SPAs for performance modelling: Lecture 5 — Tackling State Space Explosion

Jane Hillston

LFCS, School of Informatics
The University of Edinburgh
Scotland

12th April 2013

THE UNIVERSITY of EDINBURGH

Outline

Outline

State Space Explosion

The numerical solution of CTMC models such as those built using stochastic Petri nets and stochastic process algebras, like PEPA, relies on construction of the $N \times N$ infinitesimal generator matrix $\mathbf{Q}$, and the N-dimensional probability vector π, where N is the size of the state space.

State Space Explosion

The numerical solution of CTMC models such as those built using stochastic Petri nets and stochastic process algebras, like PEPA, relies on construction of the $N \times N$ infinitesimal generator matrix $\mathbf{Q}$, and the N-dimensional probability vector π, where N is the size of the state space.

Unfortunately, the size of these entities often exceeds what can be handled in memory.

State Space Explosion

The numerical solution of CTMC models such as those built using stochastic Petri nets and stochastic process algebras, like PEPA, relies on construction of the $N \times N$ infinitesimal generator matrix $\mathbf{Q}$, and the N-dimensional probability vector π, where N is the size of the state space.

Unfortunately, the size of these entities often exceeds what can be handled in memory.

This problem is known as state space explosion.

(All discrete state modelling approaches are prone to this problem.)

A simple example: processors and resources

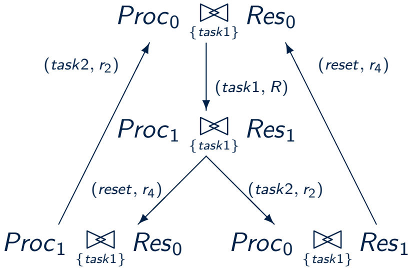
$$
\begin{aligned}
Proc_0 &\stackrel{\mathit{def}}{=} (task1, r_1).Proc_1 \\
Proc_1 &\stackrel{\mathit{def}}{=} (task2, r_2).Proc_0 \\
Res_0 &\stackrel{\mathit{def}}{=} (task1, r_3).Res_1 \\
Res_1 &\stackrel{\mathit{def}}{=} (reset, r_4).Res_0
\end{aligned}
$$

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

A simple example: processors and resources

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

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



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

A simple example: processors and resources

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

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

$$
\begin{tikzcd}
 & Proc_0 \underset{\{task1\}}{\bowtie} Res_0 \arrow[d, "{(task1, R)}"] & \\
 & Proc_1 \underset{\{task1\}}{\bowtie} Res_1 \arrow[dl, "{(reset, r_4)}"'] \arrow[dr, "{(task2, r_2)}"] & \\
Proc_1 \underset{\{task1\}}{\bowtie} Res_0 \arrow[uur, "{(task2, r_2)}"] & & Proc_0 \underset{\{task1\}}{\bowtie} Res_1 \arrow[uul, "{(reset, r_4)}"']
\end{tikzcd}
$$

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

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

Simple example : multiple instances

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

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

Simple example : multiple instances

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

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

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 : multiple instances

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

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

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

The size of state space: $2^{N_P} \times 2^{N_R}$.

Tackling state space explosion

- To overcome state-space explosion problem in CTMCs, many mathematical tools and approaches have been proposed.

Tackling state space explosion

- To overcome state-space explosion problem in CTMCs, many mathematical tools and approaches have been proposed.
- We will use the stochastic process algebra, PEPA as an example, and give an overview of four different approaches to tackling the state space explosion problem.

Tackling state space explosion

- To overcome state-space explosion problem in CTMCs, many mathematical tools and approaches have been proposed.
- We will use the stochastic process algebra, PEPA as an example, and give an overview of four different approaches to tackling the state space explosion problem.
  - state space reduction via aggregation;

Tackling state space explosion

- To overcome state-space explosion problem in CTMCs, many mathematical tools and approaches have been proposed.
- We will use the stochastic process algebra, PEPA as an example, and give an overview of four different approaches to tackling the state space explosion problem.
  - state space reduction via aggregation;
  - decomposed solution techniques

Tackling state space explosion

- To overcome state-space explosion problem in CTMCs, many mathematical tools and approaches have been proposed.
- We will use the stochastic process algebra, PEPA as an example, and give an overview of four different approaches to tackling the state space explosion problem.
  - state space reduction via aggregation;
  - decomposed solution techniques
  - stochastic simulation over the discrete state space;

Tackling state space explosion

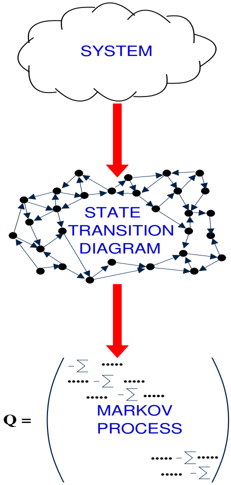
- To overcome state-space explosion problem in CTMCs, many mathematical tools and approaches have been proposed.
- We will use the stochastic process algebra, PEPA as an example, and give an overview of four different approaches to tackling the state space explosion problem.
    - state space reduction via aggregation;
    - decomposed solution techniques
    - stochastic simulation over the discrete state space;
    - fluid approximation of the state space.

Outline

Performance Modelling using CTMC

Model Construction

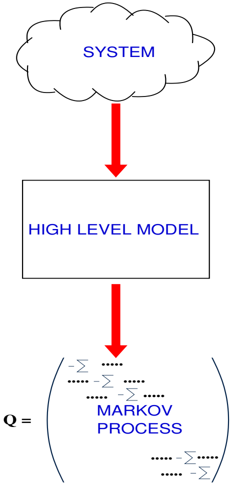
- describing the system using a high level modelling formalism
- generating the underlying CTMC



Performance Modelling using CTMC

Model Construction

- describing the system using a high level modelling formalism
- generating the underlying CTMC



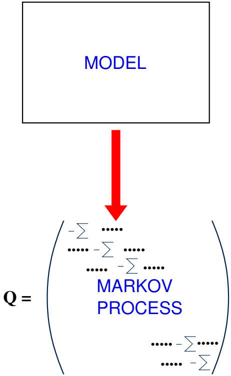
Performance Modelling using CTMC

Model Construction

- describing the system using a high level modelling formalism
- generating the underlying CTMC

Model Manipulation

- model simplification
- model aggregation



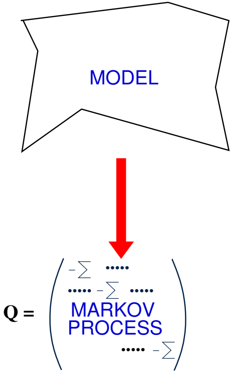
Performance Modelling using CTMC

Model Construction

- describing the system using
  a high level modelling formalism
- generating the underlying CTMC

Model Manipulation

- model simplification
- model aggregation



Performance Modelling using CTMC

Model Construction

- describing the system using a high level modelling formalism
- generating the underlying CTMC

Model Manipulation

- model simplification
- model aggregation

Model Solution

- solving the CTMC to find steady state probability distribution
- deriving performance measures

Model Manipulation

Model simplification: use a model-model equivalence to substitute one model by another which is more attractive from a solution point of view, e.g. smaller state space, special class of model, etc.

Model Manipulation

Model simplification: use a model-model equivalence to substitute one model by another which is more attractive from a solution point of view, e.g. smaller state space, special class of model, etc.

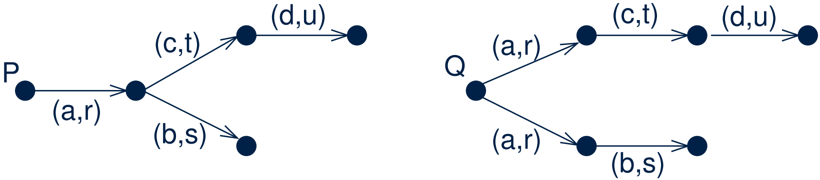
Model aggregation: use a state-state equivalence to establish a partition of the state space of a model, and replace each set of states by one macro-state, i.e. take a different stochastic representation of the same model.

Strong Equivalence in PEPA

Strong equivalence in PEPA is a bisimulation in the style of Larsen of Skou.

Strong Equivalence in PEPA

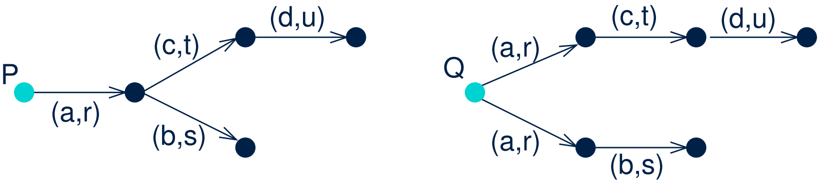
Strong equivalence in PEPA is a bisimulation in the style of Larsen of Skou.



Observability is assumed to include the ability to record timing information over a number of runs.

Strong Equivalence in PEPA

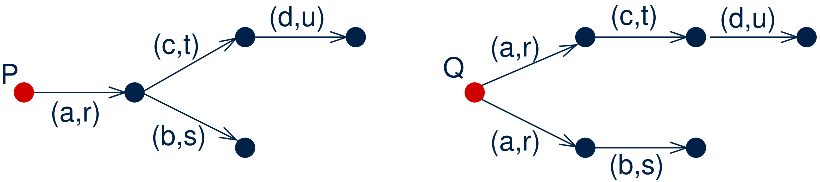
Strong equivalence in PEPA is a bisimulation in the style of Larsen of Skou.



Observability is assumed to include the ability to record timing information over a number of runs.

Strong Equivalence in PEPA

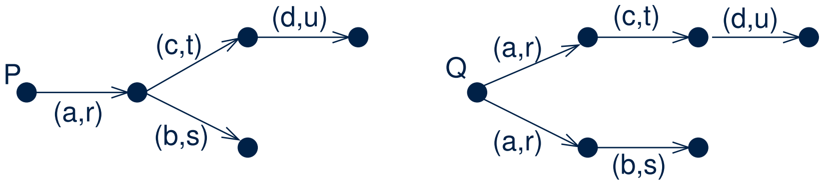
Strong equivalence in PEPA is a bisimulation in the style of Larsen of Skou.



Observability is assumed to include the ability to record timing information over a number of runs.

Strong Equivalence in PEPA

Strong equivalence in PEPA is a bisimulation in the style of Larsen of Skou.

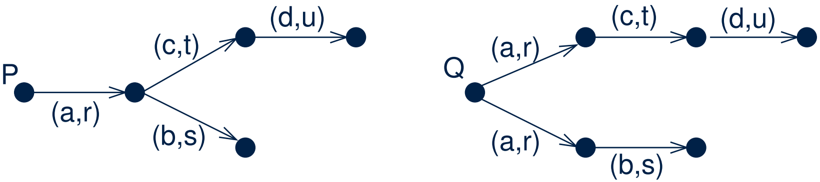


Observability is assumed to include the ability to record timing information over a number of runs.

Two processes are equivalent if they can undertake the same actions, at the same rate, and arrive at processes that are equivalent.

Strong Equivalence in PEPA

Strong equivalence in PEPA is a bisimulation in the style of Larsen of Skou.



Observability is assumed to include the ability to record timing information over a number of runs.

Two processes are equivalent if they can undertake the same actions, at the same rate, and arrive at processes that are equivalent.

Expressed as rates to equivalence classes of processes

Strong Equivalence in PEPA

Definition

An equivalence relation $\mathcal{R} \subseteq \mathcal{C} \times \mathcal{C}$ is a *strong equivalence* if whenever $(P, Q) \in \mathcal{R}$ then for all $\alpha \in \mathcal{A}$ and for all $S \in \mathcal{C}/\mathcal{R}$

$$q[P, S, \alpha] = q[Q, S, \alpha].$$

where

$$q[C_i, S, \alpha] = \sum_{C_j \in S} q(C_i, C_j, \alpha)$$

Strong Equivalence and Lumpability

- Given this definition it is fairly straightforward to show that if we consider strong equivalence of states within a single model, it induces an ordinarily lumpable partition on the state space of the underlying Markov chain.

Strong Equivalence and Lumpability

- Given this definition it is fairly straightforward to show that if we consider strong equivalence of states within a single model, it induces an ordinarily lumpable partition on the state space of the underlying Markov chain.
- Moreover it can be shown that strong equivalence is a congruence.

Strong Equivalence and Lumpability

- Given this definition it is fairly straightforward to show that if we consider strong equivalence of states within a single model, it induces an ordinarily lumpable partition on the state space of the underlying Markov chain.
- Moreover it can be shown that strong equivalence is a congruence.
- This means that aggregation based on lumpability can be applied component by component, avoiding the previous problem of having to construct the complete state space in order to find the lumpable partitions.

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 approach shifts to a numerical representation of states and transitions. [Jie Ding, PhD thesis, Edin. 2010]

Losing identity

The syntactic nature of PEPA makes models easily understood by humans, but not so convenient for approaches such as aggregation and simulation.

Losing identity

The syntactic nature of PEPA makes models easily understood by humans, but not so convenient for approaches such as aggregation and simulation.

In particular when we have many instances of the same component type, in the PEPA expression these instances are distinguished by their location (position from left to right) in the expression.

Losing identity

The syntactic nature of PEPA makes models easily understood by humans, but not so convenient for approaches such as aggregation and simulation.

In particular when we have many instances of the same component type, in the PEPA expression these instances are distinguished by their location (position from left to right) in the expression.

However, in general we do not care which such instance is involved in an event, just that one of them is, i.e. it is sufficient to count the instances that are in the possible local states.

Losing identity

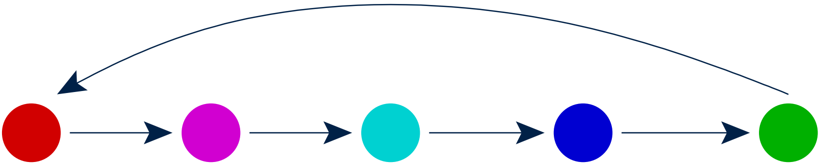
The syntactic nature of PEPA makes models easily understood by humans, but not so convenient for approaches such as aggregation and simulation.

In particular when we have many instances of the same component type, in the PEPA expression these instances are distinguished by their location (position from left to right) in the expression.

However, in general we do not care which such instance is involved in an event, just that one of them is, i.e. it is sufficient to count the instances that are in the possible local states.

Thus we change to a state representation which is a numerical state vector.

Reducing by lumpability



When we use the numerical vector state representation for PEPA we group together those expressions that have the same counts for each of the local states and we are certain that the partition that we induce on the state space is lumpable and so the lumped process is still a Markov process.

Numerical Vector Form [QEST 2005]

Definition

For an arbitrary PEPA model $\mathcal{M}$ with n component types $C_i, i = 1, 2, \cdots, n$, each with d_i distinct local derivatives, the numerical vector form of $\mathcal{M}$, $\mathbf{m}(\mathcal{M})$, is a vector with $d = \sum_{i=1}^{n} d_i$ entries.

The entry $\mathbf{m}[C_{i_j}]$ records how many instances of the jth local derivative of component type C_i are exhibited in the current state.

The entries in the system vector or a sequential component's vector are no longer syntactic terms representing the local derivative, but the number of components currently exhibiting this local derivative.

Example revisited

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

$$(Res_0 \parallel Res_0) \underset{\{task1\}}{\bowtie} (Proc_0 \parallel Proc_0)$$

Numerical vector form

For our example model:

$$\mathbf{m} = \left(\mathbf{m}[Proc_0], \mathbf{m}[Proc_1], \mathbf{m}[Res_0], \mathbf{m}[Res_1]\right).$$

Numerical vector form

For our example model:

$$\mathbf{m} = (\mathbf{m}[Proc_0], \mathbf{m}[Proc_1], \mathbf{m}[Res_0], \mathbf{m}[Res_1])\,.$$

When $N_P = N_R = 2$, the system equation of the model determines the starting state:

$$\mathbf{m} = (N_P, 0, N_R, 0) = (2, 0, 2, 0)$$

We can apply the possible activities in each of the states until we find all possible states.

$$\begin{array}{lll} \mathbf{s}_1 = (2,0,2,0), & \mathbf{s}_2 = (1,1,1,1), & \mathbf{s}_3 = (1,1,2,0), \\ \mathbf{s}_4 = (1,1,0,2), & \mathbf{s}_5 = (0,2,1,1), & \mathbf{s}_6 = (2,0,1,1), \\ \mathbf{s}_7 = (0,2,0,2), & \mathbf{s}_8 = (0,2,2,0), & \mathbf{s}_9 = (2,0,0,2). \end{array}$$

Numerical vector form

The initial state is $(2, 0, 2, 0)$ where the entries in the vector are counting the number of Res_0, Res_1, $Proc_0$, $Proc_1$ local derivatives respectively, exhibited in the current state.

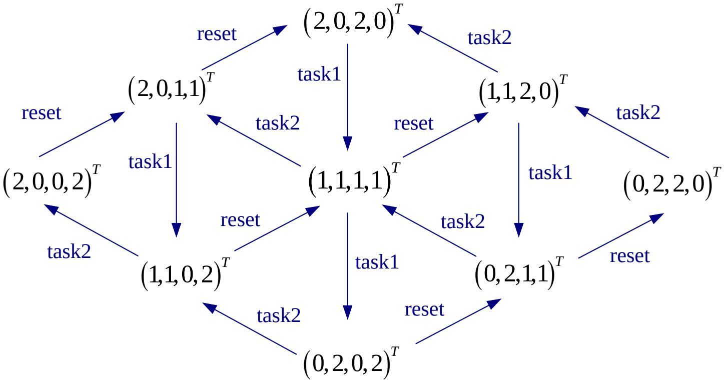
Numerical vector form

The initial state is $(2, 0, 2, 0)$ where the entries in the vector are counting the number of Res_0, Res_1, $Proc_0$, $Proc_1$ local derivatives respectively, exhibited in the current state.

If we consider the state $(1, 1, 1, 1)$ it is representing four distinct syntactic states

$$
\begin{array}{c}
(Res_0, Res_1, Proc_0, Proc_1) \\
(Res_1, Res_0, Proc_0, Proc_1) \\
(Res_0, Res_1, Proc_1, Proc_0) \\
(Res_1, Res_0, Proc_1, Proc_0)
\end{array}
$$

The resulting state space



The size of the state space: $(N_P + d_P - 1)^{d_P - 1} \times (N_R + d_R - 1)^{d_R - 1}$.

Solution of an aggregated model

Once we have the state space of the aggregated model we construct the CTMC in the obvious way — associating one state with each node in the aggregated state transition diagram.

Solution of an aggregated model

Once we have the state space of the aggregated model we construct the CTMC in the obvious way — associating one state with each node in the aggregated state transition diagram.

This CTMC will typically have a smaller state space than the one derived from the original state representation as a derivative graph, and certainly no larger.

Solution of an aggregated model

Once we have the state space of the aggregated model we construct the CTMC in the obvious way — associating one state with each node in the aggregated state transition diagram.

This CTMC will typically have a smaller state space than the one derived from the original state representation as a derivative graph, and certainly no larger.

The steady state probability distribution can then be derived in the usual way by solving the global balance equations.

Solution of an aggregated model

Once we have the state space of the aggregated model we construct the CTMC in the obvious way — associating one state with each node in the aggregated state transition diagram.

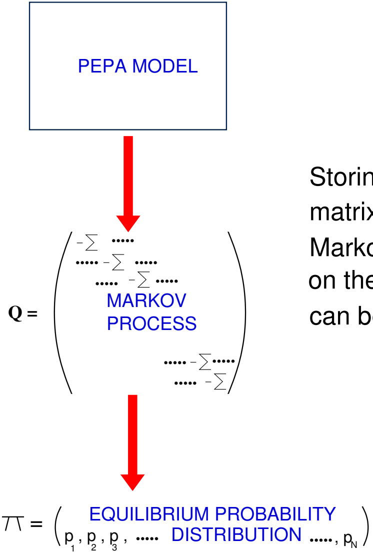
This CTMC will typically have a smaller state space than the one derived from the original state representation as a derivative graph, and certainly no larger.

The steady state probability distribution can then be derived in the usual way by solving the global balance equations.

The solution gives you the probability of being in the set of states that have the same behaviour.

Outline

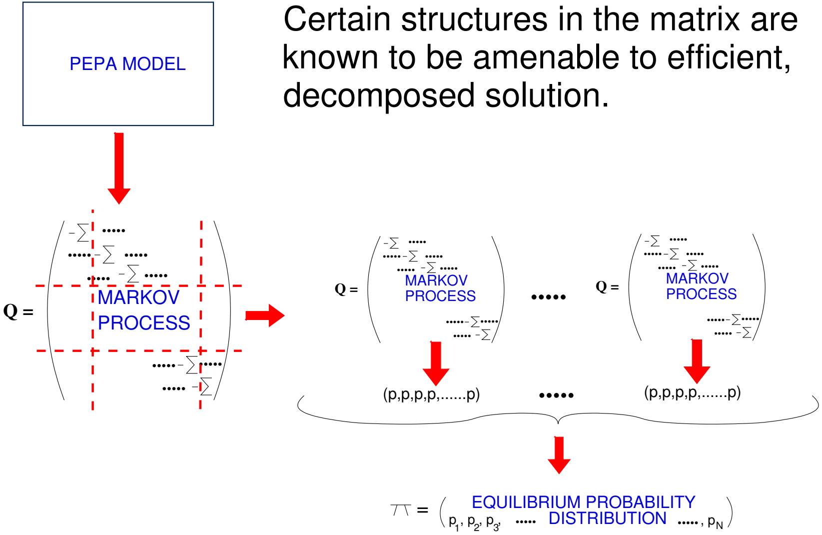
Characterising efficient solution



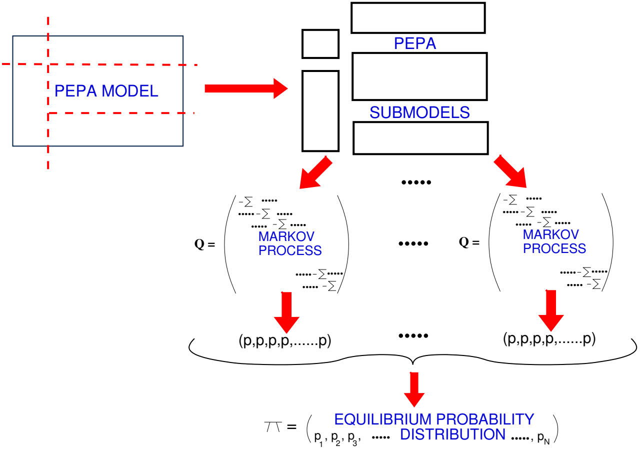
Storing and manipulating the matrix which represents the Markov process places limitations on the size of model which can be analysed.

Characterising efficient solution

Certain structures in the matrix are known to be amenable to efficient, decomposed solution.

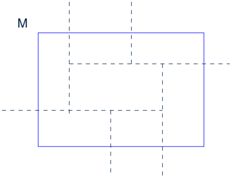


Characterising efficient solution

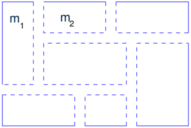


Finding the corresponding structures in the process algebra means that these techniques can be applied automatically, before the monolithic matrix is formed.

Decomposed solution: product form models



$M = (m_1, m_2, \ldots, m_n)$



p(M)

$p(M) = G \times p(m_1) \times p(m_2) \times \ldots \times p(m_n)$

Partition the model M into n statistically independent submodels $m_1, m_2, \ldots, m_n$

In isolation, find the steady state distribution p for each of the submodels m_i

Form the steady state distribution of M as the product of the solutions for each submodel m_i and a normalising constant

When do PEPA components behave as if they were statistically independent...?

Product Form PEPA Models

$$P \equiv S_1 \parallel S_2$$

Product Form PEPA Models

$$P \equiv S_1 \parallel S_2$$

Add restricted direct interaction between components with a particular structure

$$P \equiv S_1 \underset{L}{\bowtie} S_2$$

S_1, S_2 and L all restricted

Product Form PEPA Models

$$P \equiv S_1 \parallel S_2$$

Add restricted direct interaction between components with a particular structure

$$P \equiv S_1 \underset{L}{\bowtie} S_2$$

S_1, S_2 and L all restricted

- Quasi-reversibility

Product Form PEPA Models

$$P \equiv S_1 \parallel S_2$$

Add restricted direct interaction between components with a particular structure

$$P \equiv S_1 \underset{L}{\bowtie} S_2$$

S_1, S_2 and L all restricted

- Quasi-reversibility
- Reversibility

Product Form PEPA Models

$$P \equiv S_1 \parallel S_2$$

Add restricted direct interaction between components with a particular structure

$$P \underset{L}{\bowtie} S_2 \equiv S_1 \underset{L}{\bowtie} S_2$$

S_1, S_2 and L all restricted

- Quasi-reversibility
- Reversibility
- Routing process approach

Product Form PEPA Models

$$P \equiv S_1 \parallel S_2$$

Add restricted direct interaction between components with a particular structure

$$P \equiv S_1 \underset{L}{\bowtie} S_2$$

S_1, S_2 and L all restricted

- Quasi-reversibility
- Reversibility
- Routing process approach

Add indirect interaction via a third component with a particular structure and type of interaction

$$P \equiv (S_1 \parallel S_2) \underset{L}{\bowtie} R$$

L and R restricted (wrt S_1 and S_2)

Product Form PEPA Models

$$P \equiv S_1 \parallel S_2$$

Add restricted direct interaction between components with a particular structure

$$P \equiv S_1 \underset{L}{\bowtie} S_2$$

S_1, S_2 and L all restricted

- Quasi-reversibility
- Reversibility
- Routing process approach

Add indirect interaction via a third component with a particular structure and type of interaction

$$P \equiv (S_1 \parallel S_2) \underset{L}{\bowtie} R$$

L and R restricted (wrt S_1 and S_2)

- Boucherie resource contention

Product Form PEPA Models

$$P \equiv S_1 \parallel S_2$$

Add restricted direct interaction between components with a particular structure

$$P \equiv S_1 \underset{L}{\bowtie} S_2$$

S_1, S_2 and L all restricted

- Quasi-reversibility
- Reversibility
- Routing process approach

Add indirect interaction via a third component with a particular structure and type of interaction

$$P \equiv (S_1 \parallel S_2) \underset{L}{\bowtie} R$$

L and R restricted (wrt S_1 and S_2)

- Boucherie resource contention
- Queueing discipline models

Product Form PEPA Models

$$P \equiv S_1 \parallel S_2$$

Add restricted direct interaction between components with a particular structure

$$P \equiv S_1 \underset{L}{\bowtie} S_2$$

S_1, S_2 and L all restricted

- Quasi-reversibility
- Reversibility
- Routing process approach

Add indirect interaction via a third component with a particular structure and type of interaction

$$P \equiv (S_1 \parallel S_2) \underset{L}{\bowtie} R$$

L and R restricted (wrt S_1 and S_2)

- Boucherie resource contention
- Queueing discipline models
- Quasi-separability

Approximate solutions

Numerical solution of the full Markov process is regarded as the exact result and aggregation based on lumpability is also regarded as being exact.

Approximate solutions

Numerical solution of the full Markov process is regarded as the exact result and aggregation based on lumpability is also regarded as being exact.

However due to the difficulties of staying within the confines of the Markov property most techniques for tackling state space explosion are not exact.

Approximate solutions

Numerical solution of the full Markov process is regarded as the exact result and aggregation based on lumpability is also regarded as being exact.

However due to the difficulties of staying within the confines of the Markov property most techniques for tackling state space explosion are not exact.

We are sometimes prepared to trade exactness for tractability.

Time Scale Decomposition

- Based on a well-established technique in the underlying CTMC, this approach partitions the states of the process according to the rates at which they undertake activities.

Time Scale Decomposition

- Based on a well-established technique in the underlying CTMC, this approach partitions the states of the process according to the rates at which they undertake activities.
- The assumption is that the model will form local equilibria within "islands" of fast activities, before occasitionally moving on to a different "island" via a slow activity.

Time Scale Decomposition

- Based on a well-established technique in the underlying CTMC, this approach partitions the states of the process according to the rates at which they undertake activities.
- The assumption is that the model will form local equilibria within “islands” of fast activities, before occasitionally moving on to a different “island” via a slow activity.
- Fast interacting states are modelled in detail in isolation, and an aggregated model captures the transitions between the clusters of states.

Time Scale Decomposition in SPA [Mertsiotakis 98]

- Activities are partitioned into fast activities and slow activities.

Time Scale Decomposition in SPA [Mertsiotakis 98]

- Activities are partitioned into fast activities and slow activities.
- In SPA models, null cooperation over slow action types is used to identify the islands of fast activity, and each is solved as a separate model.

Time Scale Decomposition in SPA [Mertsiotakis 98]

- Activities are partitioned into fast activities and slow activities.
- In SPA models, null cooperation over slow action types is used to identify the islands of fast activity, and each is solved as a separate model.
- Similarly the aggregated model, with one state per cluster of fast activity is found by disabling the fast activities, again through null cooperation.

Time Scale Decomposition in SPA [Mertsiotakis 98]

- Activities are partitioned into fast activities and slow activities.
- In SPA models, null cooperation over slow action types is used to identify the islands of fast activity, and each is solved as a separate model.
- Similarly the aggregated model, with one state per cluster of fast activity is found by disabling the fast activities, again through null cooperation.
- Each of the resulting CTMCs is solved and the results combined to give the overall solution.

Throughput Approximation in SPA [Mertsiotakis 98]

- The model is partitioned into two in such a way that there is a one flow each way between them.
- In SPA terms this means that the two subcomponents interact between a pair of actions, each passive with respect to one of them.
- Each subcomponent is solved in isolation to give an estimate of the throughput of the interface activity for which it is active, and assuming a rate for the interface activity for which it is passive.
- This pair of solutions is carried out iteratively, each time updating the passive rate according to the previous solution until convergence is achieved.

Outline

Fluid Approximation

The fourth approach to tackling state space explosion that we consider is the use of fluid or continuous approximation.

Fluid Approximation

The fourth approach to tackling state space explosion that we consider is the use of fluid or continuous approximation.

Here the key idea is to approximate the behaviour of a discrete event system which jumps between discrete states by a continuous system which moves smoothly over a continuous state space.

Continuously varying counting variables

When this is applied in performance models the state space is usually characterised by counting variables:

- the number of customers in a queue,
- the number of servers who are busy, or
- the number of local derivatives in a particular state in a PEPA model.

Continuously varying counting variables

When this is applied in performance models the state space is usually characterised by counting variables:

- the number of customers in a queue,
- the number of servers who are busy, or
- the number of local derivatives in a particular state in a PEPA model.

Allowing continuous variables for these quantities might seem odd to begin with — what does it mean for 0.65 servers to be busy? — but when we think of it as the expectation it becomes easier to interpret.

Simple illustrative example

Tiny example

$$P_1 \stackrel{\mathit{def}}{=} (\mathit{start}, r).P_2 \qquad P_2 \stackrel{\mathit{def}}{=} (\mathit{run}, r).P_3 \qquad P_3 \stackrel{\mathit{def}}{=} (\mathit{stop}, r).P_1$$

$$\mathit{System} \stackrel{\mathit{def}}{=} (P_1 \parallel P_1)$$

Simple illustrative example

Tiny example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

This example defines a system with nine reachable states:

1. $P_1 \parallel P_1$
2. $P_1 \parallel P_2$
3. $P_1 \parallel P_3$
4. $P_2 \parallel P_1$
5. $P_2 \parallel P_2$
6. $P_2 \parallel P_3$
7. $P_3 \parallel P_1$
8. $P_3 \parallel P_2$
9. $P_3 \parallel P_3$

The transitions between states have quantified duration r which can be evaluated against a CTMC or ODE interpretation.

Analysis based on Continuous-time Markov Chains

Tiny example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

Using transient analysis we can evaluate the probability of each state with respect to time. For $t = 0$:

1 1.0000	4 0.0000	7 0.0000
2 0.0000	5 0.0000	8 0.0000
3 0.0000	6 0.0000	9 0.0000

Analysis based on Continuous-time Markov Chains

Tiny example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

Using transient analysis we can evaluate the probability of each state with respect to time. For $t = 1$:

1 0.1642	4 0.1567	7 0.0842
2 0.1567	5 0.1496	8 0.0804
3 0.0842	6 0.0804	9 0.0432

Analysis based on Continuous-time Markov Chains

Tiny example

$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$

$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

Using transient analysis we can evaluate the probability of each state with respect to time. For $t = 2$:

1. 0.1056
2. 0.1159
3. 0.1034
4. 0.1159
5. 0.1272
6. 0.1135
7. 0.1034
8. 0.1135
9. 0.1012

Analysis based on Continuous-time Markov Chains

Tiny example

$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$

$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

Using transient analysis we can evaluate the probability of each state with respect to time. For $t = 3$:

1. 0.1082
2. 0.1106
3. 0.1100
4. 0.1106
5. 0.1132
6. 0.1125
7. 0.1100
8. 0.1125
9. 0.1119

Analysis based on Continuous-time Markov Chains

Tiny example

$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$

$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

Using transient analysis we can evaluate the probability of each state with respect to time. For $t = 4$:

1. 0.1106
2. 0.1108
3. 0.1111
4. 0.1108
5. 0.1110
6. 0.1113
7. 0.1111
8. 0.1113
9. 0.1116

Analysis based on Continuous-time Markov Chains

Tiny example

$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$

$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

Using transient analysis we can evaluate the probability of each state with respect to time. For $t = 5$:

1. 0.1111
2. 0.1110
3. 0.1111
4. 0.1110
5. 0.1110
6. 0.1111
7. 0.1111
8. 0.1111
9. 0.1111

Analysis based on Continuous-time Markov Chains

Tiny example

$$P_1 \stackrel{\mathit{def}}{=} (\mathit{start}, r).P_2 \qquad P_2 \stackrel{\mathit{def}}{=} (\mathit{run}, r).P_3 \qquad P_3 \stackrel{\mathit{def}}{=} (\mathit{stop}, r).P_1$$

$$\mathit{System} \stackrel{\mathit{def}}{=} (P_1 \parallel P_1)$$

Using transient analysis we can evaluate the probability of each state with respect to time. For $t = 6$:

1. 0.1111
2. 0.1111
3. 0.1111
4. 0.1111
5. 0.1110
6. 0.1111
7. 0.1111
8. 0.1111
9. 0.1111

Analysis based on Continuous-time Markov Chains

Tiny example

$P_1 \stackrel{\text{def}}{=} (start, r).P_2 \qquad P_2 \stackrel{\text{def}}{=} (run, r).P_3 \qquad P_3 \stackrel{\text{def}}{=} (stop, r).P_1$

$$System \stackrel{\text{def}}{=} (P_1 \parallel P_1)$$

Using transient analysis we can evaluate the probability of each state with respect to time. For $t = 7$:

1. 0.1111
2. 0.1111
3. 0.1111
4. 0.1111
5. 0.1111
6. 0.1111
7. 0.1111
8. 0.1111
9. 0.1111

Analysis based on Ordinary Differential Equations

Tiny example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

Using the ordinary differential equation semantics we can compute the expected number of each type of component.

For $t = 0$:

P_1	2.0000
P_2	0.0000
P_3	0.0000

Analysis based on Ordinary Differential Equations

Tiny example

$$P_1 \stackrel{\mathit{def}}{=} (start, r).P_2 \qquad P_2 \stackrel{\mathit{def}}{=} (run, r).P_3 \qquad P_3 \stackrel{\mathit{def}}{=} (stop, r).P_1$$

$$System \stackrel{\mathit{def}}{=} (P_1 \parallel P_1)$$

Using the ordinary differential equation semantics we can compute the expected number of each type of component.

For $t = 1$:

P_1	0.8121
P_2	0.7734
P_3	0.4144

Analysis based on Ordinary Differential Equations

Tiny example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

Using the ordinary differential equation semantics we can compute the expected number of each type of component.

For $t = 2$:

P_1	0.6490
P_2	0.7051
P_3	0.6457

Analysis based on Ordinary Differential Equations

Tiny example

$$P_1 \stackrel{\text{def}}{=} (start, r).P_2 \qquad P_2 \stackrel{\text{def}}{=} (run, r).P_3 \qquad P_3 \stackrel{\text{def}}{=} (stop, r).P_1$$

$$System \stackrel{\text{def}}{=} (P_1 \parallel P_1)$$

Using the ordinary differential equation semantics we can compute the expected number of each type of component.

For $t = 3$:

P_1	0.6587
P_2	0.6719
P_3	0.6692

Analysis based on Ordinary Differential Equations

Tiny example

$$P_1 \stackrel{\mathit{def}}{=} (\mathit{start}, r).P_2 \qquad P_2 \stackrel{\mathit{def}}{=} (\mathit{run}, r).P_3 \qquad P_3 \stackrel{\mathit{def}}{=} (\mathit{stop}, r).P_1$$

$$\mathit{System} \stackrel{\mathit{def}}{=} (P_1 \parallel P_1)$$

Using the ordinary differential equation semantics we can compute the expected number of each type of component.

For $t = 4$:	P_1	0.6648
	P_2	0.6665
	P_3	0.6685

Analysis based on Ordinary Differential Equations

Tiny example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

Using the ordinary differential equation semantics we can compute the expected number of each type of component.

For $t = 5$:	P_1	0.6666
	P_2	0.6663
	P_3	0.6669

Analysis based on Ordinary Differential Equations

Tiny example

$$P_1 \stackrel{\mathit{def}}{=} (\mathit{start}, r).P_2 \qquad P_2 \stackrel{\mathit{def}}{=} (\mathit{run}, r).P_3 \qquad P_3 \stackrel{\mathit{def}}{=} (\mathit{stop}, r).P_1$$

$$\mathit{System} \stackrel{\mathit{def}}{=} (P_1 \parallel P_1)$$

Using the ordinary differential equation semantics we can compute the expected number of each type of component.

For $t = 6$:

P_1	0.6666
P_2	0.6666
P_3	0.6666

Analysis based on Ordinary Differential Equations

Tiny example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1)$$

Using the ordinary differential equation semantics we can compute the expected number of each type of component.

For $t = 7$:

P_1	0.6666
P_2	0.6666
P_3	0.6666

Analysis based on Ordinary Differential Equations

Slightly larger example

$$P_1 \stackrel{\mathit{def}}{=} (start, r).P_2 \qquad P_2 \stackrel{\mathit{def}}{=} (run, r).P_3 \qquad P_3 \stackrel{\mathit{def}}{=} (stop, r).P_1$$

$$System \stackrel{\mathit{def}}{=} (P_1 \parallel P_1 \parallel P_1)$$

A slightly larger example with a third copy of the process also initiated in state P_1.

$$\text{For } t = 0: \qquad \begin{array}{ll} P_1 & 3.0000 \\ P_2 & 0.0000 \\ P_3 & 0.0000 \end{array}$$

Analysis based on Ordinary Differential Equations

Slightly larger example

$$P_1 \stackrel{\mathit{def}}{=} (\mathit{start}, r).P_2 \qquad P_2 \stackrel{\mathit{def}}{=} (\mathit{run}, r).P_3 \qquad P_3 \stackrel{\mathit{def}}{=} (\mathit{stop}, r).P_1$$

$$\mathit{System} \stackrel{\mathit{def}}{=} (P_1 \parallel P_1 \parallel P_1)$$

A slightly larger example with a third copy of the process also initiated in state P_1.

For $t = 1$:

P_1	1.1782
P_2	1.1628
P_3	0.6590

Analysis based on Ordinary Differential Equations

Slightly larger example

$$P_1 \stackrel{\mathit{def}}{=} (\mathit{start}, r).P_2 \qquad P_2 \stackrel{\mathit{def}}{=} (\mathit{run}, r).P_3 \qquad P_3 \stackrel{\mathit{def}}{=} (\mathit{stop}, r).P_1$$

$$\mathit{System} \stackrel{\mathit{def}}{=} (P_1 \parallel P_1 \parallel P_1)$$

A slightly larger example with a third copy of the process also initiated in state P_1.

For $t = 2$:

P_1	0.9766
P_2	1.0754
P_3	0.9479

Analysis based on Ordinary Differential Equations

Slightly larger example

$$P_1 \stackrel{\text{def}}{=} (start, r).P_2 \qquad P_2 \stackrel{\text{def}}{=} (run, r).P_3 \qquad P_3 \stackrel{\text{def}}{=} (stop, r).P_1$$

$$System \stackrel{\text{def}}{=} (P_1 \parallel P_1 \parallel P_1)$$

A slightly larger example with a third copy of the process also initiated in state P_1.

For $t = 3$:		
	P_1	0.9838
	P_2	1.0142
	P_3	1.0020

Analysis based on Ordinary Differential Equations

Slightly larger example

$$P_1 \stackrel{\mathit{def}}{=} (\mathit{start}, r).P_2 \qquad P_2 \stackrel{\mathit{def}}{=} (\mathit{run}, r).P_3 \qquad P_3 \stackrel{\mathit{def}}{=} (\mathit{stop}, r).P_1$$

$$\mathit{System} \stackrel{\mathit{def}}{=} (P_1 \parallel P_1 \parallel P_1)$$

A slightly larger example with a third copy of the process also initiated in state P_1.

For $t = 4$:

P_1	0.9981
P_2	0.9995
P_3	1.0023

Analysis based on Ordinary Differential Equations

Slightly larger example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1 \parallel P_1)$$

A slightly larger example with a third copy of the process also initiated in state P_1.

For $t = 5$:

P_1	1.0001
P_2	0.9996
P_3	1.0003

Analysis based on Ordinary Differential Equations

Slightly larger example

$$P_1 \stackrel{\text{def}}{=} (start, r).P_2 \qquad P_2 \stackrel{\text{def}}{=} (run, r).P_3 \qquad P_3 \stackrel{\text{def}}{=} (stop, r).P_1$$

$$System \stackrel{\text{def}}{=} (P_1 \parallel P_1 \parallel P_1)$$

A slightly larger example with a third copy of the process also initiated in state P_1.

For $t = 6$:

P_1	1.0001
P_2	0.9999
P_3	1.0000

Analysis based on Ordinary Differential Equations

Slightly larger example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1 \parallel P_1)$$

A slightly larger example with a third copy of the process also initiated in state P_1.

For $t = 7$:		
	P_1	1.0000
	P_2	0.9999
	P_3	0.9999

Analysis based on Ordinary Differential Equations

Slightly larger example

$$P_1 \stackrel{def}{=} (start, r).P_2 \qquad P_2 \stackrel{def}{=} (run, r).P_3 \qquad P_3 \stackrel{def}{=} (stop, r).P_1$$

$$System \stackrel{def}{=} (P_1 \parallel P_1 \parallel P_1)$$

A slightly larger example with a third copy of the process also initiated in state P_1.

For $t = 8$:

P_1	1.0000
P_2	1.0000
P_3	1.0000

Isn't this just the Chapman-Kolmogorov equations?

It is possible to perform transient analysis of a continuous-time Markov chain by solving the Chapman-Kolmogorov differential equations:

$$\frac{d\pi(t)}{dt} = \pi(t)Q$$

[Stewart, 1994]

Isn't this just the Chapman-Kolmogorov equations?

It is possible to perform transient analysis of a continuous-time Markov chain by solving the Chapman-Kolmogorov differential equations:

$$\frac{d\pi(t)}{dt} = \pi(t)Q$$

[Stewart, 1994]

That's not what we're doing. We go directly to ODEs.

Fluid approximation

- In a PEPA model the state at any current time is the local derivative or state of each component of the model.

Fluid approximation

- In a PEPA model the state at any current time is the local derivative or state of each component of the model.
- We can represent the state of the system as the count of the current number of each possible local derivative or component type.

Fluid approximation

- In a PEPA model the state at any current time is the local derivative or state of each component of the model.
- We can represent the state of the system as the count of the current number of each possible local derivative or component type.
- We can approximate the behaviour of the model by treating each count as a continuous variable, and the state of the model as a whole as the set of such variables.

Fluid approximation

- In a PEPA model the state at any current time is the local derivative or state of each component of the model.
- We can represent the state of the system as the count of the current number of each possible local derivative or component type.
- We can approximate the behaviour of the model by treating each count as a continuous variable, and the state of the model as a whole as the set of such variables.
- The evolution of each count variable can then be described by an ordinary differential equation

Fluid approximation

- In a PEPA model the state at any current time is the local derivative or state of each component of the model.
- We can represent the state of the system as the count of the current number of each possible local derivative or component type.
- We can approximate the behaviour of the model by treating each count as a continuous variable, and the state of the model as a whole as the set of such variables.
- The evolution of each count variable can then be described by an ordinary differential equation (assuming rates are deterministic).

Fluid approximation

- In a PEPA model the state at any current time is the local derivative or state of each component of the model.
- We can represent the state of the system as the count of the current number of each possible local derivative or component type.
- We can approximate the behaviour of the model by treating each count as a continuous variable, and the state of the model as a whole as the set of such variables.
- The evolution of each count variable can then be described by an ordinary differential equation (assuming rates are deterministic).

Appropriate for models in which there are large numbers of components of the same type.

Differential equations from PEPA models

- The PEPA definitions of the component specify the activities which can increase or decrease the number of components exhibited in the current state.
- The cooperations show when the number of instances of another component will have an influence on the evolution of this component.

Example revisited

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

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

Example revisited

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

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

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

We can capture exactly this relationship between activities and components the activity matrix which has one row for each component and one column for each activity.

Example revisited

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

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

	r_1	r_2	r_4	
$Proc_0$	-1	1	0	
$Proc_1$	1	-1	0	
Res_0	-1	0	1	
Res_1	1	0	-1	

We can capture exactly this relationship between activities and components the activity matrix which has one row for each component and one column for each activity.

Example revisited

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

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

ODE interpretation

$$
\begin{aligned}
\frac{dx_1}{dt} &= -r_1 \min(x_1, x_3) + r_2\, x_2 \\
& \quad x_1 = \text{no. of } Proc_1 \\
\frac{dx_2}{dt} &= r_1 \min(x_1, x_3) - r_2\, x_2 \\
& \quad x_2 = \text{no. of } Proc_2 \\
\frac{dx_3}{dt} &= -r_1 \min(x_1, x_3) + r_4\, x_4 \\
& \quad x_3 = \text{no. of } Res_0 \\
\frac{dx_4}{dt} &= r_1 \min(x_1, x_3) - r_4\, x_4 \\
& \quad x_4 = \text{no. of } Res_1
\end{aligned}
$$

We can capture exactly this relationship between activities and components the activity matrix which has one row for each component and one column for each activity.

Differential equations from PEPA models

- As we have already seen in deriving the activity matrix, the PEPA definitions of the component specify the activities which can increase or decrease the number of components exhibited in the current state.
- Moreover we can see for each component, which activities are entry activities and exit activities respectively.
- The cooperations show when the number of instances of another component will have an influence on the evolution of this component.

Differential equations from PEPA models

- As we have already seen in deriving the activity matrix, the PEPA definitions of the component specify the activities which can increase or decrease the number of components exhibited in the current state.
- Moreover we can see for each component, which activities are entry activities and exit activities respectively.
- The cooperations show when the number of instances of another component will have an influence on the evolution of this component.

In the following derivation we restrict to the case where all components that cooperate on an activity have the same rate for that activity.

Differential equations from PEPA models

Let $N(\mathcal{C}_{i_j}, t)$ denote the number of $\mathcal{C}_{i_j}$ type components at time t.

Differential equations from PEPA models

Let $N(\mathcal{C}_{i_j}, t)$ denote the number of $\mathcal{C}_{i_j}$ type components at time t.

Consider the change in a small time δt:

$$N(\mathcal{C}_{i_j}, t + \delta t) \; - \; N(\mathcal{C}_{i_j}, t) =$$

$$- \underbrace{\sum_{(\alpha,r)\in Ex(\mathcal{C}_{i_j})} r \times \min_{\mathcal{C}_{k_l}\in pre(\alpha,r)} (N(\mathcal{C}_{k_l}, t))\,\delta t}_{\text{exit activities}}$$

$$+ \underbrace{\sum_{(\alpha,r)\in En(\mathcal{C}_{i_j})} r \times \min_{\mathcal{C}_{k_l}\in pre(\alpha,r)} (N(\mathcal{C}_{k_l}, t))\,\delta t}_{\text{entry activities}}$$

Differential equations from PEPA models

Let $N(\mathcal{C}_{i_j}, t)$ denote the number of $\mathcal{C}_{i_j}$ type components at time t.

Consider the change in a small time δt:

$$N(\mathcal{C}_{i_j}, t+\delta t) \; - \; N(\mathcal{C}_{i_j}, t) =$$

$$- \underbrace{\sum_{(\alpha,r)\in Ex(\mathcal{C}_{i_j})} r \times \min_{\mathcal{C}_{k_l}\in pre(\alpha,r)} (N(\mathcal{C}_{k_l}, t))\,\delta t}_{\text{exit activities}}$$

$$+ \underbrace{\sum_{(\alpha,r)\in En(\mathcal{C}_{i_j})} r \times \min_{\mathcal{C}_{k_l}\in pre(\alpha,r)} (N(\mathcal{C}_{k_l}, t))\,\delta t}_{\text{entry activities}}$$

Differential equations from PEPA models

Let $N(\mathcal{C}_{i_j}, t)$ denote the number of $\mathcal{C}_{i_j}$ type components at time t.

Consider the change in a small time δt:

$$N(\mathcal{C}_{i_j}, t + \delta t) \; - \; N(\mathcal{C}_{i_j}, t) =$$

$$- \underbrace{\sum_{(\alpha,r) \in Ex(\mathcal{C}_{i_j})} r \times \min_{\mathcal{C}_{k_l} \in pre(\alpha,r)} (N(\mathcal{C}_{k_l}, t)) \, \delta t}_{\text{exit activities}}$$

$$+ \underbrace{\sum_{(\alpha,r) \in En(\mathcal{C}_{i_j})} r \times \min_{\mathcal{C}_{k_l} \in pre(\alpha,r)} (N(\mathcal{C}_{k_l}, t)) \, \delta t}_{\text{entry activities}}$$

Differential equations from PEPA models

Let $N(\mathcal{C}_{i_j}, t)$ denote the number of $\mathcal{C}_{i_j}$ type components at time t.

Consider the change in a small time δt:

$$N(\mathcal{C}_{i_j}, t+\delta t) \; - \; N(\mathcal{C}_{i_j}, t) =$$

$$- \underbrace{\sum_{(\alpha,r)\in Ex(\mathcal{C}_{i_j})} r \times \min_{\mathcal{C}_{k_l}\in pre(\alpha,r)} (N(\mathcal{C}_{k_l}, t))\,\delta t}_{\text{exit activities}}$$

$$+ \underbrace{\sum_{(\alpha,r)\in En(\mathcal{C}_{i_j})} r \times \min_{\mathcal{C}_{k_l}\in pre(\alpha,r)} (N(\mathcal{C}_{k_l}, t))\,\delta t}_{\text{entry activities}}$$

Differential equations from PEPA models

Let $N(\mathcal{C}_{i_j}, t)$ denote the number of $\mathcal{C}_{i_j}$ type components at time t.

Consider the change in a small time δt:

$$N(\mathcal{C}_{i_j}, t+\delta t) - N(\mathcal{C}_{i_j}, t) =$$

$$- \underbrace{\sum_{(\alpha,r)\in Ex(\mathcal{C}_{i_j})} r \times \min_{\mathcal{C}_{k_l}\in pre(\alpha,r)} (N(\mathcal{C}_{k_l}, t))\,\delta t}_{\text{exit activities}}$$

$$+ \underbrace{\sum_{(\alpha,r)\in En(\mathcal{C}_{i_j})} r \times \min_{\mathcal{C}_{k_l}\in pre(\alpha,r)} (N(\mathcal{C}_{k_l}, t))\,\delta t}_{\text{entry activities}}$$

Differential equations from PEPA models

Let $N(\mathcal{C}_{i_j}, t)$ denote the number of $\mathcal{C}_{i_j}$ type components at time t.

Dividing by δt and taking the limit, $\delta t \longrightarrow 0$:

$$\begin{aligned}\frac{dN(C_{i_j}, t)}{dt} = - \sum_{(\alpha,r)\in Ex(\mathcal{C}_{i_j})} r \times \min_{\mathcal{C}_{k_l}\in pre(\alpha,r)} (N(C_{k_l}, t)) \\ + \sum_{(\alpha,r)\in En(\mathcal{C}_{i_j})} r \times \min_{\mathcal{C}_{k_l}\in pre(\alpha,r)} (N(C_{k_l}, t))\end{aligned}$$

Activity matrix

Derivation of the system of ODEs representing the PEPA model can proceed via the activity matrix which records the influence of each activity on each component type/derivative.

The matrix has one row for each component type and one column for each activity type.

One ODE is generated corresponding to each row of the matrix, taking into account the negative entries in the non-zero columns as these are the components for which this is an exit activity.

Activity matrix for the small example

	$task_1$	$task_2$	$reset$	
$Proc_0$	-1	$+1$	0	x_1
$Proc_1$	$+1$	-1	0	x_2
Res_0	-1	0	$+1$	x_3
Res_1	$+1$	0	-1	x_4

Activity matrix to ODEs

The entry in the (i, j)-th position in the matrix can be $-1, 0$, or 1.

- If the entry is -1 it means that this local state undertakes an activity of that type and so when the activity is completed there will be one less instance of this local state.

Activity matrix to ODEs

The entry in the (i, j)-th position in the matrix can be $-1, 0$, or 1.

- If the entry is -1 it means that this local state undertakes an activity of that type and so when the activity is completed there will be one less instance of this local state.
- If the entry is 0 this local state is not involved in this activity.

Activity matrix to ODEs

The entry in the (i, j)-th position in the matrix can be $-1, 0$, or 1.

- If the entry is -1 it means that this local state undertakes an activity of that type and so when the activity is completed there will be one less instance of this local state.
- If the entry is 0 this local state is not involved in this activity.
- If the entry is 1 it means that this local state is produced when the activity of that type is completed, so there will be one more instance of this local state.

ODEs

$$
\begin{aligned}
\frac{dx_1(t)}{dt} &= -r_1 \min(x_1(t), x_3(t)) + r_2 x_2(t) \\
\frac{dx_2(t)}{dt} &= r_1 \min(x_1(t), x_3(t)) - r_2 x_2(t) \\
\frac{dx_3(t)}{dt} &= -r_1 \min(x_1(t), x_3(t)) + s x_4(t) \\
\frac{dx_4(t)}{dt} &= x_1 \min(x_1(t), x_3(t)) - s x_4(t)
\end{aligned}
$$

- The form of ODEs is independent of the number of instances of components in the model.

ODEs

$$
\begin{aligned}
\frac{dx_1(t)}{dt} &= -r_1 \min(x_1(t), x_3(t)) + r_2 x_2(t) \\
\frac{dx_2(t)}{dt} &= r_1 \min(x_1(t), x_3(t)) - r_2 x_2(t) \\
\frac{dx_3(t)}{dt} &= -r_1 \min(x_1(t), x_3(t)) + s x_4(t) \\
\frac{dx_4(t)}{dt} &= x_1 \min(x_1(t), x_3(t)) - s x_4(t)
\end{aligned}
$$

- The form of ODEs is independent of the number of instances of components in the model.
- The only impact of changing the number of instances is to alter the initial conditions.

Initialising the ODEs

Consider the model $Proc_0[100] \underset{\{task1\}}{\bowtie} Res_0[80]$.

There are initially 100 processors, all starting in state $Proc_0$ and 80 resources, all of which start in state Res_0.

Initialising the ODEs

Consider the model $Proc_0[100] \underset{\{task1\}}{\bowtie} Res_0[80]$.

There are initially 100 processors, all starting in state $Proc_0$ and 80 resources, all of which start in state Res_0.

Then we set the initial conditions of the ODEs to be:

$$x_1(0) = 100 \quad x_2(0) = 0 \quad x_3(0) = 80 \quad x_4(0) = 0$$

Initialising the ODEs

Consider the model $Proc_0[100] \underset{\{task1\}}{\bowtie} Res_0[80]$.

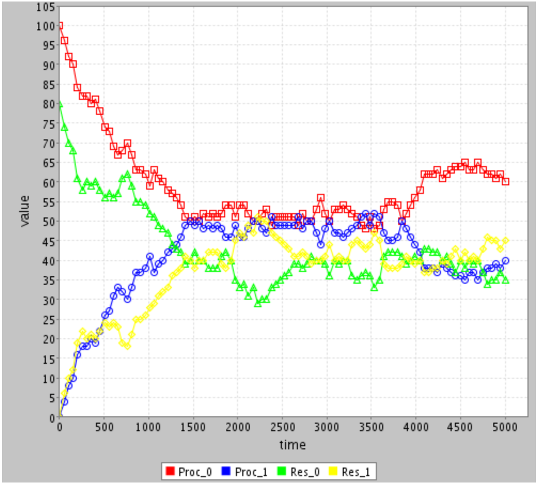
There are initially 100 processors, all starting in state $Proc_0$ and 80 resources, all of which start in state Res_0.

Then we set the initial conditions of the ODEs to be:

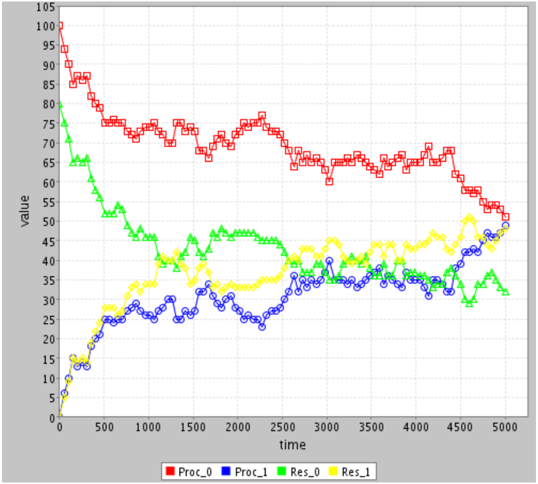
$$x_1(0) = 100 \quad x_2(0) = 0 \quad x_3(0) = 80 \quad x_4(0) = 0$$

The system of ODEs can then be given to any suitable numerical solver as an initial value problem.

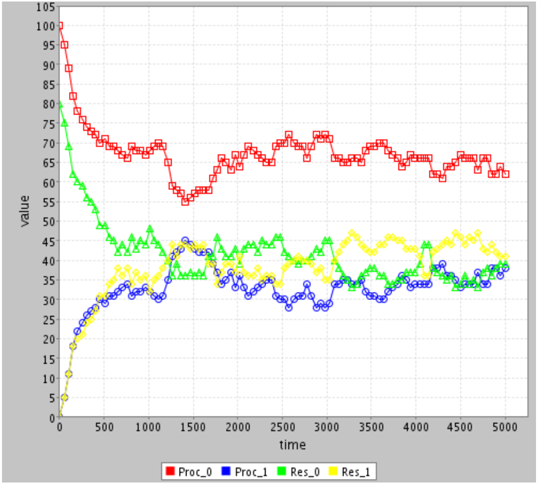
100 processors and 80 resources (simulation run A)



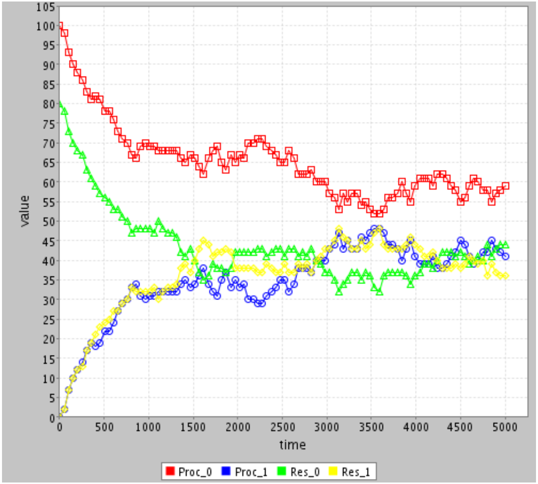
100 processors and 80 resources (simulation run B)



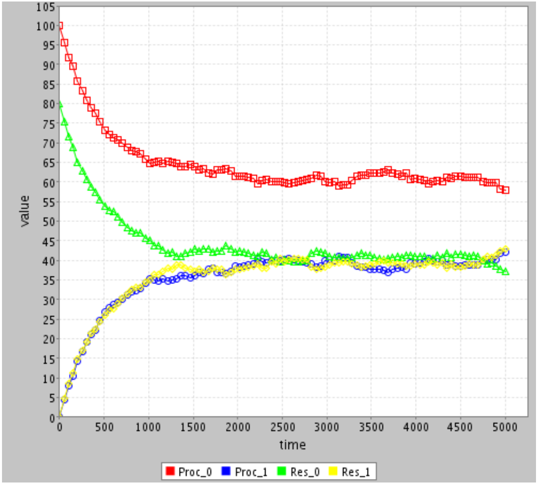
100 processors and 80 resources (simulation run C)



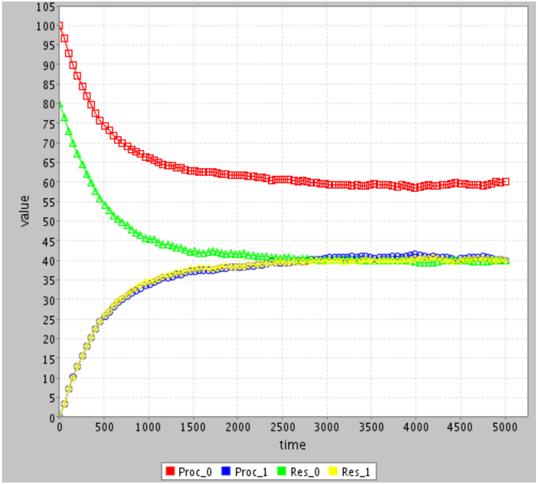
100 processors and 80 resources (simulation run D)



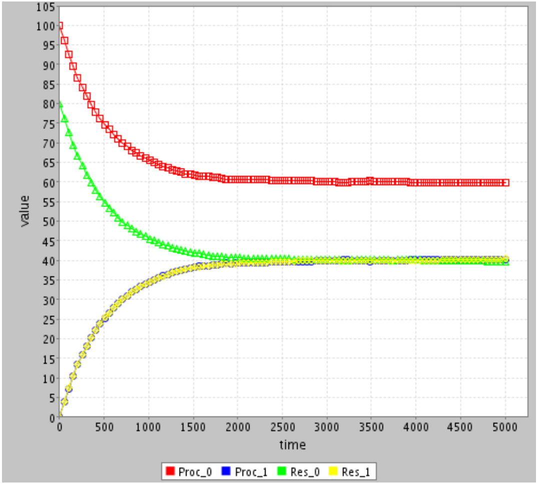
100 processors and 80 resources (average of 10 runs)



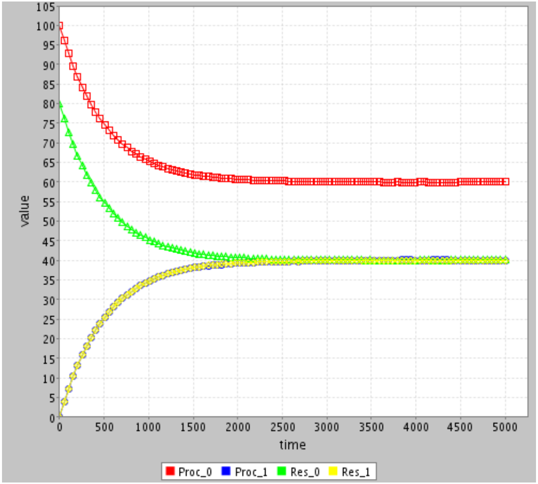
100 Processors and 80 resources (average of 100 runs)



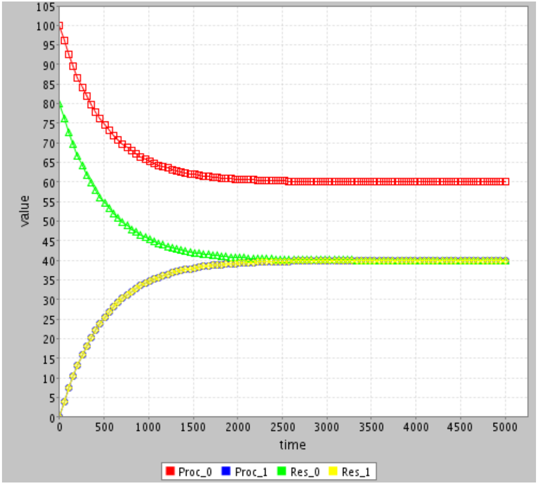
100 processors and 80 resources (average of 1000 runs)



100 processors and 80 resources (average of 10000 runs)

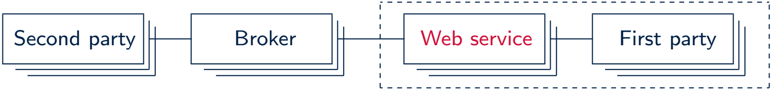


100 processors and 80 resources (ODE solution)



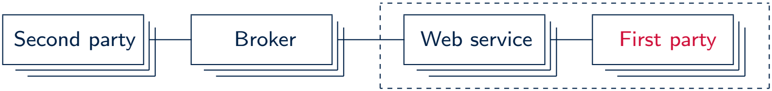
Outline

Example: Secure Web Service use



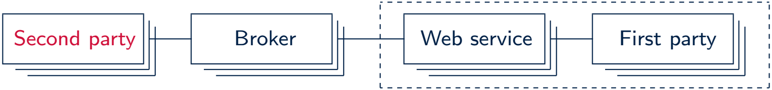
- The example which we consider is a Web service which has two types of clients:
  - first party application clients which access the web service across a secure intranet, and
  - second party browser clients which access the Web service across the Internet.
- Second party clients route their service requests via trusted brokers.

Example: Secure Web Service use



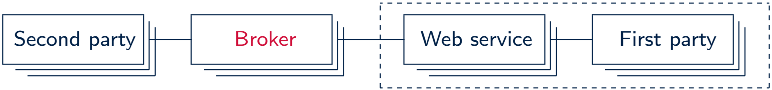
- The example which we consider is a Web service which has two types of clients:
  - first party application clients which access the web service across a secure intranet, and
  - second party browser clients which access the Web service across the Internet.
- Second party clients route their service requests via trusted brokers.

Example: Secure Web Service use



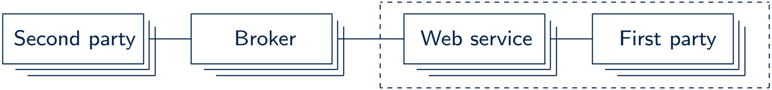
- The example which we consider is a Web service which has two types of clients:
  - first party application clients which access the web service across a secure intranet, and
  - second party browser clients which access the Web service across the Internet.
- Second party clients route their service requests via trusted brokers.

Example: Secure Web Service use



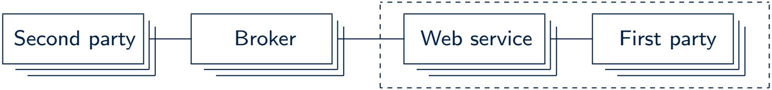
- The example which we consider is a Web service which has two types of clients:
  - first party application clients which access the web service across a secure intranet, and
  - second party browser clients which access the Web service across the Internet.
- Second party clients route their service requests via trusted brokers.

Scalability and replication



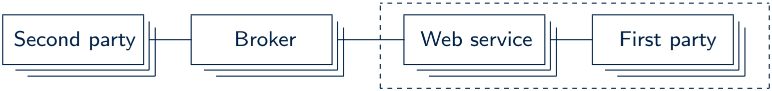
- To ensure scalability the Web service is replicated across multiple hosts.

Scalability and replication



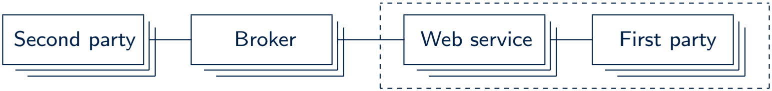
- To ensure scalability the Web service is replicated across multiple hosts.
- Multiple brokers are available.

Scalability and replication



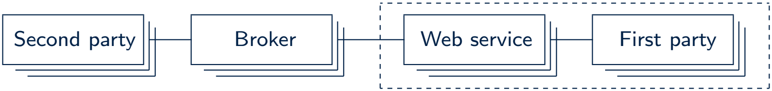
- To ensure scalability the Web service is replicated across multiple hosts.
- Multiple brokers are available.
- There are numerous first party clients behind the firewall using the service via remote method invocations across the secure intranet.

Scalability and replication



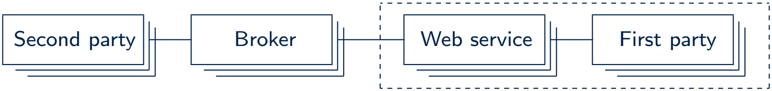
- To ensure scalability the Web service is replicated across multiple hosts.
- Multiple brokers are available.
- There are numerous first party clients behind the firewall using the service via remote method invocations across the secure intranet.
- There are numerous second party clients outside the firewall.

Security and use of encryption



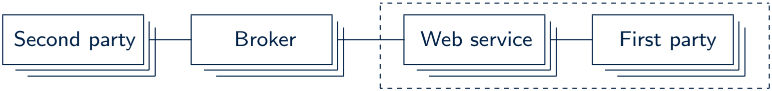
- Second party clients need to use encryption to ensure authenticity and confidentiality. First party clients do not.

Security and use of encryption



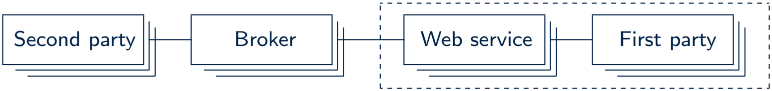
- Second party clients need to use encryption to ensure authenticity and confidentiality. First party clients do not.
- Brokers add decryption and encryption steps to build end-to-end security from point-to-point security.

Security and use of encryption



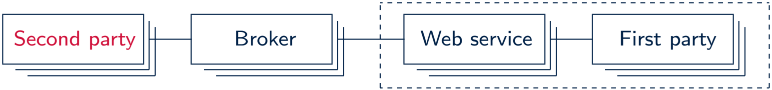
- Second party clients need to use encryption to ensure authenticity and confidentiality. First party clients do not.
- Brokers add decryption and encryption steps to build end-to-end security from point-to-point security.
  - When processing a request from a second party client brokers decrypt the request before re-encrypting it for the Web service.

Security and use of encryption



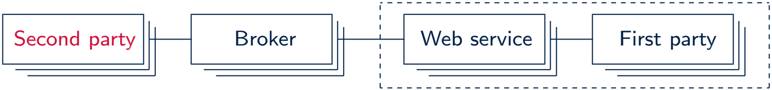
- Second party clients need to use encryption to ensure authenticity and confidentiality. First party clients do not.
- Brokers add decryption and encryption steps to build end-to-end security from point-to-point security.
    - When processing a request from a second party client brokers decrypt the request before re-encrypting it for the Web service.
    - When the response to a request is returned to the broker it decrypts the response before re-encrypting it for the client.

PEPA model: Second party clients and Brokers



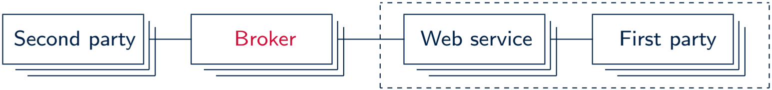
- A second party client composes service requests, encrypts these and sends them to its broker.

PEPA model: Second party clients and Brokers



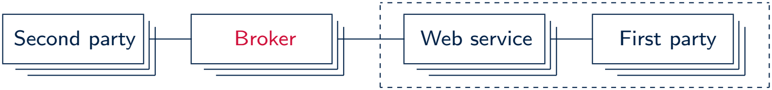
- A second party client composes service requests, encrypts these and sends them to its broker.
- It then waits for a response from the broker.

PEPA model: Second party clients and Brokers



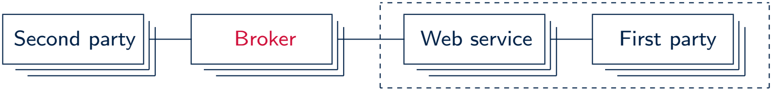
- A second party client composes service requests, encrypts these and sends them to its broker.
- It then waits for a response from the broker.
- Brokers add decryption and encryption steps to build end-to-end security from point-to-point security.

PEPA model: Second party clients and Brokers



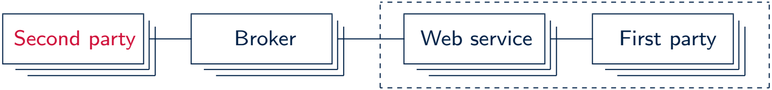
- A second party client composes service requests, encrypts these and sends them to its broker.
- It then waits for a response from the broker.
- Brokers add decryption and encryption steps to build end-to-end security from point-to-point security.
- The broker forwards the request to the Web service and then waits for a response.

PEPA model: Second party clients and Brokers



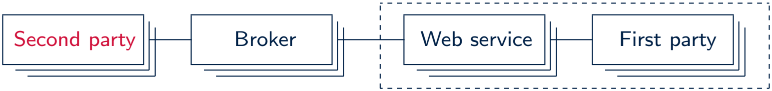
- A second party client composes service requests, encrypts these and sends them to its broker.
- It then waits for a response from the broker.
- Brokers add decryption and encryption steps to build end-to-end security from point-to-point security.
- The broker forwards the request to the Web service and then waits for a response.
- Decryption and re-encrytion are performed before returning the response to the client.

PEPA model: Second party clients



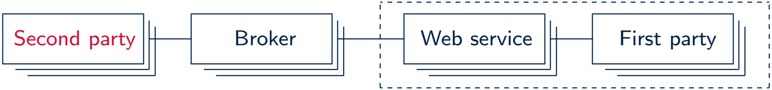
- A second party client composes service requests, encrypts these and sends them to its broker.

PEPA model: Second party clients



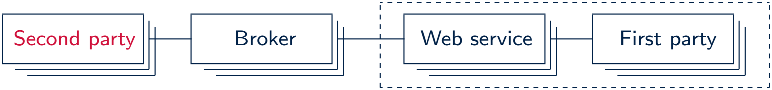
- A second party client composes service requests, encrypts these and sends them to its broker.
- It then waits for a response from the broker.

PEPA model: Second party clients



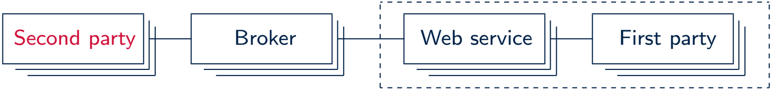
- A second party client composes service requests, encrypts these and sends them to its broker.
- It then waits for a response from the broker.
- The rate at which the first three activities happen is under the control of the client.

PEPA model: Second party clients



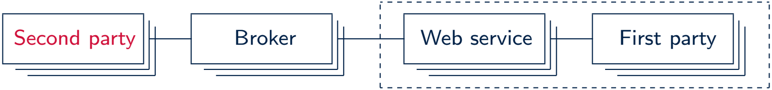
- A second party client composes service requests, encrypts these and sends them to its broker.
- It then waits for a response from the broker.
- The rate at which the first three activities happen is under the control of the client.
- The rate at which responses are produced is determined by the interaction of the broker and the service endpoint.

PEPA model: Second party clients



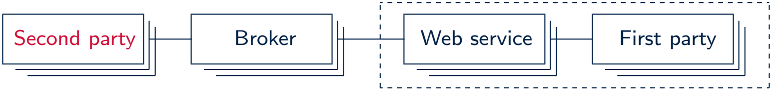
$$
\begin{aligned}
SPC_{idle} &\stackrel{def}{=} (compose_{sp}, r_{sp\_cmp}).SPC_{enc} \\
SPC_{enc} &\stackrel{def}{=} (encrypt_{b}, r_{sp\_encb}).SPC_{sending} \\
SPC_{sending} &\stackrel{def}{=} (request_{b}, r_{sp\_req}).SPC_{waiting} \\
SPC_{waiting} &\stackrel{def}{=} (response_{b}, \top).SPC_{dec} \\
SPC_{dec} &\stackrel{def}{=} (decrypt_{b}, r_{sp\_decb}).SPC_{idle}
\end{aligned}
$$

PEPA model: Second party clients



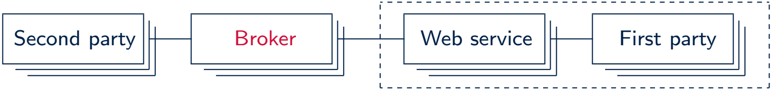
$$
\begin{aligned}
SPC_{idle} &\stackrel{def}{=} (compose_{sp}, r_{sp\_cmp}).SPC_{enc} \\
SPC_{enc} &\stackrel{def}{=} (encrypt_{b}, r_{sp\_encb}).SPC_{sending} \\
SPC_{sending} &\stackrel{def}{=} (request_{b}, r_{sp\_req}).SPC_{waiting} \\
SPC_{waiting} &\stackrel{def}{=} (response_{b}, \top).SPC_{dec} \\
SPC_{dec} &\stackrel{def}{=} (decrypt_{b}, r_{sp\_decb}).SPC_{idle}
\end{aligned}
$$

PEPA model: Second party clients



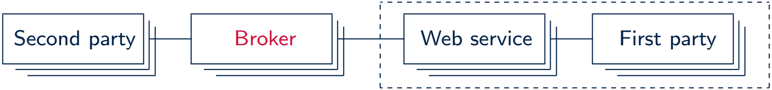
$$
\begin{aligned}
SPC_{idle} &\stackrel{def}{=} (compose_{sp}, r_{sp\_cmp}).SPC_{enc} \\
SPC_{enc} &\stackrel{def}{=} (encrypt_{b}, r_{sp\_encb}).SPC_{sending} \\
SPC_{sending} &\stackrel{def}{=} (request_{b}, r_{sp\_req}).SPC_{waiting} \\
SPC_{waiting} &\stackrel{def}{=} (response_{b}, \top).SPC_{dec} \\
SPC_{dec} &\stackrel{def}{=} (decrypt_{b}, r_{sp\_decb}).SPC_{idle}
\end{aligned}
$$

PEPA model: Brokers



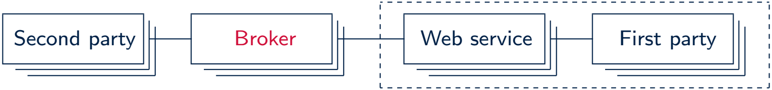
- The broker is inactive until it receives a request.

PEPA model: Brokers



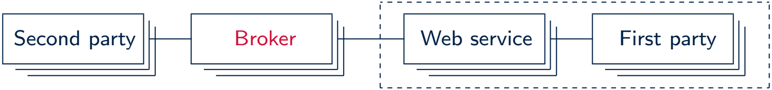
- The broker is inactive until it receives a request.
- It then decrypts the request before re-encrypting it for the Web service to ensure end-to-end security.

PEPA model: Brokers



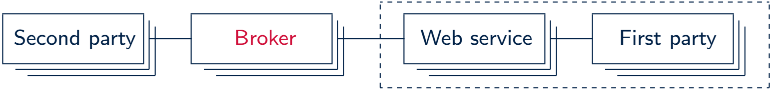
- The broker is inactive until it receives a request.
- It then decrypts the request before re-encrypting it for the Web service to ensure end-to-end security.
- It forwards the request to the Web service and then waits for a response.

PEPA model: Brokers



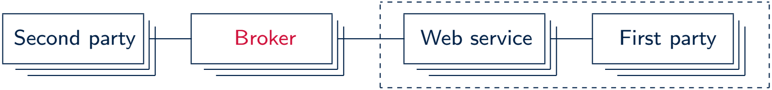
- The broker is inactive until it receives a request.
- It then decrypts the request before re-encrypting it for the Web service to ensure end-to-end security.
- It forwards the request to the Web service and then waits for a response.
- The corresponding decryption and re-encrytion are performed before returning the response to the client.

PEPA model: Brokers



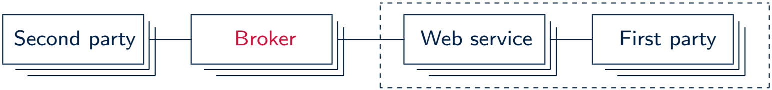
$$
\begin{aligned}
Broker_{idle} &\stackrel{def}{=} (request_b, \top).Broker_{dec\_input} \\
Broker_{dec\_input} &\stackrel{def}{=} (decrypt_{sp}, r_{b\_dec\_sp}).Broker_{enc\_input} \\
Broker_{enc\_input} &\stackrel{def}{=} (encrypt_{ws}, r_{b\_enc\_ws}).Broker_{sending} \\
Broker_{sending} &\stackrel{def}{=} (request_{ws}, r_{b\_req}).Broker_{waiting} \\
Broker_{waiting} &\stackrel{def}{=} (response_{ws}, \top).Broker_{dec\_resp} \\
Broker_{dec\_resp} &\stackrel{def}{=} (decrypt_{ws}, r_{b\_dec\_ws}).Broker_{enc\_resp} \\
Broker_{enc\_resp} &\stackrel{def}{=} (encrypt_{sp}, r_{b\_enc\_sp}).Broker_{replying} \\
Broker_{replying} &\stackrel{def}{=} (response_b, r_{b\_resp}).Broker_{idle}
\end{aligned}
$$

PEPA model: Brokers



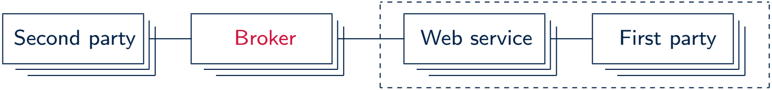
$$
\begin{aligned}
Broker_{idle} &\stackrel{def}{=} (request_b, \top).Broker_{dec\_input} \\
Broker_{dec\_input} &\stackrel{def}{=} (decrypt_{sp}, r_{b\_dec\_sp}).Broker_{enc\_input} \\
Broker_{enc\_input} &\stackrel{def}{=} (encrypt_{ws}, r_{b\_enc\_ws}).Broker_{sending} \\
Broker_{sending} &\stackrel{def}{=} (request_{ws}, r_{b\_req}).Broker_{waiting} \\
Broker_{waiting} &\stackrel{def}{=} (response_{ws}, \top).Broker_{dec\_resp} \\
Broker_{dec\_resp} &\stackrel{def}{=} (decrypt_{ws}, r_{b\_dec\_ws}).Broker_{enc\_resp} \\
Broker_{enc\_resp} &\stackrel{def}{=} (encrypt_{sp}, r_{b\_enc\_sp}).Broker_{replying} \\
Broker_{replying} &\stackrel{def}{=} (response_b, r_{b\_resp}).Broker_{idle}
\end{aligned}
$$

PEPA model: Brokers



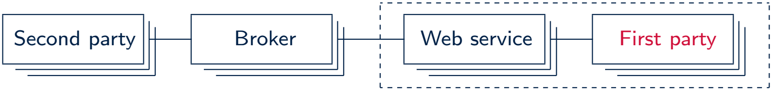
$$
\begin{aligned}
Broker_{idle} &\stackrel{def}{=} (request_b, \top).Broker_{dec\_input} \\
Broker_{dec\_input} &\stackrel{def}{=} (decrypt_{sp}, r_{b\_dec\_sp}).Broker_{enc\_input} \\
Broker_{enc\_input} &\stackrel{def}{=} (encrypt_{ws}, r_{b\_enc\_ws}).Broker_{sending} \\
Broker_{sending} &\stackrel{def}{=} (request_{ws}, r_{b\_req}).Broker_{waiting} \\
Broker_{waiting} &\stackrel{def}{=} (response_{ws}, \top).Broker_{dec\_resp} \\
Broker_{dec\_resp} &\stackrel{def}{=} (decrypt_{ws}, r_{b\_dec\_ws}).Broker_{enc\_resp} \\
Broker_{enc\_resp} &\stackrel{def}{=} (encrypt_{sp}, r_{b\_enc\_sp}).Broker_{replying} \\
Broker_{replying} &\stackrel{def}{=} (response_b, r_{b\_resp}).Broker_{idle}
\end{aligned}
$$

PEPA model: Brokers



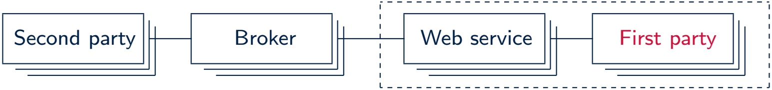
$$
\begin{aligned}
Broker_{idle} &\stackrel{def}{=} (request_b, \top).Broker_{dec\_input} \\
Broker_{dec\_input} &\stackrel{def}{=} (decrypt_{sp}, r_{b\_dec\_sp}).Broker_{enc\_input} \\
Broker_{enc\_input} &\stackrel{def}{=} (encrypt_{ws}, r_{b\_enc\_ws}).Broker_{sending} \\
Broker_{sending} &\stackrel{def}{=} (request_{ws}, r_{b\_req}).Broker_{waiting} \\
Broker_{waiting} &\stackrel{def}{=} (response_{ws}, \top).Broker_{dec\_resp} \\
Broker_{dec\_resp} &\stackrel{def}{=} (decrypt_{ws}, r_{b\_dec\_ws}).Broker_{enc\_resp} \\
Broker_{enc\_resp} &\stackrel{def}{=} (encrypt_{sp}, r_{b\_enc\_sp}).Broker_{replying} \\
Broker_{replying} &\stackrel{def}{=} (response_b, r_{b\_resp}).Broker_{idle}
\end{aligned}
$$

PEPA model: First party clients



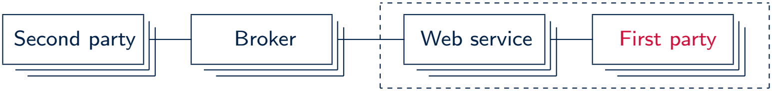
- The lifetime of a first party client mirrors that of a second party client except that encryption need not be used when all of the communication is conducted across a secure intranet.

PEPA model: First party clients



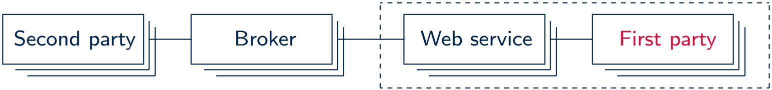
- The lifetime of a first party client mirrors that of a second party client except that encryption need not be used when all of the communication is conducted across a secure intranet.
- Also the service may be invoked by a remote method invocation to the host machine instead of via HTTP.

PEPA model: First party clients



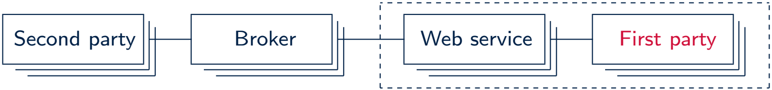
- The lifetime of a first party client mirrors that of a second party client except that encryption need not be used when all of the communication is conducted across a secure intranet.
- Also the service may be invoked by a remote method invocation to the host machine instead of via HTTP.
- Thus the first party client experiences the Web service as a blocking remote method invocation.

PEPA model: First party clients



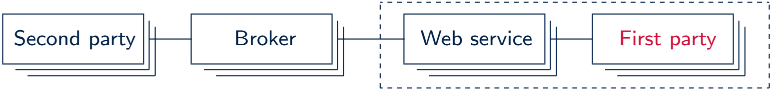
$$\begin{aligned} FPC_{idle} &\stackrel{def}{=} (compose_{fp}, r_{fp\_cmp}).FPC_{calling} \\ FPC_{calling} &\stackrel{def}{=} (invoke_{ws}, r_{fp\_inv}).FPC_{blocked} \\ FPC_{blocked} &\stackrel{def}{=} (result_{ws}, \top).FPC_{idle} \end{aligned}$$

PEPA model: First party clients



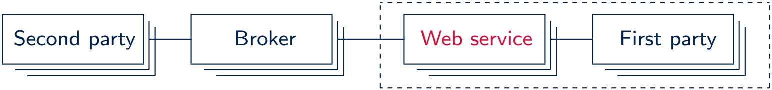
$$
\begin{aligned}
FPC_{idle} &\stackrel{def}{=} (compose_{fp}, r_{fp\_cmp}).FPC_{calling} \\
FPC_{calling} &\stackrel{def}{=} (invoke_{ws}, r_{fp\_inv}).FPC_{blocked} \\
FPC_{blocked} &\stackrel{def}{=} (result_{ws}, \top).FPC_{idle}
\end{aligned}
$$

PEPA model: First party clients



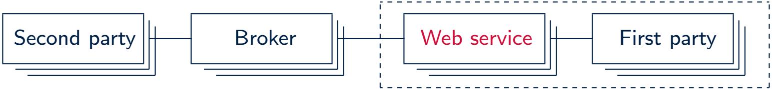
$$
\begin{aligned}
FPC_{idle} &\stackrel{def}{=} (compose_{fp}, r_{fp\_cmp}).FPC_{calling} \\
FPC_{calling} &\stackrel{def}{=} (invoke_{ws}, r_{fp\_inv}).FPC_{blocked} \\
FPC_{blocked} &\stackrel{def}{=} (result_{ws}, \top).FPC_{idle}
\end{aligned}
$$

PEPA model: Web service



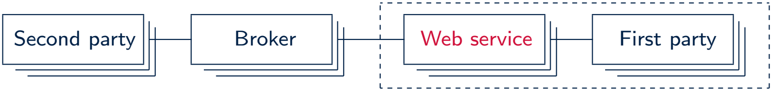
- There are two ways in which the service is executed, leading to a choice in the process algebra model taking the service process into one or other of its two modes of execution.

PEPA model: Web service



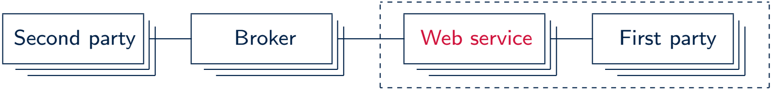
- There are two ways in which the service is executed, leading to a choice in the process algebra model taking the service process into one or other of its two modes of execution.
- In either case, the duration of the execution of the service itself is unchanged.

PEPA model: Web service



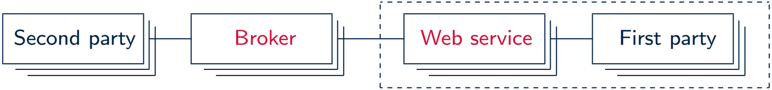
- There are two ways in which the service is executed, leading to a choice in the process algebra model taking the service process into one or other of its two modes of execution.
- In either case, the duration of the execution of the service itself is unchanged.
- The difference is only in whether encryption is needed and whether the result is delivered via HTTP or not.

PEPA model: Web service



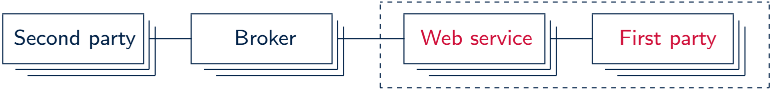
$$
\begin{aligned}
WS_{idle} &\stackrel{def}{=} (request_{ws}, \top).WS_{decoding} \\
&\phantom{\stackrel{def}{=}}+ (invoke_{ws}, \top).WS_{method} \\
WS_{decoding} &\stackrel{def}{=} (decryptReq_{ws}, r_{ws\_dec\_b}).WS_{execution} \\
WS_{execution} &\stackrel{def}{=} (execute_{ws}, r_{ws\_exec}).WS_{securing} \\
WS_{securing} &\stackrel{def}{=} (encryptResp_{ws}, r_{ws\_enc\_b}).WS_{responding} \\
WS_{responding} &\stackrel{def}{=} (response_{ws}, r_{ws\_resp\_b}).WS_{idle} \\
WS_{method} &\stackrel{def}{=} (execute_{ws}, r_{ws\_exec}).WS_{returning} \\
WS_{returning} &\stackrel{def}{=} (result_{ws}, r_{ws\_res}).WS_{idle}
\end{aligned}
$$

PEPA model: Web service



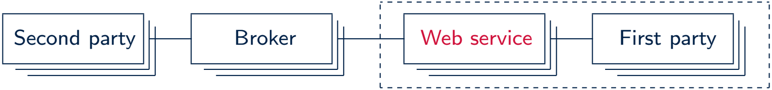
$$
\begin{aligned}
WS_{idle} &\stackrel{def}{=} (request_{ws}, \top).WS_{decoding} \\
&+ (invoke_{ws}, \top).WS_{method} \\
WS_{decoding} &\stackrel{def}{=} (decryptReq_{ws}, r_{ws\_dec\_b}).WS_{execution} \\
WS_{execution} &\stackrel{def}{=} (execute_{ws}, r_{ws\_exec}).WS_{securing} \\
WS_{securing} &\stackrel{def}{=} (encryptResp_{ws}, r_{ws\_enc\_b}).WS_{responding} \\
WS_{responding} &\stackrel{def}{=} (response_{ws}, r_{ws\_resp\_b}).WS_{idle} \\
WS_{method} &\stackrel{def}{=} (execute_{ws}, r_{ws\_exec}).WS_{returning} \\
WS_{returning} &\stackrel{def}{=} (result_{ws}, r_{ws\_res}).WS_{idle}
\end{aligned}
$$

PEPA model: Web service



$$
\begin{aligned}
WS_{idle} &\stackrel{def}{=} (request_{ws}, \top).WS_{decoding} \\
&\quad + (invoke_{ws}, \top).WS_{method} \\
WS_{decoding} &\stackrel{def}{=} (decryptReq_{ws}, r_{ws\_dec\_b}).WS_{execution} \\
WS_{execution} &\stackrel{def}{=} (execute_{ws}, r_{ws\_exec}).WS_{securing} \\
WS_{securing} &\stackrel{def}{=} (encryptResp_{ws}, r_{ws\_enc\_b}).WS_{responding} \\
WS_{responding} &\stackrel{def}{=} (response_{ws}, r_{ws\_resp\_b}).WS_{idle} \\
WS_{method} &\stackrel{def}{=} (execute_{ws}, r_{ws\_exec}).WS_{returning} \\
WS_{returning} &\stackrel{def}{=} (result_{ws}, r_{ws\_res}).WS_{idle}
\end{aligned}
$$

PEPA model: Web service



$$
\begin{aligned}
WS_{idle} &\stackrel{def}{=} (request_{ws}, \top).WS_{decoding} \\
&\phantom{\stackrel{def}{=}}+ (invoke_{ws}, \top).WS_{method} \\
WS_{decoding} &\stackrel{def}{=} (decryptReq_{ws}, r_{ws\_dec\_b}).WS_{execution} \\
WS_{execution} &\stackrel{def}{=} (execute_{ws}, r_{ws\_exec}).WS_{securing} \\
WS_{securing} &\stackrel{def}{=} (encryptResp_{ws}, r_{ws\_enc\_b}).WS_{responding} \\
WS_{responding} &\stackrel{def}{=} (response_{ws}, r_{ws\_resp\_b}).WS_{idle} \\
WS_{method} &\stackrel{def}{=} (execute_{ws}, r_{ws\_exec}).WS_{returning} \\
WS_{returning} &\stackrel{def}{=} (result_{ws}, r_{ws\_res}).WS_{idle}
\end{aligned}
$$

PEPA model: System composition

In the initial state of the system model we represent each of the four component types being initially in their idle state.

$$System \stackrel{def}{=} (SPC_{idle} \underset{\mathcal{K}}{\bowtie} Broker_{idle}) \underset{\mathcal{L}}{\bowtie} (WS_{idle} \underset{\mathcal{M}}{\bowtie} FPC_{idle})$$

$$\text{where} \quad \begin{aligned} \mathcal{K} &= \{\, request_b, response_b \,\} \\ \mathcal{L} &= \{\, request_{ws}, response_{ws} \,\} \\ \mathcal{M} &= \{\, invoke_{ws}, result_{ws} \,\} \end{aligned}$$

PEPA model: System composition

In the initial state of the system model we represent each of the four component types being initially in their idle state.

$$System \stackrel{def}{=} (SPC_{idle} \underset{\mathcal{K}}{\bowtie} Broker_{idle}) \underset{\mathcal{L}}{\bowtie} (WS_{idle} \underset{\mathcal{M}}{\bowtie} FPC_{idle})$$

$$\text{where} \quad \begin{array}{rcl} \mathcal{K} & = & \{\, request_b, response_b \,\} \\ \mathcal{L} & = & \{\, request_{ws}, response_{ws} \,\} \\ \mathcal{M} & = & \{\, invoke_{ws}, result_{ws} \,\} \end{array}$$

This model represents the smallest possible instance of the system, where there is one instance of each component type. We evaluate the system as the number of clients, brokers, and copies of the service increase.

Cost of analysis

- We compare ODE-based evaluation against other techniques which could be used to analyse the model.

Cost of analysis

- We compare ODE-based evaluation against other techniques which could be used to analyse the model.
  - Steady-state and transient analysis as implemented by the PRISM probabilistic model-checker.

Cost of analysis

- We compare ODE-based evaluation against other techniques which could be used to analyse the model.
  - Steady-state and transient analysis as implemented by the PRISM probabilistic model-checker.
  - Monte Carlo Markov Chain simulation (a Java implementation of Gillespie's Direct Method).

Running times from analyses (in seconds)

Second party clients	Brokers	Web service instances	First party clients	Number of states in the full state-space	Number of states in the aggregated state-space	Sparse matrix steady-state	Matrix/MTBDD steady-state	Transient solution for time $t = 100$	MCMC simulation one run to $t = 100$	ODE solution
1	1	1	1	48	48	1.04	1.10	1.01	2.47	2.81

Running times from analyses (in seconds)

Second party clients	Brokers	Web service instances	First party clients	Number of states in the full state-space	Number of states in the aggregated state-space	Sparse matrix steady-state	Matrix/MTBDD steady-state	Transient solution for time $t = 100$	MCMC simulation one run to $t = 100$	ODE solution
1	1	1	1	48	48	1.04	1.10	1.01	2.47	2.81
2	2	2	2	6,304	860	2.15	2.26	2.31	2.45	2.81

Running times from analyses (in seconds)

Second party clients	Brokers	Web service instances	First party clients	Number of states in the full state-space	Number of states in the aggregated state-space	Sparse matrix steady-state	Matrix/MTBDD steady-state	Transient solution for time $t = 100$	MCMC simulation one run to $t = 100$	ODE solution
1	1	1	1	48	48	1.04	1.10	1.01	2.47	2.81
2	2	2	2	6,304	860	2.15	2.26	2.31	2.45	2.81
3	3	3	3	1,130,496	161,296	172.48	255.48	588.80	2.48	2.83

Running times from analyses (in seconds)

Second party clients	Brokers	Web service instances	First party clients	Number of states in the full state-space	Number of states in the aggregated state-space	Sparse matrix steady-state	Matrix/MTBDD steady-state	Transient solution for time $t = 100$	MCMC simulation one run to $t = 100$	ODE solution
1	1	1	1	48	48	1.04	1.10	1.01	2.47	2.81
2	2	2	2	6,304	860	2.15	2.26	2.31	2.45	2.81
3	3	3	3	1,130,496	161,296	172.48	255.48	588.80	2.48	2.83
4	4	4	4	$>$234M	–	–	–	–	2.44	2.85

Running times from analyses (in seconds)

Second party clients	Brokers	Web service instances	First party clients	Number of states in the full state-space	Number of states in the aggregated state-space	Sparse matrix steady-state	Matrix/MTBDD steady-state	Transient solution for time $t = 100$	MCMC simulation one run to $t = 100$	ODE solution
1	1	1	1	48	48	1.04	1.10	1.01	2.47	2.81
2	2	2	2	6,304	860	2.15	2.26	2.31	2.45	2.81
3	3	3	3	1,130,496	161,296	172.48	255.48	588.80	2.48	2.83
4	4	4	4	>234M	–	–	–	–	2.44	2.85
100	100	100	100	–	–	–	–	–	2.78	2.78

Running times from analyses (in seconds)

Second party clients	Brokers	Web service instances	First party clients	Number of states in the full state-space	Number of states in the aggregated state-space	Sparse matrix steady-state	Matrix/MTBDD steady-state	Transient solution for time $t = 100$	MCMC simulation one run to $t = 100$	ODE solution
1	1	1	1	48	48	1.04	1.10	1.01	2.47	2.81
2	2	2	2	6,304	860	2.15	2.26	2.31	2.45	2.81
3	3	3	3	1,130,496	161,296	172.48	255.48	588.80	2.48	2.83
4	4	4	4	>234M	–	–	–	–	2.44	2.85
100	100	100	100	–	–	–	–	–	2.78	2.78
1000	100	500	1000	–	–	–	–	–	3.72	2.77

Running times from analyses (in seconds)

Second party clients	Brokers	Web service instances	First party clients	Number of states in the full state-space	Number of states in the aggregated state-space	Sparse matrix steady-state	Matrix/MTBDD steady-state	Transient solution for time $t = 100$	MCMC simulation one run to $t = 100$	ODE solution
1	1	1	1	48	48	1.04	1.10	1.01	2.47	2.81
2	2	2	2	6,304	860	2.15	2.26	2.31	2.45	2.81
3	3	3	3	1,130,496	161,296	172.48	255.48	588.80	2.48	2.83
4	4	4	4	>234M	–	–	–	–	2.44	2.85
100	100	100	100	–	–	–	–	–	2.78	2.78
1000	100	500	1000	–	–	–	–	–	3.72	2.77
1000	1000	1000	1000	–	–	–	–	–	5.44	2.77

Running times from analyses (in seconds)

Second party clients	Brokers	Web service instances	First party clients	Number of states in the full state-space	Number of states in the aggregated state-space	Sparse matrix steady-state	Matrix/MTBDD steady-state	Transient solution for time $t = 100$	MCMC simulation one run to $t = 100$	ODE solution
1	1	1	1	48	48	1.04	1.10	1.01	2.47	2.81
2	2	2	2	6,304	860	2.15	2.26	2.31	2.45	2.81
3	3	3	3	1,130,496	161,296	172.48	255.48	588.80	2.48	2.83
4	4	4	4	>234M	–	–	–	–	2.44	2.85
100	100	100	100	–	–	–	–	–	2.78	2.78
1000	100	500	1000	–	–	–	–	–	3.72	2.77
1000	1000	1000	1000	–	–	–	–	–	5.44	2.77

Time series analysis via ODEs

- We assume a system in which the number of clients of both kinds, brokers, and web service instances are all 1000.

Time series analysis via ODEs

- We assume a system in which the number of clients of both kinds, brokers, and web service instances are all 1000.
- We present the results from our ODE integrator as time-series plots of the number of each type of component behaviour as a function of time, as time runs from $t = 0$ to $t = 100$.

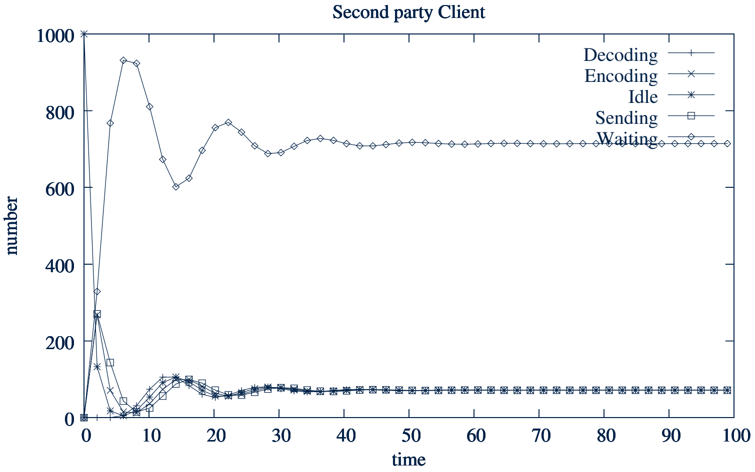
Time series analysis via ODEs

- We assume a system in which the number of clients of both kinds, brokers, and web service instances are all 1000.
- We present the results from our ODE integrator as time-series plots of the number of each type of component behaviour as a function of time, as time runs from $t = 0$ to $t = 100$.
- The graphs show fluctuations in the numbers of components with respect to time.

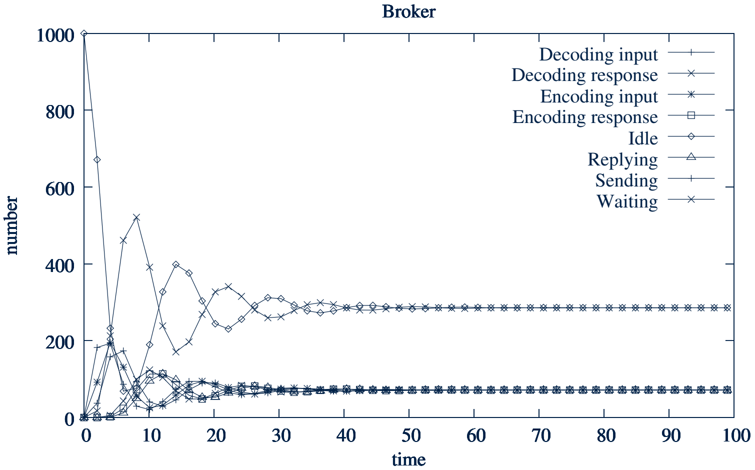
Time series analysis via ODEs

- We assume a system in which the number of clients of both kinds, brokers, and web service instances are all 1000.
- We present the results from our ODE integrator as time-series plots of the number of each type of component behaviour as a function of time, as time runs from $t = 0$ to $t = 100$.
- The graphs show fluctuations in the numbers of components with respect to time.
- We can observe an initial flurry of activity until the system stabilises into its steady-state equilibrium at time (around) $t = 50$.

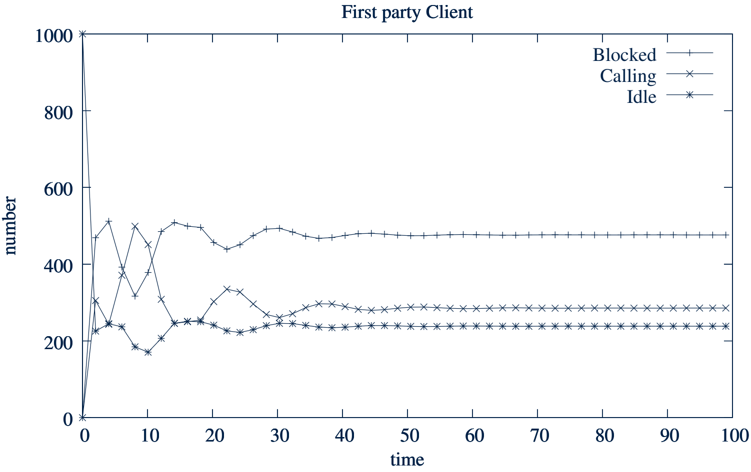
Second party clients



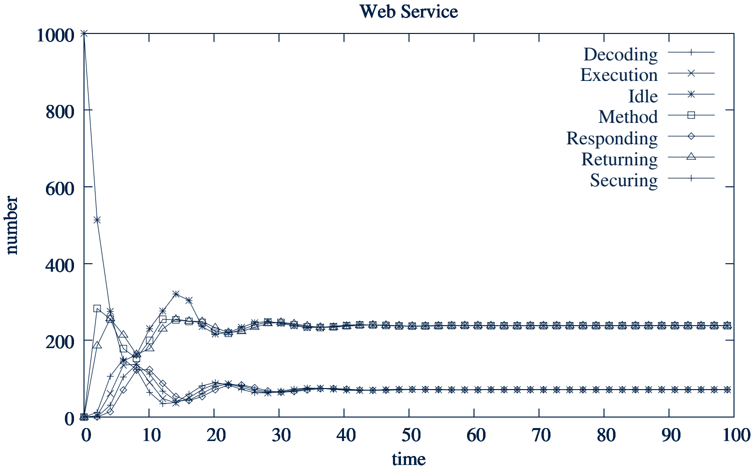
Brokers


Broker
number
time
0
200
400
600
800
1000
0
10
20
30
40
50
60
70
80
90
100
Decoding input
Decoding response
Encoding input
Encoding response
Idle
Replying
Sending
Waiting

First party clients



Web service


Web Service
number
time
0
200
400
600
800
1000
0
10
20
30
40
50
60
70
80
90
100
Decoding
Execution
Idle
Method
Responding
Returning
Securing

Outline

Deriving quantitative data

PEPA models can be analysed for quantified dynamic behaviour in a number of different ways.

Deriving quantitative data

PEPA models can be analysed for quantified dynamic behaviour in a number of different ways.

The language may be used to generate a Markov Process (CTMC).

Deriving quantitative data

PEPA models can be analysed for quantified dynamic behaviour in a number of different ways.

The language may be used to generate a Markov Process (CTMC).

PEPA MODEL $\xrightarrow{\text{SOS rules}}$ LABELLED TRANSITION SYSTEM $\xrightarrow[\text{diagram}]{\text{state transition}}$ CTMC **Q**

Deriving quantitative data

PEPA models can be analysed for quantified dynamic behaviour in a number of different ways.

The language may be used to generate a system of ordinary differential equations (ODEs).

Deriving quantitative data

PEPA models can be analysed for quantified dynamic behaviour in a number of different ways.

The language may be used to generate a system of ordinary differential equations (ODEs).

$$\text{PEPA MODEL} \xrightarrow[\text{analysis}]{\text{syntactic}} \text{ACTIVITY MATRIX} \xrightarrow[\text{interpretation}]{\text{continuous}} \text{ODEs}$$

Deriving quantitative data

PEPA models can be analysed for quantified dynamic behaviour in a number of different ways.

The language may be used to generate a system of ordinary differential equations (ODEs).

$$\text{PEPA MODEL} \xrightarrow[\text{analysis}]{\text{syntactic}} \text{ACTIVITY MATRIX} \xrightarrow[\text{interpretation}]{\text{continuous}} \text{ODEs}$$

Each of these has tool support so that the underlying model is derived automatically according to the predefined rules.