The PEPA Performance Modelling Tools

Graham Clark Stephen Gilmore Jane Hillston

Laboratory for Foundations of Computer Science,
The University of Edinburgh.
Email: {gcla, stg, jeh}@dcs.ed.ac.uk.

May 17, 1999

Abstract

The PEPA language [11] is supported by a suite of modelling tools which assist in the solution and analysis of PEPA models. The design and development of these tools have been influenced by a variety of factors, including the wishes of other users of the tools to use the language for purposes which were not anticipated by the tool designers. The suite of PEPA tools has adapted and grown to attempt to serve these needs, while continuing to support the language designers themselves.

1 Introduction

PEPA (Performance Evaluation Process Algebra) extends classical process algebra with the capacity to assign rates to the activities which are described in an abstract model of a system. Taken together, the information about the rates of performance of activities and the definition of the outcome of performing an activity specify a stochastic process and thus PEPA is said to be a *stochastic process algebra.* The PEPA language has been used to investigate the behaviour and performance of a diversity of distributed and concurrent systems [12, 9, 5, 4, 1, 15].

PEPA is a concise formal language with a small number of grammar rules which define the well-formed terms in the language. An activity α performed at rate r preceding S is denoted by $(\alpha, r).S$. Choices are separated by $+$. Cooperation between P and Q over L is $P \underset{L}{\bowtie} Q$ or $P \parallel Q$ if L is empty. Hiding the activities in L gives P/L. PEPA serves as a high-level notation for Markov modelling because it is possible to generate directly from a PEPA model a continuous-time Markov process which faithfully encodes the temporal aspects of the PEPA model. Through the analysis and solution of this Markov process the modeller can undertake an experimental investigation of the system which the model represents.

We have implemented a suite of integrated tools which perform well-formedness checking of PEPA models and allow a modeller to investigate their model through generation and solution of the corresponding Markov process or through simulation. These tools are freely available, and have been modified and improved in the light of feedback from the user community. In this paper, we describe the PEPA modelling tools in sufficient detail to allow the reader to understand their capabilities, and to provide some context for the tools demonstration. The main sections of this paper deal in turn with each of the tools to be demonstrated.

2 The PEPA modelling tools

During the tools session of this workshop, we will demonstrate four tools from the PEPA suite. These are:

1. the Workbench;
2. the state finder;
3. the reward assessor; and
4. the discrete event simulator.

More PEPA tools exist, and details of these can be found in [3].

2.1 The PEPA Workbench

The purpose of the PEPA Workbench is to check that a PEPA model is well-formed, and to generate its corresponding Markov process. The Workbench detects faults such as deadlocks and cooperations which do not involve active participants, and is described in full in an earlier paper [7].

In essence, the translation process of the Workbench accepts a PEPA model as input and produces a matrix containing the Markov process encoding of the given model. An example of this relationship is depicted in Figure 1. In the example shown there, the

$$P_1 \stackrel{def}{=} (run, r_1).P_2 \qquad P_2 \stackrel{def}{=} (stop, r_2).P_1$$

$$P_1 \parallel P_1$$

$$\Downarrow$$

$$\begin{pmatrix} -2r_1 & r_1 & r_1 & 0 \\ r_2 & -r_1 - r_2 & 0 & r_1 \\ r_2 & 0 & -r_1 - r_2 & r_1 \\ 0 & r_2 & r_2 & -2r_2 \end{pmatrix}$$

Figure 1: The PEPA Workbench schema

PEPA model has two independent components which execute in parallel, and do not *cooperate.* Each component is initiated in state P_1, and each has two states meaning the generator matrix for the corresponding Markov process has dimension four. For larger examples, it quickly becomes apparent that constructing a Markov process matrix by hand directly is an error-prone activity, since the matrix is an unstructured representation of the system devoid of names, and only containing rates.

The investigation into properties of interest to the performance modeller working with PEPA proceeds by determining the steady-state probability distribution for the system. The purpose of the analysis performed by the Workbench has been to reject PEPA models which are not well-formed. A model which was not well-formed would not correspond to an irreducible Markov process over a finite state space. This is a condition necessary to guarantee that all of the states of the Markov process are positive-recurrent, and thus that it reaches an equilibrium state.

The steady-state distribution may be found by applying any one of a number of linear algebra solution methods to the generator matrix, although as we note in Section 3, some of these methods are of limited use for PEPA models of even modest size. For this reason we have recently extended the PEPA Workbench with the capability to reduce models to a canonical form internally, thereby automatically aggregating the model [10].

2.2 The PEPA State Finder

The PEPA State Finder allows a PEPA modeller to investigate the state space which has been generated by the PEPA Workbench. The state finder provides a simple pattern language which the modeller can use to describe states of interest. Continuing the example from the previous section, a modeller interested in testing how often either component in the tiny example was component P_1 would prepare an input with the following contents.

```
% We are checking for P1 in either case
test:  P1|*
test:  *|P1
```

The PEPA State Finder would generate the function shown in Figure 2. Notice that the function correctly avoids counting the state $P_1 \parallel P_1$ twice. This mistake could easily be made if the function is built from a number of smaller functions, defined separately. Typically, the PEPA State Finder is used when investigating the results obtained after solving the Markov process produced by the Workbench. Another use is to pose questions about the reachability of certain states within the model. The analysis which can be performed in this way is a weak form of liveness analysis.

```
test := proc (π) (     π[1]       #  P1 || P1
                     + π[2]       #  P1 || P2
                     + π[3]   )   #  P2 || P1
end:
```

Figure 2: A function built by the PEPA State Finder

A significant disadvantage of the pattern language is that it is too syntactic, distinguishing between states which are (logically) equivalent. The PEPA Reward Assessor provides a more expressive logical notation for characterising states.

2.3 The PEPA Reward Assessor

Classical process algebras are complemented by modal logics, which formally express properties of a model in terms of its behaviour. A technique is described in [2] which utilises a simple formalism, based on Hennessy-Milner logic. It was developed with the explicit intention of supporting behavioural reasoning about the quantitative behaviour of PEPA models. A reward is assigned to a component of the model dependent on the behaviour of the component, in keeping with the process algebra approach. Formulae are built from Boolean connectives and modal operators thus:

$$\begin{array}{lcl}\Phi\ \mathsf{(formula)} & ::= & \mathtt{tt} \mid \mathtt{ff} \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \\ & & \Phi_1 \vee \Phi_2 \mid [K]\Phi \mid \langle K\rangle\Phi\end{array}$$

where K ranges over action types. This logic has a simple semantics; for example a PEPA model satisfies the formula $\langle \mathtt{a}\rangle\mathtt{tt}$ if it may perform an activity of type `a` such that its derivative satisfies `tt` (which every model does, trivially). This formula specifies that a process may perform an `a` activity.

The PEPA Reward Assessor is an experimental tool which allows rewards to be associated with model states which satisfy a given logical formula. Using the PEPA Reward Assessor, it is possible to isolate states which regular expressions alone could not capture. Furthermore, use of this technique confers the added advantage that reward specifications need not be modified to correctly compute performance measures for models which have been reduced, for example by using the aggregation method described in [10]. However this extra expressibility may require more experience to use correctly. Figure 3 shows a throughput analysis of a simple model using the PEPA Workbench. *Dev* attempts to transmit a packet consisting of a connection header and some data. However it may fail to connect, in which case it immediately retries. We would only wish to consider the throughput of legitimate data, and not failed connection attempts; therefore the logical expression selects only those states in which *Dev* has made a connection, and is about to transmit data. These states are assigned the data transmission rate as a reward. The six-state model has a reward assigned to states 4 and 6 only.

% We only measure throughput
% of successful transmissions

throughput is
$\langle trans\rangle\mathtt{tt} \wedge [trans][trans]\mathtt{ff} = rate(trans)$

$$\begin{array}{lcl} Dev & \stackrel{def}{=} & (check, r_1).Try \\ Try & \stackrel{def}{=} & (trans, fail).Try \\ & + & (trans, succeed).Con \\ Con & \stackrel{def}{=} & (trans, data).Dev \\ Bus & \stackrel{def}{=} & (check, \top).(reset, r_2).Bus \end{array}$$

$$Dev \underset{\{check\}}{\bowtie} Bus$$

$$\Downarrow$$

```
R[1] := R[1]+0.0:   %  Dev|Bus
R[2] := R[2]+0.0:   %  Try|(reset,r2).Bus
R[3] := R[3]+0.0:   %  Dev|(reset,r2).Bus
R[4] := R[4]+data:  %  Con|(reset,r2).Bus
R[5] := R[5]+0.0:   %  Try|Bus
R[6] := R[6]+data:  %  Con|Bus
```

Figure 3: The PEPA Reward Assessor schema

2.4 The PEPAroni discrete-event simulator

A theme of recent stochastic process algebra research has been the introduction of *generally distributed* activities, in order to lead to a formalism with improved expressibility. The addition of such general probability distributions renders false the convenient Markovian assumption, and obtaining exact solutions for such models is difficult in general. One approach is to use stochastic simulation, deriving performance measures from particular simulation runs of the model. If done with care, this can yield a useful degree of confidence in the model's behaviour. We chose not to build our simulator from scratch, but to build upon a framework already under development called *simjava* [13]. Implemented in Java, it provides a framework for discrete event simulation, and, if the developer feels particularly creative, supports the deployment of animations as applets. The fact that all simulations would be based upon Java might have disadvantages, but in practice, this has not turned out to be the case. Speed is an issue when long simulation runs are desirable for statistical accuracy. However, although Java applications have in the past tended to run considerably more slowly than native compiled code, this problem is alleviated by the recent appearance of just-in-time compilers. Our choice of language did mark a departure from the favoured use of the Standard ML programming language. Fortunately this has not been a serious issue either, since the PEPA interface is being developed in Pizza [17], a superset of Java. Pizza extends Java with several features which allows objects to be built and processed in a very similar fashion to the use of datatypes of Standard ML. This elegance gave us a degree of confidence in the correctness of our implementation.

We adopted the syntax of the PEPA Workbench and extended it to allow for the expression of more general probability distributions. For example, Figure 4 shows one of the components of Figure 3, with changes to incorporate some uniformly distributed activities. If we wished to compare the exponential version of *Dev* in Figure 3 with that in Figure 4, then it would make sense to ensure the mean of each uniform distribution was equal to the mean of its corresponding exponential version; for example, for the second activity, that $1/data = 3.5$. To analyse the performance of the non-exponential model, we are able to execute a long simulation run for the non-exponential model, and then employ the method of *batched means*, as described in [14]. The simulator maintains a record of simulation time passed, and also the residence time spent in each state of the model. We discard a transient interval, and then split the run into a number of subsamples. We can show that for the simple example of *Dev* above, the non-exponential model appears to be identical in steady-state to the exponential version. In this case this is not a surprise, since it is easily shown that *Dev* is *insensitive* to the distributions of the non-exponential activities. However, we plan to use PEPAroni to investigate more interesting cases, where components may cooperate with others. Interesting semantic issues continue to present themselves; this is work in progress, and PEPAroni is currently under active development.

$$
\begin{array}{rcl}
Dev & \stackrel{def}{=} & (check, \mathrm{Uni}(0.5; 1.5)).Try \\
Try & \stackrel{def}{=} & (trans, fail).Try \\
 & + & (trans, succeed).Con \\
Con & \stackrel{def}{=} & (trans, \mathrm{Uni}(3; 4)).Dev
\end{array}
$$

$$\Downarrow$$

```
simderiv is Dev
event fired - activity (check,Uni(0.5;1.5))
 after 1.2216
simderiv is Try
event fired - activity (trans,3.0)
 after 0.1130
simderiv is Con
event fired - activity (trans,Uni(3;4))
 after 3.1851 ...
```

Figure 4: Modelling non-exponential activities

3 Solution methods and tools

The benefits to be gained from using a stochastic process algebra to model a computer system come from the insights into the dynamic behaviour of the system. By alteration of the rates of activities, the modeller may directly witness the effect on the steady-state probability distribution, and therefore study bottlenecks, mean waiting times, and other measures.

Before we gained experience in the use of our performance modelling tools we had pessimistically assumed that limits on the effectiveness of our tools would be set by their own inability to generate Markov process representations of PEPA models. In contrast to our expectations, we have discovered that the limits on the applicability of our tools come from the inability of other packages to solve the Markov process representations which we generate. For small examples such as the ones which we include in expository papers and articles we have found that the Maple computer algebra package provides an excellent working environment for analysis of model solutions. Maple has a profound advantage over many of its competitors since it can provide *symbolic* solutions in terms of the activity rates from the model. However, solutions which can be computed are typically symbolic in only one of the rates, and further, this limits the size of the models for which it is possible to compute a solution (a few thousand states). Larger models may fall within the scope of numerical computing systems such as Matlab and Mathematica. Both [12] and [6] used Matlab to solve their models and analyse the results. Here the size of the models which could be solved increased ten-fold from those which could be solved by Maple. If one is willing to forgo some of the comforts of these mathematical computing environments, it is possible to analyse larger models or to process them more efficiently. One of the most effective solution methods which we have used is the preconditioned biconjugate gradient method. We obtained a finely tuned implementation of this from the Numerical Recipes collection [19]. For a model of nearly 20,000 states we reduced the solution time by a factor of fifteen when compared with the Matlab solution time for the steady-state distribution of the same model.

4 Implementation of the tools

With only a few exceptions, the PEPA modelling tools are implemented in the functional programming language Standard ML [16]. This language has allowed the production of reliable tools which are flexible enough to permit revision and modification to adapt to the needs of different user groups. We have found that the language's strong type system is a significant aid in eliminating a number of errors which would have manifested themselves at run-time if working in an untyped language. Two exceptions to this have been the C implementation of the biconjugate gradient method solver and the use of a UNIX tool (`grep`) called from the PEPA State Finder. This latter excursion from Standard ML proved to be foolish since we discovered that it gave rise to problems for users whose version of `grep` differed from ours. This necessitated a re-write to eliminate the need to use regular expressions to generate the model functions. On the final occasion we did not develop in Standard ML, the ML-like features provided by the Pizza programming language helped provide confidence in the correctness of our implementation.

4.1 Availability of the modelling tools

The PEPA modelling tools, together with user documentation, papers and example PEPA models are available from the PEPA Web page at the address `http://www.dcs.ed.ac.uk/pepa`.

5 Further work

Development of the existing PEPA modelling tools is ongoing and other directions will also be explored, including *debugging*. The performance measures which are calculated by the solution of the generated Markov process, or by simulation, are only as good as the model from which they were derived. If there is a logical flaw in the model of the system then the results could be entirely worthless, irrespective of whether they were derived by solution or simulation. Thus the suite of PEPA modelling tools seeks to give the modeller a number of methods of uncovering errors in their models. These include state-space construction, reachability analysis, satisfaction of a logical description, examination of simulation runs and will also include interactive debugging. The PEPA debugger is not yet complete but will eventually allow modellers to set breakpoints on both states and activities, run their models until a breakpoint is encountered and then fire selected transitions and continue. The degree of coverage of the state space of the model will be estimated.

Acknowledgements

Jane Hillston is supported by the ESPRC 'COMPA' grant, and Stephen Gilmore by the 'Distributed Commit Protocols' grant from the EPSRC and by Esprit Working group FIREworks. Graham Clark was supported by EPSRC studentship 95306547.

References

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

[2] G. Clark. Formalising the specification of rewards with PEPA. In M. Ribaudo, editor, *Proceedings of the Fourth Annual Workshop on Process Algebra and Performance Modelling*, pages 139–160. Dipartimento di Informatica, Universitá di Torino, CLUT, July 1996.

[3] G. Clark, S. Gilmore, J. Hillston, and N. Thomas. Experiences with the PEPA performance modelling tools. *IEE Proceedings—Software*, 146(1):11–19, Feb. 1999. Special issue of papers from the Fourteenth UK Performance Engineering Workshop.

[4] E. W. Dempster, N. T. Tomov, J. Lü, C. S. Pua, M. H. Williams, A. Burger, H. Taylor, and P. Broughton. Verifying a performance estimator for parallel DBMSs. In *Proceedings of EuroPar (EuroPar'98)*, Sept. 1998.

[5] A. El-Rayes, M. Kwiatkowska, and S. Minton. Analysing performance of lift systems in PEPA. In Pooley and Hillston [18], pages 83–100.

[6] A. El-Rayes, M. Kwiatkowska, and S. Minton. Analysing performance of lift systems in PEPA. In Pooley and Hillston [18], pages 83–100.

[7] S. Gilmore and J. Hillston. The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling. In *Proceedings of the Seventh International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, number 794 in Lecture Notes in Computer Science, pages 353–368, Vienna, May 1994. Springer-Verlag.

[8] S. Gilmore and J. Hillston, editors. *Proceedings of the Third International Workshop on Process Algebras and Performance Modelling*. Special Issue of *The Computer Journal*, 38(7), Dec. 1995.

[9] S. Gilmore, J. Hillston, D. Holton, and M. Rettelbach. Specifications in Stochastic Process Algebra for a Robot Control Problem. *International Journal of Production Research*, 34(4):1065–1080, 1996.

[10] S. Gilmore, J. Hillston, and M. Ribaudo. An efficient algorithm for aggregating PEPA models. Submitted for publication, 1999.

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

[12] D. Holton. A PEPA specification of an industrial production cell. In Gilmore and Hillston [8], pages 542–551.

[13] F. Howell and R. McNab. *simjava*: a discrete event simulation package for Java with applications in computer systems modelling. *First International Conference on Web-based Modelling and Simulation, San Diego CA*, 1998.

[14] R. Jain. *The art of computer systems performance analysis : techniques for experimental design, measurement, simulation, and modeling*. John Wiley and Sons, Inc., New York, 1991.

[15] L. Kloul, J. Fourneau, and F. Valois. Performance modelling of hierarchical cellular networks using PEPA. In J. Hillston, editor, *Proceedings of the Seventh International Workshop on Process Algebras and Performance Modelling*, Zaragosa, Spain, Sept. 1999.

[16] R. Milner, M. Tofte, R. Harper, and D. MacQueen. *The Definition of Standard ML: Revised 1997*. The MIT Press, 1997.

[17] M. Odersky and P. Wadler. Pizza into Java: Translating theory into practice. *Proc. 24th ACM Symposium on Principles of Programming Languages*, January 1997.

[18] R. Pooley and J. Hillston, editors. *Proc. of 12th UK Performance Engineering Workshop*, Dept. of Computer Science, The Univ. of Edinburgh, Sept. 1996.

[19] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, second edition, 1992.