Fluid Flow Approximation of PEPA models

J. Hillston
LFCS, School of Informatics
University of Edinburgh, UK
jeh@inf.ed.ac.uk

Abstract

In this paper we present a novel performance analysis technique for large-scale systems modelled in the stochastic process algebra PEPA. In contrast to the well-known approach of analysing via continuous time Markov chains, our underlying mathematical representation is a set of coupled ordinary differential equations (ODEs). This analysis process supports all of the combinators of the PEPA algebra and is well-suited to systems with large numbers of replicated components. The paper presents an elegant procedure for the generation of the ODEs and compares the results of this analysis with more conventional methods.

1 Introduction

Over the last decade process algebras in which quantified durations are associated with activities have enjoyed considerable success. In many cases the duration is assumed to be an exponentially distributed random variable resulting in an underlying mathematical model which is a continuous time Markov chain. Such models may be *solved* for either steady state or transient probability distributions via linear algebra. When more general random variables are assumed the underlying mathematical model is a generalised semi-Markov process and simulation will typically be the means of analysis. In either case there is a problem of *state space explosion*. Like most discrete state-based modelling formalisms, process algebras are prone to the failing that apparently simple models can readily generate extremely large state spaces making numerical solution via linear algebra very costly or even intractable, and simulation time-consuming and potentially inaccurate. Much research ingenuity has gone into tackling the problem of state space explosion for continuous time Markov chains (e.g. [8, 3, 13]), and consequently the size of process which can be solved does continue to grow, albeit slowly.

In this paper we propose a radically different approach to tackling this problem when modelling with a process algebra such as PEPA. The approach is based on two shifts from the usual perspective:

- Firstly, we do not aim to calculate the probability distribution over the entire state space of the model. We choose a more abstract state representation in terms of state variables, quantifying the types of behaviour evident in the model.
- Secondly, we assume that these state variables are subject to *continuous* rather than *discrete* change.

Once these adjustments are made the system is amenable to efficient solution as a set of ordinary differential equations (ODEs), leading to the evaluation of transient, and in the limit, steady state measures.

The remainder of this paper is organised as follows. Section 2 presents some background information on PEPA and introduces the new state representation, *numerical vector form*. The shift to continuous analysis is explained in Section 3 and illustrated by an example in Section 4. The relationship of the approach to other techniques is discussed in Section 5, while Section 6 concludes the paper.

2 PEPA

PEPA has been used to study the performance of a wide variety of systems [11, 1, 2, 21, 12]. As in all process algebras, systems are represented in PEPA as the composition of *components* which undertake *actions*. In PEPA the actions are assumed to have a duration, or delay. Thus the expression $(\alpha, r).P$ denotes a component which can undertake an α action, at rate r to evolve into a component P. Here $\alpha \in \mathcal{A}$ where $\mathcal{A}$ is the set of action types and $P \in \mathcal{C}$ where $\mathcal{C}$ is the set of component types.

PEPA has a small set of combinators, allowing system descriptions to be built up as the concurrent execution and interaction of simple sequential components. We informally introduce the syntax below. More detail can be found in [10]. The structured operational semantics are shown in Figure 1.

Prefix

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

Choice

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

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

Cooperation

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

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

$$\frac{E \xrightarrow{(\alpha,r_1)} E' \quad F \xrightarrow{(\alpha,r_2)} F'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,R)} E' \underset{L}{\bowtie} F'} \ (\alpha \in L)$$

$$\text{where } R = \frac{r_1}{r_\alpha(E)} \frac{r_2}{r_\alpha(F)} \min(r_\alpha(E), r_\alpha(F))$$

Hiding

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

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

Constant

$$\frac{E \xrightarrow{(\alpha,r)} E'}{A \xrightarrow{(\alpha,r)} E'} \ (A \overset{def}{=} E)$$

Figure 1. PEPA Structured Operational Semantics

Prefix: The basic mechanism for describing the behaviour of a system with a PEPA model is to give a component a designated first action using the prefix combinator, denoted by a full stop, which was introduced above. As explained, $(\alpha, r).P$ carries out an α action with rate r, and it subsequently behaves as P.

Choice: The component $P + Q$ represents a system which may behave either as P or as Q. The activities of both P and Q are enabled. The first activity to complete distinguishes one of them: the other is discarded. The system will behave as the derivative resulting from the evolution of the chosen component.

Constant: It is convenient to be able to assign names to patterns of behaviour associated with components. Constants are components whose meaning is given by a defining equation. The notation for this is $X \overset{def}{=} E$. The name X is in scope in the expression on the right hand side meaning that, for example, $X \overset{def}{=} (\alpha, r).X$ performs α at rate r forever.

Hiding: The possibility to abstract away some aspects of a component's behaviour is provided by the hiding operator, denoted P/L. Here, the set L identifies those activities which are to be considered internal or private to the component and which will appear as the unknown type τ.

Cooperation: We write $P \underset{L}{\bowtie} Q$ to denote cooperation between P and Q over L. The set which is used as the subscript to the cooperation symbol, the *cooperation set* L, determines those activities on which the *cooperands* are forced to synchronise. For action types not in L, the components proceed independently and concurrently with their enabled activities. We write $P \parallel Q$ as an abbreviation for $P \underset{L}{\bowtie} Q$ when L is empty.

If a component enables an activity whose action type is in the cooperation set it will not be able to proceed with that activity until the other component also enables an activity of that type. The two components then proceed together to complete the *shared activity*. The rate of the shared activity may be altered to reflect the work carried out by both components to complete the activity. The total capacity of a component C to carry out activities of type α is termed the *apparent rate* of α in P, denoted $r_\alpha(P)$. Unlike some other stochastic process algebras, PEPA assumes *bounded capacity*: a component cannot be made to perform an activity faster by cooperation, so the rate of a shared activity is the minimum of the rates of the activity in the cooperating components.

In some cases, when an activity is known to be carried out in cooperation with another component, a component may be *passive* with respect to that activity. This means that the rate of the activity is left unspecified (denoted $\top$) and is determined upon cooperation, by the rate of the activity in the other component. All passive actions must be synchronised in the final model.

The syntax may be formally introduced by means of the following grammar:

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

where S denotes a *sequential component* and P denotes a *model component* which executes in parallel. C stands for a constant which denotes either a sequential component or a model component. C_S stands for constants which denote sequential components. The effect of this syntactic separation between these types of constants is to constrain legal PEPA components to be cooperations of sequential processes, a necessary condition for an ergodic underlying Markov process.

2.1 State representation

In process algebra models the usual state representation is in terms of the syntactic forms of the model expression. The structured operational semantics define how a model may evolve and these may be applied exhaustively to form a *labelled transition system* (usually termed the *derivation graph* in PEPA) representing the state space of the model. This is a graph in which each node is a distinct syntactic form or *derivative* (or equivalence class of syntactic expressions up to strong equivalence) and each arc represents a possible activity causing the state change. It is important to note that in PEPA the state representation is in fact a labelled *multi-transition* system because recording the multiplicity of arcs is crucial, particularly when repeated components are involved.

When a Markovian interpretation is put on the PEPA model the duration of each activity is assumed to be a random variable governed by a negative exponential distribution. In this case the derivation graph can be considered to be the state transition diagram of a continuous time Markov chain. Thus one Markovian state is associated with each syntactic term. Performance analysis is then carried out in terms of the steady state probability distribution, or the transient probability distribution, which is extremely costly when the state space is large.

Rather than the complete syntactic form, since the static cooperation combinators remain unchanged in all states, it is often convenient to represent the states of the model in *vector form*. The state vector records one entry for each sequential component of the PEPA model. These components will be present in each derivative of the model, although they will change their local state or derivative. Thus the global state can be represented as a vector or sequence of local derivatives.[1]

If a model contains equivalent components there may be multiple states within the model which exhibit the same behaviour and so we may *aggregate* the model. The derivation graph is then constructed in terms of equivalence classes of syntactic terms and this is used as the basis of the CTMC construction [7]. At the heart of this technique is the use of a *canonical state vector* to capture the syntactic form of a model expression. If two states have the same canonical state vector they are equivalent and need not be distinguished in the aggregated derivation graph. Canonicalisation involves reordering entries within the vector in a way that strong equivalence, the Markovian bisimulation of PEPA models, is respected, but which places elements within subvectors of equivalent components in lexicographical order. Further details can be found in [7].

In this paper we propose an alternative vector form for capturing the state information of models with repeated components. In the state vector form, even when the canonical representation is used there is one entry in the vector for each sequential component in the model. When the number of repeated components becomes large this can be prohibitively expensive in terms of storage. In the alternative vector form there is one entry for each local derivative of each type of component in the model. Two components have the same type if their derivation graphs are isomorphic. The entries in the vector are no longer syntactic terms representing the local derivative of the sequential component, but the *number* of components currently exhibiting this local derivative.

To clarify the distinction between the two vector forms consider the small example defined below, consisting of interacting processors and resources:

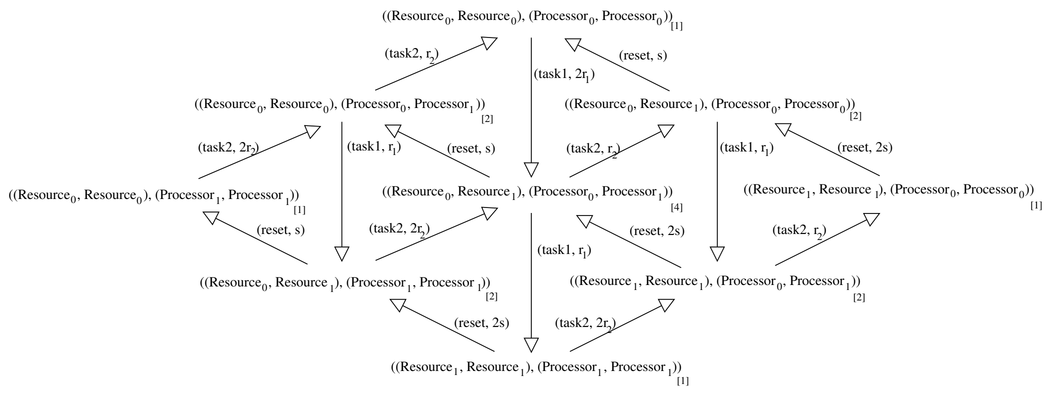
$$
\begin{aligned}
\mathit{Processor}_0 &\overset{\mathit{def}}{=} (\mathit{task}1, r_1).\mathit{Processor}_1\\
\mathit{Processor}_1 &\overset{\mathit{def}}{=} (\mathit{task}2, r_2).\mathit{Processor}_0\\
\mathit{Resource}_0 &\overset{\mathit{def}}{=} (\mathit{task}1, r_1).\mathit{Resource}_1\\
\mathit{Resource}_1 &\overset{\mathit{def}}{=} (\mathit{reset}, s).\mathit{Resource}_0
\end{aligned}
$$

$$(\mathit{Resource}_0 \parallel \mathit{Resource}_0) \underset{\{\mathit{task}1\}}{\bowtie} (\mathit{Processor}_0 \parallel \mathit{Processor}_0)$$

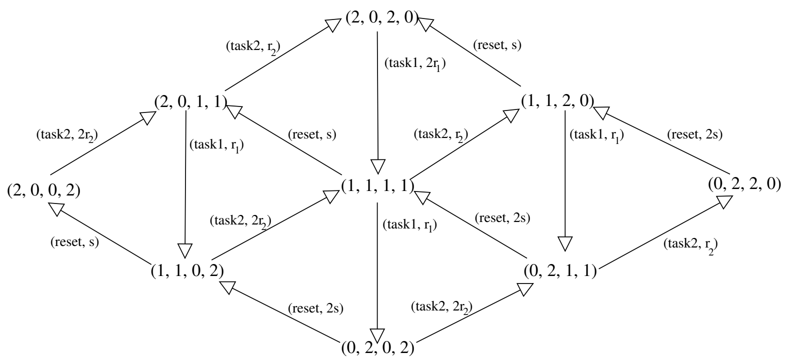
The canonical state vector form corresponding to this example with the given configuration is shown in Figure 2a). Here the initial state is represented explicitly as $((\mathit{Resource}_0, \mathit{Resource}_0), (\mathit{Processor}_0, \mathit{Processor}_0))_{\{\mathit{task}1\}}$. In contrast, in the numerical vector form, shown in Figure 2b), the initial state is $(2, 0, 2, 0)$ where the entries in the vector are counting the number of $\mathit{Resource}_0$, $\mathit{Resource}_1$, $\mathit{Processor}_0$, $\mathit{Processor}_1$ local derivatives respectively, exhibited in the current state. In the canonical state vector representation we record the number of elements in each equivalence class (shown in square brackets in Figure 2a). The total rate of the transitions between the canonical states is derived from this number of instances, the number of enabled activities and their relative probabilities. In the numerical state vector representation each vector is a single state and the rates of the transitions between states are derived directly from the vector and the activity rate, as explained below.

In the current configuration of the model, with two instances of each component type, it is clear that the state vector form and the numerical vector form each have four elements, but if we consider a configuration with ten instances of each component type it becomes clear that the numerical form is much more compact. Moreover, it is without any significant loss of information.

[1]For the remainder of this paper we use the term *local derivative* to refer to the local state of a single sequential component, whereas *derivative* will be used to refer to a global state represented in its syntactic form.


((Resource$_0$, Resource$_0$), (Processor$_0$, Processor$_0$))$_{[1]}$
(task2, r$_2$)
(task1, 2r$_1$)
(reset, s)
((Resource$_0$, Resource$_0$), (Processor$_0$, Processor$_1$))$_{[2]}$
((Resource$_0$, Resource$_1$), (Processor$_0$, Processor$_0$))$_{[2]}$
(task2, 2r$_2$)
(task1, r$_1$)
(reset, s)
(task2, r$_2$)
(task1, r$_1$)
(reset, 2s)
((Resource$_0$, Resource$_0$), (Processor$_1$, Processor$_1$))$_{[1]}$
((Resource$_0$, Resource$_1$), (Processor$_0$, Processor$_1$))$_{[4]}$
((Resource$_1$, Resource$_1$), (Processor$_0$, Processor$_0$))$_{[1]}$
(reset, s)
(task2, 2r$_2$)
(task1, r$_1$)
(reset, 2s)
(task2, r$_2$)
((Resource$_0$, Resource$_1$), (Processor$_1$, Processor$_1$))$_{[2]}$
((Resource$_1$, Resource$_1$), (Processor$_0$, Processor$_1$))$_{[2]}$
(reset, 2s)
(task2, 2r$_2$)
((Resource$_1$, Resource$_1$), (Processor$_1$, Processor$_1$))$_{[1]}$

a) Aggregated state space in canonical form



b) Vector state space

Figure 2. Illustrative example of contrasting state representations

The numerical vector form for an arbitrary PEPA model is defined as follows.

Definition 2.1 (Numerical Vector Form) *For an arbitrary PEPA model $\mathcal{M}$ with n component types $\mathcal{C}_i, i = 1, \ldots, n$, each with N_i distinct derivatives, the* numerical vector form *of $\mathcal{M}$, $\mathcal{V}(\mathcal{M})$, is a vector with $N = \sum_{i=1}^{n} N_i$ entries. The entry v_{i_j} records how many instances of the jth local derivative of component type $\mathcal{C}_i$ are exhibited in the current state.*

3 Continuous State Space Approximation

The numerical vector form presents an alternative way of presenting the state space of a PEPA model but can nevertheless be used as the basis of the usual Markovian analysis, albeit in terms of an aggregated model. Since the aggregation is based on strong equivalence we know that the aggregated CTMC is lumpably equivalent to the original CTMC derived from the derivation graph [9]. The aggregated model is isomorphic to that produced by the canonical vector form and the automatic aggregation presented in [7].

However, the focus of this paper is the use of this vector form as the basis for a fluid approximation when each component type in the model is replicated a large number of times. If this is the case the domain of values of each entry in $\mathcal{V}(\mathcal{M})$ is large. If K_i is the number of components of type $\mathcal{C}_i$ in the initial configuration of the model then each entry in the ith subvector will have domain $0, \ldots, K_i$.

Consider an arbitrary state $\mathcal{M}'$ of the model $\mathcal{M}$ which has the particular numerical vector representation $\mathcal{V}(\mathcal{M}')$. When a state change occurs it can happen in two distinct ways:

- A single sequential component, an instance of component type $\mathcal{C}_i$ may engage in an *individual action*. In this case the impact on $\mathcal{V}(\mathcal{M}')$ is that within the ith subvector one entry is incremented by one while another is decremented by one, reflecting the evolution of this single component from one local derivative to another.
- Alternatively a *shared action* may be performed resulting in the simultaneous evolution of two or more sequential components of distinct types (since we as-

sume that replicated components are independent of each other). Thus a number of distinct subvectors may need to be updated within $\mathcal{V}(\mathcal{M}')$. However in each case one entry is incremented by one and one entry is decremented by one.

The system is inherently discrete with the entries within the numerical vector form always being non-negative integers and always being incremented or decremented in steps of one. When the numbers of components are large these steps are relatively small and we can approximate the behaviour by considering the movement between states to be continuous, rather than occurring in discontinuous jumps. Thus our objective is to replace the discrete event system represented by the derivation graph of a PEPA process by a continuous model, represented by a set of coupled ordinary differential equations. The numerical vector form of state representation is an intermediate step to achieving that.

We start with some preliminary definitions. Consider a local derivative D of a sequential component. An activity (α, r) is an *exit activity* of D if D enables (α, r), i.e. there is a transition $D \xrightarrow{(\alpha,r)}$ in the labelled transition system of D. We denote the set of exit activities of D by $Ex(D)$. Conversely, we denote the set of local derivatives for which (α, r) is an exit activity by $Ex(\alpha, r)$. Similarly, an activity (β, s) is an *entry activity* of D if there is a derivative D' which enables (β, s) and D is the one-step β-derivative of D', i.e. $D' \xrightarrow{(\beta,s)} D$ is in the labelled transition system of D'. We use $En(D)$ to denote the set of entry activities of D.

This categorisation of activities is needed because it is important to record the impact of each activity on each local derivative, C_{i_j}, in the model. Recall that in the vector form the numbers of these derivatives, $N(C_{i_j})$, have become our state variables.

Let us consider the evolution of the numerical state vector. Let $v_{i_j}(t) = N(\mathcal{C}_{i_j}, t)$ denote the jth entry of the ith subvector at time t, i.e. the number of instances of the jth local derivative of sequential component $\mathcal{C}_i$. In a short time δt the change to this arbitrary vector entry will be:

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

The first term records the impact of exit activities. If the exit activity is an individual activity of this component $Ex(\alpha, r) = \{\mathcal{C}_{i_j}\}$ and $\min(N(\mathcal{C}_{k_l}, t)) = N(\mathcal{C}_{i_j}, t)$ i.e. there will be $N(\mathcal{C}_{i_j}, t)$ instances of the local derivative each proceeding with the individual activity concurrently. When $Ex(\alpha, r) \neq \{\mathcal{C}_{i_j}\}$ the activity is a shared activity involving local derivatives from two or more component types in a multiway synchronisation. By the definition of apparent rate in PEPA, if there are N replicated instances of a component offering an activity (α, r), the apparent rate of the activity will be $N \times r$. By the semantics, the apparent rate of a synchronised activity is the minimum of the apparent rates of the cooperating components. The second term is explained similarly, noting that the rate of an entry activity will be determined by the number of components for which this is an *exit* activity, in accordance with the semantics of the language.

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

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

In the following subsection we show how these equations may be derived automatically from the model definition in a straightforward way. To fully specify the system of ODEs it only remains to set the initial values of the state variables, i.e. $N(C_{i_j}, 0)$ for $i_j = 1, \ldots, N$. These are easily recorded from the initial model configuration.

3.1 Automatically deriving ODEs

The impact of activities on derivatives can be recorded in either a graph or a matrix form, easily derived from the syntactic presentation of the model, as defined below.

Definition 3.1 (Activity Graph) *An* activity graph *is a bipartite graph (N, A). The nodes N are partitioned into N_a, the* activities, *and N_d, the* derivatives. *$A \subseteq (N_a \times N_d) \cup (N_d \times N_a)$, where $a = (n_a, n_d) \in A$ if n_a is an exit activity of derivative n_d, and $a = (n_d, n_a) \in A$ if n_a is an entry activity of derivative n_d.*

The same information can be represented in a matrix, termed the *activity matrix*.

Definition 3.2 (Activity Matrix) *For a model with N_A activities and N_D distinct local derivatives, the* activity matrix *M_a is an $N_D \times N_A$ matrix, and the entries are defined as follows.*

$$(d_i, a_j) = \begin{cases} +1 & \text{if } a_j \text{ is an entry activity of } d_i \\ -1 & \text{if } a_j \text{ is an exit activity of } d_i \\ 0 & \text{otherwise.} \end{cases}$$

```
//Form one ODE for each local derivative/state variable
For $i = 1...N_D$
  //Find the activities involving this derivative
    For $j = 1....N_A$
      If $M_a(i,j) \neq 0$
        //Form exit set $Ex(j)$ for activity $j$
          $Ex(j) = \emptyset$
            For $k = 1...N_D$
              If $M_a(k,j) = -1$
                $Ex(j) = Ex(j) \cup \{k\}$
        //Record the impact of each such activity
        If $M_a(i,j) = +1$
            Add
                    $+r_j \times \min_{k \in Ex(j)} (n_k(t))$
            to the equation
        If $M_a(i,j) = -1$
            Add
                    $-r_j \times \min_{k \in Ex(j)} (n_k(t))$
            to the equation
```

Figure 3. Pseudo-code for generating the set of ODEs

In the activity matrix each row corresponds to a single local derivative. In the representation of the model as a system of ODEs there is one equation for each state variable, i.e. for the current number of each local derivative exhibited. This equation details the impact of the rest of the system on the value of that state variable. This can be derived automatically from the activity matrix when we associate a state variable n_i with each row of the matrix and a rate constant r_j with each column of the matrix. The number of terms in the ODE will be equal to the number of non-zero entries in the corresponding row, each term being based on the rate of the activity associated with that column. As explained above, by the semantics of PEPA, the actual rate of change caused by each activity will be the rate multiplied by the minimum of the current number of local derivatives enabling that activity in parallel, for each cooperating component type. The identity of these derivatives can be found in the column corresponding to the activity, a negative entry indicating that this derivative participates in that activity. There will be one ODE in the system for each row of the matrix

3.2 Small example revisited

Let us consider again the small example considered earlier, assuming now that there are large numbers of processors and resources:

$$\begin{aligned}
Processor_0 &\overset{def}{=} (task1, r_1).Processor_1\\
Processor_1 &\overset{def}{=} (task2, r_2).Processor_0\\
Resource_0 &\overset{def}{=} (task1, r_1).Resource_1\\
Resource_1 &\overset{def}{=} (reset, s).Resource_0
\end{aligned}$$

$$(Resource_0 \parallel \cdots \parallel Resource_0) \underset{\{task1\}}{\bowtie} (Processor_0 \parallel \cdots \parallel Processor_0)$$

The activity graph is as depicted in Figure 4 while the activity matrix is as depicted in Figure 5.

From the matrix, we derive each differential equation in turn. For state variable n_i, consider row i. Each non-zero entry in the row will results in one term within the equation.

$$\begin{aligned}
\frac{dn_1(t)}{dt} &= -r_1 \min(n_1(t), n_3(t)) + r_2 n_2(t)\\
\frac{dn_2(t)}{dt} &= r_1 \min(n_1(t), n_3(t)) - r_2 n_2(t)\\
\frac{dn_3(t)}{dt} &= -r_1 \min(n_1(t), n_3(t)) + s n_4(t)\\
\frac{dn_4(t)}{dt} &= r_1 \min(n_1(t), n_3(t)) - s n_4(t)
\end{aligned}$$

Note that the form of the system of equations is independent of the number of components included in the initial

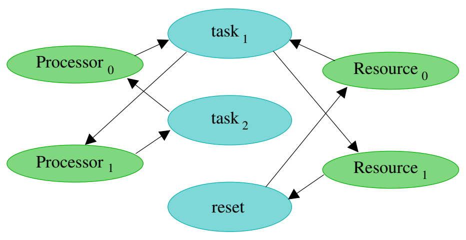


Figure 4. Activity diagram for the simple Processor-Resource model

	task$_1$	*task*$_2$	*reset*	
Processor$_0$	-1	$+1$	0	n_1
Processor$_1$	$+1$	-1	0	n_2
Resource$_0$	-1	0	$+1$	n_3
Resource$_1$	$+1$	0	-1	n_4

Figure 5. Activity matrix for the simple Processor-Resource model

configuration of the model. The only impact of changing the number of instances of each component type is to alter the initial conditions. Thus, if there are initially 200 processors, all starting in state *Processor*$_0$ and 120 resources, 50% of which start in state *Resource*$_0$ and 50% in state *Resource*$_1$, the initial conditions will be:

$$n_1(0) = 200 \quad n_2(0) = 0 \quad n_3(0) = 60 \quad n_4(0) = 60$$

4 Example

We present an example model which demonstrates the use of the ODE-based analysis procedure and relates this to existing performance analysis methods for PEPA models.

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.

To ensure scalability the Web service is replicated across multiple hosts. Multiple brokers are available too. 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. Second party clients need to use encryption to ensure authenticity and confidentiality properties whereas 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 second party client.

A schematic of the communication topology of the system is shown in Figure 6.

The model files processed by the modelling tools use long descriptive identifiers such `FirstPartyClient`. For brevity in presenting the model on the printed page we abbreviate "first party client" and "second party client" to *FPC* and *SPC* respectively. We abbreviate "Web service" to *WS*.

4.1 The PEPA model

A second party client cycles through a lifetime of composing service requests, encrypting these and sending 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. As usual with PEPA models this component contains some individual activities which it itself performs (the composition and encryption) and some activities which are performed in co-operation with another component (the request and response are in co-operation with the broker).

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

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.

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

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. The method of invoking the Web service may also be different because the service may be invoked by a remote method invocation to the host machine instead of via an HTTP request. Thus the first party client experiences the Web service as a blocking remote method invocation.

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

We model a thread of execution on the 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 as an HTTP

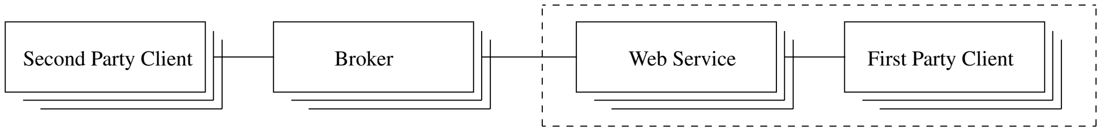


Figure 6. Web service with first and second party clients and brokers

response or as a direct value.

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

In the initial state of the system model we represent each of the four component types being initially in their idle state. The synchronisation sets $\mathcal{K}$, $\mathcal{L}$ and $\mathcal{M}$ synchronise the components on their common activity names.

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

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

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.[2]

4.2 Cost of analysis

Performance models admit many different types of analysis. Some have lower evaluation cost, but are less informative, such as steady-state analysis. Others have higher evaluation cost, but are more informative, such as transient analysis. We compare ODE-based evaluation against other techniques which could be used to analyse the model. We compare against steady-state and transient analysis as implemented by the PRISM probabilistic model-checker [17], which provides PEPA as one of its input languages. We also compare against Monte Carlo Markov Chain simulation.

[2]The example here has been chosen for its simplicity. It consists of components with only simple cycles of behaviour but this is not a restriction of the technique. The approach works equally well with components with any sequential form.

Note that Figure 7 reports only a single run of the transient analysis and simulation. In practice, due to the stochastic nature of the analyses, these would need to be re-run multiple times to produce results comparable to the ODE-based analysis. Moreover, note that the number of ODEs is constant regardless of the number of components in the system, whilst the state space grows dramatically.

The ODE integrator which we used is a Java implementation of the Dormand-Prince fifth-order Runge-Kutta solver [4]. The version of PRISM which we used is 2.1.dev4 [19]. The Monte-Carlo Markov Chain simulator is a Java implementation of Gillespie's Direct method [5]. The runtimes which are reported are elapsed (wall clock) times as reported by GNU time version 1.7. All timings were made on a 1.60GHz Pentium IV with 1Gb RAM running Red Hat Linux 9, averaged over a number of runs.

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 computation (time in seconds)	Hybrid matrix/MTBDD steady-state computation (time in seconds)	Transient solution for time $t = 100$ (time in seconds)	Monte Carlo MC simulation one run to time $t = 100$ (time in seconds)	ODE solution (time in seconds)
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	234,702,336	30,251,627	–	–	–	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

Figure 7. Running times from analyses

The observation from Figure 7 is that the running time to obtain results by ODE solution compares favourably to the other approaches tried and scales better than others as the number of instances of components increases.

4.3 Comparison of results

In Figure 8 we show the results from our solution of the PEPA Web Service model as a system of ODEs with the number of clients of both kinds, brokers, and web service instances all 1000. The results as presented from our ODE integrator are time-series plots of the number of each type of component behaviour as a function of time. The graphs show fluctuations in the numbers of components with respect to time from $t = 0$ to $t = 100$ for estimated values of rates for the activities of the system. We can observe an

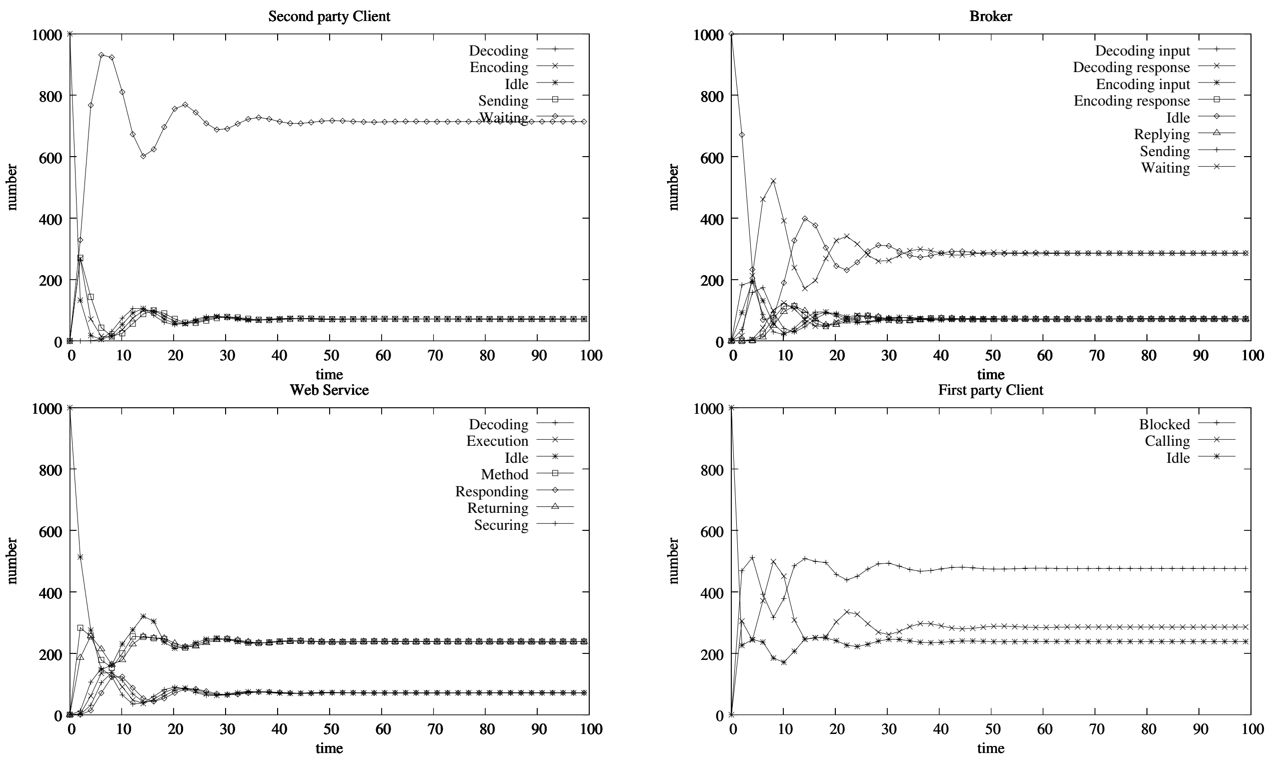


Figure 8. Time series plots of components

initial flurry of activity until the system stabilises into its steady-state equilibrium at time (around) $t = 50$.

We compared the steady-state likelihoods of component types with the probabilities computed using the PEPA Workbench [6] on the smallest instance of the model (one of each type of component) and scaling the probabilities by multiplying by 1000. (Such an approach is legitimate in the absence of blocking when the numbers of each component are equal but could not have been used for a model with different numbers of clients, brokers and services.) We found good agreement with the results obtained by ODE solution after the system stabilises into its steady-state equilibrium.

5 Discussion

The approach that we take is quite distinct from that for fluid queues [16] and fluid stochastic Petri nets [22]. In those modelling techniques the model has a mixture of fluid or continuous elements and the usual discrete state space elements. For example in a fluid queue the service process may be modelled as a fluid process, servicing a buffer which is the continuous element, but the arrival process itself will in general be governed by a (discrete) Markov process. Similarly in a fluid stochastic Petri net only some places will be fluid, having a continuous marking domain, other places remaining discrete in nature.

This is in contrast with our approach in which all state variables are approximated by continuous variables. Thus it is closer to the *diffusion* approximation technique for queueing networks [14, 15].

6 Concluding remarks and future work

The problem of state space explosion has challenged numerical solution of Markovian models for a generation. In this paper we propose a means of avoiding this problem for large scale models of repeated components, represented in PEPA. By adopting a continuous approximation of the behaviour of the model we are able to analyse systems of arbitrarily large scale. Presently we make some assumptions about the form of the PEPA models: that components of the same type do not cooperate and that all cooperating components have the same local rate for shared activities. However, work is progressing on relaxing these assumptions.

Whilst the class of models currently considered is restricted it is not without interest. Many application domains naturally give rise to models in this style. For example,

- epidemiology, both for natural populations with respect to disease and for computer software populations with respect to worms and viruses;

- wireless networks;
- Grid computing and other large scale service environments.

If the rates of change within the model are generalised to allow activity rates to be governed by probability distributions rather than being deterministic the evolution of the system can be described by a set of random differential equations [20]. A further generalisation, introducing more uncertainty, is offered by *stochastic differential equations*[18]. In future work we plan to investigate the use of these more sophisticated forms of differential equations to evaluate the behaviour of PEPA models.

Acknowledgements

The author would like to thank Stephen Gilmore for his helpful discussions in shaping this work, Muffy Calder for providing the initial stimulus, and Jeremy Bradley for useful comments on an earlier draft. This work was supported in part by the ENHANCE project (EPSRC project GR/S21717/01) and in part by the SENSORIA project (EU FET-IST Global Computing 2 project 016004).

References

[1] H. Bowman, J. Bryans, and J. Derrick. Analysis of a multimedia stream using stochastic process algebra. In C. Priami, editor, *6th Int. Workshop on Process Algebras and Performance Modelling*, pages 51–69, Nice, September 1998.

[2] J. T. Bradley, N. J. Dingle, S. T. Gilmore, and W. J. Knottenbelt. Derivation of passage-time densities in PEPA models using ipc: the Imperial PEPA Compiler. In G. Kotsis, editor, *Proc. of the 11th IEEE/ACM Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunications Systems*, pages 344–351, October 2003. IEEE Computer Society Press.

[3] D. Deavours and W. Sanders. An efficient disk-based tool for solving large Markov models. *Performance Evaluation*, 33:67–84, 1998.

[4] J. Dormand and P. J. Prince. A family of embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, 6:19–26, 1980.

[5] D. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical species. *J. Comp. Phys.*, 22:403–434, 1976.

[6] S. Gilmore and J. Hillston. The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling. In *Proc. of 7th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, number 794 in LNCS, pages 353–368, Vienna, May 1994. Springer-Verlag.

[7] S. Gilmore, J. Hillston, and M. Ribaudo. An efficient algorithm for aggregating PEPA models. *IEEE Transactions on Software Engineering*, 27(5):449–464, May 2001.

[8] H. Hermanns, J. Meyer-Kayser, and M. Siegle. Multi-terminal binary decision diagrams to represent and analyse continuous time Markov chains. In *Proc. of 3rd Intl. Workshop on the Numerical Solution of Markov Chains*, pages 188–207, 1999.

[9] J. Hillston. Compositional Markovian modelling using a process algebra. In W. Stewart, editor, *Proc. of 2nd Int. Workshop on Numerical Solution of Markov Chains: Computations with Markov Chains*, Raleigh, North Carolina, Jan. 1995. Kluwer Academic Press.

[10] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[11] J. Hillston. Tuning systems: From composition to performance. *The Computer Journal*, 2005. To appear.

[12] J. Hillston, L. Kloul, and A. Mokhtari. Towards a feasible active networking scenario. *Telecommunication Systems*, 27(2–4):413–438, 2004.

[13] W. J. Knottenbelt. *Parallel Performance Analysis of Large Markov Models*. PhD thesis, Department of Computing, Imperial College, 1999.

[14] H. Kobayashi. Application of the diffusion approximation to queueing networks I: Equilibrium queue distributions. *J. ACM*, 21(2):316–328, 1974.

[15] H. Kobayashi. Application of the diffusion approximation to queueing networks II: Nonequilibrium distributions and applications to computer modeling. *J. ACM*, 21(3):459–469, 1974.

[16] V. Kulkarni. Fluid models for single buffer systems. In *Frontiers in Queueing, Models and Applications in Science and Engineering*, Probability and Stochastic Series, pages 321–338. CRC Press, 1997.

[17] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. In *Proc. 8th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)*, volume 2280 of *LNCS*, pages 52–66. Springer, April 2002.

[18] B. Oksendal. *Stochastic Differential Equations*. Springer, 2003.

[19] D. Parker, G. Norman, and M. Kwiatkowska. *PRISM 2.1.dev4 User's Guide*. School of Computer Science, University of Birmingham, Jan. 2005.

[20] T. Soong. *Random Differential Equations in Science and Engineering*. Academic Press, 1973.

[21] N. Thomas, J. T. Bradley, and W. J. Knottenbelt. Stochastic analysis of scheduling strategies in a grid-based resource model. *IEE Software Engineering*, 151(5):232–239, September 2004.

[22] K. Trivedi and V. Kulkarni. FSPNs: Fluid stochastic Petri nets. In M. A. Marsan, editor, *Application and Theory of Petri Nets, Proc. 14th Int. Conference*, volume 691 of *LNCS*, pages 24–31, Chicago, Illinois, USA, 1993. Springer-Verlag.