Stochastic Process Algebras — From Individuals to Populations

Jane Hillston

Laboratory for Foundations of Computer Science
University of Edinburgh

24th February 2011

Outline

Outline

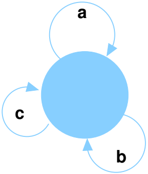
Process Algebra

- Models consist of agents which engage in actions.

$$\alpha.P$$

action type or name

agent/ component



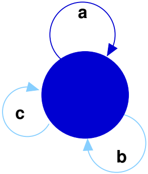
Process Algebra

- Models consist of agents which engage in actions.

$$\alpha.P$$

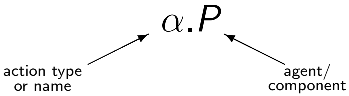
action type or name

agent/ component



Process Algebra

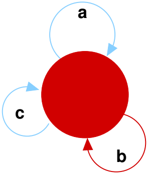
- Models consist of agents which engage in actions.



$$\alpha.P$$

action type
or name

agent/
component



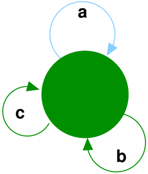
Process Algebra

- Models consist of agents which engage in actions.

$$\alpha.P$$

action type or name

agent/ component



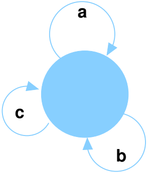
Process Algebra

- Models consist of agents which engage in actions.

$$\alpha.P$$

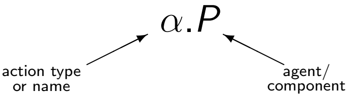
action type or name

agent/ component



Process Algebra

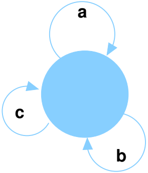
- Models consist of agents which engage in actions.



$\alpha.P$

action type or name

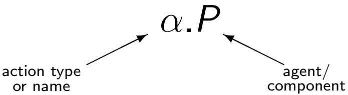
agent/ component



- The structured operational (interleaving) semantics of the language is used to generate a labelled transition system.

Process Algebra

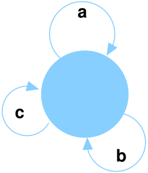
- Models consist of agents which engage in actions.



$\alpha.P$

action type or name

agent/ component

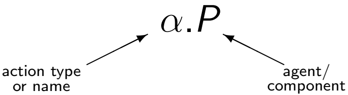


- The structured operational (interleaving) semantics of the language is used to generate a labelled transition system.

Process algebra
model

Process Algebra

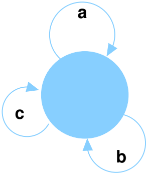
- Models consist of agents which engage in actions.



$\alpha.P$

action type or name

agent/ component

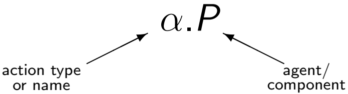


- The structured operational (interleaving) semantics of the language is used to generate a labelled transition system.

Process algebra model $\xrightarrow{\text{SOS rules}}$

Process Algebra

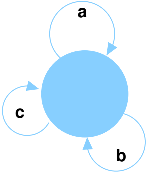
- Models consist of agents which engage in actions.



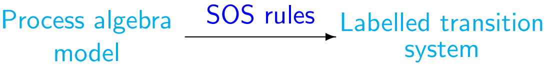
$\alpha.P$

action type or name

agent/ component



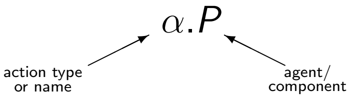
- The structured operational (interleaving) semantics of the language is used to generate a labelled transition system.



Process algebra model $\xrightarrow{\text{SOS rules}}$ Labelled transition system

Process Algebra

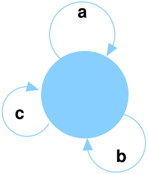
- Models consist of agents which engage in actions.



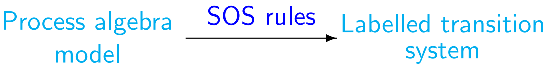
$\alpha.P$

action type
or name

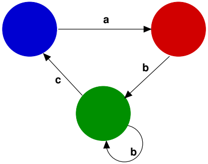
agent/
component



- The structured operational (interleaving) semantics of the language is used to generate a labelled transition system.



Process algebra model $\xrightarrow{\text{SOS rules}}$ Labelled transition system



A simple example: processors and resources

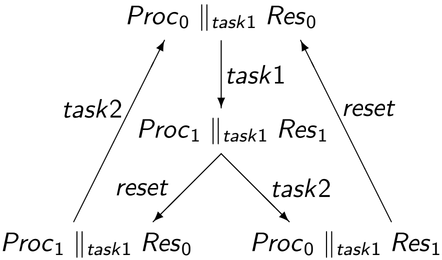
$$\begin{aligned}
Proc_0 &\stackrel{\mathit{def}}{=} task1.Proc_1\\
Proc_1 &\stackrel{\mathit{def}}{=} task2.Proc_0\\
Res_0 &\stackrel{\mathit{def}}{=} task1.Res_1\\
Res_1 &\stackrel{\mathit{def}}{=} reset.Res_0
\end{aligned}$$

$$Proc_0 \parallel_{task1} Res_0$$

A simple example: processors and resources

$$\begin{aligned}
Proc_0 &\stackrel{def}{=} task1.Proc_1 \\
Proc_1 &\stackrel{def}{=} task2.Proc_0 \\
Res_0 &\stackrel{def}{=} task1.Res_1 \\
Res_1 &\stackrel{def}{=} reset.Res_0
\end{aligned}$$

$$Proc_0 \parallel_{task1} Res_0$$



Stochastic process algebras

Process algebras where models are decorated with quantitative information used to generate a stochastic process are stochastic process algebras (SPA).

Stochastic process algebras

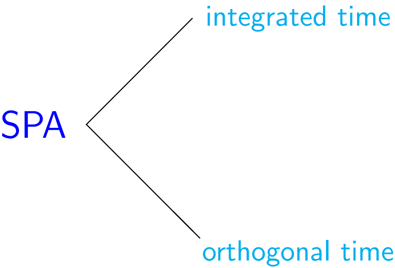
Process algebras where models are decorated with quantitative information used to generate a stochastic process are stochastic process algebras (SPA).

This extension was motivated by a desire to bring this formal and compositional approach to modelling to bear in performance analysis supporting the derivation of measures such as throughput, utlisation and response time.

SPA Languages

SPA

SPA Languages

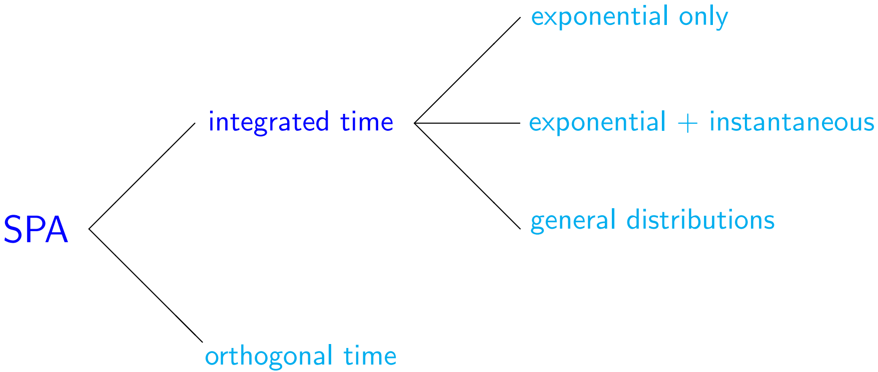


SPA

integrated time

orthogonal time

SPA Languages



SPA

integrated time

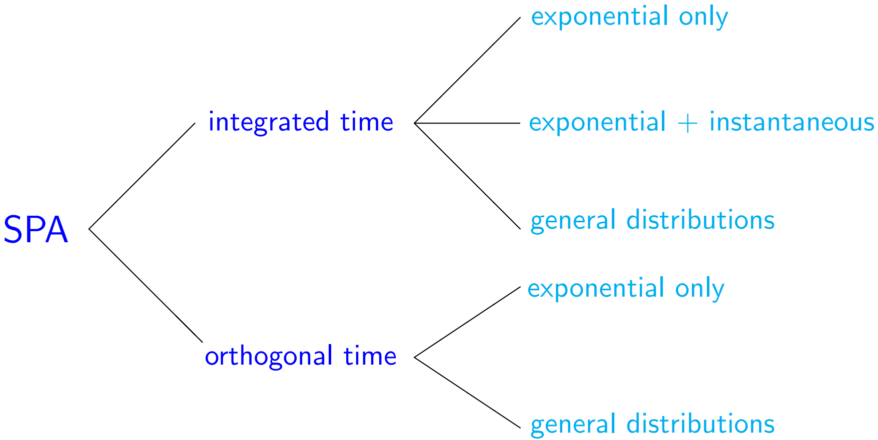
exponential only

exponential + instantaneous

general distributions

orthogonal time

SPA Languages



SPA

integrated time

exponential only

exponential + instantaneous

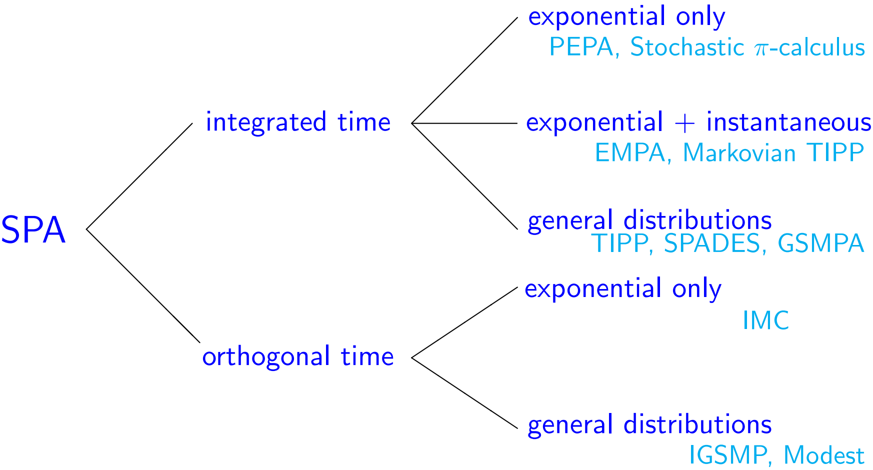
general distributions

orthogonal time

exponential only

general distributions

SPA Languages



SPA

integrated time

exponential only
PEPA, Stochastic π-calculus

exponential + instantaneous
EMPA, Markovian TIPP

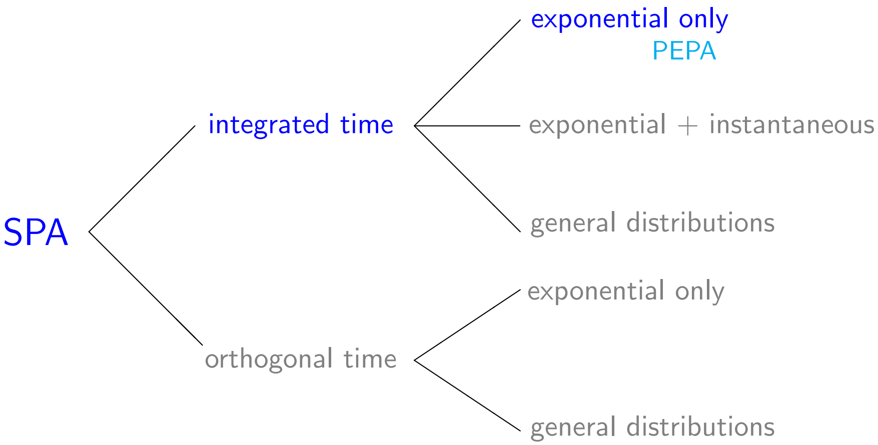
general distributions
TIPP, SPADES, GSMPA

orthogonal time

exponential only
IMC

general distributions
IGSMP, Modest

SPA Languages



SPA

integrated time

exponential only

PEPA

exponential + instantaneous

general distributions

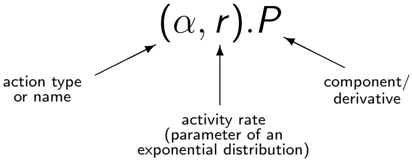
orthogonal time

exponential only

general distributions

Performance Evaluation Process Algebra

- Models are constructed from components which engage in activities.



$$(\alpha, r).P$$

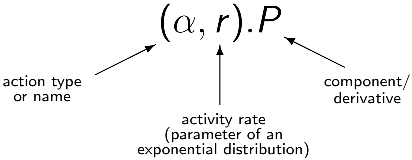
action type
or name

activity rate
(parameter of an
exponential distribution)

component/
derivative

Performance Evaluation Process Algebra

- Models are constructed from components which engage in activities.



$$(\alpha, r).P$$

action type or name

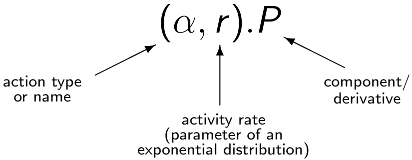
activity rate (parameter of an exponential distribution)

component/ derivative

- The language is used to generate a Continuous Time Markov Chain (CTMC) for performance modelling.

Performance Evaluation Process Algebra

- Models are constructed from components which engage in activities.



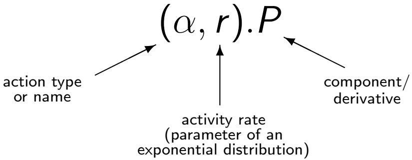
$$(\alpha, r).P$$

- The language is used to generate a Continuous Time Markov Chain (CTMC) for performance modelling.

PEPA
MODEL

Performance Evaluation Process Algebra

- Models are constructed from components which engage in activities.



$$(\alpha, r).P$$

action type or name

activity rate (parameter of an exponential distribution)

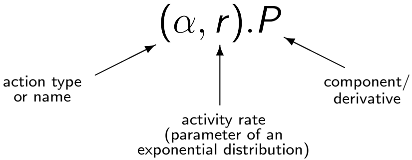
component/ derivative

- The language is used to generate a Continuous Time Markov Chain (CTMC) for performance modelling.

PEPA MODEL $\xrightarrow{\text{SOS rules}}$

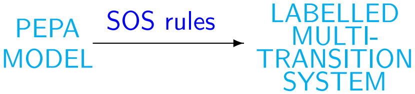
Performance Evaluation Process Algebra

- Models are constructed from components which engage in activities.



$$(\alpha, r).P$$

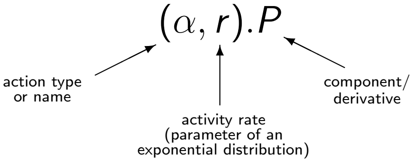
- The language is used to generate a Continuous Time Markov Chain (CTMC) for performance modelling.



PEPA MODEL $\xrightarrow{\text{SOS rules}}$ LABELLED MULTI-TRANSITION SYSTEM

Performance Evaluation Process Algebra

- Models are constructed from components which engage in activities.



$$(\alpha, r).P$$

action type or name

activity rate (parameter of an exponential distribution)

component/ derivative

- The language is used to generate a Continuous Time Markov Chain (CTMC) for performance modelling.

PEPA MODEL

SOS rules

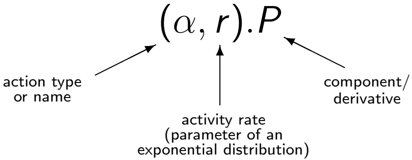
LABELLED MULTI-TRANSITION SYSTEM

state transition

diagram

Performance Evaluation Process Algebra

- Models are constructed from components which engage in activities.



$(\alpha, r).P$

action type or name

activity rate (parameter of an exponential distribution)

component/ derivative

- The language is used to generate a Continuous Time Markov Chain (CTMC) for performance modelling.

PEPA MODEL

SOS rules

LABELLED MULTI-TRANSITION SYSTEM

state transition

diagram

CTMC **Q**

Integrated analysis

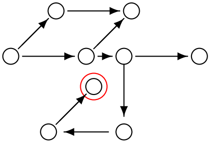
Qualitative verification can now be complemented by quantitative verification.

Integrated analysis

Qualitative verification can now be complemented by quantitative verification.

Reachability analysis

How long will it take for the system to arrive in a particular state?

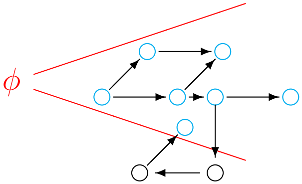


Integrated analysis

Qualitative verification can now be complemented by quantitative verification.

Model checking

Does a given property ϕ hold within the system with a given probability?

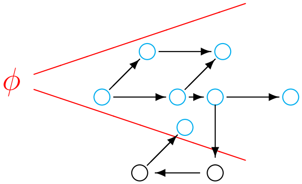


Integrated analysis

Qualitative verification can now be complemented by quantitative verification.

Model checking

For a given starting state
how long is it until
a given property ϕ holds?



Performance Evaluation Process Algebra

PEPA components perform activities either independently or in co-operation with other components.

Performance Evaluation Process Algebra

PEPA components perform activities either independently or in co-operation with other components.

$(\alpha, f).P$	Prefix
$P_1 + P_2$	Choice
$P_1 \underset{L}{\bowtie} P_2$	Co-operation
P/L	Hiding
C	Constant

Performance Evaluation Process Algebra

PEPA components perform activities either independently or in co-operation with other components.

$(\alpha, f).P$	Prefix
$P_1 + P_2$	Choice
$P_1 \underset{L}{\bowtie} P_2$	Co-operation
P/L	Hiding
C	Constant

Performance Evaluation Process Algebra

PEPA components perform activities either independently or in co-operation with other components.

$(\alpha, f).P$	Prefix
$P_1 + P_2$	Choice
$P_1 \underset{L}{\bowtie} P_2$	Co-operation
P/L	Hiding
C	Constant

Performance Evaluation Process Algebra

PEPA components perform activities either independently or in co-operation with other components.

$(\alpha, f).P$	Prefix
$P_1 + P_2$	Choice
$P_1 \underset{L}{\bowtie} P_2$	Co-operation
P/L	Hiding
C	Constant

Performance Evaluation Process Algebra

PEPA components perform activities either independently or in co-operation with other components.

$(\alpha, f).P$	Prefix
$P_1 + P_2$	Choice
$P_1 \underset{L}{\bowtie} P_2$	Co-operation
P/L	Hiding
C	Constant

Performance Evaluation Process Algebra

PEPA components perform activities either independently or in co-operation with other components.

$(\alpha, f).P$	Prefix
$P_1 + P_2$	Choice
$P_1 \underset{L}{\bowtie} P_2$	Co-operation
P/L	Hiding
C	Constant

Performance Evaluation Process Algebra

PEPA components perform activities either independently or in co-operation with other components.

$(\alpha, f).P$	Prefix
$P_1 + P_2$	Choice
$P_1 \underset{L}{\bowtie} P_2$	Co-operation
P/L	Hiding
C	Constant

$P_1 \parallel P_2$ is a derived form for $P_1 \underset{\emptyset}{\bowtie} P_2$.

Performance Evaluation Process Algebra

PEPA components perform activities either independently or in co-operation with other components.

$(\alpha, f).P$	Prefix
$P_1 + P_2$	Choice
$P_1 \underset{L}{\bowtie} P_2$	Co-operation
P/L	Hiding
C	Constant

$P_1 \parallel P_2$ is a derived form for $P_1 \underset{\emptyset}{\bowtie} P_2$.

When working with large numbers of entities, we write $P[n]$ to denote an array of n copies of P executing in parallel.

Performance Evaluation Process Algebra

PEPA components perform activities either independently or in co-operation with other components.

$(\alpha, f).P$	Prefix
$P_1 + P_2$	Choice
$P_1 \underset{L}{\bowtie} P_2$	Co-operation
P/L	Hiding
C	Constant

$P_1 \parallel P_2$ is a derived form for $P_1 \underset{\emptyset}{\bowtie} P_2$.

When working with large numbers of entities, we write $P[n]$ to denote an array of n copies of P executing in parallel.

$$P[5] \equiv (P \parallel P \parallel P \parallel P \parallel P)$$

Rates of interaction: bounded capacity

Stochastic process algebras differ in how they define the rate of synchronised actions. In PEPA cooperation between components gives rise to shared actions, the rate of which are governed by the assumption of bounded capacity.

Rates of interaction: bounded capacity

Stochastic process algebras differ in how they define the rate of synchronised actions. In PEPA cooperation between components gives rise to shared actions, the rate of which are governed by the assumption of bounded capacity.

Bounded capacity

No component can be made to carry out an action in cooperation faster than its own defined rate for the action.

Rates of interaction: bounded capacity

Stochastic process algebras differ in how they define the rate of synchronised actions. In PEPA cooperation between components gives rise to shared actions, the rate of which are governed by the assumption of bounded capacity.

Bounded capacity

No component can be made to carry out an action in cooperation faster than its own defined rate for the action.

Thus shared actions proceed at the minimum of the rates in the participating components.

Rates of interaction: bounded capacity

Stochastic process algebras differ in how they define the rate of synchronised actions. In PEPA cooperation between components gives rise to shared actions, the rate of which are governed by the assumption of bounded capacity.

Bounded capacity

No component can be made to carry out an action in cooperation faster than its own defined rate for the action.

Thus shared actions proceed at the minimum of the rates in the participating components.

In contrast independent actions do not constrain each other and if there are multiple copies of a action enabled in independent concurrent components their rates are summed.

A simple example: processors and resources

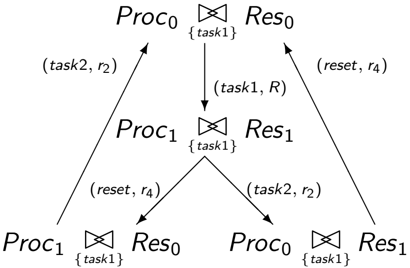
$$\begin{aligned}
Proc_0 &\overset{def}{=} (task1, r_1).Proc_1 \\
Proc_1 &\overset{def}{=} (task2, r_2).Proc_0 \\
Res_0 &\overset{def}{=} (task1, r_3).Res_1 \\
Res_1 &\overset{def}{=} (reset, r_4).Res_0
\end{aligned}$$

$$Proc_0 \underset{\{task1\}}{\bowtie} Res_0$$

A simple example: processors and resources

$$
\begin{aligned}
Proc_0 &\overset{def}{=} (task1, r_1).Proc_1\\
Proc_1 &\overset{def}{=} (task2, r_2).Proc_0\\
Res_0 &\overset{def}{=} (task1, r_3).Res_1\\
Res_1 &\overset{def}{=} (reset, r_4).Res_0
\end{aligned}
$$

$$Proc_0 \underset{\{task1\}}{\bowtie} Res_0$$



$Proc_0 \underset{\{task1\}}{\bowtie} Res_0$

$(task2, r_2)$

$(task1, R)$

$(reset, r_4)$

$Proc_1 \underset{\{task1\}}{\bowtie} Res_1$

$(reset, r_4)$

$(task2, r_2)$

$Proc_1 \underset{\{task1\}}{\bowtie} Res_0$

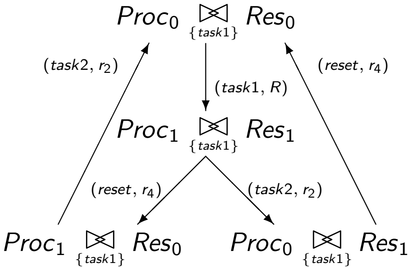
$Proc_0 \underset{\{task1\}}{\bowtie} Res_1$

$$R = \min(r_1, r_3)$$

A simple example: processors and resources

$$
\begin{aligned}
Proc_0 &\stackrel{\mathit{def}}{=} (task1, r_1).Proc_1 \\
Proc_1 &\stackrel{\mathit{def}}{=} (task2, r_2).Proc_0 \\
Res_0 &\stackrel{\mathit{def}}{=} (task1, r_3).Res_1 \\
Res_1 &\stackrel{\mathit{def}}{=} (reset, r_4).Res_0
\end{aligned}
$$

$$Proc_0 \underset{\{task1\}}{\bowtie} Res_0$$

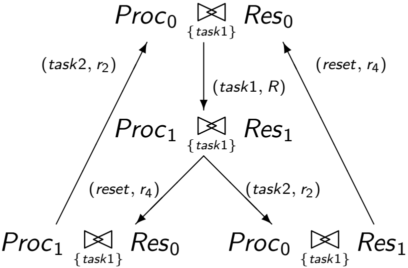


$R = \min(r_1, r_3)$

A simple example: processors and resources

$$
\begin{aligned}
Proc_0 &\stackrel{\mathit{def}}{=} (task1, r_1).Proc_1 \\
Proc_1 &\stackrel{\mathit{def}}{=} (task2, r_2).Proc_0 \\
Res_0 &\stackrel{\mathit{def}}{=} (task1, r_3).Res_1 \\
Res_1 &\stackrel{\mathit{def}}{=} (reset, r_4).Res_0
\end{aligned}
$$

$$Proc_0 \underset{\{task1\}}{\bowtie} Res_0$$



$Proc_0 \underset{\{task1\}}{\bowtie} Res_0$

$(task2, r_2)$

$(task1, R)$

$(reset, r_4)$

$Proc_1 \underset{\{task1\}}{\bowtie} Res_1$

$(reset, r_4)$

$(task2, r_2)$

$Proc_1 \underset{\{task1\}}{\bowtie} Res_0$

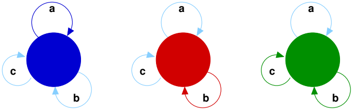
$Proc_0 \underset{\{task1\}}{\bowtie} Res_1$

$R = \min(r_1, r_3)$

$$
\mathbf{Q} = \begin{pmatrix}
-R & R & 0 & 0 \\
0 & -(r_2 + r_4) & r_4 & r_2 \\
r_2 & 0 & -r_2 & 0 \\
r_4 & 0 & 0 & -r_4
\end{pmatrix}
$$

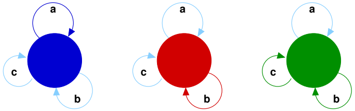
Outline

Solving discrete state models

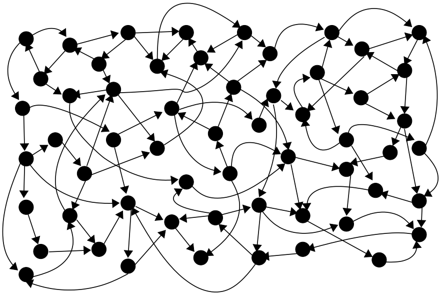


Under the SOS semantics a SPA model is mapped to a CTMC with global states determined by the local states of all the participating components.

Solving discrete state models



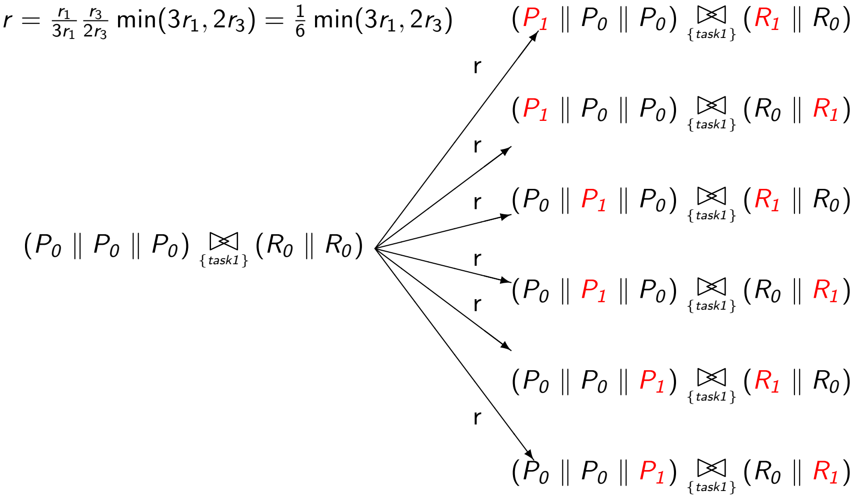
Under the SOS semantics a SPA model is mapped to a CTMC with global states determined by the local states of all the participating components.



Modelling at the level of individuals

$$\begin{tikzcd}
& (P_1 \parallel P_0 \parallel P_0) \underset{\{task1\}}{\bowtie} (R_1 \parallel R_0) \\
& (P_1 \parallel P_0 \parallel P_0) \underset{\{task1\}}{\bowtie} (R_0 \parallel R_1) \\
(P_0 \parallel P_0 \parallel P_0) \underset{\{task1\}}{\bowtie} (R_0 \parallel R_0) \arrow[uur, "r"] \arrow[ur, "r"] \arrow[r, "r"] \arrow[dr, "r"] \arrow[ddr, "r"] \arrow[dddr, "r"] & (P_0 \parallel P_1 \parallel P_0) \underset{\{task1\}}{\bowtie} (R_1 \parallel R_0) \\
& (P_0 \parallel P_1 \parallel P_0) \underset{\{task1\}}{\bowtie} (R_0 \parallel R_1) \\
& (P_0 \parallel P_0 \parallel P_1) \underset{\{task1\}}{\bowtie} (R_1 \parallel R_0) \\
& (P_0 \parallel P_0 \parallel P_1) \underset{\{task1\}}{\bowtie} (R_0 \parallel R_1)
\end{tikzcd}$$

Modelling at the level of individuals



$$r = \frac{r_1}{3r_1}\frac{r_3}{2r_3}\min(3r_1, 2r_3) = \frac{1}{6}\min(3r_1, 2r_3)$$

$$(P_0 \parallel P_0 \parallel P_0) \underset{\{task1\}}{\bowtie} (R_0 \parallel R_0)$$

$$(P_1 \parallel P_0 \parallel P_0) \underset{\{task1\}}{\bowtie} (R_1 \parallel R_0)$$

$$(P_1 \parallel P_0 \parallel P_0) \underset{\{task1\}}{\bowtie} (R_0 \parallel R_1)$$

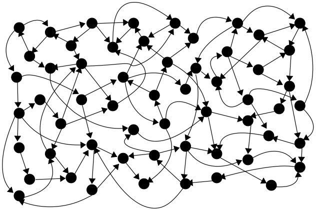
$$(P_0 \parallel P_1 \parallel P_0) \underset{\{task1\}}{\bowtie} (R_1 \parallel R_0)$$

$$(P_0 \parallel P_1 \parallel P_0) \underset{\{task1\}}{\bowtie} (R_0 \parallel R_1)$$

$$(P_0 \parallel P_0 \parallel P_1) \underset{\{task1\}}{\bowtie} (R_1 \parallel R_0)$$

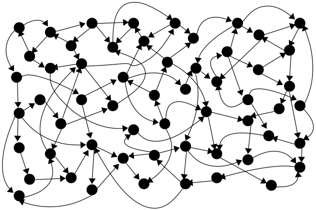
$$(P_0 \parallel P_0 \parallel P_1) \underset{\{task1\}}{\bowtie} (R_0 \parallel R_1)$$

Solving discrete state models



When the size of the state space is not too large they are amenable to numerical solution (linear algebra) to determine a steady state or transient probability distribution.

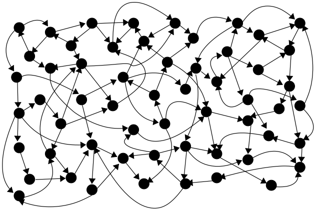
Solving discrete state models



When the size of the state space is not too large they are amenable to numerical solution (linear algebra) to determine a steady state or transient probability distribution.

$$Q = \begin{pmatrix} q_{1,1} & q_{1,2} & \cdots & q_{1,N} \\ q_{2,1} & q_{2,2} & \cdots & q_{2,N} \\ \vdots & \vdots & & \vdots \\ q_{N,1} & q_{N,2} & \cdots & q_{N,N} \end{pmatrix}$$

Solving discrete state models



When the size of the state space is not too large they are amenable to numerical solution (linear algebra) to determine a steady state or transient probability distribution.

$$Q = \begin{pmatrix} q_{1,1} & q_{1,2} & \cdots & q_{1,N} \\ q_{2,1} & q_{2,2} & \cdots & q_{2,N} \\ \vdots & \vdots & & \vdots \\ q_{N,1} & q_{N,2} & \cdots & q_{N,N} \end{pmatrix}$$

$$\pi(t) = (\pi_1(t), \pi_2(t), \ldots, \pi_N(t))$$

State space explosion

As the number of components, or the complexity of behaviour within components, grows the state space may become so large that it is infeasible to solve the underlying CTMC.

$$
\begin{array}{rcl}
Proc_0 & \stackrel{def}{=} & (task1, r_1).Proc_1 \\
Proc_1 & \stackrel{def}{=} & (task2, r_2).Proc_0 \\
Res_0 & \stackrel{def}{=} & (task1, r_3).Res_1 \\
Res_1 & \stackrel{def}{=} & (reset, r_4).Res_0
\end{array}
$$

$$Proc_0[N_P] \underset{\{task1\}}{\bowtie} Res_0[N_R]$$

State space explosion

As the number of components, or the complexity of behaviour within components, grows the state space may become so large that it is infeasible to solve the underlying CTMC.

$$
\begin{aligned}
Proc_0 &\overset{def}{=} (task1, r_1).Proc_1 \\
Proc_1 &\overset{def}{=} (task2, r_2).Proc_0 \\
Res_0 &\overset{def}{=} (task1, r_3).Res_1 \\
Res_1 &\overset{def}{=} (reset, r_4).Res_0
\end{aligned}
$$

$$Proc_0[N_P] \underset{\{task1\}}{\bowtie} Res_0[N_R]$$

CTMC interpretation

Processors (N_P)	Resources (N_R)	States ($2^{N_P+N_R}$)
1	1	4
2	1	8
2	2	16
3	2	32
3	3	64
4	3	128
4	4	256
5	4	512
5	5	1024
6	5	2048
6	6	4096
7	6	8192
7	7	16384
8	7	32768
8	8	65536
9	8	131072
9	9	262144
10	9	524288
10	10	1048576

Achieving aggregation

- If we sacrifice looking at the identity of each component we can often achieve substantial state space reduction by aggregation.

Achieving aggregation

- If we sacrifice looking at the identity of each component we can often achieve substantial state space reduction by aggregation.
- This is supported by a shift in how we view the state of a model, based on a counting abstraction.

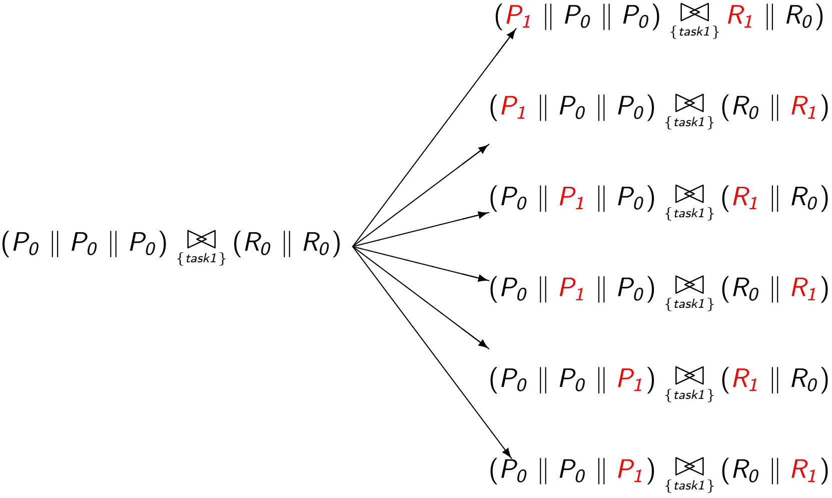
Achieving aggregation

- If we sacrifice looking at the identity of each component we can often achieve substantial state space reduction by aggregation.
- This is supported by a shift in how we view the state of a model, based on a counting abstraction.
- The syntactic nature of PEPA (and other SPAs) makes models easily understood by humans, but not so convenient for computers to directly apply these tools and approaches.

Achieving aggregation

- If we sacrifice looking at the identity of each component we can often achieve substantial state space reduction by aggregation.
- This is supported by a shift in how we view the state of a model, based on a counting abstraction.
- The syntactic nature of PEPA (and other SPAs) makes models easily understood by humans, but not so convenient for computers to directly apply these tools and approaches.
- By shifting to a *numerical state representation* we can more readily exploit results such as aggregation and access to alternative mathematical interpretations (i.e. fluid approximation).

Counting abstraction to generate the *Lumped* CTMC



$$(P_0 \parallel P_0 \parallel P_0) \underset{\{task1\}}{\bowtie} (R_0 \parallel R_0)$$

$$(P_1 \parallel P_0 \parallel P_0) \underset{\{task1\}}{\bowtie} R_1 \parallel R_0)$$

$$(P_1 \parallel P_0 \parallel P_0) \underset{\{task1\}}{\bowtie} (R_0 \parallel R_1)$$

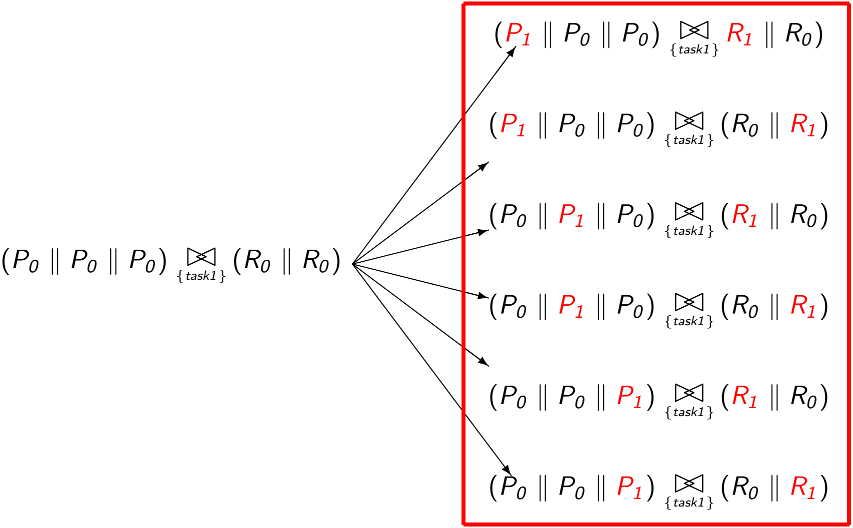
$$(P_0 \parallel P_1 \parallel P_0) \underset{\{task1\}}{\bowtie} (R_1 \parallel R_0)$$

$$(P_0 \parallel P_1 \parallel P_0) \underset{\{task1\}}{\bowtie} (R_0 \parallel R_1)$$

$$(P_0 \parallel P_0 \parallel P_1) \underset{\{task1\}}{\bowtie} (R_1 \parallel R_0)$$

$$(P_0 \parallel P_0 \parallel P_1) \underset{\{task1\}}{\bowtie} (R_0 \parallel R_1)$$

Counting abstraction to generate the *Lumped* CTMC



$(P_0 \parallel P_0 \parallel P_0) \underset{\{task1\}}{\bowtie} (R_0 \parallel R_0)$

$(P_1 \parallel P_0 \parallel P_0) \underset{\{task1\}}{\bowtie} R_1 \parallel R_0)$

$(P_1 \parallel P_0 \parallel P_0) \underset{\{task1\}}{\bowtie} (R_0 \parallel R_1)$

$(P_0 \parallel P_1 \parallel P_0) \underset{\{task1\}}{\bowtie} (R_1 \parallel R_0)$

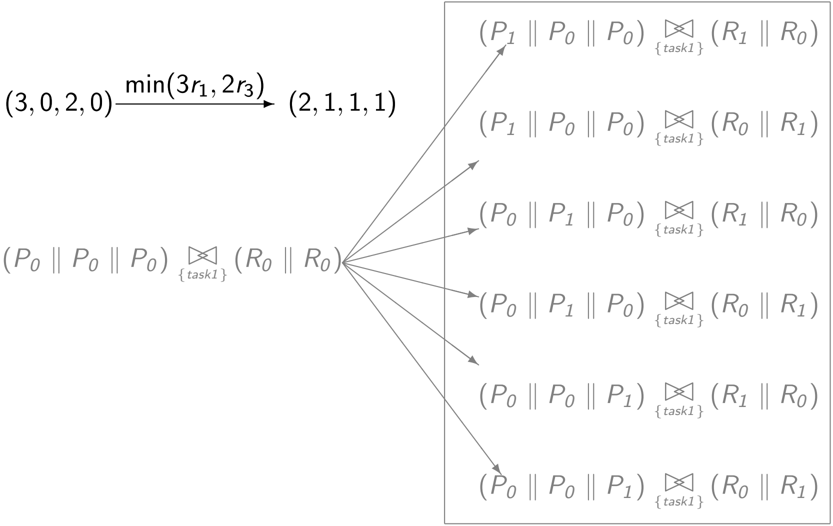
$(P_0 \parallel P_1 \parallel P_0) \underset{\{task1\}}{\bowtie} (R_0 \parallel R_1)$

$(P_0 \parallel P_0 \parallel P_1) \underset{\{task1\}}{\bowtie} (R_1 \parallel R_0)$

$(P_0 \parallel P_0 \parallel P_1) \underset{\{task1\}}{\bowtie} (R_0 \parallel R_1)$

Counting abstraction to generate the *Lumped* CTMC

$$(3,0,2,0) \xrightarrow{\min(3r_1, 2r_3)} (2,1,1,1)$$



$$(P_0 \parallel P_0 \parallel P_0) \underset{\{task1\}}{\bowtie} (R_0 \parallel R_0)$$

$$(P_1 \parallel P_0 \parallel P_0) \underset{\{task1\}}{\bowtie} (R_1 \parallel R_0)$$

$$(P_1 \parallel P_0 \parallel P_0) \underset{\{task1\}}{\bowtie} (R_0 \parallel R_1)$$

$$(P_0 \parallel P_1 \parallel P_0) \underset{\{task1\}}{\bowtie} (R_1 \parallel R_0)$$

$$(P_0 \parallel P_1 \parallel P_0) \underset{\{task1\}}{\bowtie} (R_0 \parallel R_1)$$

$$(P_0 \parallel P_0 \parallel P_1) \underset{\{task1\}}{\bowtie} (R_1 \parallel R_0)$$

$$(P_0 \parallel P_0 \parallel P_1) \underset{\{task1\}}{\bowtie} (R_0 \parallel R_1)$$

Using this result in practice

There are well-known algorithms such as Paige and Tarjan for finding the maximal partition of a graph according to some equivalence.

Using this result in practice

There are well-known algorithms such as Paige and Tarjan for finding the maximal partition of a graph according to some equivalence.

However in practice we would much rather construct the aggregated state space directly.

Using this result in practice

There are well-known algorithms such as Paige and Tarjan for finding the maximal partition of a graph according to some equivalence.

However in practice we would much rather construct the aggregated state space directly.

The first approach to this used canonical forms but still worked syntactically to identify states. [Gilmore, Hillston and Ribaudo, IEEE TSE 2001].

Using this result in practice

There are well-known algorithms such as Paige and Tarjan for finding the maximal partition of a graph according to some equivalence.

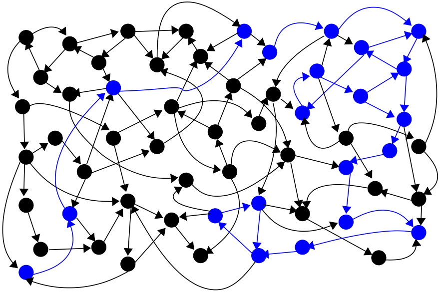
However in practice we would much rather construct the aggregated state space directly.

The first approach to this used canonical forms but still worked syntactically to identify states. [Gilmore, Hillston and Ribaudo, IEEE TSE 2001].

A more recent approaches shift to a counting abstraction and a numerical representation of states and transitions.

Solving discrete state models

Even with aggregation models may become too large to solve the underlying CTMC. As an alternative they may be studied using stochastic simulation. Each run generates a single trajectory through the state space. Many runs are needed in order to obtain average behaviours.



Discrete Event Simulation

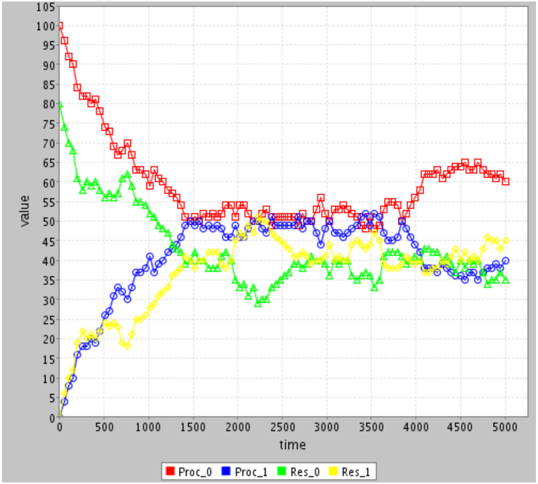
The numerical solution of the CTMC is seen as being the exact result.

Discrete Event Simulation

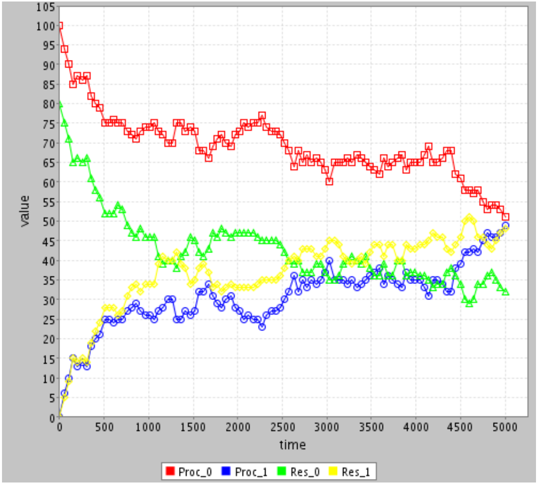
The numerical solution of the CTMC is seen as being the exact result.

Analysing a model via discrete event simulation can also produce useful results, often with some measure of confidence in the results. This is achieved by repeatedly taking random walks through the state space and observing the results, which can be very computationally intensive.

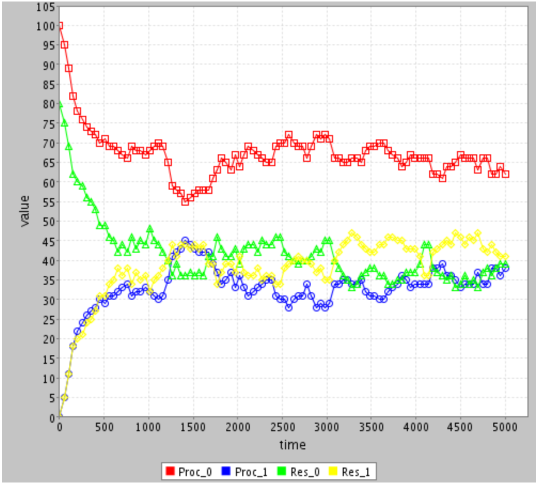
100 processors and 80 resources (simulation run A)



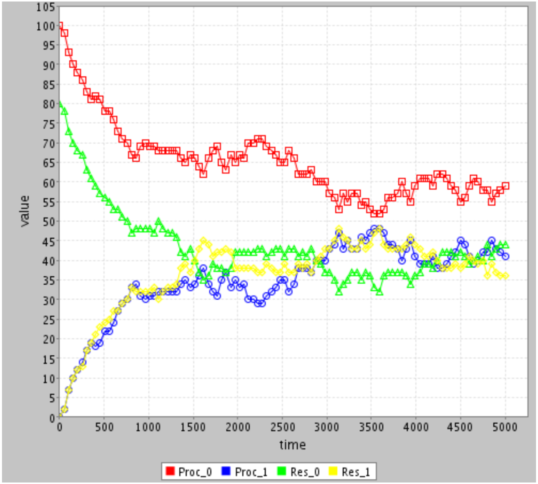
100 processors and 80 resources (simulation run B)



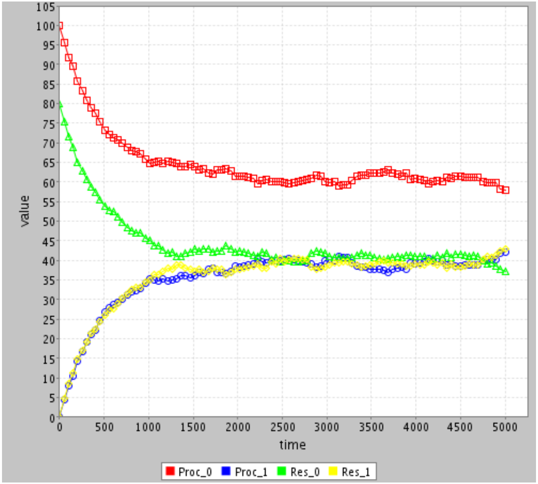
100 processors and 80 resources (simulation run C)



100 processors and 80 resources (simulation run D)



100 processors and 80 resources (average of 10 runs)



Collective dynamics

For some SPA models we can make considerable gains in efficiency when solving the model if we take a collective dynamics view of the system.

Collective dynamics

For some SPA models we can make considerable gains in efficiency when solving the model if we take a collective dynamics view of the system.

Collective dynamics considers the behaviour of populations of similar entities which can interactive with each other in seemingly simple ways to produce phenomena at the population level.

Collective dynamics

For some SPA models we can make considerable gains in efficiency when solving the model if we take a collective dynamics view of the system.

Collective dynamics considers the behaviour of populations of similar entities which can interactive with each other in seemingly simple ways to produce phenomena at the population level.

In this case we lose the identity of components and even individuality, but for many models this is an approximation we are willing to make for the efficiency, or even tractability, of the models.

Collective Behaviour

In the natural world there are many instances of collective behaviour and its consequences:

Collective Behaviour

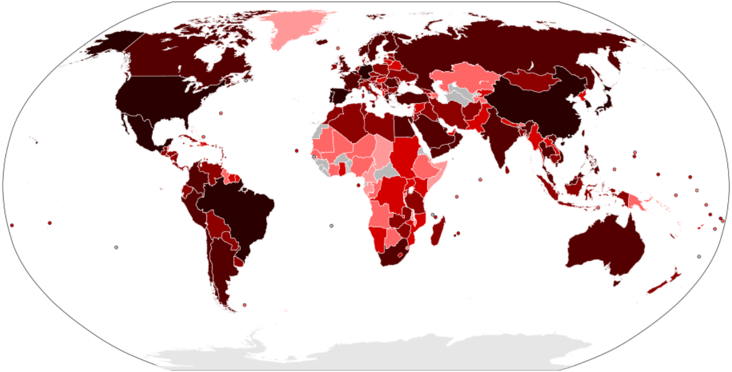
In the natural world there are many instances of collective behaviour and its consequences:

Collective Behaviour

In the natural world there are many instances of collective behaviour and its consequences:

Collective Dynamics

This is also true in the man-made and engineered world:



Spread of H1N1 virus in 2009

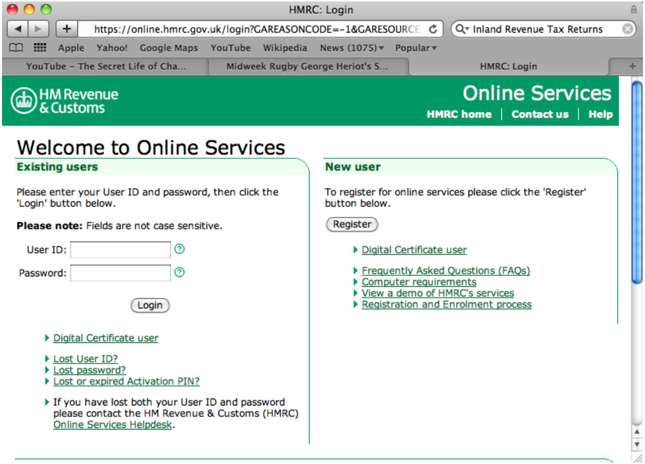
Collective Dynamics

This is also true in the man-made and engineered world:

Love Parade, Germany 2006

Collective Dynamics

This is also true in the man-made and engineered world:



Welcome to Online Services

Existing users

Please enter your User ID and password, then click the 'Login' button below.

Please note: Fields are not case sensitive.

User ID:

Password:

Login

- Digital Certificate user
- Lost User ID?
- Lost password?
- Lost or expired Activation PIN?
- If you have lost both your User ID and password please contact the HM Revenue & Customs (HMRC) Online Services Helpdesk.

New user

To register for online services please click the 'Register' button below.

Register

- Digital Certificate user
- Frequently Asked Questions (FAQs)
- Computer requirements
- View a demo of HMRC's services
- Registration and Enrolment process

Self assessment tax returns 31st January each year

Process Algebra and Collective Dynamics

Some large process algebra models can be considered to exhibit collective dynamics

Process Algebra and Collective Dynamics

Some large process algebra models can be considered to exhibit collective dynamics

- Each component type captures the behaviour of one type of individual;

Process Algebra and Collective Dynamics

Some large process algebra models can be considered to exhibit collective dynamics

- Each component type captures the behaviour of one type of individual;
- The compositional structure of the model makes explicit interaction between component types;

Process Algebra and Collective Dynamics

Some large process algebra models can be considered to exhibit collective dynamics

- Each component type captures the behaviour of one type of individual;
- The compositional structure of the model makes explicit interaction between component types;
- When there are many instances of the individual component types these may be regarded as a population;

Process Algebra and Collective Dynamics

Some large process algebra models can be considered to exhibit collective dynamics

- Each component type captures the behaviour of one type of individual;
- The compositional structure of the model makes explicit interaction between component types;
- When there are many instances of the individual component types these may be regarded as a population;
- Through the interactions of these populations group or complex behaviours may emerge at the population level.

Population statistics: emergent behaviour

A shift in perspective allows us to model the interactions between individual components but then only consider the system as a whole as an interaction of populations.

Population statistics: emergent behaviour

A shift in perspective allows us to model the interactions between individual components but then only consider the system as a whole as an interaction of populations.

This allows us to model much larger systems than previously possible but in making the shift we are no longer able to collect any information about individuals in the system.

Population statistics: emergent behaviour

A shift in perspective allows us to model the interactions between individual components but then only consider the system as a whole as an interaction of populations.

This allows us to model much larger systems than previously possible but in making the shift we are no longer able to collect any information about individuals in the system.

To characterise the behaviour of a population we count the number of individuals within the population that are exhibiting certain behaviours rather than tracking individuals directly.

Population statistics: emergent behaviour

A shift in perspective allows us to model the interactions between individual components but then only consider the system as a whole as an interaction of populations.

This allows us to model much larger systems than previously possible but in making the shift we are no longer able to collect any information about individuals in the system.

To characterise the behaviour of a population we count the number of individuals within the population that are exhibiting certain behaviours rather than tracking individuals directly.

Furthermore we make a continuous approximation of how the counts vary over time.

Performance as an emergent behaviour

In this framework we must think about the performance of a system from the collective point of view. Service providers often want to do this in any case. For example making contracts in terms of service level agreements.

Performance as an emergent behaviour

In this framework we must think about the performance of a system from the collective point of view. Service providers often want to do this in any case. For example making contracts in terms of service level agreements.

Example Service Level Agreement

90% of requests receive a response within 3 seconds.

Performance as an emergent behaviour

In this framework we must think about the performance of a system from the collective point of view. Service providers often want to do this in any case. For example making contracts in terms of service level agreements.

Example Service Level Agreement

90% of requests receive a response within 3 seconds.

Qualitative Service Level Agreement

Less than 1% of the responses received within 3 seconds will read "System is overloaded, try again later".

Outline

Continuous Approximation

Use continuous state variables to approximate the discrete state space.

Continuous Approximation

Use continuous state variables to approximate the discrete state space.

Continuous Approximation

Use continuous state variables to approximate the discrete state space.

Continuous Approximation

Use continuous state variables to approximate the discrete state space.

Continuous Approximation

Use continuous state variables to approximate the discrete state space.

Continuous Approximation

Use continuous state variables to approximate the discrete state space.

Continuous Approximation

Use continuous state variables to approximate the discrete state space.

Continuous Approximation

Use continuous state variables to approximate the discrete state space.

Continuous Approximation

Use continuous state variables to approximate the discrete state space.

Continuous Approximation

Use continuous state variables to approximate the discrete state space.

Continuous Approximation

Use continuous state variables to approximate the discrete state space.

Continuous Approximation

Use continuous state variables to approximate the discrete state space.

Use ordinary differential equations to represent the evolution of those variables over time.

New mathematical structures: differential equations

1. Use a counting abstraction rather than the CTMC complete state space.

New mathematical structures: differential equations

1. Use a counting abstraction rather than the CTMC complete state space.
2. Assume that these state variables are subject to continuous rather than discrete change.

New mathematical structures: differential equations

1. Use a counting abstraction rather than the CTMC complete state space.
2. Assume that these state variables are subject to continuous rather than discrete change.
3. No longer aim to calculate the probability distribution over the entire state space of the model.

New mathematical structures: differential equations

1. Use a counting abstraction rather than the CTMC complete state space.
2. Assume that these state variables are subject to continuous rather than discrete change.
3. No longer aim to calculate the probability distribution over the entire state space of the model.
4. Instead the trajectory of the ODEs estimates the expected behaviour of the CTMC.

Models suitable for counting abstraction

- In the PEPA language multiple instances of components are represented explicitly — we write $P[n]$ to denote an array of n copies of P executing in parallel.

$$P[5] \equiv (P \parallel P \parallel P \parallel P \parallel P)$$

Models suitable for counting abstraction

- In the PEPA language multiple instances of components are represented explicitly — we write $P[n]$ to denote an array of n copies of P executing in parallel.

$$P[5] \equiv (P \parallel P \parallel P \parallel P \parallel P)$$

- The impact of an action on a counting variable is

Models suitable for counting abstraction

- In the PEPA language multiple instances of components are represented explicitly — we write $P[n]$ to denote an array of n copies of P executing in parallel.

$$P[5] \equiv (P \parallel P \parallel P \parallel P \parallel P)$$

- The impact of an action on a counting variable is
  - decrease by 1 if the component participates in the action

Models suitable for counting abstraction

- In the PEPA language multiple instances of components are represented explicitly — we write $P[n]$ to denote an array of n copies of P executing in parallel.

$$P[5] \equiv (P \parallel P \parallel P \parallel P \parallel P)$$

- The impact of an action on a counting variable is
  - decrease by 1 if the component participates in the action
  - increase by 1 if the component is the result of the action

Models suitable for counting abstraction

- In the PEPA language multiple instances of components are represented explicitly — we write $P[n]$ to denote an array of n copies of P executing in parallel.

$$P[5] \equiv (P \parallel P \parallel P \parallel P \parallel P)$$

- The impact of an action on a counting variable is
  - decrease by 1 if the component participates in the action
  - increase by 1 if the component is the result of the action
  - zero if the component is not involved in the action.

Simple example revisited

$$
\begin{aligned}
Proc_0 &\stackrel{\mathit{def}}{=} (task1, r_1).Proc_1 \\
Proc_1 &\stackrel{\mathit{def}}{=} (task2, r_2).Proc_0 \\
Res_0 &\stackrel{\mathit{def}}{=} (task1, r_3).Res_1 \\
Res_1 &\stackrel{\mathit{def}}{=} (reset, r_4).Res_0
\end{aligned}
$$

$$Proc_0[N_P] \underset{\{task1\}}{\bowtie} Res_0[N_R]$$

Simple example revisited

$$
\begin{aligned}
Proc_0 &\overset{def}{=} (task1, r_1).Proc_1 \\
Proc_1 &\overset{def}{=} (task2, r_2).Proc_0 \\
Res_0 &\overset{def}{=} (task1, r_3).Res_1 \\
Res_1 &\overset{def}{=} (reset, r_4).Res_0
\end{aligned}
$$

$$Proc_0[N_P] \underset{\{task1\}}{\bowtie} Res_0[N_R]$$

CTMC interpretation

Processors (N_P)	Resources (N_R)	States ($2^{N_P+N_R}$)
1	1	4
2	1	8
2	2	16
3	2	32
3	3	64
4	3	128
4	4	256
5	4	512
5	5	1024
6	5	2048
6	6	4096
7	6	8192
7	7	16384
8	7	32768
8	8	65536
9	8	131072
9	9	262144
10	9	524288
10	10	1048576

Simple example revisited

$$\begin{aligned}
Proc_0 &\overset{def}{=} (task1, r_1).Proc_1 \\
Proc_1 &\overset{def}{=} (task2, r_2).Proc_0 \\
Res_0 &\overset{def}{=} (task1, r_3).Res_1 \\
Res_1 &\overset{def}{=} (reset, r_4).Res_0
\end{aligned}$$

$$Proc_0[N_P] \underset{\{task1\}}{\bowtie} Res_0[N_R]$$

- $task1$ decreases $Proc_0$ and Res_0
- $task1$ increases $Proc_1$ and Res_1
- $task2$ decreases $Proc_1$
- $task2$ increases $Proc_0$
- $reset$ decreases Res_1
- $reset$ increases Res_0

Simple example revisited

$$\begin{array}{rcl} Proc_0 & \stackrel{def}{=} & (task1, r_1).Proc_1 \\ Proc_1 & \stackrel{def}{=} & (task2, r_2).Proc_0 \\ Res_0 & \stackrel{def}{=} & (task1, r_3).Res_1 \\ Res_1 & \stackrel{def}{=} & (reset, r_4).Res_0 \end{array}$$

$$Proc_0[N_P] \underset{\{task1\}}{\bowtie} Res_0[N_R]$$

$$\frac{\mathrm{d}x_1}{\mathrm{d}t} = -\min(r_1\, x_1, r_3\, x_3) + r_2\, x_2$$

$$x_1 = \text{no. of } Proc_1$$

- *task*1 decreases $Proc_0$
- *task*1 is performed by $Proc_0$ and Res_0
- *task*2 increases $Proc_0$
- *task*2 is performed by $Proc_1$

Simple example revisited

$$
\begin{aligned}
Proc_0 &\overset{def}{=} (task1, r_1).Proc_1 \\
Proc_1 &\overset{def}{=} (task2, r_2).Proc_0 \\
Res_0 &\overset{def}{=} (task1, r_3).Res_1 \\
Res_1 &\overset{def}{=} (reset, r_4).Res_0
\end{aligned}
$$

$$Proc_0[N_P] \underset{\{task1\}}{\bowtie} Res_0[N_R]$$

ODE interpretation

$$\frac{dx_1}{dt} = -\min(r_1\,x_1, r_3\,x_3) + r_2\,x_2$$

$x_1 = \text{no. of } Proc_1$

$$\frac{dx_2}{dt} = \min(r_1\,x_1, r_3\,x_3) - r_2\,x_2$$

$x_2 = \text{no. of } Proc_2$

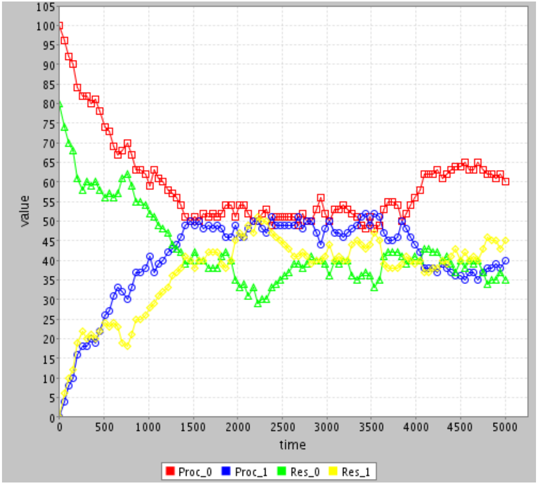
$$\frac{dx_3}{dt} = -\min(r_1\,x_1, r_3\,x_3) + r_4\,x_4$$

$x_3 = \text{no. of } Res_0$

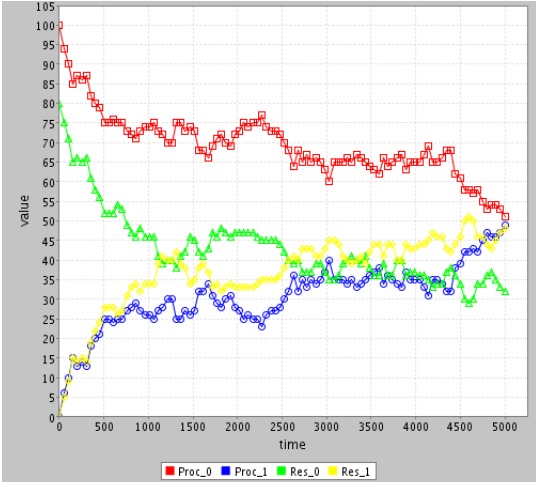
$$\frac{dx_4}{dt} = \min(r_1\,x_1, r_3\,x_3) - r_4\,x_4$$

$x_4 = \text{no. of } Res_1$

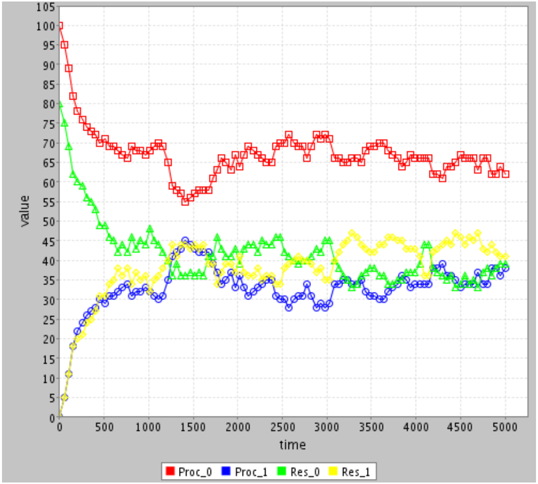
100 processors and 80 resources (simulation run A)



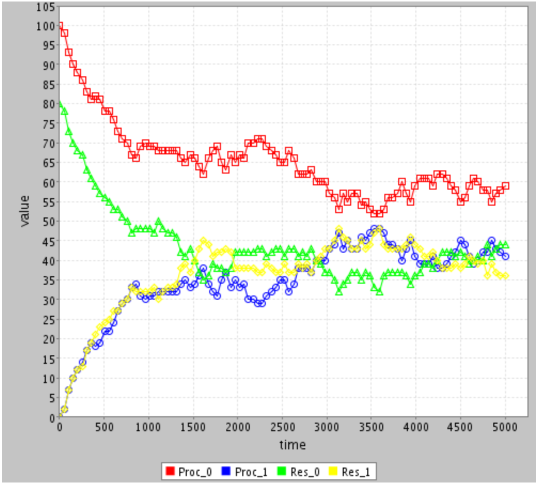
100 processors and 80 resources (simulation run B)



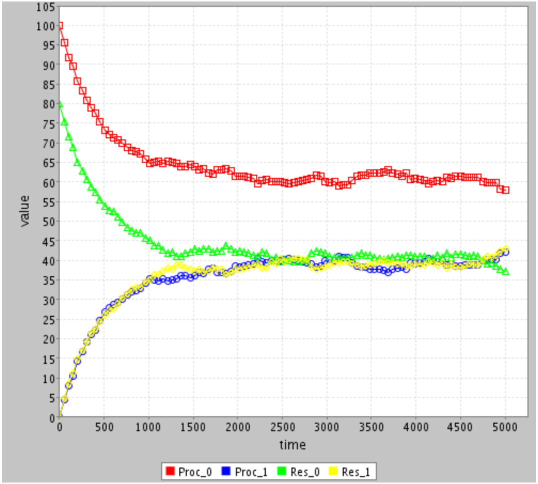
100 processors and 80 resources (simulation run C)



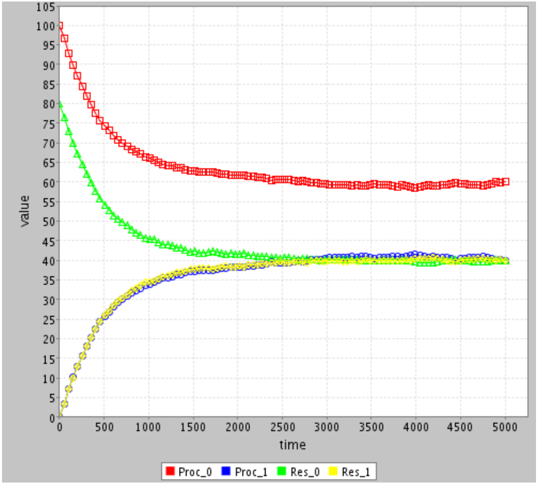
100 processors and 80 resources (simulation run D)



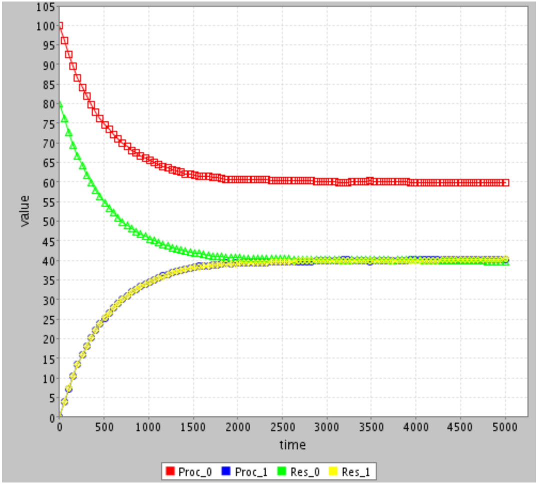
100 processors and 80 resources (average of 10 runs)



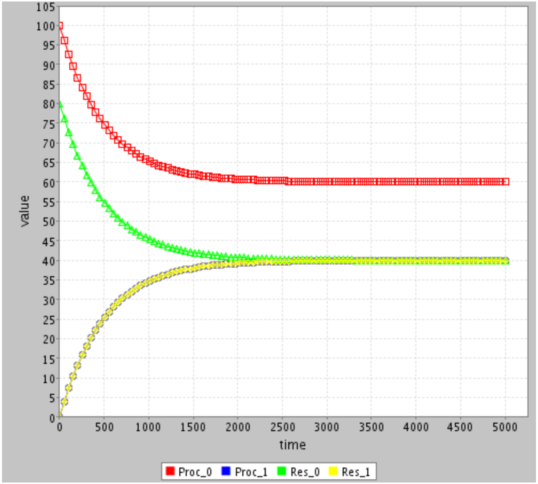
100 Processors and 80 resources (average of 100 runs)



100 processors and 80 resources (average of 1000 runs)



100 processors and 80 resources (ODE solution)



Outline

Case study: A Virtual University

Location, Time, and Size

Case study: e-University Course Selection

The e-University was one of the case studies considered in the recently completed SENSORIA project, an EU project focussed on service-oriented computing.

Case study: e-University Course Selection

The e-University was one of the case studies considered in the recently completed SENSORIA project, an EU project focussed on service-oriented computing.

In the Course Selection scenario students obtain information about the courses available at their education establishment and may enrol in those for which specific requirements are satisfied.

Case study: e-University Course Selection

The e-University was one of the case studies considered in the recently completed SENSORIA project, an EU project focussed on service-oriented computing.

In the Course Selection scenario students obtain information about the courses available at their education establishment and may enrol in those for which specific requirements are satisfied.

The model will not consider other services which may be deployed in an actual application (e.g. authentication services) because their impact on performance is assumed to be negligible.

Case study: e-University Course Selection

The e-University was one of the case studies considered in the recently completed SENSORIA project, an EU project focussed on service-oriented computing.

In the Course Selection scenario students obtain information about the courses available at their education establishment and may enrol in those for which specific requirements are satisfied.

The model will not consider other services which may be deployed in an actual application (e.g. authentication services) because their impact on performance is assumed to be negligible.

We assume a constant population of students e.g. the application is only accessible after the university's matriculation process is complete.

The model: services

Access to the system is through the University Portal. There are four services in this model:

Course Browsing	allows the user to navigate through the University's course offerings;
Course Selection	allows the user to submit a tentative course plan which will be validated against the University's requirements and the student's curriculum;
Student Confirmation	will enforce the student to check relevant personal details;
Course Registration	will confirm the student's selection.

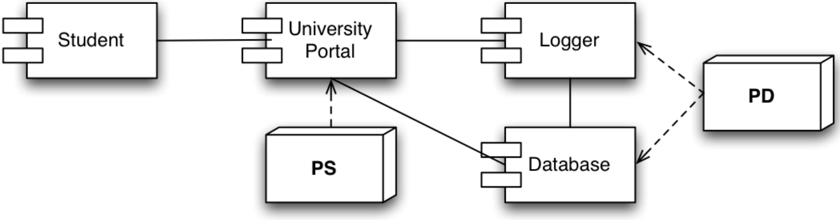
The model: services

Access to the system is through the University Portal. There are four services in this model:

Course Browsing	allows the user to navigate through the University's course offerings;
Course Selection	allows the user to submit a tentative course plan which will be validated against the University's requirements and the student's curriculum;
Student Confirmation	will enforce the student to check relevant personal details;
Course Registration	will confirm the student's selection.

These components make use of a Database service, which in turn maintains an event log through a Logger service.

The model: deployment diagram



Solid lines show request/response pairings whilst dashed lines show deployment on processors.

The model: multi-threading

- The University Portal instantiates a pool of threads.
- Each thread deals with one request from one student for one of the services offered.
- During processing the thread is locked in the sense that it cannot be acquired by further incoming requests.
- When the request is fulfilled the thread clears its state and becomes available again.

The model: multi-threading

- The University Portal instantiates a pool of threads.
- Each thread deals with one request from one student for one of the services offered.
- During processing the thread is locked in the sense that it cannot be acquired by further incoming requests.
- When the request is fulfilled the thread clears its state and becomes available again.

Analogous multi-threaded behaviour will be given to Database and Logger.

The model: multi-threading

- The University Portal instantiates a pool of threads.
- Each thread deals with one request from one student for one of the services offered.
- During processing the thread is locked in the sense that it cannot be acquired by further incoming requests.
- When the request is fulfilled the thread clears its state and becomes available again.

Analogous multi-threaded behaviour will be given to Database and Logger.

If all threads are busy requests are queued. There will also be contention with many threads running on the same processor.

The model: modelling processors and services

$$
\begin{aligned}
Processor_1 &\stackrel{def}{=} (acq, r_{acq}).Processor_2 \\
Processor_2 &\stackrel{def}{=} (type_1, r_1).Processor_1 \\
&+ (type_2, r_2).Processor_1 \\
&+ \dots \\
&+ (type_n, r_n).Processor_1
\end{aligned}
$$

The model: modelling processors and services

$$
\begin{array}{rcl}
Processor_1 & \stackrel{def}{=} & (acq, r_{acq}).Processor_2 \\
Processor_2 & \stackrel{def}{=} & (type_1, r_1).Processor_1 \\
 & + & (type_2, r_2).Processor_1 \\
 & + & \ldots \\
 & + & (type_n, r_n).Processor_1
\end{array}
$$

$$A \stackrel{def}{=} (req_{A,B}, r_{reqA}).(reply_{A,B}, r_{repA}).A'$$

$$B \stackrel{def}{=} (req_{A,B}, r_{reqB}).(execute, r).(reply_{A,B}, r_{repB}).B'$$

The model: modelling processors and services

$$\begin{array}{rcl}
Processor_1 & \stackrel{def}{=} & (acq, r_{acq}).Processor_2 \\
Processor_2 & \stackrel{def}{=} & (type_1, r_1).Processor_1 \\
 & + & (type_2, r_2).Processor_1 \\
 & + & \ldots \\
 & + & (type_n, r_n).Processor_1
\end{array}$$

$$A \stackrel{def}{=} (req_{A,B}, r_{reqA}).(reply_{A,B}, r_{repA}).A'$$

$$B \stackrel{def}{=} (req_{A,B}, r_{reqB}).(execute, r).(reply_{A,B}, r_{repB}).B'$$

The communication between A and B will be expressed by means of the cooperation operator $A \underset{L}{\bowtie} B, L = \{req_{A,B}, reply_{A,B}\}$.

The model: University Portal

A single thread of execution for the application layer University Portal is implemented as a sequential component which initially accepts requests for any of the services provided:

$$\begin{array}{rcl} \mathit{Portal} & \stackrel{\mathit{def}}{=} & (\mathit{req}_{\mathit{student},\mathit{browse}}, \nu).\mathit{Browse} \\ & + & (\mathit{req}_{\mathit{student},\mathit{select}}, \nu).\mathit{Select} \\ & + & (\mathit{req}_{\mathit{student},\mathit{confirm}}, \nu).\mathit{Confirm} \\ & + & (\mathit{req}_{\mathit{student},\mathit{register}}, \nu).\mathit{Register} \end{array}$$

The rate ν is used throughout this model in all the request/reply activities.

The model: Course Browsing (example service)

The action type acq_{ps} is used to obtain exclusive access to processor PS.

$$
\begin{aligned}
Browse &\stackrel{\mathit{def}}{=} (acq_{ps}, \nu).Cache \\
Cache &\stackrel{\mathit{def}}{=} (cache, 0.95 r_{cache}).Internal \\
&\quad + \; (cache, 0.05 r_{cache}).External \\
Internal &\stackrel{\mathit{def}}{=} (acq_{ps}, \nu).(internal, r_{int}).BrowseRep \\
External &\stackrel{\mathit{def}}{=} (req_{external,read}, \nu).(reply_{external,read}, \nu). \\
&\qquad (acq_{ps}, \nu).(external, r_{ext}).BrowseRep \\
BrowseRep &\stackrel{\mathit{def}}{=} (reply_{student,browse}, \nu).Portal
\end{aligned}
$$

The model: Course Browsing (example service)

The action type acq_{ps} is used to obtain exclusive access to processor PS.

$$
\begin{aligned}
Browse &\overset{def}{=} (acq_{ps}, \nu).Cache \\
Cache &\overset{def}{=} (cache, 0.95 r_{cache}).Internal \\
&\quad + \ (cache, 0.05 r_{cache}).External \\
Internal &\overset{def}{=} (acq_{ps}, \nu).(internal, r_{int}).BrowseRep \\
External &\overset{def}{=} (req_{external,read}, \nu).(reply_{external,read}, \nu). \\
&\qquad (acq_{ps}, \nu).(external, r_{ext}).BrowseRep \\
BrowseRep &\overset{def}{=} (reply_{student,browse}, \nu).Portal
\end{aligned}
$$

The other services, Course Selection, Student Confirmation and Course Registration are modelled analogously.

The model: the database

The Database offers two functions: reading and writing.

The model: the database

The Database offers two functions: reading and writing.

$$
\begin{array}{rcl}
\mathit{Database} & \stackrel{\mathit{def}}{=} & (\mathit{req}_{\mathit{external},\mathit{read}}, \nu).\mathit{Read} \\
 & + & (\mathit{req}_{\mathit{register},\mathit{write}}, \nu).\mathit{Write} \\
\mathit{Read} & \stackrel{\mathit{def}}{=} & (\mathit{acq}_{\mathit{pd}}, \nu).(\mathit{read}, r_{\mathit{read}}).\mathit{ReadReply} \\
\mathit{ReadReply} & \stackrel{\mathit{def}}{=} & (\mathit{reply}_{\mathit{external},\mathit{read}}, \nu).\mathit{Database} \\
\mathit{Write} & \stackrel{\mathit{def}}{=} & (\mathit{acq}_{\mathit{pd}}, \nu).(\mathit{write}, r_{\mathit{write}}).\mathit{LogWrite} \\
\mathit{LogWrite} & \stackrel{\mathit{def}}{=} & (\mathit{req}_{\mathit{database},\mathit{log}}, \nu). \\
 & & (\mathit{reply}_{\mathit{database},\mathit{log}}, \nu).\mathit{WriteReply} \\
\mathit{WriteReply} & \stackrel{\mathit{def}}{=} & (\mathit{reply}_{\mathit{register},\mathit{write}}, \nu).\mathit{Database}
\end{array}
$$

The model: the database

The Database offers two functions: reading and writing.

$$
\begin{array}{rcl}
\mathit{Database} & \stackrel{\mathit{def}}{=} & (\mathit{req}_{\mathit{external},\mathit{read}}, \nu).\mathit{Read} \\
 & + & (\mathit{req}_{\mathit{register},\mathit{write}}, \nu).\mathit{Write} \\
\mathit{Read} & \stackrel{\mathit{def}}{=} & (\mathit{acq}_{\mathit{pd}}, \nu).(\mathit{read}, r_{\mathit{read}}).\mathit{ReadReply} \\
\mathit{ReadReply} & \stackrel{\mathit{def}}{=} & (\mathit{reply}_{\mathit{external},\mathit{read}}, \nu).\mathit{Database} \\
\mathit{Write} & \stackrel{\mathit{def}}{=} & (\mathit{acq}_{\mathit{pd}}, \nu).(\mathit{write}, r_{\mathit{write}}).\mathit{LogWrite} \\
\mathit{LogWrite} & \stackrel{\mathit{def}}{=} & (\mathit{req}_{\mathit{database},\mathit{log}}, \nu). \\
 & & (\mathit{reply}_{\mathit{database},\mathit{log}}, \nu).\mathit{WriteReply} \\
\mathit{WriteReply} & \stackrel{\mathit{def}}{=} & (\mathit{reply}_{\mathit{register},\mathit{write}}, \nu).\mathit{Database}
\end{array}
$$

Reading is a purely local computation.
Writing additionally uses the Logger service.

The model: the Logger

The Logger service accepts requests from Student Confirmation and Database respectively. It is deployed on the same processor as Database, (processor *PD*) so one thread execution may be modelled as:

$$\begin{array}{rcl}
\mathit{Logger} & \stackrel{\mathit{def}}{=} & (\mathit{req}_{\mathit{confirm},\mathit{log}}, \nu).\mathit{LogConfirm} \\
 & + & (\mathit{req}_{\mathit{database},\mathit{log}}, \nu).\mathit{LogDatabase} \\
\mathit{LogConfirm} & \stackrel{\mathit{def}}{=} & (\mathit{acq}_{\mathit{pd}}, \nu). \\
 & & (\mathit{log}_{\mathit{conf}}, r_{\mathit{lgc}}).\mathit{ReplyConfirm} \\
\mathit{ReplyConfirm} & \stackrel{\mathit{def}}{=} & (\mathit{reply}_{\mathit{confirm},\mathit{log}}, \nu).\mathit{Logger} \\
\mathit{LogDatabase} & \stackrel{\mathit{def}}{=} & (\mathit{acq}_{\mathit{pd}}, \nu). \\
 & & (\mathit{log}_{\mathit{db}}, r_{\mathit{lgd}}).\mathit{ReplyDatabase} \\
\mathit{ReplyDatabase} & \stackrel{\mathit{def}}{=} & (\mathit{reply}_{\mathit{database},\mathit{log}}, \nu).\mathit{Logger}
\end{array}$$

The model: The processor *PD*

$$\begin{aligned} PD_1 &\overset{def}{=} (acq_{pd}, \nu).PD_2 \\ PD_2 &\overset{def}{=} (read, r_{read}).PD_1 + (write, r_{write}).PD_1 \\ &\phantom{\overset{def}{=}}\; + (log_{conf}, r_{lgc}).PD_1 + (log_{db}, r_{lgd}).PD_1 \end{aligned}$$

The model: Student Workload

A student is modelled as a sequential component which interacts with the university portal and accesses all of the services available. The behaviour is cyclic and the student interposes some think time between successive requests.

The model: Student Workload

A student is modelled as a sequential component which interacts with the university portal and accesses all of the services available. The behaviour is cyclic and the student interposes some think time between successive requests.

$$\begin{aligned}
StdThink &\stackrel{def}{=} (think, r_{think}).StdBrowse \\
StdBrowse &\stackrel{def}{=} (req_{student,browse}, \nu).(reply_{student,browse}, \nu).StdSelect \\
StdSelect &\stackrel{def}{=} (req_{student,select}, \nu).(reply_{student,select}, \nu).StdConfirm \\
StdConfirm &\stackrel{def}{=} (req_{student,confirm}, \nu).(reply_{student,confirm}, \nu).StdRegister \\
StdRegister &\stackrel{def}{=} (req_{student,register}, \nu).(reply_{student,register}, \nu).StdThink
\end{aligned}$$

The model: System Equation

The system equation captures the multiplicity of threads and processors.

$$StdThink[N_S] \underset{*}{\bowtie} \Big(\big(Portal[N_P] \underset{M_1}{\bowtie} ValUni[N_P] \underset{M_1}{\bowtie} ValCur[N_P] \big) \underset{M_2}{\bowtie} Database[N_D] \underset{M_3}{\bowtie} Logger[N_L] \Big) \underset{*}{\bowtie} \Big(PS_1[N_{PS}] \underset{\emptyset}{\bowtie} PD_1[N_{PD}] \Big)$$

where

$$M_1 = \{fork, join\}$$
$$M_2 = \{req_{external,read}, reply_{external,read}, req_{register,write}, reply_{register,write}\}$$
$$M_3 = \{req_{confirm,log}, reply_{confirm,log}, req_{database,log}, reply_{database,log}\}$$

The separate validating threads *ValUni* and *ValCur* inherit the multiplicity levels of the thread *Portal* which spawns them.

Qualitative Analysis

The state space size for this model grows very rapidly — for example, adding one portal thread can result in an increase in state space size by a factor of ten.

N_S	N_P	N_D	N_L	N_{PS}	N_{PD}	*Size*
1	any	any	any	1	any	48
1	any	any	any	≥ 2	any	49
2	1	1	1	1	1	230
3	1	1	1	1	1	680
3	2	2	2	2	2	5540
10	2	2	2	2	2	512116
10	3	2	2	2	2	5075026

Qualitative Analysis

The state space size for this model grows very rapidly — for example, adding one portal thread can result in an increase in state space size by a factor of ten.

Analysis based on the explicit representation of the state space is nevertheless a valuable tool for validating the correctness of the model.

Qualitative Analysis

The state space size for this model grows very rapidly — for example, adding one portal thread can result in an increase in state space size by a factor of ten.

Analysis based on the explicit representation of the state space is nevertheless a valuable tool for validating the correctness of the model.

When there is only one student the state space is fairly small regardless of the multiplicity levels of threads and processors because at most only one of them will be used.

Qualitative Analysis

The state space size for this model grows very rapidly — for example, adding one portal thread can result in an increase in state space size by a factor of ten.

Analysis based on the explicit representation of the state space is nevertheless a valuable tool for validating the correctness of the model.

When there is only one student the state space is fairly small regardless of the multiplicity levels of threads and processors because at most only one of them will be used.

Another form of qualitative analysis can be based on visual inspection of the reachability graph, which can be walked through to generate possible trajectories of the system.

Model Verification

We may verify that the model respects policies of exclusive access to threads and processors by direct inspection of the state space.

Student	*Portal*	*Portal*	*PS*
Action type cache			
$(reply_{student,browse}, \nu).StdSelect$	*Cache*	*Portal*	PS_2
$(reply_{student,browse}, \nu).StdSelect$	*Portal*	*Cache*	PS_2
Action type prepare			
$(reply_{student,select}, \nu).StdConfirm$	*Portal*	$(prepare, r_{prep}).ForkPrepare$	PS_2
$(reply_{student,select}, \nu).StdConfirm$	$(prepare, r_{prep}).ForkPrepare$	*Portal*	PS_2
Action type confirm			
$(reply_{student,confirm}, \nu).StdRegister$	*Portal*	$(confirm, r_{con}).LogStudent$	PS_2
$(reply_{student,confirm}, \nu).StdRegister$	$(confirm, r_{con}).LogStudent$	*Portal*	PS_2
Action type register			
$(reply_{student,register}, \nu).StdThink$	$(register, r_{reg}).Store$	*Portal*	PS_2
$(reply_{student,register}, \nu).StdThink$	*Portal*	$(register, r_{reg}).Store$	PS_2

This fragment of the state space when there are two threads for the portal and one instance of all other components.

Model Verification

We may verify that the model respects policies of exclusive access to threads and processors by direct inspection of the state space.

Student	*Portal*	*Portal*	*PS*
Action type cache			
$(\mathit{reply}_{\mathit{student},\mathit{browse}}, \nu).\mathit{StdSelect}$	*Cache*	*Portal*	PS_2
$(\mathit{reply}_{\mathit{student},\mathit{browse}}, \nu).\mathit{StdSelect}$	*Portal*	*Cache*	PS_2
Action type prepare			
$(\mathit{reply}_{\mathit{student},\mathit{select}}, \nu).\mathit{StdConfirm}$	*Portal*	$(\mathit{prepare}, r_{\mathit{prep}}).\mathit{ForkPrepare}$	PS_2
$(\mathit{reply}_{\mathit{student},\mathit{select}}, \nu).\mathit{StdConfirm}$	$(\mathit{prepare}, r_{\mathit{prep}}).\mathit{ForkPrepare}$	*Portal*	PS_2
Action type confirm			
$(\mathit{reply}_{\mathit{student},\mathit{confirm}}, \nu).\mathit{StdRegister}$	*Portal*	$(\mathit{confirm}, r_{\mathit{con}}).\mathit{LogStudent}$	PS_2
$(\mathit{reply}_{\mathit{student},\mathit{confirm}}, \nu).\mathit{StdRegister}$	$(\mathit{confirm}, r_{\mathit{con}}).\mathit{LogStudent}$	*Portal*	PS_2
Action type register			
$(\mathit{reply}_{\mathit{student},\mathit{register}}, \nu).\mathit{StdThink}$	$(\mathit{register}, r_{\mathit{reg}}).\mathit{Store}$	*Portal*	PS_2
$(\mathit{reply}_{\mathit{student},\mathit{register}}, \nu).\mathit{StdThink}$	*Portal*	$(\mathit{register}, r_{\mathit{reg}}).\mathit{Store}$	PS_2

This fragment of the state space when there are two threads for the portal and one instance of all other components.

A necessary condition for correctness is that if one thread is engaged in an activity the other is idle.

Model Verification

In general we do not wish to conduct such verification by hand but it may be readily undertaken using automatic tools such as the probabilistic model checker, PRISM.

Model Verification

In general we do not wish to conduct such verification by hand but it may be readily undertaken using automatic tools such as the probabilistic model checker, PRISM.

Indeed PRISM supports PEPA as one of its input languages.

Markovian Analysis: Performance Bounds

Since exact Markovian analysis relies on an explicit enumeration of the state space it is limited to relatively small-scale systems.

Markovian Analysis: Performance Bounds

Since exact Markovian analysis relies on an explicit enumeration of the state space it is limited to relatively small-scale systems.

But performance analysis of even these small scale systems can offer valuable insight into the behaviour of the system — for the e-University system it may be used to derive performance bound estimates.

Markovian Analysis: Performance Bounds

Since exact Markovian analysis relies on an explicit enumeration of the state space it is limited to relatively small-scale systems.

But performance analysis of even these small scale systems can offer valuable insight into the behaviour of the system — for the e-University system it may be used to derive performance bound estimates.

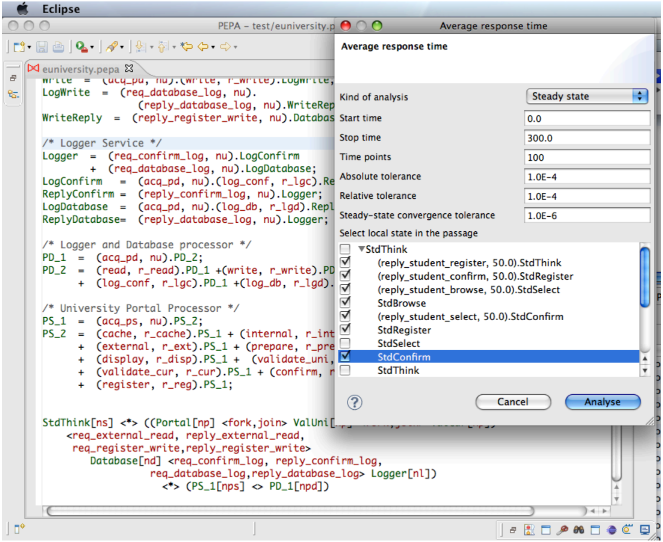
If we consider the system with only one student, the performance in terms of response time perceived by the user will be optimal because there is no contention for resources — the response time for larger populations of students can only get worse.

Average response times

With only one student the state space is kept manageable and bounds can be computed quickly and accurately.

System Configuration								
N_P	N_D	N_L	N_{PD}	N_{PS}	$N_S = 1$	$N_S = 2$	$N_S = 3$	$N_S = 4$
1	1	1	1	1	3.195	3.694	4.390	5.357
1	1	1	1	2	3.064	3.522	4.155	5.032
3	3	3	3	3	3.064	3.065	3.066	3.074

Calculating average response time



```
Write   =  (acq_pd, nu).(write, r_write).LogWrite;
LogWrite  =  (req_database_log, nu).
                (reply_database_log, nu).WriteRep
WriteReply  =  (reply_register_write, nu).Databas

/* Logger Service */
Logger  =  (req_confirm_log, nu).LogConfirm
        +  (req_database_log, nu).LogDatabase;
LogConfirm   =  (acq_pd, nu).(log_conf, r_lgc).Re
ReplyConfirm =  (reply_confirm_log, nu).Logger;
LogDatabase  =  (acq_pd, nu).(log_db, r_lgd).Repl
ReplyDatabase=  (reply_database_log, nu).Logger;

/* Logger and Database processor */
PD_1  =  (acq_pd, nu).PD_2;
PD_2  =  (read, r_read).PD_1 +(write, r_write).PD
      +  (log_conf, r_lgc).PD_1 +(log_db, r_lgd).

/* University Portal Processor */
PS_1  =  (acq_ps, nu).PS_2;
PS_2  =  (cache, r_cache).PS_1 + (internal, r_int
      +  (external, r_ext).PS_1 + (prepare, r_pre
      +  (display, r_disp).PS_1 +  (validate_uni,
      +  (validate_cur, r_cur).PS_1 + (confirm, r
      +  (register, r_reg).PS_1;


StdThink[ns] <*> ((Portal[np] <fork,join> ValUni[
    <req_external_read, reply_external_read,
     req_register_write,reply_register_write>
        Database[nd] <req_confirm_log, reply_confirm_log,
                  req_database_log,reply_database_log> Logger[nl])
                    <*> (PS_1[nps] <> PD_1[npd])
```

Average response time

Kind of analysis: Steady state
Start time: 0.0
Stop time: 300.0
Time points: 100
Absolute tolerance: 1.0E-4
Relative tolerance: 1.0E-4
Steady-state convergence tolerance: 1.0E-6

Select local state in the passage

- ☐ StdThink
  - ☑ (reply_student_register, 50.0).StdThink
  - ☑ (reply_student_confirm, 50.0).StdRegister
  - ☑ (reply_student_browse, 50.0).StdSelect
  - ☑ StdBrowse
  - ☑ (reply_student_select, 50.0).StdConfirm
  - ☑ StdRegister
  - ☐ StdSelect
  - ☑ StdConfirm
  - ☐ StdThink

Scalability and Optimisation

To study the system under realistically sized user workloads many more instances of the components need to be considered.

Scalability and Optimisation

To study the system under realistically sized user workloads many more instances of the components need to be considered.

This gives rise to a model whose state space would far exceed the size that can be handled via explicit state space techniques such as Markovian analysis.

Scalability and Optimisation

To study the system under realistically sized user workloads many more instances of the components need to be considered.

This gives rise to a model whose state space would far exceed the size that can be handled via explicit state space techniques such as Markovian analysis.

Fortunately, as multiple instances of components are involved we can consider this as a system exhibiting collective behaviour and resort to fluid approximation.

Scalability and Optimisation

To study the system under realistically sized user workloads many more instances of the components need to be considered.

This gives rise to a model whose state space would far exceed the size that can be handled via explicit state space techniques such as Markovian analysis.

Fortunately, as multiple instances of components are involved we can consider this as a system exhibiting collective behaviour and resort to fluid approximation.

The model gives rise to a set of 63 coupled ODEs which can be solved, for example using a fifth-order Runge-Kutta numerical integrator.

Scalability and Optimisation

To study the system under realistically sized user workloads many more instances of the components need to be considered.

This gives rise to a model whose state space would far exceed the size that can be handled via explicit state space techniques such as Markovian analysis.

Fortunately, as multiple instances of components are involved we can consider this as a system exhibiting collective behaviour and resort to fluid approximation.

The model gives rise to a set of 63 coupled ODEs which can be solved, for example using a fifth-order Runge-Kutta numerical integrator.

Of course an alternative would be to use stochastic simulation.

Scalability analysis

$$N_P = N_D = N_L = 80, N_{PD} = 40, N_{PS} = 40.$$

N_S	ODE	Runtime	CTMC	Runtime	Error
1	2.975	2.7 s	3.091	436 s	3.74%
300	2.975	2.7 s	3.105	2656 s	4.17%
325	2.975	2.6 s	3.329	3017 s	10.62%
350	3.686	2.9 s	3.863	6505 s	4.57%
400	5.999	7.3 s	5.993	4465 s	0.10%
500	10.623	6.6 s	10.534	3845 s	0.84%
600	15.248	6.6 s	15.233	2985 s	0.10%

Stochastic simulation was conducted using the method of batch means, terminated when the 95% confidence interval was $\leq 1\%$ of the average.

Optimisation

In addition to the workload parameter N_S there are 20 other parameters in the model: 15 rate parameters and 5 concurrency levels for threads and processors.

Optimisation

In addition to the workload parameter N_S there are 20 other parameters in the model: 15 rate parameters and 5 concurrency levels for threads and processors.

A typical scenario may be that the rate parameters are hardware-dependent and cannot be revised, and the workload is fixed, determined by the student population.

Optimisation

In addition to the workload parameter N_S there are 20 other parameters in the model: 15 rate parameters and 5 concurrency levels for threads and processors.

A typical scenario may be that the rate parameters are hardware-dependent and cannot be revised, and the workload is fixed, determined by the student population.

The modeller's role is then to determine an optimal configuration for the concurrency levels of the system with respect to user-perceived performance, e.g. average response time.

Optimisation

In addition to the workload parameter N_S there are 20 other parameters in the model: 15 rate parameters and 5 concurrency levels for threads and processors.

A typical scenario may be that the rate parameters are hardware-dependent and cannot be revised, and the workload is fixed, determined by the student population.

The modeller's role is then to determine an optimal configuration for the concurrency levels of the system with respect to user-perceived performance, e.g. average response time.

Let us assume a workload population of 350 students and assume that the response time previously found for this workload is acceptable.

Optimisation

The table shows the response times calculated with different system configurations of similar size.

Conf.	N_P	N_D	N_L	N_{PS}	N_{PD}	*Response time*
A	80	80	80	40	40	3.686
B	70	70	70	40	40	3.686
C	60	60	60	40	40	4.506
D	70	70	70	35	35	5.998
E	70	50	50	40	40	3.686
F	70	20	20	40	40	3.686
G	70	20	15	40	40	4.278
H	70	15	20	40	40	5.024

Optimisation

The table shows the response times calculated with different system configurations of similar size.

Conf.	N_P	N_D	N_L	N_{PS}	N_{PD}	*Response time*
A	80	80	80	40	40	3.686
B	70	70	70	40	40	3.686
C	60	60	60	40	40	4.506
D	70	70	70	35	35	5.998
E	70	50	50	40	40	3.686
F	70	20	20	40	40	3.686
G	70	20	15	40	40	4.278
H	70	15	20	40	40	5.024

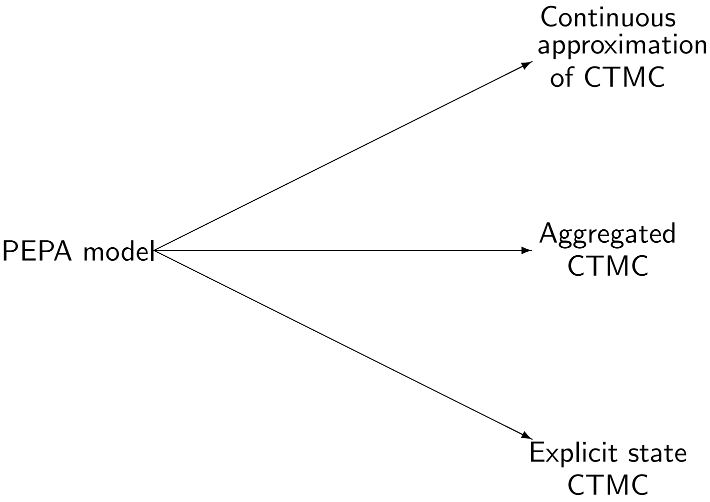
Optimisation

The table shows the response times calculated with different system configurations of similar size.

Conf.	N_P	N_D	N_L	N_{PS}	N_{PD}	*Response time*
A	80	80	80	40	40	3.686
B	70	70	70	40	40	3.686
C	60	60	60	40	40	4.506
D	70	70	70	35	35	5.998
E	70	50	50	40	40	3.686
F	70	20	20	40	40	3.686
G	70	20	15	40	40	4.278
H	70	15	20	40	40	5.024

Outline

Scalable Representations



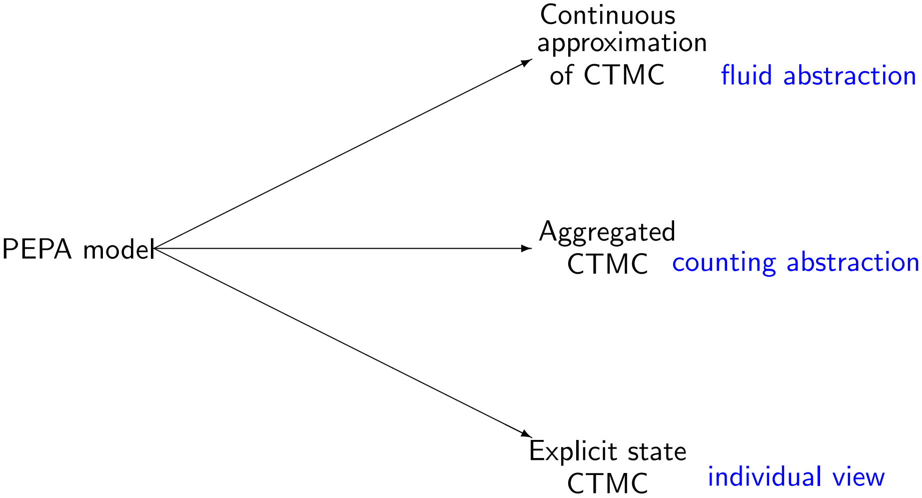
PEPA model

Continuous approximation of CTMC

Aggregated CTMC

Explicit state CTMC

Scalable Representations



PEPA model

Continuous approximation of CTMC

fluid abstraction

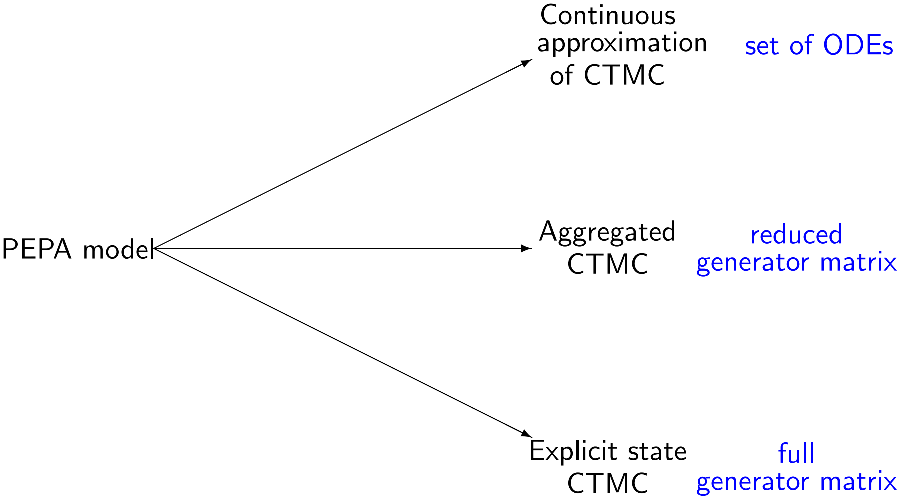
Aggregated CTMC

counting abstraction

Explicit state CTMC

individual view

Scalable Representations



PEPA model

Continuous approximation of CTMC

set of ODEs

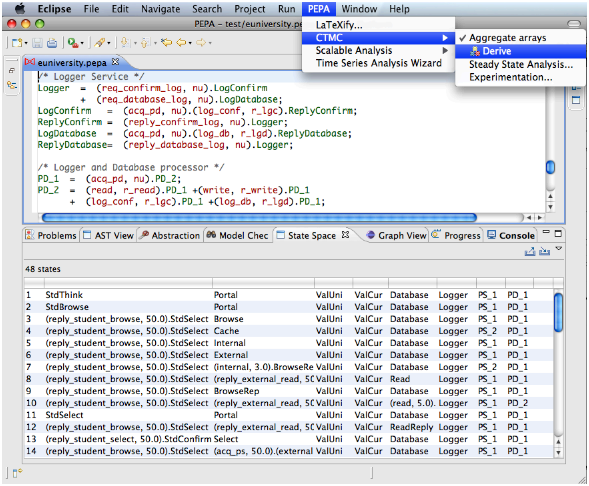
Aggregated CTMC

reduced generator matrix

Explicit state CTMC

full generator matrix

Eclipse Plug-in for PEPA



```
/* Logger Service */
Logger   =  (req_confirm_log, nu).LogConfirm
         +  (req_database_log, nu).LogDatabase;
LogConfirm   =  (acq_pd, nu).(log_conf, r_lgc).ReplyConfirm;
ReplyConfirm =  (reply_confirm_log, nu).Logger;
LogDatabase  =  (acq_pd, nu).(log_db, r_lgd).ReplyDatabase;
ReplyDatabase=  (reply_database_log, nu).Logger;

/* Logger and Database processor */
PD_1  =  (acq_pd, nu).PD_2;
PD_2  =  (read, r_read).PD_1 +(write, r_write).PD_1
      +  (log_conf, r_lgc).PD_1 +(log_db, r_lgd).PD_1;
```

48 states

1	StdThink	Portal	ValUni	ValCur	Database	Logger	PS_1	PD_1
2	StdBrowse	Portal	ValUni	ValCur	Database	Logger	PS_1	PD_1
3	(reply_student_browse, 50.0).StdSelect	Browse	ValUni	ValCur	Database	Logger	PS_1	PD_1
4	(reply_student_browse, 50.0).StdSelect	Cache	ValUni	ValCur	Database	Logger	PS_2	PD_1
5	(reply_student_browse, 50.0).StdSelect	Internal	ValUni	ValCur	Database	Logger	PS_1	PD_1
6	(reply_student_browse, 50.0).StdSelect	External	ValUni	ValCur	Database	Logger	PS_1	PD_1
7	(reply_student_browse, 50.0).StdSelect	(internal, 3.0).BrowseRe	ValUni	ValCur	Database	Logger	PS_2	PD_1
8	(reply_student_browse, 50.0).StdSelect	(reply_external_read, 50	ValUni	ValCur	Read	Logger	PS_1	PD_1
9	(reply_student_browse, 50.0).StdSelect	BrowseRep	ValUni	ValCur	Database	Logger	PS_1	PD_1
10	(reply_student_browse, 50.0).StdSelect	(reply_external_read, 50	ValUni	ValCur	(read, 5.0).	Logger	PS_1	PD_2
11	StdSelect	Portal	ValUni	ValCur	Database	Logger	PS_1	PD_1
12	(reply_student_browse, 50.0).StdSelect	(reply_external_read, 50	ValUni	ValCur	ReadReply	Logger	PS_1	PD_1
13	(reply_student_select, 50.0).StdConfirm	Select	ValUni	ValCur	Database	Logger	PS_1	PD_1
14	(reply_student_browse, 50.0).StdSelect	(acq_ps, 50.0).(external	ValUni	ValCur	Database	Logger	PS_1	PD_1

Robust tool support is essential to make develop techniques practical.

Other applications

PEPA, and the associated analysis techniques, were originally developed with the objective of studying computer systems.

However, it has also be adopted by modelling a wide-range of other types of system:

- Locks and movable bridges in inland shipping in Belgium (Knapen, Hasselt)
- Automotive on-board diagnostics expert systems (Console, Picardi and Ribaudo)
- Biological cell signalling pathways (Calder, Duguid, Gilmore and Hillston)
- Crowd dynamics in informatic environments (Harrison, Latella and Massink)

Conclusions

- Many interesting and important systems can be regarded as examples of collective dynamics and emergent behaviour.

Conclusions

- Many interesting and important systems can be regarded as examples of collective dynamics and emergent behaviour.
- Process algebras, such as PEPA, are well-suited to modelling the behaviour of such systems in terms of the individuals and their interactions.

Conclusions

- Many interesting and important systems can be regarded as examples of collective dynamics and emergent behaviour.
- Process algebras, such as PEPA, are well-suited to modelling the behaviour of such systems in terms of the individuals and their interactions.
- Continuous approximation allows a rigorous mathematical analysis of the average behaviour of such systems.

Conclusions

- Many interesting and important systems can be regarded as examples of collective dynamics and emergent behaviour.
- Process algebras, such as PEPA, are well-suited to modelling the behaviour of such systems in terms of the individuals and their interactions.
- Continuous approximation allows a rigorous mathematical analysis of the average behaviour of such systems.
- This alternative view of systems has opened up many and exciting new research directions.

Thanks!

Thanks!

Acknowledgements: collaborators

Thanks to many co-authors and collaborators: Jeremy Bradley, Luca Bortolussi, Allan Clark, Graham Clark, Adam Duguid, Vashti Galpin, Nil Gesweiller, **Stephen Gilmore**, Marco Stenico, Nigel Thomas, **Mirco Tribastone**, and others.

Thanks!

Acknowledgements: collaborators

Thanks to many co-authors and collaborators: Jeremy Bradley, Luca Bortolussi, Allan Clark, Graham Clark, Adam Duguid, Vashti Galpin, Nil Gesweiller, **Stephen Gilmore**, Marco Stenico, Nigel Thomas, **Mirco Tribastone**, and others.

Acknowledgements: funding

The PEPA project has received funding from the EPRSC, BBSRC, the Royal Society and the CEC IST-FET programme.

Thanks!

Acknowledgements: collaborators
Thanks to many co-authors and collaborators: Jeremy Bradley, Luca Bortolussi, Allan Clark, Graham Clark, Adam Duguid, Vashti Galpin, Nil Gesweiller, **Stephen Gilmore**, Marco Stenico, Nigel Thomas, **Mirco Tribastone**, and others.

Acknowledgements: funding
The PEPA project has received funding from the EPRSC, BBSRC, the Royal Society and the CEC IST-FET programme.

More information:
http://www.dcs.ed.ac.uk/pepa