Performance Evaluation Process Algebra

Polverelli Enrico

Alma Mater Studiorum Università di Bologna

Seconda Facoltà di Ingegneria

Linguaggi e Modelli Computazionali LM

a.a. 2011-2012

Indice

1 Performance Evaluation Process Algebra **1**

1.1 Introduzione 1

1.2 Sintassi e semantica 2

2 PEPA Eclipse Plugin **10**

2.1 Caso di studio. 10

3 Fluid Flow Approximation **18**

3.1 Motivazioni 18

3.2 Generazione del sistema di ODE 20

3.3 Caso di studio 25

4 Conclusioni e Bibliografia **28**

4.1 Conclusioni 28

4.2 Bibliografia 29

Capitolo 1

Performance Evaluation Process Algebra

1.1 Introduzione

Performance Evaluation Process Algebra (PEPA) è una tecnica algebrica di descrizione basata sulla classica algebra di processo, migliorata con l'aggiunta di informazioni temporali stocastiche. Questa estensione permette di descrivere modelli che posso essere usati per calcolare misure di performance e dedurre proprietà funzionali di un sistema.

Le algebre di processo sono entità matematiche che modellano sistemi concorrenti e forniscono un framework concettuale per analizzare la struttura e il comportamento di un dato sistema. Nelle algebre di processo classiche, come ad esempio CCS (Calculus of Communicating System), il tempo è astratto, cioè le azioni descritte sono considerate istantanee ed importa solo il loro ordine relativo, inoltre le scelte sono generalmente non deterministiche. In PEPA invece viene usata una variabile con distribuzione esponenziale negativa per specificare la durata di ogni azione in modo tale da poter effettuare un mapping fra l'algebra di processo e la sottostante CTMC (Continuos Time Markov Chain). [1]

Gli elementi base di PEPA sono i *componenti* e le *attività*, che corrispondono agli *stati* e alle *transizioni* della corrispondente CTMC. Ogni attività è rappresentata da due informazioni: il tipo di azione, che la identifica, e il *rate* dell'attività cioè il parametro della distribuzione esponenziale negativa che regola la durata dell'attività stessa. Ogni componente può effettuare un insieme di azioni $a \in A$, dove A è l'insieme di tutte le possibili azioni, ogni azione a è rappresentata da una coppia (α, r) dove α è il tipo di azione da compiere ed $r \in R+$ è il corrispondente *rate*. Quando non è importante specificare il tipo di azione, ma solo un'azione generica allora si usa il simbolo τ. PEPA supporta anche un particolare rate,

detto rate passivo, che indica durata ignota dell'attività ed è usato come importante meccanismo di sincronizzazione fra processi, in questo caso l'attività sarà rappresentata dalla coppia (α ,T), dove T indica il rate passivo dell'attività.

Se un componente esegue un'attività $a = (\alpha, r)$, per completarla impiegherà un certo periodo di tempo, governato dalla distribuzione esponenziale negativa con parametro r, cioè la probabilità che un'attività completi la sua azione entro un tempo t è pari ad $1 - e^{-rt}$. In PEPA è fondamentale che la distribuzione sia di tipo esponenziale negativa perché è l'unica distribuzione che permette un mapping diretto con le CTMC, che sono entità matematiche trattabili dal punto di vista analitico, infatti calcolare la distribuzione stazionaria (se esiste) di una CTMC è un problema risolvibile con un sistema di equazioni lineari [6], per il quale esistono metodi di risoluzione molto performanti.

1.2 Sintassi e semantica

Il linguaggio PEPA fornisce un insieme ristretto di operatori che permettono di descrivere un sistema tramite la definizione del comportamento di ogni singolo componente, delle attività che intraprendono e le reciproche interazioni. La sintassi del linguaggio viene descritta in modo formale dalla seguente grammatica:

$$S ::= (\alpha, r).S \mid S + S \mid C_S$$
$$P ::= P \underset{L}{\bowtie} P \mid P/L \mid C$$

dove S indica un componente *sequenziale* mentre P indica un componente del *modello* che viene eseguito in parallelo. C indica una costante che può rappresentare sia un componente sequenziale che un componente del modello mentre C_s indica una costante che rappresenta un componente sequenziale. Serve adesso una descrizione degli operatori:

- Prefix: $(\alpha, r).S$

Indicato dal simbolo ".", è il meccanismo base per descrivere il comportamento di un sistema. Un dato componente porta avanti l'azione *(α, r)* e successivamente si comporta come *S*. L'operatore di prefisso può essere combinato più volte per descrivere il ciclo di vita di un componente, ad esempio:

$$Comp =_{\text{def}} (error, e).(repair, p).Comp$$

indica un componente il cui comportamento è quello di eseguire l'azione *error* con rate *e*, eseguire l'azione *repair* con rate *p* e ripetere questo comportamento.

- Choice: $S + S$

  Indicato dal simbolo "+", serve per descrivere comportamenti più complessi, infatti permettere di descrivere la possibilità di competizione o selezione fra differenti attività. Il componente il cui comportamento è $S_1 + S2$ può comportarsi sia come S1 che come S2, a seconda di quale delle due attività viene completata per prima, l'altra viene scartata. Il sistema si comporterà in modo conseguente all'evoluzione del componente.

- Constant: *C*

  A volte è conveniente assegnare nomi a pattern di comportamento associati a componenti e questo è proprio quello che fa l'operatore di costante. Ad esempio $C =_{\text{def}} P$ assegna alla costante *C* il comportamento del comportamento del componente *P*.

- Cooperation: $P \underset{L}{\bowtie} P$

  Questo operatore permette di descrivere interazione diretta, o *cooperazione*, fra componenti. L'insieme *L* di azioni visibili è determinante in quanto i componenti saranno forzati a cooperare solo sulle attività presenti all'interno di tale insieme, tutte le altre azioni che non sono comprese in *L* vengono portate avanti in modo indipendente da ogni componente. Quando un componente completa un'attività che è all'interno dell'insieme delle azioni visibili *L* non sarà in grado di proseguire fino a quando anche l'altro componente coinvolto non avrà completato un'attività di quel tipo. I due

componenti procedono quindi al completamento dell'attività condivisa. Il rate dell'azione condivisa può essere alterato per riflettere il lavoro portato avanti da entrambi i componenti per completare l'attività. Come esempio prendiamo un sistema che necessita di cooperare con una risorsa per completare un certo task, questa cooperazione viene espressa come:

$$System \stackrel{\text{def}}{=} Comp \underset{\{task\}}{\bowtie} Res$$

Il sistema potrebbe a sua volta cooperare con un'entità esterna, ad esempio:

$$System \underset{\{repair\}}{\bowtie} Repman$$

oppure, in modo totalmente equivalente:

$$(Comp \underset{\{task\}}{\bowtie} Res) \underset{\{repair\}}{\bowtie} Repman$$

In alcuni casi un componente può essere passivo rispetto all'attività che viene indicata come (α ,T), questo significa che il rate non viene specificato ed è determinato dal rate della corrispondente attività del componente attivo nell'atto della cooperazione.

Se l'insieme L è vuoto allora i due componenti procedono in modo indipendente, senza attività condivise e vengono rappresentati con la notazione compatta P || Q.

- Hiding: *P / L*

  Indicato dal simbolo " / " questo operatore serve a rendere private e non visibili ad un osservatore esterno tutte le azioni specificate nel set L, queste azioni non saranno quindi utilizzabili come azioni di cooperazione.

Fra tutti gli operatori il più prioritario è l'operatore di Hiding, seguito dall'operatore Prefix e da quello di Cooperation, l'operatore Choice è il meno prioritario. Ovviamente la priorità fra operatori può essere modificata tramite l'uso di parentesi.

I componenti del modello catturano la struttura del sistema in termini di componenti statici, il comportamento dinamico del sistema è rappresentato dall'evoluzione di questi componenti, sia individualmente che

in cooperazione fra loro. La forma di questa evoluzione è governata da un insieme di regole formali che danno semantica operazionale ai termini PEPA [2]:

Prefix

$$\frac{}{(\alpha, r).E \xrightarrow{(\alpha,r)} E}$$

Choice

$$\frac{E \xrightarrow{(\alpha,r)} E'}{E + F \xrightarrow{(\alpha,r)} E'} \qquad \frac{F \xrightarrow{(\alpha,r)} F'}{E + F \xrightarrow{(\alpha,r)} F'}$$

Cooperation $(\alpha \notin L)$

$$\frac{E \xrightarrow{(\alpha,r)} E'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,r)} E' \underset{L}{\bowtie} F} \qquad \frac{F \xrightarrow{(\alpha,r)} F'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,r)} E \underset{L}{\bowtie} F'}$$

Cooperation $(\alpha \in L)$

$$\frac{E \xrightarrow{(\alpha,r_1)} E' \quad F \xrightarrow{(\alpha,r_2)} F'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,R)} E' \underset{L}{\bowtie} F'} \qquad R = \frac{r_1}{r_\alpha(E)} \frac{r_2}{r_\alpha(F)} r_m \qquad r_m = \min(r_\alpha(E), r_\alpha(F))$$

Hiding

$$\frac{E \xrightarrow{(\alpha,r)} E'}{E/L \xrightarrow{(\alpha,r)} E'/L} \ (\alpha \notin L) \qquad \frac{E \xrightarrow{(\alpha,r)} E'}{E/L \xrightarrow{(\tau,r)} E'/L} \ (\alpha \in L)$$

Constant

$$\frac{E \xrightarrow{(\alpha,r)} E'}{C \xrightarrow{(\alpha,r)} E'} \ (C \stackrel{\text{def}}{=} E)$$

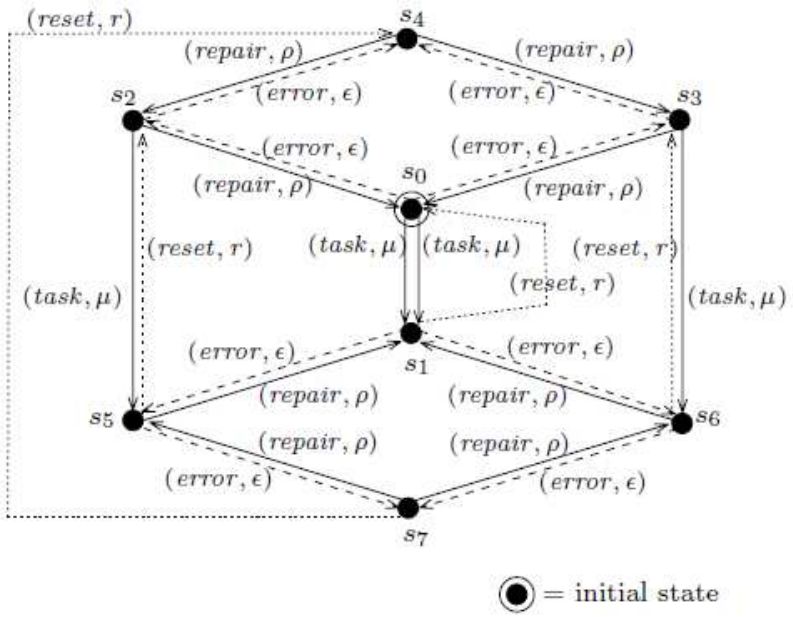
Semantica Operazionale di PEPA

Queste regole evidenziano come un componente che non cooperi su alcuna attività non possa influenzare lo stato computazionale di un altro componente. Nel caso i due componenti cooperino il rate dell'attività risultante rifletterà la capacità di ogni componente di portare avanti un'attività di quel tipo. Dato un componente *E* ed un tipo di azione α definiamo il *rate apparente* $r_\alpha(E)$ come la somma dei rate dell'attività di tipo α abilitati in E.

Come nelle classiche algebre di processo, anche in PEPA la semantica di ogni termine è data da un *labelled multi-transition system*, in cui è significativa la molteplicità degli archi in quanto la molteplicità degli archi influenza il rate apparente di una certa attività. Nel transition system ad ogni termine sintattico del linguaggio PEPA corrisponde uno stato, detto *derivative*, mentre un arco rappresenta un'attività che causa l'evoluzione di un *derivative* in un altro. L'insieme completo di tutti gli stati raggiungibili è detto *derivative set* del modello e forma i nodi del *derivation graph*, costruito applicando in modo esaustivo tutte le regole semantiche [3]. Ad esempio il grafo di derivazione per il sistema:

$$
\begin{aligned}
Comp &\stackrel{\text{def}}{=} (error, \epsilon).(repair, \rho).Comp \\
&\quad + \quad (task, \mu).Comp \\
Res &\stackrel{\text{def}}{=} (task, \top).(reset, r).Res \\
Repman &\stackrel{\text{def}}{=} (repair, \top).Repman \\
((Comp \parallel Comp) &\underset{\{task\}}{\bowtie} Res) \underset{\{repair\}}{\bowtie} Repman
\end{aligned}
$$

È il seguente:



Mentre questo è l'insieme completo degli stati del grafo di derivazione:

State	Corresponding derivative
s_0	$((Comp \parallel Comp) \underset{\{task\}}{\bowtie} Res) \underset{\{repair\}}{\bowtie} Repman$
s_1	$((Comp \parallel Comp) \underset{\{task\}}{\bowtie} (reset, r).Res) \underset{\{repair\}}{\bowtie} Repman$
s_2	$(((repair, \rho).Comp \parallel Comp) \underset{\{task\}}{\bowtie} Res) \underset{\{repair\}}{\bowtie} Repman$
s_3	$((Comp \parallel (repair, \rho).Comp) \underset{\{task\}}{\bowtie} Res) \underset{\{repair\}}{\bowtie} Repman$
s_4	$(((repair, \rho).Comp \parallel (repair, \rho).Comp) \underset{\{task\}}{\bowtie} Res) \underset{\{repair\}}{\bowtie} Repman$
s_5	$(((repair, \rho).Comp \parallel Comp) \underset{\{task\}}{\bowtie} (reset, r).Res) \underset{\{repair\}}{\bowtie} Repman$
s_6	$((Comp \parallel (repair, \rho).Comp) \underset{\{task\}}{\bowtie} (reset, r).Res) \underset{\{repair\}}{\bowtie} Repman$
s_7	$(((repair, \rho).Comp \parallel (repair, \rho).Comp) \underset{\{task\}}{\bowtie} (reset, r).Res) \underset{\{repair\}}{\bowtie} Repman$

E’ interessante notare come nel grafo di derivazione siano presenti due archi fra lo stato iniziale s0 e lo stato s1, entrambi etichettati con *(task,μ)*, questo cattura il fatto che ci sono due distinte derivazioni dell’attività *task*, a seconda che sia il primo o il secondo componente *Comp* a completare il task in cooperazione con la risorsa, anche se il *derivative* che si ottiene è lo stesso in ogni caso.

Gli aspetti temporali del comportamento di un componente non sono rappresentati negli stati del grafo di derivazione ma vengono rappresentati sugli archi, come il parametro della distribuzione esponenziale negativa che regola la durata di ogni azione corrispondente. Quando abilitata, un’attività $a = (\alpha,r)$ impiegherà un certo tempo per essere completata, questo tempo dipende dal parametro r. Se più attività sono abilitate contemporaneamente, sia indipendentemente sia in competizione, assumiamo che esiste una *race condition* fra di loro e l’attività che impiegherà meno tempo ad essere completata sarà quella che verrà considerata come eseguita. L’evoluzione del modello determinerà se le altre attività dovranno essere invalidate oppure semplicemente interrotte a seguito del cambio di stato. In entrambi i casi il fatto che la distribuzione esponenziale negativa sia *senza memoria* permette di non dover ricordare il tempo di esecuzione precedente [6].

Quando due componenti portano avanti un’attività in cooperazione il rate dell’attività condivisa rifletterà la capacità di lavoro del componente più lento. Assumiamo che ogni componente abbia una capacità di lavoro prefissata per una certa attività di tipo α, che non può essere migliorata dal lavoro cooperativo, a meno che il componente non sia passivo rispetto a quel tipo di attività, allora questa capacità di lavoro coincide con il rate apparente di quel componente per quella data attività. Il rate apparente di α nell’attività condivisa fra due componenti P e Q sarà il minimo fra $r_\alpha(P)$ e $r_\alpha(Q)$ cioè il minimo rate apparente per α fra i due componenti P e Q. Questo ha perfettamente senso in quanto, essendo ogni attività regolata da una distribuzione esponenziale negativa, il tempo medio di esecuzione di un’azione viene definito come inverso del parametro *r* che regola la distribuzione stessa.

Il grafo di derivazione è la base della sottostante CTMC che viene usata per calcolare le misure di performance dal modello PEPA. Il grafo viene sistematicamente ridotto in una forma che possa essere trattata, dal punto di vista matematico, come il diagramma di transizione degli stati della CTMC sottostante in cui ogni *derivative* è uno stato nella CTMC.

Capitolo 2

PEPA Eclipse Plugin

Il plugin Eclipse per PEPA permette la creazione e l'analisi delle performance di sistemi descritti in un linguaggio di alto livello che ricalca molto fedelmente la sintassi del linguaggio PEPA sopra descritto.

2.1 Caso di studio

Per spiegare meglio tutte le funzionalità che questo plugin offre si analizzerà un caso di studio che rappresenta un web service bancario, tramite il quale un cliente può fare richiesta di un prestito.

Eccone la descrizione in linguaggio PEPA:

```
Customer_Idle = (request , r_request ).Customer_Entering;
Customer_Entering = (enterData, r_enterdata ).Customer_Upload;
Customer_Upload = (uploadData, r_upload ).Customer_Waiting;
Customer_Waiting = (approve, T).Customer_Idle + (decline,
T).Customer_Deciding;
Customer_Deciding = (reapply, r_reapply).Customer_Entering +
(reapply, r_reapply).Customer_Idle;

Service_Idle = (uploadData, T ).Service_Validating;
Service_Validating = (validateData, r_validate ).Service_Sending;
Service_Sending = (sendBank, r_sendBank ).Service_Idle;

Bank_Idle = (sendBank, T ).Bank_PreDecide;
Bank_PreDecide = (predecide, r_predecide).Bank_Approve
      + (predecide, r_predecide).Bank_Decline
      + (predecide, r_predecide).Bank_Employee;
Bank_Employee = (decide, r_decide).Bank_Confirm
      + (decide, r_decide).Bank_Decline;
Bank_Confirm = (decide, r_decide).Bank_Approve
      + (decide, r_decide).Bank_Decline;
Bank_Approve = (approve, r_inform).Bank_Idle;
```

```
Bank_Decline = (decline, r_inform).Bank_Idle;

Customer_Idle <uploadData, approve, decline>
                    (Service_Idle <sendBank> Bank_Idle)
```

In questo caso di studio sono presenti tre component: il cliente *Customer*, il web service *Service* e la banca *Bank*. Per prima cosa viene descritto il comportamento di ogni singolo componente:

- Il cliente inizia la procedura di richiesta del prestito inserendo i propri dati bancari ed effettuando l'upload dei dati stessi, successivamente attende una risposta dalla banca, se questa risposta è positiva il cliente conclude la transazione ed il sistema è pronto per un nuovo cliente mentre se la risposta è negativa il cliente può inserire nuovamente i dati per tentare una nuova richiesta oppure decidere di abbandonare.
- Il web service fa da tramite fra il cliente e la banca: aspetta l'upload dei dati da parte del cliente, effettua un controllo sulla correttezza dei dati inseriti e successivamente li comunica alla banca.
- La banca aspetta i dati dal web service, una volta ricevuti li analizza ed entra in una fase di pre-decisione al termine della quale il prestito può essere accettato, rifiutato oppure essere sottoposto ad un ulteriore controllo da parte di un impiegato della banca stessa. L'impiegato controlla i dati e sottopone nuovamente il prestito alla banca, che deciderà se concederlo o meno. In ogni caso il cliente riceverà alla fine una risposta positiva o negativa in merito alla sua richiesta di prestito

Successivamente viene descritto come i componenti sono connessi e su quali attività cooperano, al fine di comporre il sistema che intendiamo rappresentare: il web service e la banca cooperano sull'attività relativa alla trasmissione dei dati bancari del cliente ed il sistema risultante coopera con il cliente sulle attività di inserimento di dati e risposta alla richiesta. In questo esempio abbiamo un solo componente per ogni tipo.

Una volta inserita questa specifica all'interno dell'IDE viene fatto subito un controllo automatico sulla sintassi e sul fatto che la nostra

specifica non rappresenti un sistema che al suo interno contenga deadlock, superati questi controlli preliminari siamo pronti per operare ed effettuare le misure di performance che caratterizzano il sistema. Per prima cosa dobbiamo derivare lo spazio degli stati, cioè l'insieme dei possibili *derivative* del sistema, per il sistema in oggetto il corrispondente spazio degli stati sarà:

11 states

1	Customer_Idle	Service_Idle	Bank_Idle
2	Customer_Entering	Service_Idle	Bank_Idle
3	Customer_Upload	Service_Idle	Bank_Idle
4	Customer_Waiting	Service_Validating	Bank_Idle
5	Customer_Waiting	Service_Sending	Bank_Idle
6	Customer_Waiting	Service_Idle	Bank_PreDecide
7	Customer_Waiting	Service_Idle	Bank_Approve
8	Customer_Waiting	Service_Idle	Bank_Decline
9	Customer_Waiting	Service_Idle	Bank_Employee
10	Customer_Deciding	Service_Idle	Bank_Idle
11	Customer_Waiting	Service_Idle	Bank_Confirm

E' già possibile navigare lo spazio degli stati ed analizzare ogni singolo stato per scoprire da quali stati è raggiunto e quali stati può raggiungere.

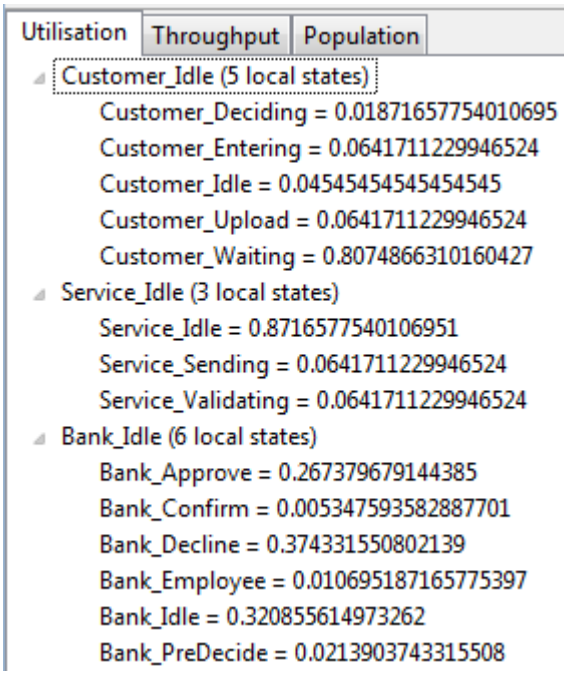
A questo punto siamo pronti per risolvere il modello ed eseguire la *steady state analysis*, cioè ricavare lo stato stazionario della sottostante CTMC. Sono disponibili diversi metodi di risoluzione, il metodo diretto è sicuramente il più corretto ma è poco performante su spazi degli stati molto grandi sui quali invece è molto performante il metodo di risoluzione Hydra AIR. In questo caso useremo il metodo diretto:

11 states

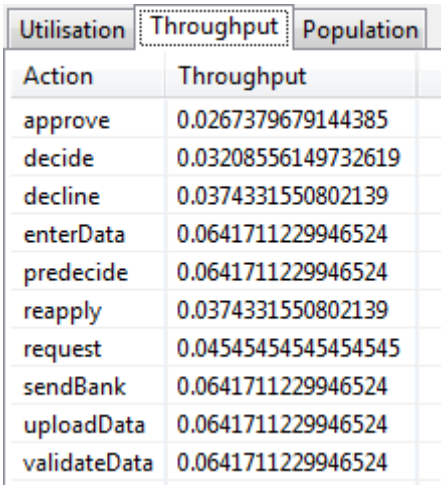
1	Customer_Idle	Service_Idle	Bank_Idle	0.04545454545454545
2	Customer_Entering	Service_Idle	Bank_Idle	0.0641711229946524
3	Customer_Upload	Service_Idle	Bank_Idle	0.0641711229946524
4	Customer_Waiting	Service_Validating	Bank_Idle	0.0641711229946524
5	Customer_Waiting	Service_Sending	Bank_Idle	0.0641711229946524
6	Customer_Waiting	Service_Idle	Bank_PreDecide	0.0213903743315508
7	Customer_Waiting	Service_Idle	Bank_Approve	0.267379679144385
8	Customer_Waiting	Service_Idle	Bank_Decline	0.374331550802139
9	Customer_Waiting	Service_Idle	Bank_Employee	0.010695187165775397
10	Customer_Deciding	Service_Idle	Bank_Idle	0.01871657754010695
11	Customer_Waiting	Service_Idle	Bank_Confirm	0.005347593582887701

Il numero accanto ad ogni stato è un probabilità e rappresenta la *steady state probability* cioè la probabilità che il sistema si trovi in quello stato quando il sistema ha raggiunto lo stato stazionario. Già risolvendo il modello sono disponibili automaticamente tre misure di performance:

- Utilisation: per ogni componente indica la quantità, in percentuale, di tempo di vita passato in ogni stato.



- Throughput: indica, per ogni tipo di attività, quante di queste vengono eseguite in una unità di tempo.



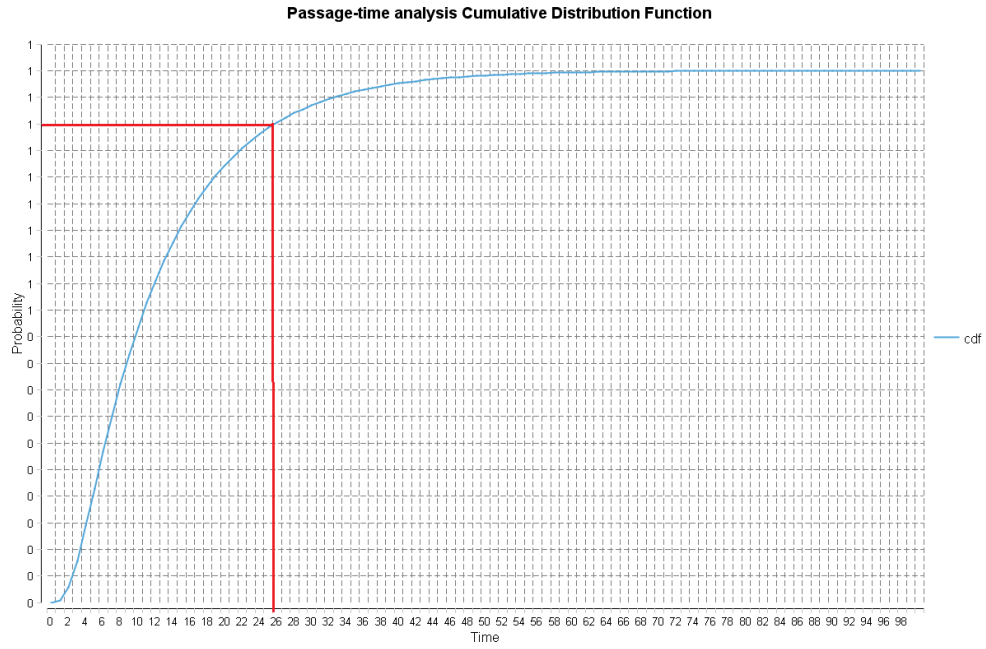
Action	Throughput
approve	0.0267379679144385
decide	0.03208556149732619
decline	0.0374331550802139
enterData	0.0641711229946524
predecide	0.0641711229946524
reapply	0.0374331550802139
request	0.04545454545454545
sendBank	0.0641711229946524
uploadData	0.0641711229946524
validateData	0.0641711229946524

- Population: indica il numero medio di copie di un componente sequenziale quando il modello raggiunge lo stato stazionario.

Utilisation | Throughput | Population

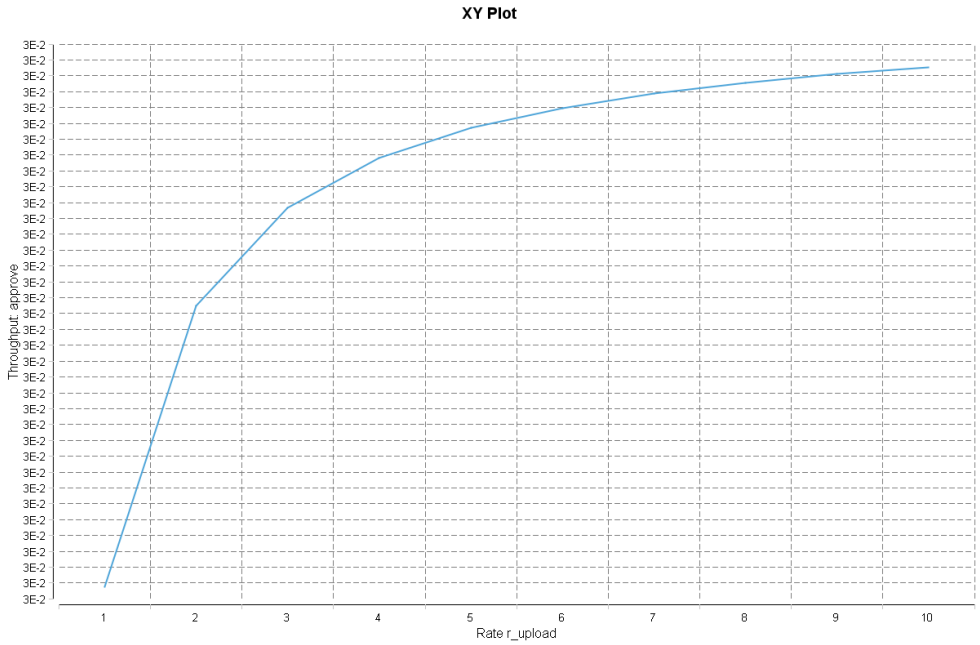
Component Name	Population
Bank_Approve	0.267379679144385
Bank_Confirm	0.005347593582887701
Bank_Decline	0.374331550802139
Bank_Employee	0.010695187165775397
Bank_Idle	0.320855614973262
Bank_PreDecide	0.0213903743315508
Customer_Deciding	0.01871657754010695
Customer_Entering	0.0641711229946524
Customer_Idle	0.04545454545454545
Customer_Upload	0.0641711229946524
Customer_Waiting	0.8074866310160427
Service_Idle	0.8716577540106951
Service_Sending	0.0641711229946524
Service_Validating	0.0641711229946524

Adesso è possibile seguire un'analisi temporale per valutare aspetti temporali del sistema, ad esempio: una volta eseguita l'azione di *uploadData* qual è la probabilità che il prestito venga accettato/rifiutato entro un certo tempo?

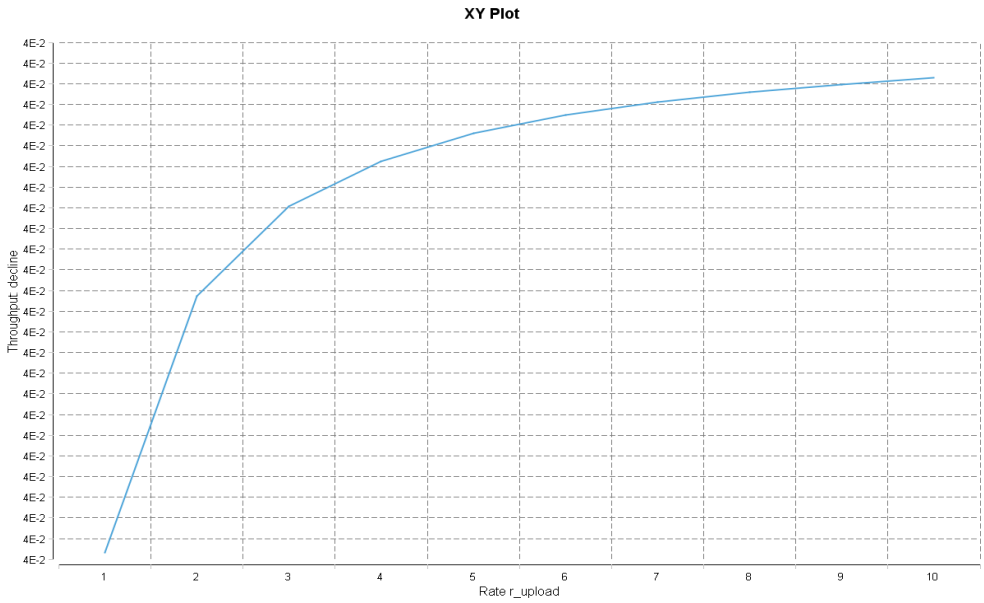


Graficando la distribuzione di probabilità cumulativa possiamo vedere come, entro 25 unità di tempo, il prestito venga accettato/rifiutato con una probabilità del 90%

Un altro interessante aspetto che è possibile analizzare è come variano le misure di performance al variare dei rate delle azioni, supponiamo di migliorare la velocità di upload dei dati, cioè aumenterà il rate dell'attività *uploadData* (e di conseguenza ne calerà la durata), come cambierà il throughput delle attività relative all'accettazione/rifiuto del prestito? Cambiano il rate dell'attività da 1.0 a 10.0 con step 1.0 otteniamo i seguenti risultati:



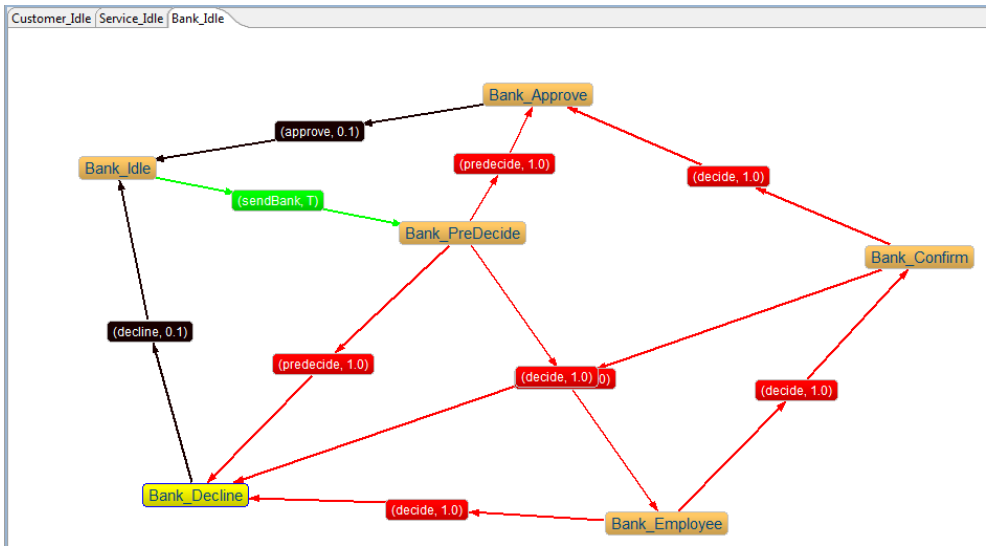
Throughput per l'attività di accettazione



Throughput per l'attività di rifiuto

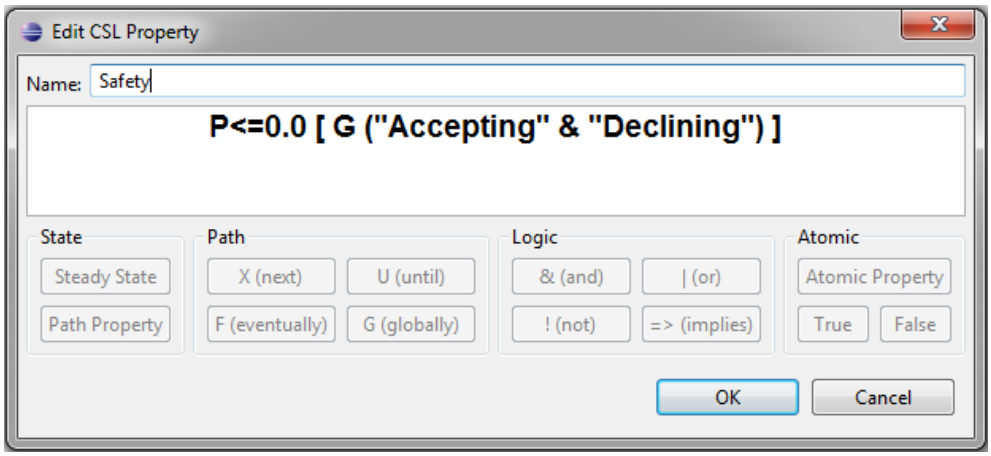
Come era possibile intuire se cala il tempo impiegato per fare l'upload dei dati allora, nella stessa unità di tempo, aumenterà il numero di altre attività eseguite. Purtroppo i grafici prodotti non danno informazioni significative su quanto cambi il throughput delle azioni dato che sull'asse Y del grafico la scala è del tutto non significativa, questo è un grosso limite del plugin Eclipse.

E' possibile anche fare del Model Cheking: aprendo la Abstraction View possiamo avere una rappresentazione grafica del sistema per ogni componente, ad esempio per il componente relativo alla banca:



Vediamo come ad ogni stato interno del componente corrisponda un nodo del grafico mentre ad ogni attività corrisponde un arco, sul quale viene indicata la corrispondente attività ed il suo rate. Anche i colori degli archi sono significativi, gli archi verdi rappresentano i rate passivi mentre i rate attivi sono rappresentati con il colore rosso, più un'attività è veloce più scura sarà la tonalità di rosso che colora l'arco. Adesso è possibile dichiarare delle proprietà che valgono negli stati, cioè "etichettare" gli stati: come esempio etichettiamo lo stato *Bank_Approve* con una proprietà *Accepting* (cioè prestito accettato) che varrà solo in quello stato ed analogamente facciamo la stessa cosa con una proprietà *Declining* (cioè prestito rifiutato) che sarà vera solo nello stato *Bank_Decline*. Questo renderà possibile fare il vero e proprio Model Checking, passando alla Model Checking View possiamo specificare delle nuove proprietà del

sistema tramite un editor guidato per il linguaggio CSL e verificarle. Ad esempio vogliamo verificare che il sistema non venga mai a trovarsi in una situazione in cui il prestito è contemporaneamente approvato e rifiutato, pertanto esprimiamo questo come:



Cioè verifichiamo che, globalmente, la probabilità che *Accepting* e *Declining* siano vere nello stesso momento sia minore o uguale a 0. Eseguendo il controllo otteniamo:

Name	CSL Property	Result
Safety	P<=0.0 [G ("Accepting" & "Declining")]	True

questo risultato conferma la bontà del modello proposto.

Un limite di questo plugin è però la possibilità di effettuare il Model Checking come sopra descritto solo se è presente un solo componente per ogni tipo, se avessi specificato il sistema in oggetto come:

```
Customer_Idle[3] <uploadData, approve, decline>
                    (Service_Idle <sendBank> Bank_Idle)
```

cioè con 3 elementi di tipo *Customer_Idle* in parallelo le viste di Abstraction View e di Model Checking View sarebbero state disabilitate e non avrei potuto verificare la proprietà sopra descritta. Le altre procedure specificate, come l'analisi dello stato stazionario o l'analisi temporale, non risentono della molteplicità dei componenti.

Capitolo 3

Fluid Flow Approximation

3.1 Motivazioni

La creazione di modelli accurati per l'analisi di performance è diventata sempre più importante a seguito della comparsa di sistemi sempre più grandi e complessi. Queste misure possono essere molto utili per individuare criticità del sistema stesso, come colli di bottiglia, e possono anche suggerire modi in cui rimediare a queste eventuali criticità.

Ci sono diversi metodi possibili per eseguire l'analisi delle performance di un sistema. Se questo sistema esiste già allora è possibile monitorarlo per un certo periodo di tempo al fine di ricavarne delle misure di performance ma è ovvio che non è possibile usare questo approccio in fase di progettazione di un sistema. Serve quindi modellare il sistema in modo formale, ad esempio in linguaggio PEPA, al fine di poterlo analizzare tramite metodi matematici come simulazioni stocastiche, metodi analitici o metodi numerici. Tutti questi metodi si basano, in ogni caso, sulla CTMC che corrisponde al sistema descritto, questo è un vantaggio in quanto, come detto anche sopra, le CTMC sono entità facilmente trattabili dal punto di vista matematico. Purtroppo però modelli realistici di sistemi complessi possono facilmente generare CTMC il cui spazio degli stati è talmente grande da essere computazionalmente intrattabile. In generale, la generazione dello spazio degli stati e il calcolo delle corrispondenti probabilità dello stato stazionario, è un problema computazionalmente troppo oneroso per essere risolto per via numerica o analitica.

Questo fenomeno, conosciuto come "esplosione dello spazio degli stati", è uno dei maggiori problemi nel campo delle analisi di performance ed è molto limitante nella dimensione dei modelli e quindi nella complessità

dei modelli che possono essere analizzati in modo efficiente, in particolar modo nell'analisi di sistemi paralleli su larga scala.

Sorge quindi la necessità di un approccio differente per poter trattare modelli di sistemi complessi, ne sono stati sviluppati diversi ma quello su cui puntiamo l'attenzione in questo elaborato è chiamato *fluid flow approximation* e consiste nell'approssimazione di uno spazio degli stati discreto con uno spazio degli stati continuo al fine di rendere il sistema rappresentabile da un sistema di equazione differenziali ordinarie (ODE) [6]. Questi sistemi di ODE sono facilmente trattabili dal punto di vista numerico (per esempio, attraverso il metodo di Eulero) e rendono questo approccio altamente scalabile, infatti è molto più veloce risolvere le equazioni differenziali piuttosto che eseguire simulazioni stocastiche secondo i metodi standard.

Con la *fluid flow approximation* si otterranno però dei valori che saranno solo delle approssimazioni dei valori attesi di certe variabili associate al modello, questo è ovviamente uno dei problemi più grossi in quanto questo valore approssimato è inutile se non è associato alla conoscenza di come il comportamento del sistema è "distribuito" intorno a questo valore. Inoltre, sebbene all'apparenza adatti ad un'approssimazione continua dello spazio degli stati, molti modelli presentano componenti intrinsecamente discreti il cui comportamento non avrebbe senso se trattato in modo continuo. Una delle maggiori aree di ricerca nel campo della *fluid flow approximation* riguarda la gestione di questi componenti, infatti sia che vengano ignorati o trattati come continui portano sempre alla generazione di un modello inaccurato.

Dato che questo approccio non è quindi universalmente applicabile, da ora in avanti tutti i modelli a cui si farà riferimento per spiegare come viene generato il sistema di ODE saranno modelli di sistemi altamente simmetrici, cioè sistemi in cui ci sono molti componenti omogenei che operano in parallelo o in cooperazione.[6]

3.2 Generazione del sistema di ODE

Dati *N* componenti seriali che operano in cooperazione, ognuno con *S* possibili *derivative*, vengono introdotti *S* contatori interi $\{c_i\}$ $(1 \leq i \leq S)$ con $0 \leq c_i \leq N$ per ogni *i*, dove c_i conta il numero di componenti seriali che si trovano in un derivative *i* in un dato istante temporale. Questo permette di contare solo quanti componenti sono in un determinato stato, senza distinguere esplicitamente quale componente sia in un determinato stato, in questo modo si genera uno spazio degli stati aggregato di dimensioni inferiori allo spazio originale. L'aggregazione dello spazio degli stati è un prerequisito fondamentale per poter generare il sistema di ODE ed è una tecnica molto importante sulla quale vale la pena soffermarsi un attimo per capire quanto possa effettivamente ridurre uno spazio degli stati al fine di poterlo maneggiare. Supponiamo di avere un sistema composto da 2 processori e 2 risorse, descritto come:

$$Processor_0 \stackrel{def}{=} (task_1, \top).Processor_1$$
$$Processor_1 \stackrel{def}{=} (task_2, r_2).Processor_0$$
$$Resource_0 \stackrel{def}{=} (task_1, r_1).Resource_1$$
$$Resource_1 \stackrel{def}{=} (reset, s).Resource_0$$

$$Sys \stackrel{def}{=} (Processor_0 \parallel Processor_0) \underset{\{task_1\}}{\bowtie} (Resource_0 \parallel Resource_0)$$

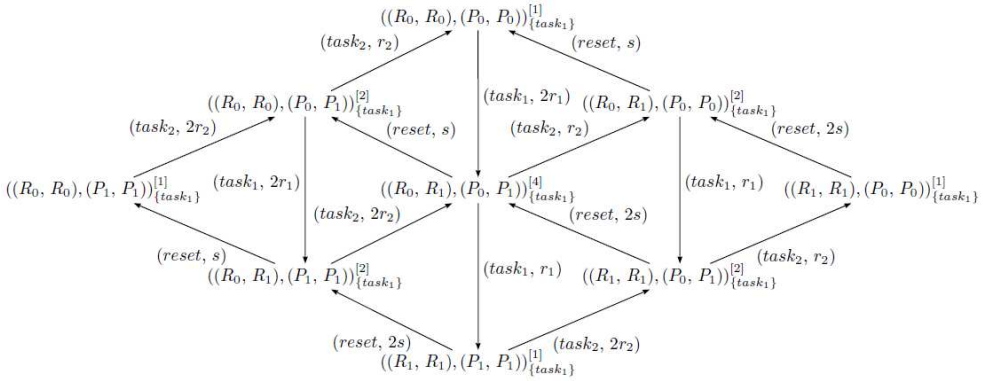
Osserviamo come la prima azione da fare sia inevitabilmente *task1* che però può avvenire in quattro modi differenti:

1. $(P_0 \parallel P_0) \underset{\{task_1\}}{\bowtie} (R_0 \parallel R_0) \xrightarrow{(task_1, r_1/2)} (P_1 \parallel P_0) \underset{\{task_1\}}{\bowtie} (R_1 \parallel R_0)$
2. $(P_0 \parallel P_0) \underset{\{task_1\}}{\bowtie} (R_0 \parallel R_0) \xrightarrow{(task_1, r_1/2)} (P_1 \parallel P_0) \underset{\{task_1\}}{\bowtie} (R_0 \parallel R_1)$
3. $(P_0 \parallel P_0) \underset{\{task_1\}}{\bowtie} (R_0 \parallel R_0) \xrightarrow{(task_1, r_1/2)} (P_0 \parallel P_1) \underset{\{task_1\}}{\bowtie} (R_1 \parallel R_0)$
4. $(P_0 \parallel P_0) \underset{\{task_1\}}{\bowtie} (R_0 \parallel R_0) \xrightarrow{(task_1, r_1/2)} (P_0 \parallel P_1) \underset{\{task_1\}}{\bowtie} (R_0 \parallel R_1)$

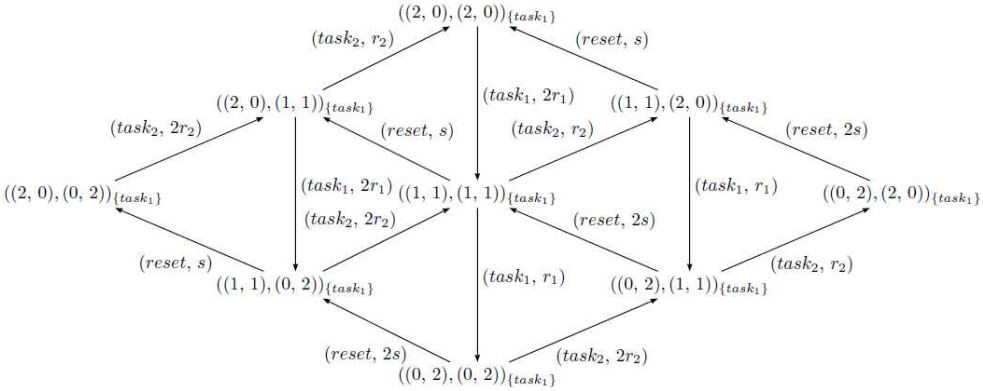
Questi quattro modi sono però logicamente equivalenti (dato che tutti gli stati che ne derivano presentano un P0, un P1, un R0 e un R1) e quindi possono essere contati come un solo nella forma:

$$((P_0,\, P_0)\,,\, (R_0,\, R_0))_{\{task_1\}} \xrightarrow{(task_1,\, 2r_1)} ((P_0,\, P_1)\,,\, (R_0,\, R_1))_{\{task_1\}}$$

Effettuando questo procedimento per tutte le possibili azioni otteniamo quindi uno spazio degli stati aggregato che viene definito in *forma vettoriale canonica*:



Questa *forma vettoriale canonica* può essere ulteriormente trasformata in una forma detta *forma vettoriale numerica* in cui viene rappresentato, per ogni componente, il numero di ogni *derivative* presenti in un dato stato:



Alla fine di questo procedimento, dati N componenti ognuno dei quali con D possibili *derivative*, lo spazio degli stati verrà ridotto da D^N a fino a $\frac{(D+N-1)!}{N!(D-1)!}$.[3][6] Per avere un esempio numerico di quanto venga ridotto lo spazio degli stati dopo l'aggregazione supponiamo di avere un sistema con 3 tipi di componenti: il primo avrà 4 *derivative*, il secondo 2 ed il terzo 8, il sistema è composto da 200 componenti del primo tipo che cooperano con 300 componenti del secondo tipo e 50 del terzo, lo spazio degli stati non aggregato risultante avrebbe:

$$4^{200}.2^{300}.8^{50} = 2^{400}.2^{300}.2^{150} = 2^{850}$$

stati, un numero palesemente ingestibile mentre dopo l'aggregazione il numero degli stati verrà ridotto a:

$$\begin{aligned}\frac{203!}{200!3!}\frac{301!}{300!}\frac{57!}{50!7!} &= \frac{203.202.201}{6}301\frac{57.56.55.54.53.52.51}{7.6.5.4.3.2} \\ &= 203.101.67.301.57.4.11.9.53.13.17 \\ &\leq 2^8.2^7.2^7.2^9.2^6.2^2.2^4.2^4.2^6.2^4.2^5 = 2^{62} \\ &\geq 2^7.2^6.2^6.2^8.2^5.2^2.2^3.2^3.2^5.2^3.2^4 = 2^{52}\end{aligned}$$

Il miglioramento è significativo, il numero degli stati è ancora troppo elevato per un'analisi numerica ma il risultato è la base da cui partire per poter applicare la tecnica di *fluid flow approximation*.

Prima di vedere come vengono generate le ODE servono ancora alcune definizioni preliminari:

- C_{ij} è il j-esimo *derivative* del tipo di componente C_i, per ogni C_{ij} sono associate attività in entrata, cioè quelle che fanno diventare un *derivative* C_{ik} un *derivative* C_{ij} ed attività in uscita, cioè quelle che fanno diventare un *derivative* C_{ij} un altro *derivative*. Queste attività avranno corrispondenti rate di entrata e di uscita.
- v_{ij} come il numero di istanze del j-esimo *derivative* locale del tipo di componente C_i presenti nello stato corrente. L'idea è quella che la variazione dinamica di v_{ij} possa essere descritta come:

$$\begin{aligned}\frac{dv_{ij}(t)}{dt} = &- \sum_{k:C_{ij}\xrightarrow{(a,.)}C_{ik}} \text{exit rate from component } C_{ij} \text{ to } C_{ik} \\ &+ \sum_{k:C_{ik}\xrightarrow{(b,.)}C_{ij}} \text{entry rate into component } C_{ij} \text{ from } C_{ik}\end{aligned}$$

- *Time-based system equation*, rappresentata come P(t), è l'equazione che descrive un modello PEPA, con l'aggiunta di un'indicazione temporale. P(t) indica il modello PEPA al tempo t.
- *Derivative tuple* ($N(S_1, t), \ldots, N(S_N, t)$)$_L$ conta in numero di *derivative* S_i in un cluster di componenti S_L (un cluster di componenti è un insieme omogeneo di componenti S che cooperano su un set di azioni L). La funzione *N(S, t)* rappresenta il numero di

componenti che sono nello stato S al tempo t, dato un modello PEPA rappresentato da un'equaizone *time-based* come quella sopra descritta.

- *Component counting function*, rappresentata come $v_a(C, P(t))$, conta il numero di componenti che un'azione individuale fa evolvere.
- *Component rate function:* rappresentata con $\rho_a(C,P)$, se applicata ad uno stato del tipo di componente C all'interno di un sistema P calcola il rate osservato localmente dell'azione *a*, che può differire dal rate globale o dal rate apparente della stessa azione. Quindi se *a* parteciperà ad attività di cooperazione dovrà riflettere il lavoro portato avanti dai componenti in cooperazione (come definito dal rate apparente introdotto precedentemente) mentre se non è parte di alcuna cooperazione dipenderà dal numero di componenti del cluster che possono eseguire quell'azione. Questa funzione è fondamentale nella generazione del sistema di ODE ed è definita come:

$$\rho_a\left(C,\, P_1(t) \underset{L}{\bowtie} P_2(t)\right) := \begin{cases} \frac{\rho_a(C,P_i(t))}{r_a(P_i(t))} \min(r_a(P_i(t)),\, r_a(P_j(t))) & \text{if } a \in L,\, C \in \mathcal{V}_a(P_i) \\ \rho_a(C,\, P_i(t)) & \text{if } a \in L,\, C \in \mathcal{I}n(P_i),\, C \notin \mathcal{V}_a(P_i) \\ & \text{or } a \notin L,\, C \in \mathcal{I}n(P_i) \\ 0 & \text{otherwise} \end{cases}$$

Dove *In(P)* rappresenta l'insieme degli stati dei componenti sequenziali che possono essere esibiti in P mentre $V_a(P)$ rappresenta i componenti sequenziali all'interno di P che possono partecipare ad *a*.

Esaurite queste definizioni siamo adesso in grado di definire un'approssimazione che rappresenti la variazione di $N(C_{ij}, t)$ all'interno di *P(t)*:

$$N(C_{ij},\, t+\delta t) - N(C_{ij},\, t) = \underbrace{-\sum_{k:C_{ij} \xrightarrow{(a,\,.)} C_{ik}} p_a(C_{ij},\, C_{ik})\rho_a(C_{ij},\, P(t))v_a(C_{ij},\, P(t))\delta t}_{\text{exit activities}} \underbrace{+\sum_{k:C_{ik} \xrightarrow{(b,\,.)} C_{ij}} p_b(C_{ik},\, C_{ij})\rho_b(C_{ik},\, P(t))v_b(C_{ik},\, P(t))\delta t}_{\text{entry activities}}$$

dove $p_a(S,T)$ è la probabilità che S evolva in uno stato T al seguito dell'esecuzione di un'azione di tipo *a*:

$$p_a(S,T) := \frac{1}{r_a(S)} \sum_{S \xrightarrow{(a,\lambda_i)} T} \lambda_i$$

Dividendo per δt e facendo tendere δt a 0 e ponendo $v_{ij}(t) = N(C_{ij}, t)$ si ha finalmente il sistema di equazioni differenziali che rappresenta il sistema:

$$\frac{dv_{ij}(t)}{dt} = \underbrace{- \sum_{k:C_{ij} \xrightarrow{(a,.)} C_{ik}} p_a(C_{ij}, C_{ik}) \rho_a(C_{ij}, P(t)) v_a(C_{ij}, P(t))}_{\text{exit activities}} + \underbrace{\sum_{k:C_{ik} \xrightarrow{(b,.)} C_{ij}} p_b(C_{ik}, C_{ij}) \rho_b(C_{ik}, P(t)) v_b(C_{ik}, P(t))}_{\text{entry activities}}$$

Per comprendere meglio come avviene la generazione torniamo all'esempio sopra citato del sistema con due risorse e due processori, per quanto detto possiamo descriverne la *time-based equation* come:

$$Sys(t) \stackrel{def}{=} P_1(t) \underset{\{task_1\}}{\bowtie} P_2(t)$$

$$P_1(t) \stackrel{def}{=} (N(Processor_0, t), N(Processor_1, t))_{\emptyset}$$

$$P_2(t) \stackrel{def}{=} (N(Resource_0, t), N(Resource_1, t))_{\emptyset}$$

dalla quale deriva il seguente sistema di ODE.

$$\frac{dv_{11}(t)}{dt} = -r_1 I_{11}(t) v_{21}(t) + r_2 v_{12}(t)$$

$$\frac{dv_{12}(t)}{dt} = r_1 I_{11}(t) v_{21}(t) - r_2 v_{12}(t)$$

$$\frac{dv_{21}(t)}{dt} = -r_1 I_{11}(t) v_{21}(t) + s v_{22}(t)$$

$$\frac{dv_{22}(t)}{dt} = r_1 I_{11}(t) v_{21}(t) - s v_{22}(t)$$

nel quale v_{11} indica il numero di componenti Processor0, v_{12} il numero di Process1, v_{21} il numero di Resource0 e v_{22} il numero di Resource1 e

$$I_{ij}(t) := \begin{cases} 1 & \text{if } v_{ij}(t) > 0 \\ 0 & \text{if } v_{ij}(t) = 0 \end{cases}$$

3.3 Caso di studio

Il caso di studio presentato per sfruttare le potenzialità dell'approccio ad ODE fa riferimento ad un modello che va a descrivere l'evoluzione di un sistema sottoposto ad un attacco di rete del tipo DDOS (Distribuited Denial Of Service), il sistema è *Internet-scale* cioè può comprendere migliaia di entità connesse al sistema e connessioni fra queste entità.

I tipi di compenti del sistema sono:

- S: un componente di tipo S è un computer, connesso alla rete, che è suscettibile ad infezione da parte di un virus.
- I: un componente di tipo I rappresenta un computer infettato dal virus, una volta infetto il computer può infettare altri computer, eseguire in attacco DDOS oppure essere riparato per "curare" l'infezione.
- A: un componente di tipo A rappresenta un computer infettato che esegue attacchi DDOS, ad esempio richiedendo una certa pagina o eseguendo un *ping*. Può essere riparato per porre fine agli attacchi.
- R: un componente di tipo R rappresenta un computer che è stato "curato" ed una volta eseguita la cura non potrà più essere soggetto ad infezione da parte di questo virus né eseguire altri attacchi DDOS sulla rete.
- Net: un componente di tipo Net rappresenta una connessione di rete che può essere sfruttata da un computer infetto per infettare altri computer o compiere un attacco DDOS. Sia l'infezione che l'attacco possono fallire a causa di problemi di rete.
- V: un componente V rappresenta l'attacco DDOS vero e proprio in cui il computer attaccante tiene occupato per un certo periodo di tempo il computer attaccato.

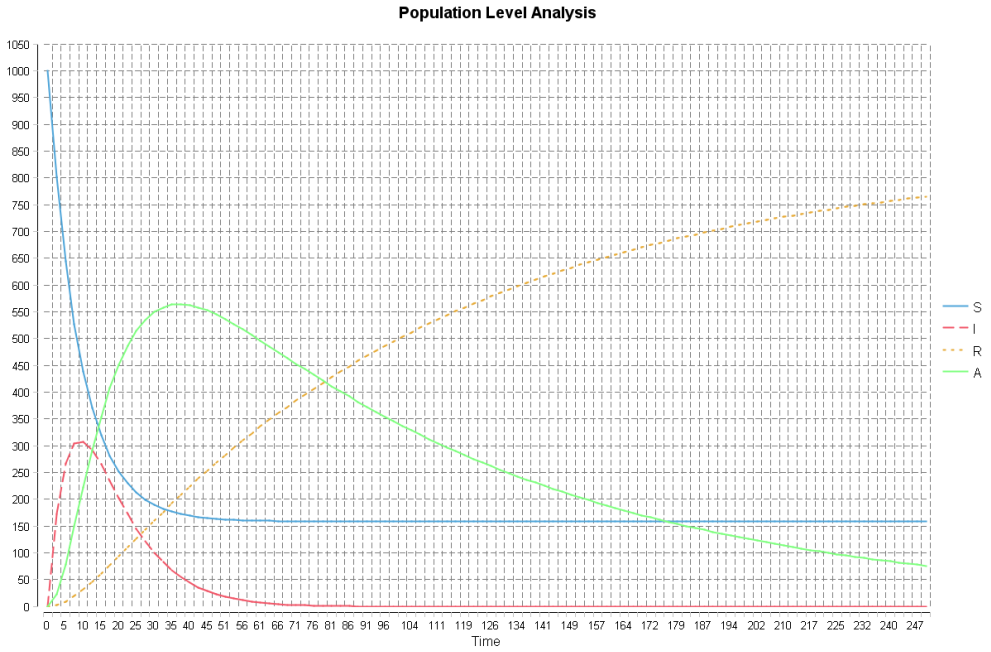
Supponiamo che il sistema sia una porzione di rete a cui sono connessi 1000 computer che viene attaccata da 10 computer infetti, su questa rete sono supportate al massimo 700 connessioni in contemporanea.

La specifica in linguaggio PEPA per Eclipse del sistema è la seguente:

```
beta = 10.0;
gamma = 0.01;
delta = 0.5;
x = 0.1;
lambda = 0.1;
p = 0.1;
sigma = 0.01;

S = (infectS, 10.0).I;
I = (infectI, beta).I + (infectS,10.0).I + (patch, gamma).R
      + (attack, x).A;
A = (attackA, lambda).A + (patch, gamma).R;
R = (stop,1.0).R;

Net = (infectI,10.0).Net0 + (attackA, 1.0).Net1;
Net0 = (infectS, beta).Net + (failR, delta).Net;
Net1 = (attackV, p).Net + (failR, delta).Net;

V = (attackV, 1.0).V0;
V0 = (release, sigma).V;

(S[1000] <> I[10] <> V)
      <infectI, infectS, attackA, attackV> Net[700]
```

Il modello espresso potrebbe essere più espressivo se fosse possibile usare il rate passivo ma il plugin Eclipse usato per eseguire l'analisi ODE non supporta l'uso del rate passivo. I rate espressi numericamente all'interno dei comportamenti sono in sostituzione dei rate passivi mentre quelli dichiarati sono rate attivi.

Il calcolo dello spazio degli stati è impossibile da fare ma per modelli di questo tipo può eccedere facilmente i 10^{10000} stati, eseguendo l'analisi ODE otteniamo:



Come si può vedere il numero di computer suscettibili ad infezione cala rapidamente nelle prime fasi di vita del sistema, unitamente alla rapida ascesa del numero di computer infetti e al conseguente inizio degli attacchi DDOS. Insieme all'infezione inizia però anche il processo di riparazione del sistema che porta all'eliminazione del virus, infatti il numero di computer suscettibili si stabilizza quando il numero di computer infetti scende a zero e successivamente continuerà a riparare i computer attaccanti fino a farli scendere a zero. Rimarrà un certo numero di computer non riparati ed ancora soggetti a possibili infezioni da parte di questo virus.

Capitolo 4

Conclusioni e Bibliografia

4.1 Conclusioni

PEPA si è dimostrato un buon metodo di descrizione, utile per modellare in maniera piuttosto sintetica sistemi complessi grazie alla sua sintassi compatta e al basso numero di operatori utilizzabili. Possiede una buona capacità espressiva, aumentata dalla possibilità di poter specificare informazioni temporali sulle azioni, anche se risulta complesso parametrizzare i modelli.

Il plugin Eclipse per PEPA è semplice da usare e racchiude in un unico tool diversi strumenti sviluppati per eseguire delle analisi su PEPA, come ogni tool che fa queste analisi è soggetto a problemi di memoria quando lo spazio degli stati da trattare è troppo elevato. La disponibilità dell'analisi ad ODE riesce a sopperire in modo parziale a questo problema dato che rende possibile l'analisi di sistemi molto grandi ma solo per alcuni tipi di sistemi, come precedentemente descritto, inoltre il plugin non supporta l'uso di rate passivi in queste analisi, limitando l'espressività dei modelli stessi.

L'approccio ad ODE si è rivelato fondamentale per trattare spazi degli stati di grandi dimensioni e prevedere l'evoluzione di sistemi su larga scala in modo veloce al fine di comprenderne i meccanismi in anticipo e poter agire di conseguenza. L'approccio ad ODE è attualmente lo stato dell'arte ma in letteratura vengono presentati anche altri approcci come quello ad SDE (*Stochastic Differential Equation)* [6] in cui ogni variabile usata nel sistema di equazioni è a sua volta un processo stocastico, in modo da non perdere la stocasticità intrinseca del sistema quando lo si va ad approssimare in modo continuo, ed un approccio ibrido che permette di trattare i componenti intrinsecamente discreti non trattabili tramite

ODE/SDE grazie ad una approssimazione dei componenti “continui” tramite moto Browniano [6] in modo da poter lasciare invariati i componenti discreti e poter generare il set di SDE.

4.2 Bibliografia

[1] J. Hillstone. *Tuning system: from composition to performance.* Computer Journal, 2005.

[2] J. Hillstone. *Process Algebras for quantitavive analysis.* Lics, 2005.

[3] J. Hillstone, S. Gilmore, M. Ribaudo. *An efficient Algorithm for aggregating PEPA models.* IEEE Transaction on software engineering, (27), May 2001.

[4] J. Hillstone, S. Donatelli, M. Ribaudo. *A Comparison of Performance Evaluation Process Algebra and Generalized Stochastic Petri nets.* In Proc. 6th Intern. Workshop on Petri Nets and Performance Models, pages 158-168, Durham, NC, USA, October 1995.

[5] J. Hillstone. *Fluid flow approximation of PEPA models.* QEST, 2005.

[6] R. Hayden. *Addressing the state space explosion problem for PEPA models through fluid-flow approximation.*