An approximate solution of PEPA models using component substitution

Nigel Thomas[*], Jeremy Bradley[†] and David Thornley[†]

Abstract

Performance models specified using compositional algebra suffer the well-known state space explosion problem, where a relatively small definition leads to a Markov chain with a large state space that is problematic to solve. As a result it is widely recognised that the development of techniques to solve performance models efficiently is of particular practical importance. Recently the notion of behavioural independence was introduced to exploit the structure of Markovian process algebra models in order to solve models in a compositional manner. In this paper the opposite property, namely control, is used to solve models by substituting components in the model with simpler versions. The approach is validated through two examples and by deriving a variety of performance measures.

1 Introduction

Stochastic process algebra are widely recognised as a good way of specifying performance models since they allow the modeller to exploit the compositionality of systems in the specification. However, this advantageous property for specification often leads to difficulty in solution, as a

[*]Department of Computer Science, University of Durham. *Nigel.Thomas@durham.ac.uk*

[†]Department of Computing, Imperial College of Science, Technology and Medecine, University of London

model with several components will generally lead to a large underlying continuous time Markov chain (CTMC) that requires significant effort to solve. As a result a key research topic in recent years has been to identify efficient methods for analysing and solving stochastic process algebra models by decomposition (see [1, 2]). This means using the component structure of models to drive the solution, ideally generating isolated solutions for individual components that can be combined to solve the entire model without needing to generate the global state space of the complete model. One element of this work has been the identification and characterisation of product form solutions, using properties such as reversibility [3] and quasi-reversibility [4]. Another focus has been the study of classes of model, which do not have a product form solution, but can nevertheless be decomposed under certain conditions to provide solutions to certain global measures by obtaining marginal distributions for components.

In this paper a large class of models is considered where decomposition in the classical sense is not possible because there is a strong interaction between the elements. Our approach to this problem is to form simpler versions of the model with far fewer states, which can be solved to give an approximate solution of the marginal distributions for each component. The approach described here is inspired by Gribaudi and Sereno [5] who used a similar technique when solving certain Petri net models. A similar approach was explored by Mertziotakis [6], using the TIPP stochastic process algebra, based on earlier Petri net work by Campos et al [7] and queueing network results from Agrawal et al [8]. All these approaches are based on the notion of *response time preservation*, which in turn is similar in concept to the notion of *flow equivalent servers* used to efficiently solve certain product form queueing networks (see [9] for example). The basis for this method is a classical divide and conquer algorithm, where a number of service centres in a model are replaced with a single queue which has the same response time as that estimated for the original service centres. This approach is repeated for all the service centres in the model until a convergence is reached and hence an approximation for the response time for the network

is obtained.

Stochastic process algebra models do not generally exhibit the flow condition that queueing network models do, hence it is necessary to introduce a notion of influence by which a concept of sequence of actions across successive components can be observed. Mertziotakis [6] applied a separation by identifying an interface process that provided a pivot point between interacting components. This interface process was used to calculate a response time between interactions in an iterative method, whereby each component was ignored in an alternating pattern until the calculated response time reached convergence. The approach applied here uses a very similar iterative algorithm, but we do not rely on separating the interface as in [6] but rather a component substitution is made. Because of this distinction the class of model considered by Mertziotakis is somewhat different to that considered here and so, although the numerical results presented here are somewhat more impressive, the methods are applicable to subtly different scenarios.

The class of model studied here relies on a property referred to as *behavioural independence*, which has recently been used by Thomas [10, 11] to give clear definitions for certain restricted classes of model subject to decomposed solution. Essentially this property states that the behaviour of a particular group of components in a model is not influenced by the behaviour of other components in the model. This non-trivial property can be a powerful tool in identifying independent and semi-independent behaviours. If behavioural independence does not exist then by definition another component must be exerting *control*. In this paper we exploit the existence of control in models where all components not only exert control, but also have control exerted on them. A component in the model can be replaced with a simpler version of itself, thus the resultant model gives rise to a CTMC that is also much reduced in size and consequently easier to solve. This simple component offers the same set of interactions as the component it replaced so continues to exert control, but in general has far fewer states. However, in general it is not always possible to know the rates of all transitions within the simple component. Therefore an iterative

approach is used to find a convergence between two or more reduced models. This approach is validated through two examples; the first is a queueing system with blocking and the second follows a case study in multimedia modelling by Bowman et al [12].

In the following sections PEPA is summarised and a number of definitions are made which are necessary to specify the class of model to which our approach applies. The class of model to be studied is then described and the simple component and the solution method specified. To validate the approach a simple queueing example is presented followed by a more complex multimedia model and some concluding remarks.

2 PEPA

A formal presentation of PEPA is given in [13], in this section a brief informal summary is presented. PEPA, being a Markovian Process Algebra, only supports actions that occur with durations that are negative exponentially distributed. Specifications written in PEPA represent Markov processes and can be mapped to a continuous time Markov chain (CTMC). Systems are specified in PEPA in terms of *activities* and *components*. An activity (α, r) is described by the (action) *type* of the activity, α, and the *rate* of the associated negative exponential distribution, r. This rate may be any positive real number, or given as *unspecified* using the symbol $\top$. The syntax for describing components is given as:

$$P ::= (\alpha, r).P \mid P + Q \mid P/L \mid P \underset{L}{\bowtie} Q \mid A$$

The component $(\alpha, r).P$ performs the activity of type α at rate r and then behaves like P. The component $P + Q$ behaves either like P or like Q, the resultant behaviour being given by the first activity to complete. The component P/L behaves exactly like P except that the activities in the set L are concealed, their type is not visible and instead appears as the unknown type τ. Concurrent components can be synchronised, $P \underset{L}{\bowtie} Q$, such that activities in the *cooperation set* L

involve the participation of both components. In PEPA the shared activity occurs at the slowest of the rates of the participants and if a rate is unspecified in a component, the component is passive with respect to that activities of that type. The parallel combinator $\|$ is used as shorthand to denote synchronisation with no shared activities, i.e. $P \| Q \equiv P \underset{\emptyset}{\bowtie} Q$. $A \stackrel{def}{=} P$ gives the constant A the behaviour of the component P.

2.1 Definitions

In the following sections a number of properties of PEPA models will be referred to. Since the definitions of these properties are presented in detail elsewhere [13], only a general description of them is included here. Informally a *derivative* is the "state" of a component defining the current behaviour of a model or component. For example, if $P \stackrel{def}{=} (\alpha, r).Q$ then Q is a derivative of P and if $Q \stackrel{def}{=} (\beta, r_2).R$ then R is a derivative of both Q and P, and so on. The *derivative set*, $ds(P)$, is the set of all the possible derivatives of a component, P. The *current action type set* of a component P, $\mathcal{A}(P)$, contains all the action types (but not the rates) that are enabled in the current derivative of the component P only. The *complete action type set* of a component P, $\vec{\mathcal{A}}(P)$, contains all the action types (but not the rates) that are enabled in any of the derivatives $P' \in ds(P)$, hence $\vec{\mathcal{A}}(P) = \bigcup_{P' \in ds(P)} \mathcal{A}(P')$. The *current activity multiset* of a component P, $\mathcal{A}ct(P)$, contains all the activities (action type and rate pairs) that are enabled in the current derivative of the component P only. In addition it is necessary to construct a number of additional definitions.

Definition 2.1 (Fertile activity)

An activity of type γ is said to be fertile in derivative P_i if $P_i \xrightarrow{\gamma} P_j$ and $i \neq j$.

Thus a fertile activity is one which will change the current derivative of a component. Hence, the current behaviour of a component will only change in response to the completion of a fertile activity. Furthermore, each of the transitions between states in the CTMC will map directly to

an occurrence of a fertile activity. Identifying fertile activities is therefore a useful objective if the state space of the underlying CTMC is to be reduced. The following two definitions refer to sets of fertile activities that are required to define control.

Definition 2.2 (Current fertile action type set)

The current fertile action type set of a component P, denoted $\mathcal{A}_f(P)$, is the set of all action types of activities that are fertile in the current derivative of P.

Definition 2.3 (Complete fertile action type set)

The complete fertile action type set of a component P, denoted $\vec{\mathcal{A}}_f(P)$, is the set of all action types of activities that are fertile in at least one derivative of P.

3 Behavioural independence and control

The method proposed here to tackle the state space explosion problem relies of the properties of *behavioural independence* and *control*. Put simply, behavioural independence requires that components in a model behave identically regardless of the current behaviour of other components in the model (this property is defined formally and discussed in detail in [11]). The trivial case for behavioural independence is clearly that with no shared activities, i.e. $P||Q$, however this is not the only case where components may be considered to be behaviourally independent. Furthermore, the fact that no activities are shared between two components does not mean they will always be behaviourally independent in the presence of other components. For example, in $(P||Q) \underset{L}{\bowtie} R$ the interaction between Q and R may influence the interaction between P and R, causing P and Q to be behaviourally *dependent*.

If a component is not behaviourally independent then it must be dependent on some other component to perform one or more activities during its evolution. This property is referred to as control, and is more formally defined thus:

Definition 3.1 (Control)

The component P is said to be subject to control in the model $P \underset{L}{\bowtie} Q$ if for any $P_i \in ds(P)$

$$\forall\, Q_j, Q_k \in ds(Q)\ s.t.\ (P_i \underset{L}{\bowtie} Q_j), (P_i \underset{L}{\bowtie} Q_k) \in ds(P \underset{L}{\bowtie} Q);$$
$$\mathcal{A}ct\left((P_i \underset{L}{\bowtie} Q_j)/\{\mathcal{A}(P_i \underset{L}{\bowtie} Q_j)/\{\mathcal{A}_f(P_i) \cap L\}\}\right) \neq$$
$$\mathcal{A}ct\left((P_i \underset{L}{\bowtie} Q_k)/\{\mathcal{A}(P_i \underset{L}{\bowtie} Q_k)/\{\mathcal{A}_f(P_i) \cap L\}\}\right)$$

This definition states that the fertile activities in any derivative of P (or the rates at which those activities occur) can change as the current derivative of Q evolves. By changing the set of fertile activities in P, Q is changing the future behaviour of P, i.e. exerting control. This dependence is expressed by saying that component Q *controls* component P over activities of type $K \subset L$ in $P \underset{L}{\bowtie} Q$ if the rate at which an activity of type $k \in K$ can happen in $P_i \in ds(P)$ depends on the current derivative of Q. Clearly, if Q controls P over K then P cannot be behaviourally independent, but the independence, or otherwise, of Q is not given by this statement. The *control set* of P, $\mathcal{K}(P)$ is referred to as all the action types by which it is controlled. For example, if in the model $(P \underset{L}{\bowtie} Q) \underset{M}{\bowtie} R$, Q controls P over K_1 and R controls P over K_2 then $\mathcal{K}(P) = K_1 \cup K_2$. Clearly P is behaviourally independent iff $\mathcal{K}(P) = \emptyset$.

The importance of the notion of control is two-fold. Firstly it allows where a model fails to be behaviourally independent to be explicitly stated. If the number of instances of control (the occurrence of activities through which control is exerted) is small, then it may be possible to create an approximate model where these instances are in some way ignored, or an approximate solution where the intervals between controls are long enough for the model to approach steady state behaviour. The second important use for the notion of control is in building approximations that actually seek to exploit this property. This topic is explored in the remaining sections of this paper.

3.1 Identifying control

Test cases have been developed for behavioural independence that capture many practical occurrences of that property at the component level, rather than the model level. For example, a component P will obviously be behaviourally independent in $P \underset{L}{\bowtie} Q$ if all action types $a \in L$ are defined in every derivative of Q at the same rate. Unfortunately the conditions for behavioural independence that can be applied at the component level are not complete, therefore the failure to identify behavioural independence does not imply control. In theory similar tests are possible for the notion of control, however, whilst it is a simple matter to show that a combination of components does indeed exhibit control, it is another matter to prove that the model can evolve into that combination of components without recourse to the global state space. Thus, even if $Q_i \stackrel{def}{=} (a, \alpha).Q_k$ and $Q_j \stackrel{def}{=} (a, \beta).Q_k$, the component P may still be behaviourally independent in $P \underset{L}{\bowtie} Q$ if there is no $P_i \in ds(P)$ such that both $P_i \underset{L}{\bowtie} Q_i \in ds(P \underset{L}{\bowtie} Q)$ and $P_i \underset{L}{\bowtie} Q_j \in ds(P \underset{L}{\bowtie} Q)$. The clear exception to this problem is where it can be shown that

1. Q can evolve into Q_i and Q_j using only activities that are not shared,
2. there is a $P_i \in ds(P)$ which may also be reached through internal activities,
3. a is defined passively (with rate $\top$) in P_i.

The first two of these conditions is necessary as only internal (not shared) activities can be assumed to happen, since any shared action may be blocked and it is only possible to know whether this is the case by exploring the derivatives of the model $P \underset{L}{\bowtie} Q$. The final condition, that a is defined passively in P_i, is necessary as a shared action may only be assumed to occur at a given symbolic rate if one partner is passive or both partners specify the same symbolic rate.

Clearly this approach to identifying control is restrictive, indeed it is unlikely that any automated approach at the component level is able to identify 100% of cases. Furthermore, control is subject to numerical, as well as symbolic, conditions, thus if $\alpha = \beta$ then no control is exerted in

this case. Clearly at the symbolic level of automated model inspection there is little that can be done to address numerical concerns, although it seems reasonable to assume that if the modeller has gone to the trouble of specifying different symbolic rates α and β for the same action type a, then there is a significant possibility that $\alpha \neq \beta$.

4 Approximation using reduced components

Given a model of two components, $X \underset{L}{\bowtie} Y$, such that X controls Y it is possible to generate X' such that the activities of Y are unaltered with respect to L in $X' \underset{L}{\bowtie} Y$ if for any derivative of $X \underset{L}{\bowtie} Y$ there is a corresponding derivative of $X' \underset{L}{\bowtie} Y$ such that

$$\mathcal{A}ct(X \underset{L}{\bowtie} Y) \cap L = \mathcal{A}ct(X' \underset{L}{\bowtie} Y) \cap L$$

In many situations, such as modelling resources, there will be little or no difference between X and the smallest possible X'. However, when X contains many derivatives, X_i where $\mathcal{A}ct(X_i) \cap L$ does not change, then there is potential for substantial state space reduction. The number of derivatives in X' is dependent on the number of action types in L and the number of distinct rates at which they are enabled. If the rate of each shared action is the same wherever it is enabled then the size of the reduced component X' is $2^{|L|}$. Thus the smaller the cooperation set the better the reduction in state space in the system $X' \underset{L}{\bowtie} Y$. Consider the case where $L = \{a\}$, if the rate of this action is the same in every derivative of $X \underset{\{a\}}{\bowtie} Y$ in which it is enabled then the reduced form of X, X' is given as follows.

$$X'_1 \stackrel{def}{=} (a, r1).X'_1 + (a, r2).X'_2 + (\tau, r3).X'_2$$

$$X'_2 \stackrel{def}{=} (\tau, r4).X'_1$$

The unknown action type τ is used to denote all those action types internal to X such that the derivative of X changes from X_i to X_j and $(\mathcal{A}ct(X_i) \cap L) \neq (\mathcal{A}ct(X_j) \cap L)$ (for all possible

i and j. At this stage it is not possible to know the rates $r1$, $r2$, $r3$ and $r4$ in general, although the sum of the rates $r1$ and $r2$ is clearly the rate of a in every derivative of $X \underset{\{a\}}{\bowtie} Y$ in which it is enabled. If X is behaviourally independent in $X \underset{L}{\bowtie} Y$ then X may be solved in isolation and hence the rates found from the steady state probabilities. However, if Y controls X in $X \underset{L}{\bowtie} Y$ then it is not a simple matter to solve X. Instead an iterative approach is adopted (from [5]) to tackle this problem as follows.

1. Generate the smallest X' such that for any derivative of $X \underset{L}{\bowtie} Y$ there is a corresponding derivative of $X' \underset{L}{\bowtie} Y$ such that $\mathcal{A}ct(X \underset{L}{\bowtie} Y) \cap L = \mathcal{A}ct(X' \underset{L}{\bowtie} Y) \cap L$.

2. Similarly generate the smallest Y' such that for any derivative of $X \underset{L}{\bowtie} Y$ there is a corresponding derivative of $X \underset{L}{\bowtie} Y'$ such that $\mathcal{A}ct(X \underset{L}{\bowtie} Y) \cap L = \mathcal{A}ct(X \underset{L}{\bowtie} Y') \cap L$.

3. Solve $X' \underset{L}{\bowtie} Y$ using estimates of the unknown rates in X'.

4. Use the solution of $X' \underset{L}{\bowtie} Y$ to calculate the unknown rates in Y'.

5. Solve $X \underset{L}{\bowtie} Y'$ to find new estimates of the unknown rates in X'.

6. Solve $X' \underset{L}{\bowtie} Y$ to find new estimates of the unknown rates in Y'.

7. Repeat steps 5 and 6 until convergence (or abandon).

Clearly, if one of the components is behaviourally independent then it is possible to solve this component in isolation and use its marginal probabilities to calculate the rates in the reduced model without the need for any iteration.

5 Example: A simple queueing network with blocking

Consider the following definition of a simple two queue network with feedback and blocking.

$$Queue1_0 \stackrel{def}{=} (arrival, \lambda).Queue1_1 + (feedback, \top).Queue1_1$$

$$
\begin{aligned}
Queue1_j &\stackrel{def}{=} (arrival, \lambda).Queue1_{j+1} + (service1, \mu_1).Queue1_{j-1} \\
&\quad + (feedback, \top).Queue1_{j+1} \quad , \quad 1 \leq j \leq N-1 \\
Queue1_N &\stackrel{def}{=} (service1, \mu_1).Queue1_{N-1}
\end{aligned}
$$

$$
\begin{aligned}
Queue2_0 &\stackrel{def}{=} (service1, \top).Queue2_1 \\
Queue2_j &\stackrel{def}{=} (service1, \top).Queue2_{j+1} + (feedback, q\mu_2).Queue2_{j-1} \\
&\quad + (departure, (1-q)\mu_2).Queue2_{j-1} \quad , \quad 1 \leq j \leq N-1 \\
Queue2_N &\stackrel{def}{=} (feedback, q\mu_2).Queue2_{N-1} \\
&\quad + (departure, (1-q)\mu_2).Queue2_{N-1}
\end{aligned}
$$

$$
Queue1_0 \underset{\substack{\{service1,\\ feedback\}}}{\bowtie} Queue2_0
$$

Each queue has an input action which is controlled by the other queue, i.e. it is blocked when the other queue is empty, $Queue1$ has an additional independent *arrival* process. Similarly each queue output action which is also controlled by the other queue, i.e. it is blocked when the other queue is full, $Queue2$ has an additional *departure* process. Hence, both components in this model ($Queue1$ and $Queue2$) are controlling and controlled over the co-operation set $\{service1, feedback\}$.

It is a simple matter to construct the reduced versions of $Queue1$ and $Queue2$, $Queue1'$ and $Queue2'$ respectively.

$$
\begin{aligned}
Queue1'_a &\stackrel{def}{=} (arrival, \lambda).Queue1'_b + (feedback, \top).Queue1'_b \\
Queue1'_b &\stackrel{def}{=} (arrival, p_1\lambda).Queue1'_c + (service1, (1-p_2)\mu_1).Queue1'_b \\
&\quad + (service1, p_2\mu_1).Queue1'_a + (feedback, p_1 q\mu_2).Queue1'_c \\
&\quad + (feedback, (1-p_1)q\mu_2).Queue1'_b \\
Queue1'_c &\stackrel{def}{=} (service1, \mu_1).Queue1'_b
\end{aligned}
$$

$$
\begin{aligned}
Queue2'_a &\stackrel{def}{=} (service1, \top).Queue2'_b \\
Queue2'_b &\stackrel{def}{=} (service1, (1-p_3)\mu_1).Queue2'_b \\
&\quad + (service1, p_3\mu_1).Queue2'_c + (feedback, p_4 q\mu_2).Queue2'_a \\
&\quad + (feedback, (1-p_4)q\mu_2).Queue2'_b \\
&\quad + (departure, p_4(1-q)\mu_2).Queue2'_a \\
Queue2'_c &\stackrel{def}{=} (feedback, q\mu_2).Queue2'_b \\
&\quad + (departure, (1-q)\mu_2).Queue2'_b
\end{aligned}
$$

In this example the reduced form of the components have only three derivatives as there is no possible state where both the input and output activities are blocked. Clearly the derivatives $Queue1_a$, $Queue1_c$, $Queue2_a$ and $Queue2_c$, are analogous to each queue being either empty or full and the derivatives $Queue1_b$ and $Queue2_b$ correspond to all the non-empty and non-full conditions of each queue respectively. This approximation has introduced four new probabilities:

- p_1 is the probability that an additional job entering $Queue1$ will cause it to become full,
  $p_1 = Pr[Queue1_{N-1}]/(1 - Pr[Queue1_N] - Pr[Queue1_0])$.

- p_2 is the probability that a $service1$ action causes $Queue1$ to become empty,
  $p_2 = Pr[Queue1_1]/(1 - Pr[Queue1_N] - Pr[Queue1_0])$.

- p_3 is the probability that a $service1$ action causes $Queue2$ to become full,
  $p_3 = Pr[Queue2_{N-1}]/(1 - Pr[Queue2_N] - Pr[Queue2_0])$.

- p_4 is the probability that a job leaving $Queue2$ will cause it to become empty,
  $p_4 = Pr[Queue2_1]/(1 - Pr[Queue2_N] - Pr[Queue2_0])$.

The optimal values for these probabilities must be found by iteratively solving the two reduced models:

$$Queue1_0 \underset{\substack{\{service1,\\ feedback\}}}{\bowtie} Queue2'_a$$

and

$$Queue1'_a \underset{\substack{\{service1,\\ feedback\}}}{\bowtie} Queue2_0$$

Each of these reduced models has a CTMC with $3(N+1)$ states whereas the original model has a CTMC with $(N+1)^2$ states. Clearly there is a potentially significant saving on computation in each iteration, however the overall computational efficiency of this approach is dependent not only on the value of N, but the number of iterations required to achieve convergence over the introduced probabilities. Therefore in the following subsection the accuracy of the approximation and the number of iterations required are both investigated.

5.1 Numerical results

The complete model and the reduced (iterative solution) models were solved using the PEPA Workbench [14], to derive the generator matrices, and XMaple was used to solve these numerically. In all cases 20 iterations of the algorithm were used, however in most instances a good degree of convergence (6 decimal places) was achieved in 7 or 8 iterations. Two performance measures are derived for comparison, idleness and the average number of jobs in the system, and both these measures are compared with exact results computed directly from the complete model. In the first two plots the maximum sizes of the queues, N, was 9. This figure was chosen as a compromise between ease of solution of the complete model (for comparison) and significance of the model reduction.

Figure 1 shows idleness plotted against arrival rate on a logarithmic scale to more clearly show the degree of divergence. This plot clearly shows the extremely good approximation gained at low

load, but the reduction in quality as the arrival rate increases. Idleness, the percentage of time

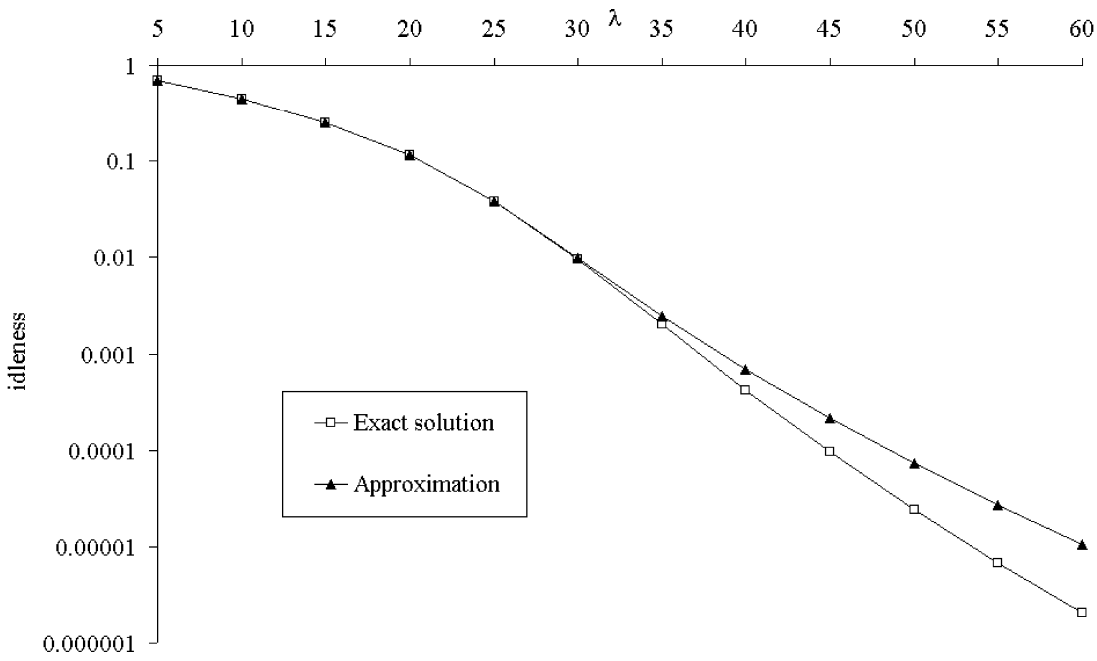

Figure 1: Idleness varied with arrival rate for exact solution and approximation

$(\mu_1 = \mu_2 = 60, q = 0.5)$

during which both queues are empty simultaneously, is calculated directly from the approximations as the average of $Pr[Queue1'_a \ \& \ Queue2_0]$ and $Pr[Queue1_0 \ \& \ Queue2'_a]$. It has been observed that when these two probabilities are very close to one another the estimated idleness is a very good approximation, and that when the degree of separation between them is larger it indicates a poor approximation. However, it has not been possible to accurately predict the error in this way and it has generally been observed that the degree of error is actually far greater than the degree of separation between the estimations. For example, in the worst case in Figure 1 the difference between the two approximations is aproximately 20%, whereas the error is approximate 420%.

The second measure is the average number of jobs in the system, shown in Figure 2, which is calculated from the marginal queue size distributions derived from each of the approximations. It is clear that, whereas there was a significant error for the estimation of idleness, even at average

load, there is no discernible error for the calculation of the average number of jobs.

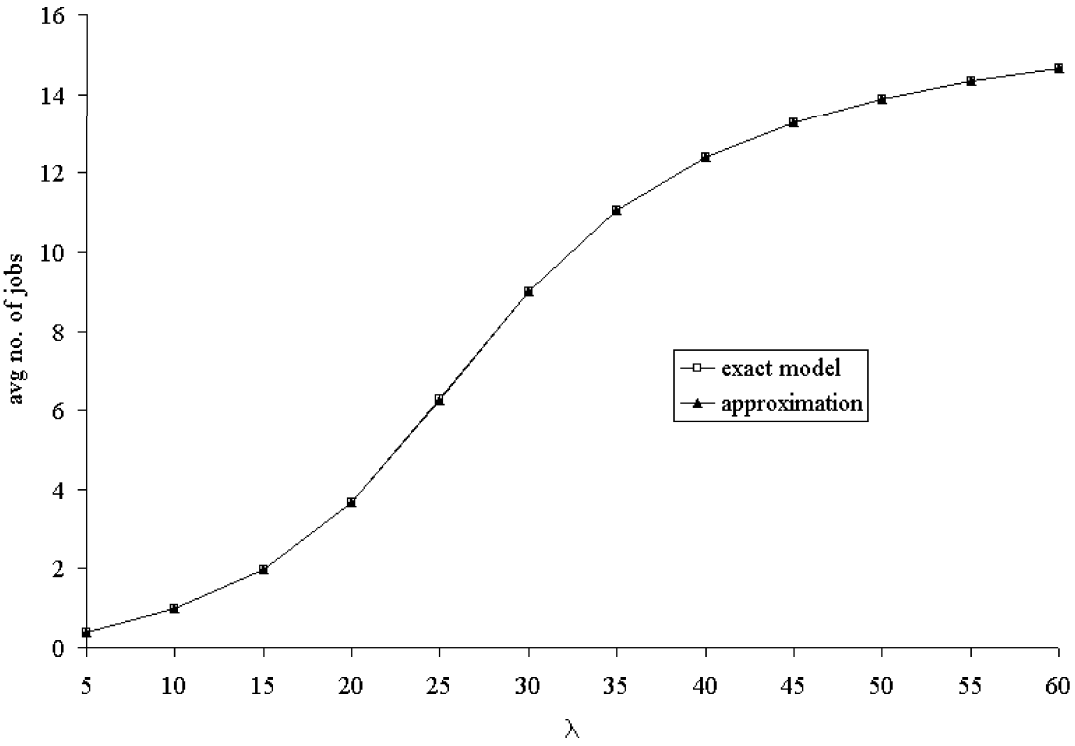

Figure 2: Average number of jobs varied with arrival rate

$(\mu_1 = \mu_2 = 60, q = 0.5)$

This is reiterated by Figure 3, which shows the percentage error from the data in Figure 2 for various values of the maximum queue size, N. Nowhere in this plot is the error worse than 0.7 percent and in most cases it is much smaller. It is perhaps surprising that an approximate model (in this case two approximate models) should give rise to a poor approximation when used directly but a very good approximation when using marginal probabilities. However, it is clear from the results presented here that, in this case at least, the errors introduced in the reduced models are almost entirely attributable to the approximated queues. Therefore that part of the reduced models which is modelled completely is actually behaving in an almost exact manner despite a crude approximation to the rest of the system.

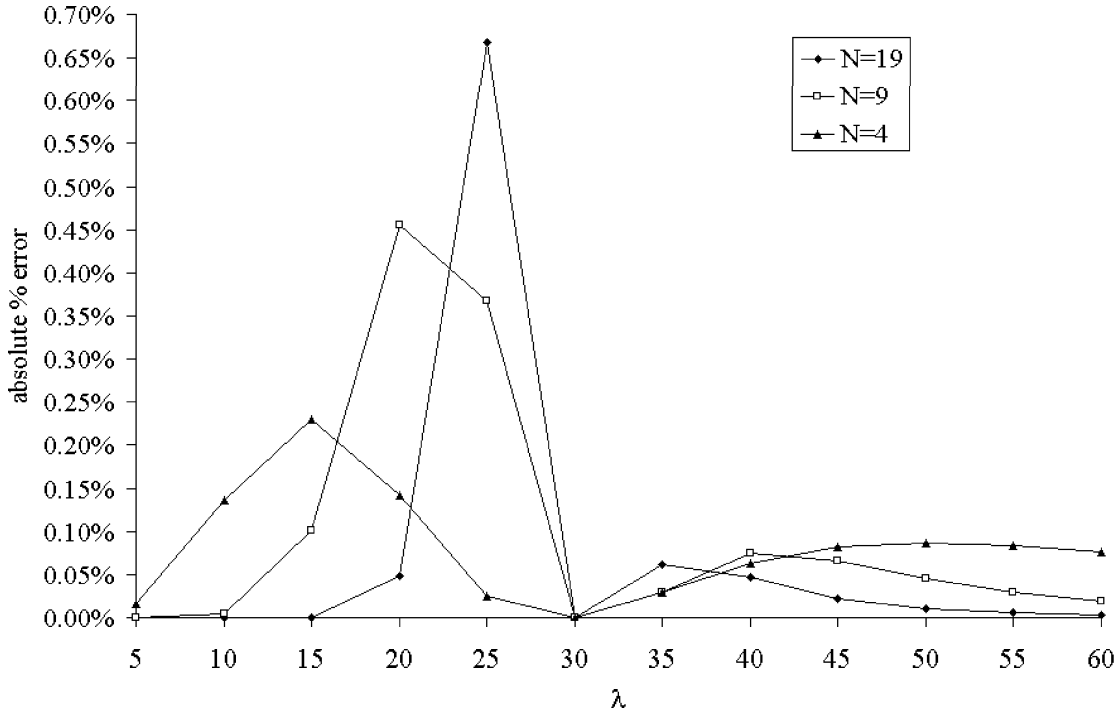

Figure 3: Percentage error of approximation of the average number of jobs varied with arrival rate

$$(\mu_1 = \mu_2 = 60, q = 0.5)$$

6 Example: A multimedia stream

The model shown here was proposed by Bowman et al [12] as part of an investigation into quality of service of multimedia systems. The model consists of four components; a source which generates traffic (multimedia frames), a channel which carries that traffic, a sink which receives the traffic and a timer which tests whether the frames are displayed within acceptable time intervals.

$$Source \stackrel{def}{=} (transmit, r_{trans}).Source$$

$$Channel \stackrel{def}{=} Channel_0$$

$$Channel_0 \stackrel{def}{=} (transmit, \top).Channel_1$$

$$Channel_i \stackrel{def}{=} (transmit, \top).Channel_{i+1} + (receive, r_{rec}).Channel_{i-1} + (loss, r_{loss}).Channel_{i-1} \quad 1 \leq i \leq 4$$

$$Channel_5 \stackrel{def}{=} (receive, r_{rec}).Channel_4 + (loss, r_{loss}).Channel_4$$

$$\begin{aligned}
Sink &\stackrel{def}{=} Sink_0 \\
Sink_0 &\stackrel{def}{=} (receive, \top).Sink_1 \\
Sink_i &\stackrel{def}{=} (receive, \top).Sink_{i+1} + (display, r_{disp}).SinkR_{i-1} \; 1 \leq i \leq 2 \\
Sink_3 &\stackrel{def}{=} (display, r_{disp}).SinkR_2 \\
SinkR_i &\stackrel{def}{=} (reset, r_{reset}).Sink_i
\end{aligned}$$

$$\begin{aligned}
Timer &\stackrel{def}{=} Timer_0 \\
Timer_0 &\stackrel{def}{=} (reset, \top).Timer_1 \\
Timer_i &\stackrel{def}{=} (tick, r_{tick}).Timer_{i+1} + Timer_0 \quad 1 \leq i \leq 5 \\
Timer_6 &\stackrel{def}{=} (error, r_{error}).Timer_1 + Timer_0
\end{aligned}$$

$$Source \underset{transmit}{\bowtie} Channel \underset{receive}{\bowtie} Sink \underset{reset}{\bowtie} Timer$$

Clearly the *Source* component in this simple model does not contribute to the global state space and can easily be considered to be part of the *Channel* component. Furthermore, whilst the *Timer* is clearly controlled by the *Sink* through the *reset* action, it does not itself exert any control. Both the *Channel* and *Sink* components exert control through the *receive* action. Hence, it is necessary to consider only reduced forms for the *Channel* and *Sink* components as follows.

$$\begin{aligned}
Channel_A &\stackrel{def}{=} (transmit, r_{trans}).Channel_B \\
Channel_B &\stackrel{def}{=} (transmit, p_1 r_{trans}).Channel_C \\
&\quad + (loss, p_2 r_{loss}).Channel_A \\
&\quad + (receive, p_2 r_{rec}).Channel_A \\
&\quad + (receive, (1-p_2) r_{rec}).Channel_B
\end{aligned}$$

$$Channel_C \stackrel{def}{=} (receive, r_{rec}).Channel_B + (loss, r_{loss}).Channel_B$$

$$\begin{aligned}
Sink_A &\stackrel{def}{=} (receive, \top).Sink_B \\
Sink_B &\stackrel{def}{=} (receive, p_3 r_{rec}).Sink_C + (receive, (1-p_3) r_{rec}).Sink_B \\
&\quad + (display, p_4 r_{disp}).SinkR_A \\
&\quad + (display, (1-p_4) r_{disp}).SinkR_B \\
Sink_C &\stackrel{def}{=} (display, r_{disp}).SinkR_B \\
SinkR_A &\stackrel{def}{=} (reset, r_{reset}).Sink_A \\
SinkR_B &\stackrel{def}{=} (reset, r_{reset}).Sink_B
\end{aligned}$$

Where,

- p_1 is the probability that a *transmit* action causes the *Channel* to become full,

  $p_1 = Pr[Channel_4]/(1 - Pr[Channel_5] - Pr[Channel_0])$,

- p_2 is the probability that a *loss* or *receive* action causes the *Channel* to become empty,

  $p_2 = Pr[Channel_1]/(1 - Pr[Channel_5] - Pr[Channel_0])$,

- p_3 is the probability that a *receive* action causes *Sink* to become full,

  $p_3 = Pr[Sink_2]/(Pr[Sink_1] + Pr[Sink_2])$,

- p_4 is the probability that a *display* action causes *Sink* to become empty,

  $p_4 = Pr[Sink_1]/(Pr[Sink_1] + Pr[Sink_2])$.

And,

- $Pr[Sink_i]$ is the probability of being in a system state where the *Sink* component is behaving as $Sink_i$,

- $Pr[Channel_i]$ is the probability of being in a system state where the *Channel* component is behaving as $Channel_i$.

These components are then used in the following model descriptions using the iterative method described above,

$$Channel_A \underset{receive}{\bowtie} Sink$$

$$Channel \underset{receive}{\bowtie} Sink_A$$

and then subsequently to find the marginal probabilities for the *Timer* component.

$$Channel_A \underset{receive}{\bowtie} Sink_A \underset{reset}{\bowtie} Timer$$

These models have 21, 30 and 105 states respectively, whereas the complete model has 294 states. Further reduction in the model involving the *Timer* is clearly possible, but this is not within the scope of this paper. In addition, if the capacity of the system were increased then the state space saving would be greater in all three reduced models, however we have used the parameters given in [12] to ensure continuity.

6.1 Numerical results

It is interesting to note that most of the performance measures employed in [12] are derived from the marginal probabilities, and hence can be repeated here without further approximation. The measures of interest are latency, throughput, jitter and error rate. These are calculated by finding the observed rates of activities, referred to as the *true* rates by Bowman et al [12]. The observed rate is the rate at which activities actually occur in the model and this differs from the given rate because activities periodically become blocked. It is a simple matter to identify all the states where an action can occur, calculate the probability of being in any of those states and multiply that probability by the given rate to obtain the observed rate. Thus,

$$true\_r_{disp} = r_{disp} \times \sum_{i=1}^{3} Pr[Sink_i]$$

Only $true\_r_{rec}$ is derived from global states, but these are easily identified directly in the reduced solutions.

The *stream latency* can then be found as the sum of the *source latency*, *channel latency* and *sink latency*, which can be found using Little's law.

$$\begin{aligned} Source\_latency &= (true\_r_{trans})^{-1} \\ Channel\_latency &= \frac{ave\_frames\_channel}{(true\_r_{loss} + true\_r_{rec})} \\ Sink\_latency &= \frac{ave\_frames\_sink}{true\_r_{disp}} \end{aligned}$$

Where, $ave\_frames\_channel$ and ave_frames_sink are the average number of frames in the *Channel* and *Sink* respectively.

The *stream jitter* is similarly the sum of the *source variance*, *channel variance* and *sink variance*. These are calculated by assuming that the observed activities are exponential.

$$\begin{aligned} source\_variance &= (true\_r_{trans})^{-2} \\ channel\_variance &= (true\_r_{rec})^{-2} \\ sink\_variance &= (true\_r_{disp})^{-2} \end{aligned}$$

Figure 1 shows results for all these measures calculated from the exact model and the approximate models using two values for the loss rate, r_{loss}. All the parameters are set to be the same as used by Bowman et al [12]; $r_{trans} = 60.0$, $r_{rec} = 30.0$, $r_{disp} = 200.0$, $r_{tick} = 100.0$, $r_{error} = 2000.0$, $r_{reset} = 2000.0$.

It is clear that the results presented are of a consistent level of accuracy as the approximation of average queue size in the previous example. The calculation of $true\_r_{rec}$ is derived from global states and yet the error is small (0.04%). However, the results from the first example would indicate that as the load (r_{trans}) increases then the error in $true\_r_{rec}$ will increase, perhaps to a significant degree. The largest errors occur in the calculation of jitter (around 0.09%). This occurs due to variance being calculated as the square of the true rate for each component, hence

	Exact Model	Approximation	Exact Model	Approximation	Max % error
rloss	0.0	0.0	40.0	40.0	n/a
true_rloss	0.0	0.0	30.6407580	30.6414347	0.002208%
true_rtrans	29.0896933	29.0766183	53.3446637	53.3413548	0.044947%
true_rrec	29.0896933	29.0769569	22.7039068	22.6996509	0.043783%
true_rdisp	29.0896932	29.0772955	22.7039068	22.6993817	0.042619%
true_rtick	99.4545996	99.4544169	99.3795095	99.3794498	0.000184%
true_rerror	10.9080074	10.9116621	12.4098107	12.4110033	0.033505%
ave. no. in source	1.0	1.0	1.0	1.0	0.0%
ave. no. in channel	4.1272616	4.1268218	2.0714651	2.0718012	0.016222%
ave. no. in sink	0.1712637	0.1711826	0.1285243	0.1285634	0.047345%
ave. no. in stream	5.2985253	5.2980045	3.1999894	3.2003646	0.011723%
source latency	0.0343764	0.0343919	0.0187460	0.0187472	0.044967%
channel latency	0.1418805	0.1419276	0.0388317	0.0388406	0.033144%
sink latency	0.0058874	0.0058872	0.0056609	0.0056637	0.050357%
stream latency	0.1821444	0.1822066	0.0632386	0.0632515	0.034151%
throughput jitter	0.0011817	0.0011827	0.0019400	0.0019408	0.085292%
channel jitter	0.0011817	0.0011828	0.0019400	0.0019407	0.087624%
source jitter	0.0011817	0.0011828	0.0003514	0.0003515	0.089955%
stream jitter	0.0035452	0.0035483	0.0042314	0.0042329	0.087624%

Figure 4: Analysis of multimedia stream

the error present in the true rate is increased. These observations would suggest that metrics can only really be trusted to any degree of satisfaction when calculated directly from the marginal component probabilities only (unlike $true\_r_{rec}$) and the errors are not subject to multiplication within this calculation (unlike jitter).

7 Conclusions and further work

In this paper we have presented an iterative approximation technique that can be applied to models in which all the components exhibit the property of control. The class of models considered here includes many models that are not subject to any previous decomposition technique, hence this approach, despite being approximate, is a useful addition to the armoury of the performance

modeller. Although the examples presented here use only two components in the iterative part, the technique is also applicable to multiple compound components. In addition the heart of this technique could easily be applied to cases where one component is behaviourally independent but controls another component, with consequently large savings in computation. In the examples the accuracy of the approximation for calculating marginal component distributions is shown to be extremely good, although the approximations themselves give more variable accuracy when used directly to derive measures of interest. It is interesting to note however, that most of the performance measures of interest here are derived from the marginal probabilities and not the global probabilities.

The approach taken so far in attempting to automate the construction of reduced components has been pragmatic. If no behavioural independence is identified then the ideal form for a reduced component is attempted at the component level. The global state space is explored only if it is not possible to form such a component and derive the necessary rates. In theory it should be possible to examine the model in advance to determine whether or not the global state space needs to be explored, however this has not yet been achieved. In addition it is possible to calculate in advance the state space saving that can be achieved and hence inform the modeller as to the most profitable approach to take. The conditions for convergence have yet to be explored and, although this has not been a practical issue, it would be desirable to prove that a class of models will achieve convergence under this technique. It would also be desirable to develop heuristics to predict the probabilities involved. However, experience with the examples here has shown that accuracy to two or three decimal places is achieved within one or two iterations, so it is unlikely that a heuristic would save much computation in practice.

A further line of investigation concerns a slightly more flexible condition we have termed, *weak control*. A component Q is said to *weakly control* component P over $K \subset L$ in the model $P \underset{L}{\bowtie} Q$ if the occurrence of an action $a \in K$ affects only the rate of subsequent activities in P but not their

existence, i.e. $\mathcal{A}(P)$ does not change as Q evolves, but the rates of its activities can. In this instance a reduced model of the kind presented in Section 4 could be constructed with average rates for activities in Q, rather than a different derivative of the reduced component Q' for every different rate of a. This can be thought of as analogous to the queueing scenario of approximating Markov modulated arrivals with a Poisson stream. Clearly the accuracy of this kind of approximation is likely to be much less predictable than that presented in this paper, however the computational saving is potentially significant.

Acknowledgements

This work has been partially supported by the EPSRC funded QWiD project (GR/N01491) at the University of Durham and the EMU (GR/R02238) and MEGAN (GR/N16068) projects at Imperial College, London. The authors would like to extend special thanks to Dr Jane Hillston of the University of Edinburgh for helpful comments following an earlier version of this paper and for directing them to the earlier work by Dr Vassilios Mertziotakis. They would also like to acknowledge the support of colleagues in the ALMA group in the Department of Computing at Imperial College, in particular Dr Uli Harder for some helpful comments and Professor Peter Harrison. More recent advice from Professor Murray Woodside of Carleton University, Canada, has highlighted several opportunities for further extension. An earlier version of this paper was presented at UKPEW 2002, orgainised by Dr Mohamed Ould-Khaoula and Dr Lewis M. MacKenzie at the University of Glasgow.

Several of the papers listed in the bibliography, as well as other papers on stochastic process algebra, are available online from the PEPA homepage *http://www.dcs.ed.ac.uk/pepa* maintained by Dr Stephen Gilmore, or from Nigel Thomas' homepage *http://www.dur.ac.uk/nigel.thomas.*

References

[1] J. Hillston, Exploiting Structure in Solution: Decomposing Composed Models, in: C. Priami (ed.), *Proceedings of 6th International Workshop on Process Algebra and Performance Modelling*, 1998.

[2] J. Hillston, Exploiting Structure in Solution: Decomposing Composed Models, in: *FMPA Lecture Notes*, Springer-Verlag, 2001.

[3] J. Hillston and N. Thomas, A Syntactic Analysis of Reversible PEPA Models, in: *Proceedings of the Sixth International Workshop on Process Algebra and Performance Modelling*, 1998.

[4] P.G. Harrison and J. Hillston, Exploiting Quasi-reversible Structures in Markovian Process Algebra Models, *The Computer Journal*, **38**(7), pp. 510–520, 1995.

[5] M. Gribaudo and M. Sereno, Approximation technique of finite queueing networks exploiting Petri net analysis, in: D. Kouvatsos (ed.), *Proceedings of the Fourth International Workshop on Queueing Networks with Finite Capacity*, Ilkley, UK, 2000.

[6] V. Mertziotakis, Approximate Analysis Methods for Stochastic Process Algebras, Dissertation Doktor-Ingenieur, Universität Erlangen-Nürnberg, 1998.

[7] J. Campos, J.M. Colom, H. Jungnitz and M. Sliva, Approximate Throughput Computation of Stochastic Marked Graphs, *IEEE Transactions on Software Engineering*, **20**(7), pp. 526-535, 1994.

[8] S.C. Agrawal, J.P. Buzen and A.W. Shum, Response Time Preservation: A General Technique for Developing Approximate Algorithms for Queueing Networks, *Performance Evaluation Review*, **12**(3), pp. 63-77, 1984.

[9] P.G. Harrison and N.M. Patel, *Performance modelling of communication and computer architectures*, Addison Wesley, 1993.

[10] N. Thomas, Behavioural Independence and Control in Markovian Process Algebra, Technical Report, Department of Computer Science, University of Durham, 2002.

[11] N. Thomas, Exploiting behavioural independence and control in Markovian process algebra, in: *Proceedings of the 1st Workshop on Process Algebra with Stochastic Timed Activities*, University of Edinburgh, 2002.

[12] H. Bowman, J.W. Bryans and J. Derrick, Analysis of a Multimedia Stream using Stochastic Process Algebra, *The Computer Journal*, **44**(4), 2001.

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

[14] G. Clark, S. Gilmore, J. Hillston and N. Thomas, Experiences with the PEPA performance modelling tools, *IEE Proceedings - Software*, **146**(1), 1999.