare i percorsi metabolici, che richiedono informazioni riguardo ai controllori attivi durante l'intero processo ed ai singoli stage che lo compongono. Ogni stage richiede i dati dei catalizzatori. Importante notare che i tipi dei catalizzatori e dei controllori devono essere già noti al sistema, che altrimenti non potrebbe funzionare.

Per la lettura dei dati è stato scelto di realizzare ad hoc un lexer ed un parser, utilizzando il tool ANTLR. In tal modo è possibile interpretare e tradurre in entità del sistema, qualsiasi informazione venga ricevuta, purchè sia ben formata e rispetti le regole sintattiche definite.

3.3 Il DB della conoscenza

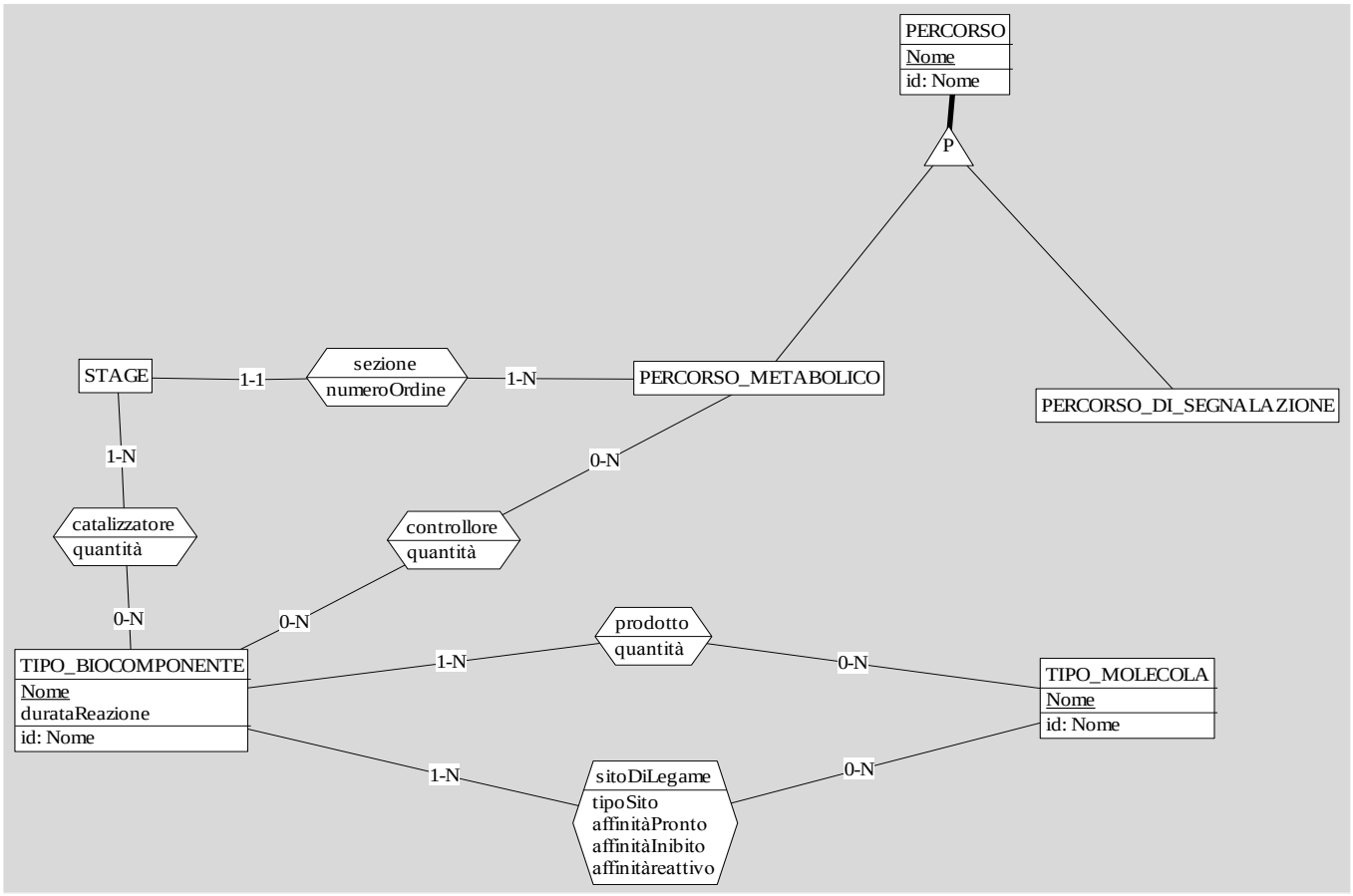

Figura 3.2 – Lo schema E/R del database dela conoscenza

Come accennato nel capitolo 3.1, nel caso di sistema multiutente, si è scelto di memorizzare tutte le informazioni che necessitano di essere condivise fra più utenti in un'unica base dati relazionale. Pertanto il database della conoscenza verrà memorizzato nella stessa macchina in cui si troverà il centro di tuple, che rappresenterà quindi il server centrale dell'architettura del sistema.

In figura 3.2 è mostrato lo schema concettuale che rappresenta la struttura del DB. Come già detto, al momento i percorsi di segnalazione sono mantenuti solamente per coerenza con la struttura del sistema biologico reale, ma non sono implementati in alcun modo. È prevista una loro eventuale implementazione futura.

Lo schema E/R rappresentato è semplicemente un sottoinsieme delle informazioni mostrate in figura 3.1, sebbene siano schematizzate con un differente formalismo.

3.4 Fault tolerance

Per garantire il corretto funzionamento del sistema ed il ripristino dello stato corrente a fronte di eventuali guasti ed interruzioni, è stato necessario mettere in piedi opportuni meccanismi. Per quanto riguarda il database, anche in questo caso i meccanismi di prevenzione e ripristino sono già forniti dal relativo DBMS. Per quanto riguarda il centro di tuple, invece, è stato necessario realizzarli appositamente. A tale scopo si è scelto di utilizzare dei file di log in cui memorizzare, in ordine di esecuzione, tutte le operazioni effettuate sul compartimento. In tal modo, in caso di interruzione del servizio, sarà sufficiente comnsultare tale file per riportare l'intero sistema allo stato antecedente al guasto. Per garantire il funzionamento anche in caso di interruzione del funzionamento del server, si è scelto di predisporre periodiche copie dei file di log (sia del database, che del centro di tuple) in una macchina collocata geograficamente distante da quella ospitante, in modo da garantire anche la disaster recovery. In tal caso tutte le modifiche avvenute fra un salvataggio e l'altro andranno perse; l'unica alternativa sarebbe stata quella di salvare in remoto ogni modifica effettuata, ma in tal caso l'overhead sarebbe stato tale da compromettere l'effettivo funzionamento del sistema. Pertanto si è scelto di tollerare un minimo livello di rischio a fronte di un miglior livello di prestazioni.

3.5 Utenti e permessi

Come in buona parte dei team di lavoro, è altamente probabile che anche i gruppi che utilizzeranno il framework in questione abbiano una struttura di tipo gerarchico, in cui alcuni utenti avranno permessi più elevati di altri. A tale scopo

si è scelto di realizzare una suddivisione statica degli utenti in vari profili, in base ai permessi che gli saranni garantiti. Tali profili sono descritti nei seguenti paragrafi, in maniera top-down (dal profilo con i permessi più elevati fino a quelli con le restrizioni maggiori).

3.5.1 Amministratore globale

È l'utente amministratore dell'intero sistema, pertanto ha tutti i permessi disponibili. Inizialmente viene definito un unico utente amministratore che potrà a sua volta crearne di nuovi. I suoi permessi comprendono le seguenti azioni:

- Creare o eliminare percorsi o tipi di biocomponente.
- Modificare qualsiasi percorso o tipo di biocomponente presente all'interno del sistema.
- Creare o eliminare utenti.
- Modificare i profili di tutti gli utenti.
- Avviare o arrestare il sistema.
- Avviare o arrestare un percorso.
- Utilizzare la GUI per inserire o rimuovere molecole.

3.5.2 Amministratore di gruppo

I vari utenti possono essere suddivisi in più gruppi di lavoro. All'interno di ogni gruppo deve esistere almeno un amministratore locale (che può essere eletto esclusivamente da un amministratore globale), che ha i medesimi permessi dell'amministratore globale (tranne quelli relativi all'editing degli utenti ed alla possibilità di avviare o arrestare il sistema), ma relativi al proprio gruppo di appartenenza anzichè all'intero sistema.

3.5.3 Utente semplice

Gli utenti semplici possono solamente avviare o interrompere percorsi ed utilizzare la GUI per introdurre o rimuovere molecole al loro interno. I processi devono essere relativi allo stesso gruppo di appartenenza dell'utente

4. Implementazione del centro di tuple

4.1 Tipologie di tuple

Tabella 5.1

Nome tupla	**richiesta**	
Descrizione	Rappresenta la richiesta di molecole, necessarie per scatenare la reazione, che viene effettuata da parte di un biocomponente al compartimento.	
Attributo	**Tipo**	**Descrizione**
Id	Int	Id del biocomponente che effettua la richiesta (non c'è rischio di ambiguità, dato che l'id è univoco ed ogni biocomponente può effettuare una sola richiesta alla volta).
Soddisfatta	Boolean	indica se la richiesta è ancora stata soddisfatta o meno.

Tabella 5.2

Nome tupla	**sito**

Descrizione	Descrive i parametri di un sito di legame.	
Attributo	**Tipo**	**Descrizione**
Tipo	String	Nome del tipo di molecola con cui il sito può legarsi
Id	Int	Id del biocomponente che ha effettuato la richiesta. Permette di effettuare l'associazione fra il sito di legame e la richiesta che permette di soddisfare.
Indice	Int	Indice del sito di legame. Permette di associare la tupla attuale alla corrispondente tupla di tipo 'sito'.
Affinita	Double	Percentuale di affinità. Direttamente proporzionale alla probbilità che il sito ottenga la molecola richiesta. 0<Aff<=1
Riempito	Bool	True se il sito si è legato con una molecola; false altrimenti.

Tabella 5.3

Nome tupla	**mols**	
Descrizione	Indica quante molecole di un certo tipo sono presenti all'interno del compartimento cellulare.	
Attributo	**Tipo**	**Descrizione**
Tipo	String	Tipo di molecole.
Numero	Int	Numero di molecole di tipo @Tipo sono presenti nel centro di tuple.

Tabella 5.4

Nome tupla	**molecola**	
Descrizione	Rappresenta una singola molecola del tipo specificato. L'unica utilità di questo tipo di tupla è l'innesco della reaction R3.	
Attributo	**Tipo**	**Descrizione**

Tipo	String	Tipo della molecola.

Tabella 5.5

Nome tupla	**molecole**	
Descrizione	Contrariamente alla tupla *mols*, che rappresenta le molecole già presentiall'interno del tuple centre, questa tupla rappresenta le molecole mentre vengono rilasciate. L'utilità è data dalla possibilità di innnescare reazioni ad hoc che permettano di gestire immediatamente le molecole appena prodotte andando a soddisfare le richieste che ne hanno bisogno	
Attributo	**Tipo**	**Descrizione**
Tipo	String	Tipo di molecole.
Numero	Int	Numero di molecole di tipo @Tipo sono presenti nel centro di tuple.

4.2 Operazioni

4.2.1 Inserimento di una richiesta

Si riporta la sequenza di azioni che viene eseguita nel momento in cui viene effettuata una nuova richiesta al compartimento:

1. Viene immessa la tupla corrispondente alla richiesta:
   out(richiesta(@Id, @Soddisfatta=false)).
   Nel momento dell'inserimento, ovviamente risulta insoddisfatta, pertanto il parametro @Soddisfatta sarà impostato a false.
2. Vengono immessi tutte le tuple corrisponde ai siti di legame che possono essere riempiti:
   out(sito(@Tipo, @Id, @Indice, @Affinita,@Riempito=false))

Nel momento dell'inserimento della tupla il parametro @Riempito vale false, in quanto per i siti già pieni non è necessario effettuare alcun tipo di richiesta.

Ogni volta che viene immesso un nuovo sito, è possibile che la molecola da esso richiesta sia immediatamente disponibile, pertanto l'inserimento di una nuova tuple di questo tipo innesca l'esecuzione della reaction R1, descritta al paragrafo 4.3.1.

4.2.2 Estrazione di una richiesta

Ogni volta che un biocomponente in attesa verifica se la propria richiesta è stata soddisfatta, effettua la seguente interrogazione:

```
in_p(
    richiesta(@Id, true)
    ),
```

dove @Id è il numero identificativo del biocomponente. Se la richiesta non è stata soddisfatta il booleano varrà false, pertanto l'interrogazione non produrrà alcun risultato. In caso contrario sarà quindi necessario verificare quali siti sono stati riempiti, rec

uperando tutte le tuple di tipo sito associate alla richiesta. Tale operazione è effettuata attraverso la seguente richiesta:

```
in(
    sito(Tipo,@Id, Indice, Affinita, Riempito)
    ).
```

Anche in questo caso @Id corrisponde all'identificativo del biocomponente.

4.2.3 Inserimento di molecole

Ogni volta che vengono rilasciate nuove molecole all'interno del centro di

tuple, sia come prodotto di una reazione da parte di un biocomponente, sia su richiesta dell'utente da GUI, è necessario verificare se esistono, all'interno del centro di tuple, richieste pendenti in attesa delle molecole appena rilasciate. Pertanto è necessario verificare l'esistenza di tuple di tipo sito il cui tipo coincide con quello delle molecole; per ognuno che si riesce a riempire, si imposta a true il booleano corrispondente sia nel sito stesso, che nella rispettiva richiesta (vedi tabelle 5.1 e 5.2). Se al termine di tale operazione ne rimangono alcune disponibili, viene incrementato il numero totale di molecole del tipo corrente all'interno del centro di tuple.

Per implementare effettivamente in Tucson tale soluzione, si è scelto, anzichè realizzare un'unica reaction complessa, di suddividere modularmente l'esecuzione in una catena di reazioni più semplici, che sono la R2 e la R3, rappresentate rispettivamente ai paragrafi 5.3.2 e 5.3.3.

Come già indicato si è scelto di effettuare una distinzione fra la tupla che rappresenta le molecole presenti nel centro, cioè *mols*, e quella che rappresenta le molecole che vengono rilasciate, chiamata appunto *molecole*. L'evento che scatena la prima reaction è quindi l'esecuzione di un comando di tipo out id una tupla di quest'ultimo tipo. Tale tupla indicherà, oltre al tipo di molecole rilasciato, anche il corrispondente numero N; la reaction R2, la scomporrà quindi in N tuple di tipo *molecola*, la cui sintassi è stata definita precedentemente. A questo punto subentrerà la reaction R3, innescata dalla produzione di ognuna delle singole molecole; per ognuna di esse andrà a verificare la presenza di una richiesta corrispondente che possa essere soddisfatta. In caso di esito positivo riempirà il sito di legame prima di passare alla molecola successiva; altrimenti si limiterà semplicemente ad incrementare il numero di molecole del tipo indicato all'interno del compartimento.

4.3 Reactions

4.3.1 Reaction R1

```
reaction(
    out(
        sito(Tipo, Id, Index, AffinityValue, false)
        ),
        (completion),
         (                                          //Si verifica se sono presenti molecole
        in(mols(Tipo, Numero)), Numero>0,           del tipo cercato

        in(richiesta(Id,X)),  out(richiesta(Id,true)),  //Si segnala che la richiesta è stata
                                                        soddisfatta

        in(sito(Tipo, Id, Index, Affinita, false)),     //Si riempie il sito di legame
        out(sito(Tipo, Id, Index, Affinita, true) ),
        N2 is Numero-1
        out(mols(Tipo,N2))                              //Si rimuove la molecola dal
        )                                               compartimento corrispondente
)
```

4.3.2 Reaction R2

La presente reaction scatta nel momento in cui vengono immesse molecole all'interno del compartimento. Il uso unico compito è quello di suddividere modularmente la tupla in tante tuple di tipo *molecola* quante sono le molecole immesse. In tal modo è possibile mantenere le reaction manutenibili in modo

migliore e permette inoltre di rendere il codice più leggibile e maggiormente comprensibile per gli altri utenti.

È necessario suddividere la reaction in due parti per garantire che, dopo aver decrementato il numero di molecole e si è raggiunto il valore 0, vengano mantenute tuple non necessarie all'interno del compartimento.

```
reaction(
    out(
        molecole(Tipo, Numero)
        ),
        (completion),
        (
        in(molecole(Tipo, Numero)),          //Si verifica che vi siano molecole
        Numero>0,                            rimanenti
                                             //Si innesca la reaction R3
        out(molecola(Tipo),
        N is Numero-1                        //Si innesca 'ricorsivamente' la reaction
        out(molecole(Tipo, N)),              R2 con un numero di molecole inferiore
        )                                    di un'unità
)

reaction(
    out(
        molecole(Tipo, Numero)
        ),
        (completion),
        (
        in(molecole(Tipo, Numero)),
        Numero=0,
```

```
    )
)
```

4.3.3 Reaction R3

Nonostante nei capitoli precedenti tale reazione sia sempre stata, per comodità, descritta come unica, in realtà questa reazione è costituita da tre reaction separate che, essendo state concepite come mutualmente esclusive, conducono in ogni casi ad un unico risultato. Si riporta il codice delle tre reaction, ognuna accompagnata da una descrizione:

- La prima reaction viene eseguita quando, per la molecola che si sta rilasciando, è presente, all'interno del centro di tuple, almeno un sito con cui legarsi.
- La seconda invece, viene lanciata quando nessun sito richiede la molecola che è stata rilasciata, pertanto ci si limita ad incrementare il numero di molecole del tipo che si sta rilasciando.
- La terza subentra nel caso in cui non sia presente alcuna tupla di tipo *mols* per il tipo che si sta rilasciando, quindi non sarebbe possibile effettuare alcun incremento. In questo caso viene introdotta tale tupla con valore di inizializzazione pari ad 1.

```
reaction(
    out(
        molecola(Tipo)
        ),
        (completion),
         (

        in(sito(Tipo, Id, Index, Affinita, false)),          //Condizione: esiste un sito
```

```
        in (molecola(Tipo),                          in attesa di una molecola
                                                     dello stesso tipo di quella
                                                     che si sta rilasciando.
        out(sito(Tipo, Id, Index, Affinita, true) ),
        in(richiesta(Id, X)                          //Si riempie il sito di legame
        out(richiesta(Id, true)                      //Si contrassegna la richiesta
        )                                            corrispondente come
)                                                    soddisfatta


reaction(
    out(
        molecola(Tipo)
        ),
        (completion),
         (

        no(sito(Tipo, Id, Index, Affinita, false)),  //Condizione: nessun sito
        in (molecola(Tipo),                          richiede una molecola di
                                                     tipo Tipo
        in (mols(Tipo,Numero)),
        N2 is Numero+1                               //Condizione: una tupla
                                                     mols, per il tipo che si sta
                                                     rilasciando, è già presente
        out (mols(Tipo,N2)                           nel centro.

        )                                            //Si incrementa il numero di
)                                                    molecole
```

```
reaction(
    out(
        molecola(Tipo)
        ),
        (completion),
         (

        no(sito(Tipo, Id, Index, Affinita, false)),     //Condizione: nessun sito
        in (molecola(Tipo),                             richiede una molecola di
                                                        tipo Tipo

        no (mols(Tipo,Numero),                          //Condizione: nessuna tupla
                                                        di tipo mols è presente per il
                                                        tipo Tipo

        out (mols(Tipo,1)                               //Si aggiunge la tupla
                                                        assegnandole  il valore 1.
        )
)
```

5. Stato dei lavori e sviluppi futuri

Al momento il sistema è stato realizzato solamente in parte. Per ottenere una prima versione funzionante da mostrare al committente nei tempi previsti, si è scelto di concentrarsi, in questa prima fase, esclusivamente nella realizzazione della versione monoutente, pertanto non sono state curate nè l'effettiva implementazione del database della conoscenza, nè la parte di comunicazione fra client e server. Allo stato attuale, il sistema si limita a leggere un file di testo (al momento definito in maniera statica come il file di nome "BS.txt" che deve essere collocato nella cartella del progetto), interpretarlo ed avviare i percorsi definiti al suo interno. Pertanto i prossimi passi necessari saranno la progettazione logica e fisica e l'effettiva realizzazione del DB della conoscenza, e l'implementazione dell'architettura client-server con il reciproco protocollo di comunicazione. Dopodichè sarà necessario effettuare una fase di test costituita da una serie di controlli atti a verificare il funzionamento del sistema multiutente e della coerenza con il corrispondente a singolo utente. In tal modo sarà possibile assicurarsi che non avvengano perdite di informazioni nella comunicazione di rete.

Parallelamente a questo, sarà necessario apportare quelle correzioni che precedentemente sono state trascurate; primo fra tutti il corretto utilizzo dei vincoli di affinità. Al momento la scelta del sito di legame che viene riempito, in fase di rilascio di una nuova molecola, è effettuata in maniera casuale. Per il

corretto funzionamento del sistema, invece, è necessario servire per prime le richieste con il maggior valore di affinità, o comunque fare in modo che l'algoritmo di assegnamento aleatorio ne tenga conto redistribuendo le probabilità in maniera proporzionale al valore di affinità.