Process Algebra for Collective Dynamics

Jane Hillston

Laboratory for Foundations of Computer Science
University of Edinburgh

joint work with Stephen Gilmore and Mirco Tribastone
Outline

1. Introduction
   - Stochastic Process Algebra
   - Collective Dynamics

2. Continuous Approximation
   - State variables
   - Numerical illustration

3. Fluid-Flow Semantics
   - Fluid Structured Operational Semantics

4. Example
   - Scalable Web Services

5. Conclusions
   - Alternative Models
1 Introduction
   - Stochastic Process Algebra
   - Collective Dynamics

2 Continuous Approximation
   - State variables
   - Numerical illustration

3 Fluid-Flow Semantics
   - Fluid Structured Operational Semantics

4 Example
   - Scalable Web Services

5 Conclusions
   - Alternative Models
Process Algebra

- Models consist of **agents** which engage in **actions**.

\[ \alpha . P \]

- \( \alpha \): action type or name
- \( P \): agent/component
Models consist of **agents** which engage in **actions**.

\[ \alpha . P \]

- action type or name
- agent/component

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

- Models consist of agents which engage in actions.

\[ \alpha . P \]

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

Process algebra model
Process Algebra

- Models consist of agents which engage in actions.

\[ \alpha. P \]

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

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

- Models consist of agents which engage in actions.

\[ \alpha \cdot P \]

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

Process algebra model $\xrightarrow{\text{SOS rules}}$ Labelled transition system
A simple example: processors and resources

\[
\begin{align*}
    \text{Proc}_0 & \overset{\text{def}}{=} \text{task1}.\text{Proc}_1 \\
    \text{Proc}_1 & \overset{\text{def}}{=} \text{task2}.\text{Proc}_0 \\
    \text{Res}_0 & \overset{\text{def}}{=} \text{task1}.\text{Res}_1 \\
    \text{Res}_1 & \overset{\text{def}}{=} \text{reset}.\text{Res}_0
\end{align*}
\]

\[\text{Proc}_0 \parallel_{\text{task1}} \text{Res}_0\]
A simple example: processors and resources

\[
\begin{align*}
\text{Proc}_0 & \overset{\text{def}}{=} \text{task1}.\text{Proc}_1 \\
\text{Proc}_1 & \overset{\text{def}}{=} \text{task2}.\text{Proc}_0 \\
\text{Res}_0 & \overset{\text{def}}{=} \text{task1}.\text{Res}_1 \\
\text{Res}_1 & \overset{\text{def}}{=} \text{reset}.\text{Res}_0
\end{align*}
\]
Stochastic process algebras

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

- Models are constructed from components which engage in activities.

\[(\alpha, r).P\]

- action type or name
- activity rate (parameter of an exponential distribution)
- component/derivative
Stochastic Process Algebra

- Models are constructed from components which engage in activities.

\[(\alpha, r).P\]

- The language is used to generate a CTMC for performance modelling.
Stochastic Process Algebra

- Models are constructed from components which engage in activities.

\[(\alpha, r).P\]

- The language is used to generate a CTMC for performance modelling.

SPA MODEL
Stochastic Process Algebra

- Models are constructed from components which engage in activities.

\[(\alpha, r).P\]

- The language is used to generate a CTMC for performance modelling.
Stochastic Process Algebra

- Models are constructed from components which engage in activities.

\[(\alpha, r).P\]

- The language is used to generate a CTMC for performance modelling.
Stochastic Process Algebra

- Models are constructed from **components** which engage in activities.

\[(\alpha, r).P\]

- The language is used to generate a **CTMC** for performance modelling.

SPA MODEL → SOS rules → LABELLED TRANSITION SYSTEM → state transition diagram
Stochastic Process Algebra

- Models are constructed from components which engage in activities.

\[(\alpha, r).P\]

- The language is used to generate a CTMC for performance modelling.
Stochastic Process Algebra

- Models are constructed from components which engage in activities.

\[(\alpha, r).P\]

- The language is used to generate a CTMC for performance modelling.

SPA MODEL \xrightarrow{\text{SOS rules}} \text{LABELLED MULTI-TRANSITION SYSTEM} \xrightarrow{\text{state transition diagram}} \text{CTMC Q}
## Integrated analysis

**Qualitative** verification can now be complemented by **quantitative** verification.
Integrated analysis

**Qualitative** verification can now be complemented by **quantitative** verification.
Integrated analysis

Qualitative verification can now be complemented by quantitative verification.

Reachability analysis

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

Qualitative verification can now be complemented by quantitative verification.

Model checking

Does a given property $\phi$ hold within the system with a given probability?
Integrated analysis

**Qualitative** verification can now be complemented by **quantitative** verification.

**Model checking**

For a given starting state how long is it until a given property $\phi$ holds?
Performance Evaluation Process Algebra

PEPA components perform activities either independently or in co-operation with other components.
Performance Evaluation Process Algebra

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

\[(\alpha, f).P\]  Prefix

\[P_1 + P_2\]  Choice

\[P_1 \parallel P_2\]  Co-operation

\[P/L\]  Hiding

\[X\]  Variable
Performance Evaluation Process Algebra

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

\((\alpha, f).P\) Prefix

\(P_1 + P_2\) Choice

\(P_1 \parallel P_2\) Co-operation

\(P/L\) Hiding

\(X\) Variable
Performance Evaluation Process Algebra

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

\( (\alpha, f).P \) Prefix
\( P_1 + P_2 \) Choice
\( P_1 \parallel P_2 \) Co-operation
\( P/L \) Hiding
\( X \) Variable
Performance Evaluation Process Algebra

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

\[(\alpha, f).P\] Prefix
\[P_1 + P_2\] Choice
\[P_1 \boxdot P_2\] Co-operation
\[P/L\] Hiding
\[X\] Variable
Performance Evaluation Process Algebra

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

\[(\alpha, f).P\] Prefix
\[P_1 + P_2\] Choice
\[P_1 \parallel P_2\] Co-operation
\[P/L\] Hiding
\[X\] Variable
Performance Evaluation Process Algebra

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

\[(\alpha, f).P\] Prefix

\[P_1 + P_2\] Choice

\[P_1 \boxtimes P_2\] Co-operation

\[P/L\] Hiding

\[X\] Variable
PEPA components perform activities either independently or in co-operation with other components.

\[(\alpha, f).P\] Prefix

\[P_1 + P_2\] Choice

\[P_1 \otimes P_2\] Co-operation

\[P/L\] Hiding

\[X\] Variable

\[P_1 \parallel P_2\] is a derived form for \[P_1 \otimes \emptyset P_2\].
Performance Evaluation Process Algebra

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

\((\alpha, f).P\) Prefix

\(P_1 + P_2\) Choice

\(P_1 \boxtimes L P_2\) Co-operation

\(P/L\) Hiding

\(X\) Variable

\(P_1 \parallel P_2\) is a derived form for \(P_1 \boxtimes_{\emptyset} P_2\).

When working with large numbers of entities, we write \(P[n]\) to denote an array of \(n\) copies of \(P\) executing in parallel.
Performance Evaluation Process Algebra

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

\[(\alpha, f).P\]  Prefix

\[P_1 + P_2\]  Choice

\[P_1 \boxtimes P_2\]  Co-operation

\[P/L\]  Hiding

\[X\]  Variable

\[P_1 \parallel P_2\] is a derived form for \[P_1 \boxtimes P_2\].

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

\[P[5] \equiv (P \parallel P \parallel P \parallel P \parallel P)\]
Rates of interaction: bounded capacity

Stochastic process algebras differ in how they define the rate of synchronised actions.
Rates of interaction: bounded capacity

Stochastic process algebras differ in how they define the rate of synchronised actions.

In PEPA the cooperation of components is assumed to give rise to **shared actions** and the rates of these shared actions are governed by the assumption of **bounded capacity**.
Rates of interaction: bounded capacity

Stochastic process algebras differ in how they define the rate of synchronised actions.

In PEPA the cooperation of components is assumed to give rise to shared actions and the rates of these shared actions are governed by the assumption of bounded capacity.

The principle of bounded capacity means that a component cannot be made to carry out an action in cooperation faster than its own defined rate for the action. Thus shared actions proceed at the minimum of the rates in the participating components.
Rates of interaction: bounded capacity

Stochastic process algebras differ in how they define the rate of synchronised actions.

In PEPA the cooperation of components is assumed to give rise to shared actions and the rates of these shared actions are governed by the assumption of bounded capacity.

The principle of bounded capacity means that a component cannot be made to carry out an action in cooperation faster than its own defined rate for the action. Thus shared actions proceed at the minimum of the rates in the participating components.

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

\[
\begin{align*}
Proc_0 & \stackrel{\text{def}}{=} (\text{task1, } r_1).Proc_1 \\
Proc_1 & \stackrel{\text{def}}{=} (\text{task2, } r_2).Proc_0 \\
Res_0 & \stackrel{\text{def}}{=} (\text{task1, } r_3).Res_1 \\
Res_1 & \stackrel{\text{def}}{=} (\text{reset, } r_4).Res_0
\end{align*}
\]

\[R = \min(r_1, r_3)
\]

\[
Q = \begin{bmatrix}
- & - & R & 0 \\
- & - & 0 & -(r_2 + r_4) \\
0 & 0 & - & r_4 \\
0 & 0 & - & -
\end{bmatrix}
\]
A simple example: processors and resources

\[ Proc_0 \overset{\text{def}}{=} (\text{task1}, r_1).Proc_1 \]
\[ Proc_1 \overset{\text{def}}{=} (\text{task2}, r_2).Proc_0 \]
\[ Res_0 \overset{\text{def}}{=} (\text{task1}, r_3).Res_1 \]
\[ Res_1 \overset{\text{def}}{=} (\text{reset}, r_4).Res_0 \]

\[ R = \min(r_1, r_3) \]
A simple example: processors and resources

\[
\begin{align*}
Proc_0 & \overset{\text{def}}{=} (\text{task1}, r_1).Proc_1 \\
Proc_1 & \overset{\text{def}}{=} (\text{task2}, r_2).Proc_0 \\
Res_0 & \overset{\text{def}}{=} (\text{task1}, r_3).Res_1 \\
Res_1 & \overset{\text{def}}{=} (\text{reset}, r_4).Res_0
\end{align*}
\]

\[R = \min(r_1, r_3)\]
A simple example: processors and resources

\[ \begin{align*}
Proc_0 & \overset{\text{def}}{=} (\text{task1}, r_1) \cdot Proc_1 \\
Proc_1 & \overset{\text{def}}{=} (\text{task2}, r_2) \cdot Proc_0 \\
Res_0 & \overset{\text{def}}{=} (\text{task1}, r_3) \cdot Res_1 \\
Res_1 & \overset{\text{def}}{=} (\text{reset}, r_4) \cdot Res_0
\end{align*} \]

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

\[ R = \min(r_1, r_3) \]
Calculating the rate of actions carried out in cooperation

\[(P_0 \parallel P_0) \{task1\} (R_0 \parallel R_0 \parallel R_0)\]

\[(P_1 \parallel P_0) \{task1\} (R_1 \parallel R_0 \parallel R_0)\]

\[(P_1 \parallel P_0) \{task1\} (R_0 \parallel R_1 \parallel R_0)\]

\[(P_0 \parallel P_0) \{task1\} (R_0 \parallel R_0 \parallel R_0)\]

\[(P_0 \parallel P_0) \{task1\} (R_0 \parallel R_0 \parallel R_1)\]

\[(P_0 \parallel P_0) \{task1\} (R_0 \parallel R_1 \parallel R_0)\]

\[(P_0 \parallel P_1) \{task1\} (R_1 \parallel R_0 \parallel R_0)\]

\[(P_0 \parallel P_1) \{task1\} (R_0 \parallel R_1 \parallel R_0)\]

\[(P_0 \parallel P_1) \{task1\} (R_0 \parallel R_0 \parallel R_1)\]
Calculating the rate of actions carried out in cooperation

\[ r = \frac{n_1}{2r_1} \frac{n_3}{3r_3} \min(2r_1, 3r_3) = \frac{1}{6} \min(2r_1, 3r_3) \]

\[ \begin{align*}
    (P_0 \parallel P_0) \{\text{task1}\} (R_0 \parallel R_0 \parallel R_0) \\
    (P_1 \parallel P_0) \{\text{task1}\} (R_0 \parallel R_0 \parallel R_0) \\
    (P_1 \parallel P_0) \{\text{task1}\} (R_0 \parallel R_1 \parallel R_0) \\
    (P_0 \parallel P_1) \{\text{task1}\} (R_1 \parallel R_0 \parallel R_0) \\
    (P_0 \parallel P_1) \{\text{task1}\} (R_0 \parallel R_1 \parallel R_0) \\
    (P_0 \parallel P_1) \{\text{task1}\} (R_0 \parallel R_0 \parallel R_1)
\end{align*} \]
Collective Dynamics

The behaviour of many systems can be interpreted as the result of the collective behaviour of a large number of interacting entities.
Collective Dynamics

The behaviour of many systems can be interpreted as the result of the collective behaviour of a large number of interacting entities.

For such systems we are often as interested in the population level behaviour as we are in the behaviour of the individual entities.
Collective Behaviour

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

In the natural world there are many instances of collective behaviour and its consequences:
Process Algebra and Collective Dynamics

Process algebra are well-suited to modelling such systems
Process Algebra and Collective Dynamics

Process algebra are well-suited to modelling such systems

- Developed to represent concurrent behaviour compositionally;
Process algebra are well-suited to modelling such systems

- Developed to represent concurrent behaviour compositionally;
- Capture the interactions between individuals explicitly;
Process Algebra and Collective Dynamics

Process algebra are well-suited to modelling such systems

- Developed to represent concurrent behaviour compositionally;
- Capture the interactions between individuals explicitly;
- Incorporate formal apparatus for reasoning about the behaviour of systems;
Process Algebra and Collective Dynamics

Process algebra are well-suited to modelling such systems

- Developed to represent concurrent behaviour compositionally;
- Capture the interactions between individuals explicitly;
- Incorporate formal apparatus for reasoning about the behaviour of systems;
- Stochastic extensions, such as PEPA, enable quantified behaviour of the dynamics of systems.
Process Algebra and Collective Dynamics

Process algebra are well-suited to modelling such systems

- Developed to represent concurrent behaviour compositionally;
- Capture the interactions between individuals explicitly;
- Incorporate formal apparatus for reasoning about the behaviour of systems;
- Stochastic extensions, such as PEPA, enable quantified behaviour of the dynamics of systems.

In the CODA project we are developing stochastic process algebras and associated theory, tailored to the construction and evaluation of the collective dynamics of large systems of interacting entities.
A shift in perspective allows us to model the interactions between individual components but then only the consider the system as a whole as an interaction of populations.
Population statistics: emergent behaviour

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

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

We must instead think about the performance of the collective point of view. Service providers often want to do this in any case. For example making contracts in terms of service level agreements.
Performance as an emergent behaviour

We must instead think about the performance of the collective point of view. Service providers often want to do this in any case. For example making contracts in terms of service level agreements.

Example Service Level Agreement
90% of requests receive a response within 3 seconds.
Performance as an emergent behaviour

We must instead think about the performance of the collective point of view. Service providers often want to do this in any case. For example making contracts in terms of service level agreements.

Example Service Level Agreement
90% of requests receive a response within 3 seconds.

Qualitative Service Level Agreement
Less than 1% of the responses received within 3 seconds will read “System is overloaded, try again later”.
Novelty

The novelty in this approach is twofold:

- Linking process algebra and continuous mathematical models for dynamic evaluation represents a paradigm shift in how such systems are studied.
- The prospect of formally-based quantified evaluation of dynamic behaviour could have significant impact in application domains such as:
Novelty

The novelty in this approach is twofold:

- Linking process algebra and continuous mathematical models for dynamic evaluation represents a paradigm shift in how such systems are studied.
Novelty

The novelty in this approach is twofold:

- Linking process algebra and continuous mathematical models for dynamic evaluation represents a paradigm shift in how such systems are studied.

- The prospect of formally-based quantified evaluation of dynamic behaviour could have significant impact in application domains such as:
Novelty

The novelty in this approach is twofold:

- Linking process algebra and continuous mathematical models for dynamic evaluation represents a paradigm shift in how such systems are studied.

- The prospect of formally-based quantified evaluation of dynamic behaviour could have significant impact in application domains such as:

  Large scale software systems
  Issues of scalability are important for user satisfaction and resource efficiency but such issues are difficult to investigate using discrete state models.
Novelty

The novelty in this approach is twofold:

- Linking process algebra and continuous mathematical models for dynamic evaluation represents a paradigm shift in how such systems are studied.

- The prospect of formally-based quantified evaluation of dynamic behaviour could have significant impact in application domains such as:

**Biochemical signalling pathways**
Understanding these pathways has the potential to improve the quality of life through enhanced drug treatment and better drug design.
Novelty

The novelty in this approach is twofold:

- Linking process algebra and continuous mathematical models for dynamic evaluation represents a paradigm shift in how such systems are studied.

- The prospect of formally-based quantified evaluation of dynamic behaviour could have significant impact in application domains such as:

  **Epidemiological systems**
  Improved modelling of these systems could lead to improved disease prevention and treatment in nature and better security in computer systems.
Novelty

The novelty in this approach is twofold:

- Linking process algebra and continuous mathematical models for dynamic evaluation represents a paradigm shift in how such systems are studied.

- The prospect of formally-based quantified evaluation of dynamic behaviour could have significant impact in application domains such as:

**Crowd dynamics**
Technology enhancement is creating new possibilities for directing crowd movements in buildings and urban spaces, for example for emergency egress, which are not yet well-understood.
# Outline

1. **Introduction**
   - Stochastic Process Algebra
   - Collective Dynamics

2. **Continuous Approximation**
   - State variables
   - Numerical illustration

3. **Fluid-Flow Semantics**
   - Fluid Structured Operational Semantics

4. **Example**
   - Scalable Web Services

5. **Conclusions**
   - Alternative Models
Solving discrete state models

Under the SOS semantics a PEPA model is mapped to a Continuous Time Markov Chain (CTMC) with global states determined by the local states of all the participating components.
Solving discrete state models

Under the SOS semantics a PEPA model is mapped to a Continuous Time Markov Chain (CTMC) with global states determined by the local states of all the participating components.

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

Under the SOS semantics a PEPA model is mapped to a **Continuous Time Markov Chain (CTMC)** with global states determined by the local states of all the participating components.

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

Alternatively they may be studied using **stochastic simulation**. Each run generates a single trajectory through the state space. Many runs are needed in order to obtain average behaviours.
Solving discrete state models

Under the SOS semantics a PEPA model is mapped to a *Continuous Time Markov Chain (CTMC)* with global states determined by the local states of all the participating components.

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

Alternatively they may be studied using *stochastic simulation*. Each run generates a single trajectory through the state space. Many runs are needed in order to obtain average behaviours.

As the size of the state space becomes large it becomes infeasible to carry out numerical solution and extremely time-consuming to conduct stochastic simulation.
Continuous Approximation

The major limitation of the CTMC approach is the state space explosion problem.

Use continuous state variables to approximate the discrete state space.

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

The major limitation of the CTMC approach is the state space explosion problem.

State space explosion becomes an ever more challenging problem as the scale and complexity of modern systems increase.
Continuous Approximation

The major limitation of the CTMC approach is the state space explosion problem.

State space explosion becomes an ever more challenging problem as the scale and complexity of modern systems increase.

Use continuous state variables to approximate the discrete state space.
Continuous Approximation

The major limitation of the CTMC approach is the state space explosion problem.

State space explosion becomes an ever more challenging problem as the scale and complexity of modern systems increase.

Use continuous state variables to approximate the discrete state space.
Continuous Approximation

The major limitation of the CTMC approach is the state space explosion problem.

State space explosion becomes an ever more challenging problem as the scale and complexity of modern systems increase.

Use continuous state variables to approximate the discrete state space.
Continuous Approximation

The major limitation of the CTMC approach is the state space explosion problem.

State space explosion becomes an ever more challenging problem as the scale and complexity of modern systems increase.

Use continuous state variables to approximate the discrete state space.
Continuous Approximation

The major limitation of the CTMC approach is the state space explosion problem.

State space explosion becomes an ever more challenging problem as the scale and complexity of modern systems increase.

Use continuous state variables to approximate the discrete state space.
Continuous Approximation

The major limitation of the CTMC approach is the state space explosion problem.

State space explosion becomes an ever more challenging problem as the scale and complexity of modern systems increase.

Use continuous state variables to approximate the discrete state space.
Continuous Approximation

The major limitation of the CTMC approach is the state space explosion problem.

State space explosion becomes an ever more challenging problem as the scale and complexity of modern systems increase.

Use continuous state variables to approximate the discrete state space.
Continuous Approximation

The major limitation of the CTMC approach is the state space explosion problem.

State space explosion becomes an ever more challenging problem as the scale and complexity of modern systems increase.

Use continuous state variables to approximate the discrete state space.
Continuous Approximation

The major limitation of the CTMC approach is the state space explosion problem.

State space explosion becomes an ever more challenging problem as the scale and complexity of modern systems increase.

Use continuous state variables to approximate the discrete state space.
Continuous Approximation

The major limitation of the CTMC approach is the state space explosion problem.

State space explosion becomes an ever more challenging problem as the scale and complexity of modern systems increase.

Use continuous state variables to approximate the discrete state space.
Continuous Approximation

The major limitation of the CTMC approach is the state space explosion problem.

State space explosion becomes an ever more challenging problem as the scale and complexity of modern systems increase.

Use continuous state variables to approximate the discrete state space.
Continuous Approximation

The major limitation of the CTMC approach is the state space explosion problem.

State space explosion becomes an ever more challenging problem as the scale and complexity of modern systems increase.

Use continuous state variables to approximate the discrete state space.

Use ordinary differential equations to represent the evolution of those variables over time.
New mathematical structures: differential equations

- Use a more abstract state representation rather than the CTMC complete state space.
New mathematical structures: differential equations

- Use a more abstract state representation rather than the CTMC complete state space.
- Assume that these state variables are subject to continuous rather than discrete change.
New mathematical structures: differential equations

- Use a more abstract state representation rather than the CTMC complete state space.
- Assume that these state variables are subject to continuous rather than discrete change.
- No longer aim to calculate the probability distribution over the entire state space of the model.
New mathematical structures: differential equations

- Use a **more abstract state representation** rather than the CTMC complete state space.

- Assume that these state variables are subject to **continuous** rather than **discrete** change.

- No longer aim to calculate the probability distribution over the entire state space of the model.

- Instead the ODEs estimate the **expected** behaviour of the CTMC.
New mathematical structures: differential equations

- Use a more abstract state representation rather than the CTMC complete state space.
- Assume that these state variables are subject to continuous rather than discrete change.
- No longer aim to calculate the probability distribution over the entire state space of the model.
- Instead the ODEs estimate the expected behaviour of the CTMC.

Appropriate for models in which there are large numbers of components of the same type, i.e. models of populations and situations of collective dynamics.
Simple example revisited

\[
\begin{align*}
\text{Proc}_0 & \overset{\text{def}}{=} (\text{task}1, r_1).\text{Proc}_1 \\
\text{Proc}_1 & \overset{\text{def}}{=} (\text{task}2, r_2).\text{Proc}_0 \\
\text{Res}_0 & \overset{\text{def}}{=} (\text{task}1, r_3).\text{Res}_1 \\
\text{Res}_1 & \overset{\text{def}}{=} (\text{reset}, r_4).\text{Res}_0
\end{align*}
\]

\[\text{Proc}_0[N_P \{\text{task}1\}] \boxtimes \text{Res}_0[N_R]\]
Simple example revisited

**CTMC interpretation**

<table>
<thead>
<tr>
<th>Processors ($N_P$)</th>
<th>Resources ($N_R$)</th>
<th>States ($2^{N_P+N_R}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>32</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>64</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>128</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>256</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>512</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1024</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>2048</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>4096</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>8192</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>16384</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>32768</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>65536</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>131072</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>262144</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>524288</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>1048576</td>
</tr>
</tbody>
</table>
Simple example revisited

\[\begin{align*}
Proc_0 & \overset{\text{def}}{=} (\text{task1}, r_1).Proc_1 \\
Proc_1 & \overset{\text{def}}{=} (\text{task2}, r_2).Proc_0 \\
Res_0 & \overset{\text{def}}{=} (\text{task1}, r_3).Res_1 \\
Res_1 & \overset{\text{def}}{=} (\text{reset}, r_4).Res_0
\end{align*}\]

ODE interpretation:

\[\begin{align*}
\frac{dx_1}{dt} &= -\min(r_1 x_1, r_3 x_3) + r_2 x_1 \\
x_1 &= \text{no. of } Proc_0
\end{align*}\]

\[\begin{align*}
\frac{dx_2}{dt} &= r_1 \min(x_1, x_3) - r_2 x_1 \\
x_2 &= \text{no. of } Proc_1
\end{align*}\]

\[\begin{align*}
\frac{dx_3}{dt} &= -r_1 \min(x_1, x_3) + r_4 x_4 \\
x_3 &= \text{no. of } Res_0
\end{align*}\]

\[\begin{align*}
\frac{dx_4}{dt} &= r_1 \min(x_1, x_3) - r_4 x_4 \\
x_4 &= \text{no. of } Res_1
\end{align*}\]
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

1. Introduction
   - Stochastic Process Algebra
   - Collective Dynamics

2. Continuous Approximation
   - State variables
   - Numerical illustration

3. Fluid-Flow Semantics
   - Fluid Structured Operational Semantics

4. Example
   - Scalable Web Services

5. Conclusions
   - Alternative Models
Deriving a Fluid Approximation of a PEPA model

The aim is to represent the CTMC implicitly (avoiding state space explosion), and to generate the set of ODEs which are the fluid limit of that CTMC.
Deriving a Fluid Approximation of a PEPA model

The aim is to represent the CTMC implicitly (avoiding state space explosion), and to generate the set of ODEs which are the fluid limit of that CTMC.

The existing (CTMC) SOS semantics is not suitable for this purpose because it constructs the state space of the CTMC explicitly.
Deriving a Fluid Approximation of a PEPA model

The aim is to represent the CTMC implicitly (avoiding state space explosion), and to generate the set of ODEs which are the fluid limit of that CTMC.

The existing (CTMC) SOS semantics is not suitable for this purpose because it constructs the state space of the CTMC explicitly.
Deriving a Fluid Approximation of a PEPA model

The aim is to represent the CTMC implicitly (avoiding state space explosion), and to generate the set of ODEs which are the fluid limit of that CTMC.

The existing (CTMC) SOS semantics is not suitable for this purpose because it constructs the state space of the CTMC explicitly.

Nevertheless we are able to define a structured operational semantics which defines the possible transitions of an arbitrary abstract state and from this derive the ODEs.
Deriving a Fluid Approximation of a PEPA model

The aim is to represent the CTMC implicitly (avoiding state space explosion), and to generate the set of ODEs which are the fluid limit of that CTMC.

The existing (CTMC) SOS semantics is not suitable for this purpose because it constructs the state space of the CTMC explicitly.

Nevertheless we are able to define a structured operational semantics which defines the possible transitions of an arbitrary abstract state and from this derive the ODEs.
Fluid Structured Operational Semantics

In order to get to the implicit representation of the CTMC we need to:

1. Remove excess components (Context Reduction)
2. Collect the transitions of the reduced context (Jump Multiset)
3. Calculate the rate of the transitions in terms of an arbitrary state of the CTMC. Once this is done we can extract the vector field $F_M(x)$ from the jump multiset.
Fluid Structured Operational Semantics

In order to get to the implicit representation of the CTMC we need to:

1. Remove excess components (Context Reduction)
In order to get to the implicit representation of the CTMC we need to:

1. Remove excess components \((\text{Context Reduction})\)
2. Collect the transitions of the reduced context \((\text{Jump Multiset})\)
Fluid Structured Operational Semantics

In order to get to the implicit representation of the CTMC we need to:

1. Remove excess components (*Context Reduction*)
2. Collect the transitions of the reduced context (*Jump Multiset*)
3. Calculate the rate of the transitions in terms of an arbitrary state of the CTMC.
Fluid Structured Operational Semantics

In order to get to the implicit representation of the CTMC we need to:

1. Remove excess components (Context Reduction)
2. Collect the transitions of the reduced context (Jump Multiset)
3. Calculate the rate of the transitions in terms of an arbitrary state of the CTMC.

Once this is done we can extract the vector field $F_{\mathcal{M}}(x)$ from the jump multiset.
Context Reduction

\[ \begin{align*}
\text{Proc}_0 & \overset{\text{def}}{=} (\text{task}1, r_1).\text{Proc}_1 \\
\text{Proc}_1 & \overset{\text{def}}{=} (\text{task}2, r_2).\text{Proc}_0 \\
\text{Res}_0 & \overset{\text{def}}{=} (\text{task}1, r_3).\text{Res}_1 \\
\text{Res}_1 & \overset{\text{def}}{=} (\text{reset}, r_4).\text{Res}_0 \\
\text{System} & \overset{\text{def}}{=} \text{Proc}_0[N_P] \ {\text{transfer}} \ \bowtie \ \text{Res}_0[N_R] \\
\downarrow \\
\mathcal{R}(\text{System}) &= \{ \text{Proc}_0, \text{Proc}_1 \} \ {\text{task}1} \ \bowtie \ \{ \text{Res}_0, \text{Res}_1 \}
\end{align*} \]
Context Reduction

\[ \begin{align*}
Proc_0 & \overset{\text{def}}{=} (\text{task}1, r_1).Proc_1 \\
Proc_1 & \overset{\text{def}}{=} (\text{task}2, r_2).Proc_0 \\
Res_0 & \overset{\text{def}}{=} (\text{task}1, r_3).Res_1 \\
Res_1 & \overset{\text{def}}{=} (\text{reset}, r_4).Res_0 \\
\text{System} & \overset{\text{def}}{=} Proc_0[N_P] \begin{array}{c}
\begin{array}{c}
\text{transfer}
\end{array}
\end{array} \begin{array}{c}
\begin{array}{c}
\end{array}
\end{array} Res_0[N_R] \\
\Downarrow
\end{align*} \]

\[ \mathcal{R}(\text{System}) = \{ Proc_0, Proc_1 \} \begin{array}{c}
\begin{array}{c}
\text{task}1
\end{array}
\end{array} \begin{array}{c}
\begin{array}{c}
\end{array}
\end{array} \{ Res_0, Res_1 \} \]

Population Vector

\[ \xi = (\xi_1, \xi_2, \xi_3, \xi_4) \]
Location Dependency

\[ System \overset{\text{def}}{=} Proc_0[N'_C] \parallel Res_0[N_S] \parallel Proc_0[N''_C] \]
Location Dependency

\[ \text{System} \overset{\text{def}}{=} \text{Proc}_0[N'_C] \parallel_{\{\text{task}1\}} \text{Res}_0[N_S] \parallel \text{Proc}_0[N''_C] \]

\[ \Downarrow \]

\[ \{\text{Proc}_0, \text{Proc}_1\} \parallel_{\{\text{task}1\}} \{\text{Res}_0, \text{Res}_1\} \parallel \{\text{Proc}_0, \text{Proc}_1\} \]
Location Dependency

\[ \text{System} \overset{\text{def}}{=} \text{Proc}_0[N'_C] \uprod \{\text{task1}\} \text{Res}_0[N_S] \parallel \text{Proc}_0[N''_C] \]

\[ \downarrow \]

\[ \curly{\text{Proc}_0, \text{Proc}_1} \uprod \{\text{task1}\} \curly{\text{Res}_0, \text{Res}_1} \parallel \curly{\text{Proc}_0, \text{Proc}_1} \]

Population Vector

\[ \xi = (\xi_1, \xi_2, \xi_3, \xi_4, \xi_5, \xi_6) \]
Fluid Structured Operational Semantics by Example

\[
\begin{align*}
\text{Proc}_0 & \overset{\text{def}}{=} (\text{task}_1, r_1).\text{Proc}_1 \\
\text{Proc}_1 & \overset{\text{def}}{=} (\text{task}_2, r_2).\text{Proc}_0 \\
\text{Res}_0 & \overset{\text{def}}{=} (\text{task}_1, r_3).\text{Res}_1 \\
\text{Res}_1 & \overset{\text{def}}{=} (\text{reset}, r_4).\text{Res}_0 \\
\text{System} & \overset{\text{def}}{=} \text{Proc}_0[N_P] \{\text{transfer}\} \text{Res}_0[N_R] \\
\xi & = (\xi_1, \xi_2, \xi_3, \xi_4)
\end{align*}
\]
Fluid Structured Operational Semantics by Example

\[
\begin{align*}
\text{Proc}_0 & \overset{\text{def}}{=} (\text{task1}, r_1).\text{Proc}_1 \\
\text{Proc}_1 & \overset{\text{def}}{=} (\text{task2}, r_2).\text{Proc}_0 \\
\text{Res}_0 & \overset{\text{def}}{=} (\text{task1}, r_3).\text{Res}_1 \\
\text{Res}_1 & \overset{\text{def}}{=} (\text{reset}, r_4).\text{Res}_0 \\
\text{System} & \overset{\text{def}}{=} \text{Proc}_0[N_P] \overset{\{\text{transfer}\}}{\oplus} \text{Res}_0[N_R] \\
\xi & = (\xi_1, \xi_2, \xi_3, \xi_4)
\end{align*}
\]

\[
\begin{align*}
\text{Proc}_0 & \xrightarrow{\text{task1}, r_1} \text{Proc}_1 \\
\text{Proc}_0 & \xrightarrow{\text{task1}, r_1\xi_1} * \text{Proc}_1
\end{align*}
\]
Fluid Structured Operational Semantics by Example

\[
\begin{align*}
\text{Proc}_0 & \overset{\text{def}}{=} (\text{task}1, r_1).\text{Proc}_1 \\
\text{Proc}_1 & \overset{\text{def}}{=} (\text{task}2, r_2).\text{Proc}_0 \\
\text{Res}_0 & \overset{\text{def}}{=} (\text{task}1, r_3).\text{Res}_1 \\
\text{Res}_1 & \overset{\text{def}}{=} (\text{reset}, r_4).\text{Res}_0 \\
\text{System} & \overset{\text{def}}{=} \text{Proc}_0[N_P] \ {\text{transfer}} \ \text{Res}_0[N_R] \\
\xi & = (\xi_1, \xi_2, \xi_3, \xi_4)
\end{align*}
\]

\[
\begin{align*}
\text{Proc}_0 & \xrightarrow{\text{task}1, r_1} \text{Proc}_1 \\
\text{Proc}_0 & \xrightarrow{\text{task}1, r_1 \xi_1} \ast \text{Proc}_1 \\
\text{Res}_0 & \xrightarrow{\text{task}1, r_3} \text{Res}_1 \\
\text{Res}_0 & \xrightarrow{\text{task}1, r_3 \xi_3} \ast \text{Res}_1
\end{align*}
\]
Fluid Structured Operational Semantics by Example

\[ \text{Proc}_0 \overset{\text{def}}{=} (\text{task}1, r_1).\text{Proc}_1 \]
\[ \text{Proc}_1 \overset{\text{def}}{=} (\text{task}2, r_2).\text{Proc}_0 \]
\[ \text{Res}_0 \overset{\text{def}}{=} (\text{task}1, r_3).\text{Res}_1 \]
\[ \text{Res}_1 \overset{\text{def}}{=} (\text{reset}, r_4).\text{Res}_0 \]
\[ \text{System} \overset{\text{def}}{=} \text{Proc}_0[N_P] \{\text{transfer}\} \text{Res}_0[N_R] \]

\[ \xi = (\xi_1, \xi_2, \xi_3, \xi_4) \]
Apparent Rate Calculation
Apparent Rate Calculation

\[
\begin{align*}
\text{Proc}_0 \xrightarrow{\text{task1}, r'_1} & \text{Proc}_1 \\
\text{Proc}_0 \xrightarrow{\text{task1}, r'_1, \xi_1} & \ast \text{Proc}_1 \\
\text{Proc}_0 & \bowtie \text{Res}_0 \xrightarrow{\text{task1}, r(\xi)} \ast \text{Proc}_1 & \bowtie \text{Res}_1 & \text{Res}_0 \xrightarrow{\text{task1}, r_3, \xi_3} \ast \text{Res}_1
\end{align*}
\]

\[
r(\xi) = \frac{r_1 \xi_1}{r^*_\text{task1} (\text{Proc}_0, \xi)} \frac{r_3 \xi_4}{r^*_\text{task1} (\text{Res}_0, \xi)} \min \left( r^*_\text{task1} (\text{Proc}_0, \xi), r^*_\text{task1} (\text{Res}_0, \xi) \right)
\]

\[
= \frac{r_1 \xi_1}{r_1 \xi_1} \frac{r_3 \xi_4}{r_3 \xi_4} \min \left( r_1 \xi_1, r_3 \xi_4 \right)
\]

\[
= \min \left( r_1 \xi_1, r_3 \xi_4 \right)
\]
\( f(\xi, l, \alpha) \) as the Generator Matrix of the \textit{Lumped} CTMC

\[
(P_0 \parallel P_0) \{\text{task1}\} (R_0 \parallel R_0 \parallel R_0)
\]

\[
(P_1 \parallel P_0) \{\text{task1}\} (R_1 \parallel R_0 \parallel R_0)
\]

\[
(P_1 \parallel P_0) \{\text{task1}\} (R_0 \parallel R_1 \parallel R_0)
\]

\[
(P_1 \parallel P_0) \{\text{task1}\} (R_0 \parallel R_0 \parallel R_1)
\]

\[
(P_0 \parallel P_1) \{\text{task1}\} (R_1 \parallel R_0 \parallel R_0)
\]

\[
(P_0 \parallel P_1) \{\text{task1}\} (R_0 \parallel R_1 \parallel R_0)
\]

\[
(P_0 \parallel P_1) \{\text{task1}\} (R_0 \parallel R_0 \parallel R_1)
\]
$f(\xi, l, \alpha)$ as the Generator Matrix of the *Lumped* CTMC

\[
\begin{align*}
& (P_0 \parallel P_0) \{\text{task1}\} (R_0 \parallel R_0 \parallel R_0) \\
& (P_1 \parallel P_0) \{\text{task1}\} (R_0 \parallel R_1 \parallel R_0) \\
& (P_1 \parallel P_0) \{\text{task1}\} (R_0 \parallel R_0 \parallel R_1) \\
& (P_0 \parallel P_1) \{\text{task1}\} (R_1 \parallel R_0 \parallel R_0) \\
& (P_0 \parallel P_1) \{\text{task1}\} (R_0 \parallel R_1 \parallel R_0) \\
& \quad \vdots \\
& (P_0 \parallel P_1) \{\text{task1}\} (R_0 \parallel R_0 \parallel R_1)
\end{align*}
\]

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

\[
(P_0 \parallel P_1) \{\text{task1}\} (R_0 \parallel R_0 \parallel R_1)
\]
$f(\xi, l, \alpha)$ as the Generator Matrix of the *Lumped* CTMC

$$\begin{align*}
\left( P_0 \parallel P_0 \right)_{\{\text{task1}\}} \left( R_0 \parallel R_0 \parallel R_0 \right)
\end{align*}$$

$$r = \frac{r_1}{2r_1} \frac{r_3}{3r_3} \ \text{min}(2r_1, 3r_3) = \frac{1}{6} \ \text{min}(2r_1, 3r_3)$$
$f(\xi, l, \alpha)$ as the Generator Matrix of the *Lumped* CTMC

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

$$(P_0 \parallel P_0) \xrightarrow{\text{task1}} (R_0 \parallel R_0 \parallel R_0)$$

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

\[ \text{Proc}_0 \quad \begin{array}{c} \bullet \\ \{\text{task}1\} \end{array} \quad \text{Res}_0 \quad \xrightarrow{\text{task}_1, r(\xi)} \quad \ast \quad \text{Proc}_1 \quad \begin{array}{c} \bullet \\ \{\text{task}1\} \end{array} \quad \text{Res}_1 \]

\[ r(\xi) = \min (r_1\xi_1, r_3\xi_3) \]
Jump Multiset

\[ \text{Proc}_0 \quad \llap{\{\text{task1}\}} \quad \text{Res}_0 \xrightarrow{\text{task1}, r(\xi)} \ast \quad \text{Proc}_1 \quad \llap{\{\text{task1}\}} \quad \text{Res}_1 \]

\[ r(\xi) = \min (r_1 \xi_1, r_3 \xi_3) \]

\[ \text{Proc}_1 \quad \llap{\{\text{task1}\}} \quad \text{Res}_0 \xrightarrow{\text{task2}, \xi_2 r_2} \ast \quad \text{Proc}_0 \quad \llap{\{\text{task1}\}} \quad \text{Res}_0 \]
Jump Multiset

\[
\begin{align*}
\text{Proc}_0 \quad \boxdot & \quad \text{Res}_0 & \xrightarrow{\text{task1}, r(\xi)} & \star & \text{Proc}_1 & \boxdot & \text{Res}_1 \\
r(\xi) &= \min (r_1\xi_1, r_3\xi_3)
\end{align*}
\]

\[
\begin{align*}
\text{Proc}_1 \quad \boxdot & \quad \text{Res}_0 & \xrightarrow{\text{task2}, \xi_2 r_2} & \star & \text{Proc}_0 & \boxdot & \text{Res}_0
\end{align*}
\]

\[
\begin{align*}
\text{Proc}_0 \quad \boxdot & \quad \text{Res}_1 & \xrightarrow{\text{reset}, \xi_4 r_4} & \star & \text{Proc}_0 & \boxdot & \text{Res}_0
\end{align*}
\]
Equivalent Transitions

Some transitions may give the same information:

\[
\begin{align*}
&\text{Proc}_0 \xrightarrow{\text{task1}} \text{Res}_1 \xrightarrow{\text{reset,ξ}_4 r_4} \ast \text{Proc}_0 \xrightarrow{\text{task1}} \text{Res}_0 \\
&\text{Proc}_1 \xrightarrow{\text{task1}} \text{Res}_1 \xrightarrow{\text{reset,ξ}_4 r_4} \ast \text{Proc}_1 \xrightarrow{\text{task1}} \text{Res}_0
\end{align*}
\]

i.e., \(\text{Res}_1\) may perform an action independently from the rest of the system.

This is captured by the procedure used for the construction of the generator function \(f(ξ, l, α)\)
Construction of $f(\xi, l, \alpha)$

$Proc_0 \{ \text{task1} \} \xrightarrow{\text{reset}, \xi_4 r_4} * Proc_0 \{ \text{task1} \}$
Construction of $f(\xi, l, \alpha)$

$Proc_0 \{\text{task1}\} \xrightarrow{\text{reset,} \xi_4 r_4} \ast Proc_0 \{\text{task1}\} \rightarrow \ast Res_1 \rightarrow \ast Proc_0 \{\text{task1}\} \rightarrow \ast Res_0$

- Take $l = (0, 0, 0, 0)$
Construction of $f(\xi, l, \alpha)$

Take $l = (0, 0, 0, 0)$

Add $-1$ to all elements of $l$ corresponding to the indices of the components in the lhs of the transition

$$l = (-1, 0, 0, -1)$$

Add $+1$ to all elements of $l$ corresponding to the indices of the components in the rhs of the transition

$$l = (-1 + 1, 0, +1, -1) = (0, 0, +1, -1)$$
Construction of $f(\xi, l, \alpha)$

$$Proc_0 \begin{array}{c} \boxtimes \end{array} Res_1 \xrightarrow{\text{reset}, \xi_4 r_4} * Proc_0 \begin{array}{c} \boxtimes \end{array} Res_0$$

- Take $l = (0, 0, 0, 0)$
- Add $-1$ to all elements of $l$ corresponding to the indices of the components in the lhs of the transition
  
  \[ l = (-1, 0, 0, -1) \]

- Add $+1$ to all elements of $l$ corresponding to the indices of the components in the rhs of the transition
  
  \[ l = (-1 + 1, 0, +1, -1) = (0, 0, +1, -1) \]

\[ f(\xi, (0, 0, +1, -1), \text{reset}) = \xi_4 r_4 \]
Construction of \( f(\xi, l, \alpha) \)
Construction of $f(\xi, l, \alpha)$

$$f(\xi, (-1, +1, -1, +1), \text{task1}) = r(\xi)$$
Construction of $f(\xi, l, \alpha)$

\[
\begin{align*}
\text{Proc}_0 \{\text{task1}\} & \quad \text{Res}_0 & \xrightarrow{\text{task1}, r(\xi)} & \quad \text{Proc}_1 \{\text{task1}\} & \quad \text{Res}_1 \\
\text{Proc}_1 \{\text{task1}\} & \quad \text{Res}_0 & \xrightarrow{\text{task2}, \xi_2 r'_2} & \quad \text{Proc}_0 \{\text{task1}\} & \quad \text{Res}_0
\end{align*}
\]

\[
\begin{align*}
 f(\xi, (-1, +1, -1, +1), \text{task1}) &= r(\xi) \\
 f(\xi, (+1, -1, 0, 0), \text{task2}) &= \xi_2 r_2
\end{align*}
\]
Construction of $f(\xi, l, \alpha)$

\[
\begin{align*}
Proc_0 \{\text{task1}\} & \quad \xrightarrow{\text{task1, } r(\xi)} \quad Proc_1 \{\text{task1}\} \\
Proc_1 \{\text{task1}\} & \quad \xrightarrow{\text{task2, } \xi_2 r'_2} \quad Proc_0 \{\text{task1}\} \\
Proc_0 \{\text{task1}\} & \quad \xrightarrow{\text{reset, } \xi_4 r_4} \quad Proc_0 \{\text{task1}\}
\end{align*}
\]

\[
\begin{align*}
f(\xi, (-1, +1, -1, +1), \text{task1}) & = r(\xi) \\
f(\xi, (+1, -1, 0, 0), \text{task2}) & = \xi_2 r_2 \\
f(\xi, (0, 0, +1, -1), \text{reset}) & = \xi_4 r_4
\end{align*}
\]
Capturing behaviour in the Generator Function

\[
\begin{align*}
\text{Proc}_0 & \overset{\text{def}}{=} (\text{task}1, r_1).\text{Proc}_1 \\
\text{Proc}_1 & \overset{\text{def}}{=} (\text{task}2, r_2).\text{Proc}_0 \\
\text{Res}_0 & \overset{\text{def}}{=} (\text{task}1, r_3).\text{Res}_1 \\
\text{Res}_1 & \overset{\text{def}}{=} (\text{reset}, r_4).\text{Res}_0 \\
\text{System} & \overset{\text{def}}{=} \text{Proc}_0[N_P] \quad \text{transfer} \quad \text{Res}_0[N_R]
\end{align*}
\]
Capturing behaviour in the Generator Function

\[
\begin{align*}
Proc_0 & \overset{\text{def}}{=} (\text{task1, } r_1).Proc_1 \\
Proc_1 & \overset{\text{def}}{=} (\text{task2, } r_2).Proc_0 \\
Res_0 & \overset{\text{def}}{=} (\text{task1, } r_3).Res_1 \\
Res_1 & \overset{\text{def}}{=} (\text{reset, } r_4).Res_0 \\
System & \overset{\text{def}}{=} Proc_0[N_P] \odot Res_0[N_R] \\
& \begin{cases} \{ \text{transfer} \} \end{cases}
\end{align*}
\]

Numerical Vector Form

\[
\xi = (\xi_1, \xi_2, \xi_3, \xi_4) \in \mathbb{N}^4, \quad \xi_1 + \xi_2 = N_P \quad \text{and} \quad \xi_3 + \xi_4 = N_R
\]
Capturing behaviour in the Generator Function

\[\begin{align*}
Proc_0 & \overset{\text{def}}{=} (\text{task}_1, r_1).Proc_1 \\
Proc_1 & \overset{\text{def}}{=} (\text{task}_2, r_2).Proc_0 \\
Res_0 & \overset{\text{def}}{=} (\text{task}_1, r_3).Res_1 \\
Res_1 & \overset{\text{def}}{=} (\text{reset}, r_4).Res_0 \\
\text{System} & \overset{\text{def}}{=} Proc_0[N_P] \ \{\text{transfer}\} \ Res_0[N_R]
\end{align*}\]

Numerical Vector Form

\[\xi = (\xi_1, \xi_2, \xi_3, \xi_4) \in \mathbb{N}^4, \quad \xi_1 + \xi_2 = N_P \quad \text{and} \quad \xi_3 + \xi_4 = N_R\]

Generator Function

\[\begin{align*}
f(\xi, (-1, 1, -1, 1), \text{task}_1) &= \min (r_1\xi_1, r_3\xi_3) \\
f(\xi, (1, -1, 0, 0), \text{task}_2) &= r_2\xi_2 \\
f(\xi, (0, 0, 1, -1), \text{reset}) &= r_4\xi_4
\end{align*}\]
Extraction of the ODE from $f$

Generator Function

\[
\begin{align*}
    f(\xi, (-1, 1, -1, 1), \text{task1}) &= \min (r_1 \xi_1, r_3 \xi_3) \\
    f(\xi, (1, -1, 0, 0), \text{task2}) &= r_2 \xi_2 \\
    f(\xi, (0, 0, 1, -1), \text{reset}) &= r_4 \xi_4
\end{align*}
\]

Differential Equation

\[
\frac{dx}{dt} = F_M(x) = \sum_{l \in \mathbb{Z}^d} \sum_{\alpha \in A} f(x, l, \alpha)
\]

\[
= (-1, 1, -1, 1) \min (r_1 x_1, r_3 x_3) + (1, -1, 0, 0) r_2 x_2 \\
+ (0, 0, 1, -1) r_4 x_4
\]
Extraction of the ODE from $f$

Generator Function

\[
\begin{align*}
  f(\xi, (-1, 1, -1, 1), task1) &= \min (r_1 \xi_1, r_3 \xi_3) \\
  f(\xi, (1, -1, 0, 0), task2) &= r_2 \xi_2 \\
  f(\xi, (0, 0, 1, -1), reset) &= r_4 \xi_4
\end{align*}
\]

Differential Equation

\[
\begin{align*}
  \frac{dx_1}{dt} &= -\min (r_1 x_1, r_3 x_3) + r_2 x_2 \\
  \frac{dx_2}{dt} &= \min (r_1 x_1, r_3 x_3) - r_2 x_2 \\
  \frac{dx_3}{dt} &= -\min (r_1 x_1, r_3 x_3) + r_4 x_4 \\
  \frac{dx_4}{dt} &= \min (r_1 x_1, r_3 x_3) - r_4 x_4
\end{align*}
\]
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td></td>
<td>- Stochastic Process Algebra</td>
</tr>
<tr>
<td></td>
<td>- Collective Dynamics</td>
</tr>
<tr>
<td>2</td>
<td>Continuous Approximation</td>
</tr>
<tr>
<td></td>
<td>- State variables</td>
</tr>
<tr>
<td></td>
<td>- Numerical illustration</td>
</tr>
<tr>
<td>3</td>
<td>Fluid-Flow Semantics</td>
</tr>
<tr>
<td></td>
<td>- Fluid Structured Operational Semantics</td>
</tr>
<tr>
<td>4</td>
<td>Example</td>
</tr>
<tr>
<td></td>
<td>- Scalable Web Services</td>
</tr>
<tr>
<td>5</td>
<td>Conclusions</td>
</tr>
<tr>
<td></td>
<td>- Alternative Models</td>
</tr>
</tbody>
</table>
Virtual University Scenario

- A **Virtual University** is a federation of *real* universities, each contributing courses and degrees.
Virtual University Scenario

- A **Virtual University** is a federation of *real* universities, each contributing courses and degrees.
- Sharing of knowledge is promoted by providing students with a wider selection of subjects.
Virtual University Scenario

- A Virtual University is a federation of real universities, each contributing courses and degrees.
- Sharing of knowledge is promoted by providing students with a wider selection of subjects.
- Services are replicated across the physical sites.
Virtual University Scenario

- A **Virtual University** is a federation of *real* universities, each contributing courses and degrees.
- Sharing of knowledge is promoted by providing students with a wider selection of subjects.
- Services are replicated across the physical sites.
- By agreement in the university, students may connect to any site to download content and use services, not just the one which is geographically closest.
Case Study: A Virtual University
Location, Time, and Size
Replicating Web Services

Two viable approaches to cope with increasing user demand:
Replicating Web Services

Two viable approaches to cope with increasing user demand:

- use a service broker for routing
Replicating Web Services

Two viable approaches to cope with increasing user demand:

- use a service broker for routing
- decentralised routing
Replicating Web Services

Two viable approaches to cope with increasing user demand:

- use a service broker for routing
- decentralised routing
A client contacts a university site to download content.
A client contacts a university site to download content.

The site either serves the request or forwards it to another site.
Decentralised Routing

1. A client contacts a university site to download content.
2. The site either serves the request or forwards it to another site.
3. The decision is made in accord with the local service policy.
Model in PEPA

Clients

\[ \text{Client}_i \overset{\text{def}}{=} (\text{connect}_1, c_{1,i}).(\text{download}_1, d_{1,i}).\text{Idle}_i \\
+ (\text{connect}_2, c_{2,i}).(\text{download}_2, d_{2,i}).\text{Idle}_i \\
\ldots \\
+ (\text{connect}_m, c_{m,i}).(\text{download}_m, d_{m,i}).\text{Idle}_i \\
+ (\text{overload, } \top).\text{Client}_i \]

\[ \text{Idle}_i \overset{\text{def}}{=} (\text{idle, } r_{\text{idle},i}).\text{Client}_i \]

\[(1 \leq i \leq k)\]
Model in PEPA

Clients

\[\begin{align*}
\text{Client}_i & \overset{\text{def}}{=} (\text{connect}_1, c_{1,i}).(\text{download}_1, d_{1,i}).\text{Idle}_i \\
& + (\text{connect}_2, c_{2,i}).(\text{download}_2, d_{2,i}).\text{Idle}_i \\
& \cdots \\
& + (\text{connect}_m, c_{m,i}).(\text{download}_m, d_{m,i}).\text{Idle}_i \\
& + (\text{overload}, \top).\text{Client}_i \\
\text{Idle}_i & \overset{\text{def}}{=} (\text{idle}, r_{\text{idle},i}).\text{Client}_i
\end{align*}\]

\((1 \leq i \leq k)\)
Model in PEPA

Clients

\[\text{Client}_i \overset{\text{def}}{=} (\text{connect}_1, c_{1,i}).(\text{download}_1, d_{1,i}).\text{Idle}_i \]
\[+ (\text{connect}_2, c_{2,i}).(\text{download}_2, d_{2,i}).\text{Idle}_i \]
\[\ldots\]
\[+ (\text{connect}_m, c_{m,i}).(\text{download}_m, d_{m,i}).\text{Idle}_i \]
\[+ (\text{overload}, \top).\text{Client}_i \]
\[\text{Idle}_i \overset{\text{def}}{=} (\text{idle}, r_{\text{idle},i}).\text{Client}_i \]

\[(1 \leq i \leq k)\]
Model in PEPA

Content mirrors

\[ \begin{align*}
\text{Mirror}_j & \quad \text{def} \quad (\text{connect}_j, f_j(s)).\text{MirrorUploading}_j \\
\text{MirrorUploading}_j & \quad \text{def} \quad (\text{download}_j, \top).\text{Mirror}_j
\end{align*} \]

(1 \leq j \leq m)
Model in PEPA

Content mirrors

\[
\text{Mirror}_j \overset{\text{def}}{=} (\text{connect}_j, f_j(s)).\text{MirrorUploading}_j \\
\text{MirrorUploading}_j \overset{\text{def}}{=} (\text{download}_j, \top).\text{Mirror}_j
\]

\[(1 \leq j \leq m)\]
Service policies as functional rates in PEPA

The Bologna policy

Serve all requests while load is less than 75%. If more, and the loads at UNIFI, UPISA, LMU and UEDIN are at least 60%, 60%, 40% and 20% then serve the request if load is less than 95%.

\[
f_{\text{UNIBO}} = \begin{cases} 
\top & \text{if MirrorUploading}_{\text{UNIBO}} < 75 \\
\top & \text{if MirrorUploading}_{\text{UNIBO}} < 95, \\
\text{MirrorUploading}_{\text{UNIFI}} \geq 60, \\
\text{MirrorUploading}_{\text{UPISA}} \geq 60, \\
\text{MirrorUploading}_{\text{LMU}} \geq 40, \\
\text{MirrorUploading}_{\text{UEDIN}} \geq 20, \\
0 & \text{otherwise}
\end{cases}
\]
Service policies as functional rates in PEPA

The Bologna policy

Serve all requests while load is less than 75%. If more, and the loads at UNIFI, UPISA, LMU and UEDIN are at least 60%, 60%, 40% and 20% then serve the request if load is less than 95%.

\[
\begin{align*}
f_{\text{UNIBO}} &= \begin{cases} 
\top & \text{if } \text{MirrorUploading}_{\text{UNIBO}} < 75 \\
\top & \text{if } \text{MirrorUploading}_{\text{UNIBO}} < 95, \\
& \text{MirrorUploading}_{\text{UNIFI}} \geq 60, \\
& \text{MirrorUploading}_{\text{UPISA}} \geq 60, \\
& \text{MirrorUploading}_{\text{LMU}} \geq 40, \\
& \text{MirrorUploading}_{\text{UEDIN}} \geq 20 \\
0 & \text{otherwise}
\end{cases}
\end{align*}
\]
Dealing with overload

\[
\text{Overload} \overset{\text{def}}{=} \langle \text{overload}, o(s) \rangle \cdot \text{Overload}
\]

\[
o(s) = \begin{cases} 
\top & f_i(s) = 0, \quad 1 \leq i \leq m \\
0 & \text{otherwise}
\end{cases}
\]
Model in PEPA

Dealing with overload

\[
\text{Overload} \overset{\text{def}}{=} (\text{overload}, o(s)) \cdot \text{Overload}
\]

\[
o(s) = \begin{cases} 
\top & f_i(s) = 0, \quad 1 \leq i \leq m \\
0 & \text{otherwise}
\end{cases}
\]

The system as a whole with client and mirror site populations

\[
(\text{Client}_1[p_1] \parallel \text{Client}_2[p_2] \parallel \ldots \parallel \text{Client}_k[p_k]) \parallel \bigotimes_l (\text{Mirror}_1[q_1] \parallel \text{Mirror}_2[q_2] \parallel \ldots \parallel \text{Mirror}_m[q_m])
\]
Numerical Results

\[ r_{idle} = 0.001 \]
Numerical Results

\[ r_{idle} = 0.01 \]
Numerical Results

\[ r_{idle} = 0.02 \]
Numerical Results

\[ r_{idle} = 0.03 \]
Numerical Results

\[ r_{idle} = 0.04 \]
Numerical Results

\[ r_{idle} = 0.05 \]
Numerical Results

\[ r_{\text{idle}} = 0.06 \]
Outline

1. Introduction
   - Stochastic Process Algebra
   - Collective Dynamics

2. Continuous Approximation
   - State variables
   - Numerical illustration

3. Fluid-Flow Semantics
   - Fluid Structured Operational Semantics

4. Example
   - Scalable Web Services

5. Conclusions
   - Alternative Models
Alternative Representations

- ODEs
- Stochastic Simulation
  - CTMC

Large PEPA model
Alternative Representations

- ODEs (population view)
- Large PEPA model
- Stochastic Simulation (CTMC) (individual view)
The vector field $\mathcal{F}(x)$ is Lipschitz continuous i.e. all the rate functions governing transitions in the process algebra satisfy local continuity conditions.
Consistency results

- The vector field $\mathcal{F}(x)$ is Lipschitz continuous i.e. all the rate functions governing transitions in the process algebra satisfy local continuity conditions.

- The generated ODEs are the fluid limit of the family of CTMCs generated by $f(\xi, l, \alpha)$: this family forms a sequence as the initial populations are scaled by a variable $n$. 

Consistency results

- The vector field $\mathcal{F}(x)$ is Lipschitz continuous i.e. all the rate functions governing transitions in the process algebra satisfy local continuity conditions.

- The generated ODEs are the fluid limit of the family of CTMCs generated by $f(\xi, l, \alpha)$: this family forms a sequence as the initial populations are scaled by a variable $n$.

Consistency results

- The vector field $\mathcal{F}(x)$ is Lipschitz continuous i.e. all the rate functions governing transitions in the process algebra satisfy local continuity conditions.

- The generated ODEs are the fluid limit of the family of CTMCs generated by $f(\xi, l, \alpha)$: this family forms a sequence as the initial populations are scaled by a variable $n$.


- Moreover Lipschitz continuity of the vector field guarantees existence and uniqueness of the solution to the initial value problem.
Conclusions

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

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

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

- Many interesting and important systems can be regarded as examples of collective dynamics and emergent behaviour.
- Process algebras, such as PEPA, are well-suited to modelling the behaviour of such systems in terms of the individuals and their interactions.
- **Continuous approximation** allows a rigorous mathematical analysis of the average behaviour of such systems.
- This alternative view of systems has opened up many and exciting new research directions.
Thanks!
Acknowledgements: collaborators
Thanks to many co-authors and collaborators: Jeremy Bradley, Luca Bortolussi, Allan Clark, Adam Duguid, Vashti Galpin, Nil Gesweiller, Stephen Gilmore, Marco Stenico, Mirco Tribastone, and others.
Thanks!

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

**Acknowledgements: funding**
Thanks to EPRSC for the Process Algebra for Collective Dynamics grant and the CEC IST-FET programme for the SENSORIA project which have supported this work.
Thanks!

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

Acknowledgements: funding
Thanks to EPRSC for the Process Algebra for Collective Dynamics grant and the CEC IST-FET programme for the SENSORIA project which have supported this work.

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

- ODEs (population view)
- Stochastic Simulation (individual view)

- Large PEPA model
  - TDSHA (hybrid automaton)
  - hybrid view

- Example

- Continuous Approximation
- Fluid-Flow Semantics
- Conclusions
Alternative Representations

Large PEPA model

- ODEs (population view)
- TDSHA (hybrid automaton) (hybrid view)
- Stochastic Simulation CTMC (individual view)