A survey of the PEPA tools

Stephen Gilmore Jane Hillston*

5th June 2003

Abstract

This paper surveys the history and the current state of tool support for modelling with the PEPA stochastic process algebra and the PEPA nets modelling language. We discuss future directions for tool support for the PEPA family of languages.

1 Introduction

The PEPA stochastic process algebra [1] is a small modelling language which is used to express models of systems which are composed of cooperating components which undertake timed activities either individually or in co-operation with other components. Constraints on the nature of the timing information (exponentially distributed random delays) mean that a PEPA model defines a Continuous-Time Markov Chain. The analysis of a PEPA model proceeds by deriving its underlying Markov chain and solving this to find the long-run probability of the states of the chain. The states of the chain are in one-to-one correspondence with the states of the derivation graph of a PEPA process as specified by the operational semantics of the language so information about the long-run behaviour of the CTMC translates directly to information about the PEPA model from which it was derived.

The benefit of using a high-level description language for Markov modelling is that it is possible to implement tools to automate the process of deriving the CTMC from the high-level model. The first tool which we implemented for PEPA did exactly this, and this tool was the PEPA Workbench [2]. The PEPA Workbench was implemented in Standard ML [3], partly because it is a beautiful programming language, and partly because we thought that this might facilitate interoperation with other process algebra tools and verification platforms such as the Concurrency Workbench [4] and HOL/ML [5]. In fact, we never connected the PEPA Workbench to either of these tools and this remains a missed opportunity to this day.

*Laboratory for Foundations of Computer Science, The University of Edinburgh, Edinburgh, EH9 3JZ. Email: {stg, jeh}@lfcs.ed.ac.uk

The ML edition of the PEPA Workbench exported its results as the infinitessimal generator matrices of CTMC, represented in the matrix formats used by numerical computing platforms such as Maple, Matlab and Mathematica. This format had the advantage that it was human-readable and even outside the Workbench the PEPA modeller worked with high-level data structures such as sparse matrices and vectors without having to deal with the concrete data structures which were used to implement these.

Having even modest tool support available allowed us and others [6, 7, 8] to solve meaningful models of realistic applications. However, some modellers wanted to make more detailed models still and more detail equates to more states in the system and in the underlying Markov chain. One development in the PEPA tools came about because the ML edition, together with Maple, was unable to solve Robert Holton's robotic workcell model [9] quickly enough to generate an acceptable set of experimental plots. For this reason we modified the PEPA Workbench to interoperate with an external solver written in C [10].

This set the pattern for much of the PEPA tool development which was to follow; we would use a simple high-level tool to gain experience and insights and then extend this to a better engineered tool later. The ML edition of the PEPA Workbench is still used in this way, as a testbed for extensions of the PEPA language such as PEPA nets [11] and for new algorithms [12].

At this time a number of users were using the PEPA Workbench on a number of platforms (Solaris, Linux and Windows) and at the same time the ML language was undergoing a significant revision (from SML'90 [13] to SML'97 [3]). In addition, Graham Clark had implemented some additional PEPA tools such as the PEPAroni simulator in Pizza [14], an extension of Java. The development of the Pizza compiler was then discontinued in favour of GJ [15], meaning that something had to be done with PEPAroni also. For these reasons we decided to port the PEPA Workbench to Java [16]. In addition we incorporated transient solution facilities [17] as well as the PEPAroni simulation capabilities [18].

The Java edition of the PEPA Workbench became a vehicle for experimentation also. The Java language is supported by an enormous set of libraries and APIs. Some of these have no equivalents in Standard ML and so some experimental extensions are easier to implement in the Java edition than in the ML edition of the workbench. One example of this would be the extension of the workbench to interoperate with Argo/UML [19] whereby UML state machines encoded in XMI format can be loaded onto the workbench and solved. This depends on an XML parsing package which is native to Java, but not to Standard ML.

2 Analysis Tools

Of course, generating the CTMC underlying a PEPA model, and finding its steady state probability vector is rarely, if ever, the objective of PEPA modelling. Formal tool support for querying performance models is an area which has received little attention until recently, despite its practical importance. Whilst some effort has been applied in this direction for PEPA models, it remains an area in which we see much scope for future work.

At the most basic level the modeller wishes to construct a *reward structure* over the state space of the CTMC, to be used in conjunction with the steady state probability vector to derive performance measures. For steady state measures the reward structure is a vector recording a "reward" for each state, although for many state the reward value will be zero. Thus the problem becomes one of identifying the appropriate set of states to attach a non-zero reward to. Clearly, when the CTMC arises from a stochastic process algebra model we prefer to characterise the state at the process algebra level. PEPA analysis tools have been developed which tackle this problem in two distinct ways.

2.1 The PEPA State Finder

The PEPA State Finder is intended to be used with the ML edition of the PEPA Workbench. It identifies subsets of states using regular expression pattern matching, applied to the table of PEPA expressions which make up the state of the model. Recall that there is a one-to-one correspondence between the syntactic forms of the PEPA process as it evolves and the states of the CTMC. The Workbench maintains a table recording this correspondence, and usig regular expression pattern matching the PEPA State Finder is able to extract the states of interest. For example, it is possible to use an expression such `*|(next,r).*` to return the state numbers of all the sate in which the second component enables a $(next, r)$ activity. This could then be used to construct a reward structure suitable for calculating the throughput of $next$ in the second component: the value r is placed in the reward vector at each position corresponding to a (numerical) state found by the PEPA State Finder.

2.2 PML$_\mu$

A more sophisticated means of specifying rewards is described in Graham Clark's PhD thesis [20], and developed around the stochastic logic PML$_\mu$. Inspired by the probabilistic model logic of Laren and Skou, PML [21], PML$_\mu$ is able to differentiate PEPA terms which perform the same activities but at different rates. The key to this is a modification to Hennessy-Milner logic in which the diamond operator becomes decorated with a rate. The semantics of an expression in the logic is a subset of states, and thus logical expressions may be used, in conjunction with a value, to specify a reward structure. Graham Clark extended the ML edition of the PEPA Workbench to include support for PML_μ and associated reward structures [20].

3 Interoperation with other tools

The new pattern of working with the PEPA tools is to connect them to, or build them into, existing tools for performance modelling. This section recounts three such efforts, all of which are still ongoing.

3.1 PEPA and Möbius

Because the Möbius modelling framework [22] is both a multi-formalism and multi-paradigm modelling tool it was an ideal place to begin integrating PEPA with another, distinctly different, modelling tool [23]. When working with a custom tool such as the PEPA Workbench, the goal is very clear: it is to produce a correct implementation of the PEPA language. Deviations of the PEPA Workbench from the PEPA semantics are simply errors which need to be fixed. The unexpected effect of integrating PEPA into Möbius was that not only was PEPA support built into Möbius but Möbius features were built into the version of PEPA which was implemented in Möbius. The reason for this was that it then became possible to share variables between components modelled in PEPA and components implemented in another Möbius formalism, such as SANs.

Building support for PEPA directly in another tool has the advantage that the language is supported efficiently, without additional overheads imposed by translation from one representation into another. However, it has the implementation cost that the implementor must be familiar not only with the concepts of the host tool but also with their representation in data structures and algorithms. This means that although this approach can produce very good results it can do so at relatively high cost.

3.2 PEPA and PRISM

We next progressed to a lighter-weight, component-based method where we attempted to work with a relatively loose coupling between the language and the host solver. This was the approach when we produced a binding for the PEPA language in the PRISM [24] probabilistic model checker. This proceeded in two stages. Firstly, Dave Parker and Gethin Norman extended the PRISM model checker to support PEPA's combinators (parallel and hiding). Then an existing PEPA tool, the PEPA-to-Ada translator [25], was adapted to form the PEPA-to-PRISM Compiler. This involved re-targeting the compiler to generate as its output format the reactive modules language supported by PRISM, including the extension to the PEPA modelling constructs.

One aspect of this work which came as a surprise was the extent to which one could have small, but problematic, mismatches between one modelling language and another. One of the most obvious is that PEPA defines synchronisation between active participants (via *apparent rates* [1]) differently from PRISM (the rate of the synchronised activity is the product of the rates of the participants in the synchronisation). For this reason the PEPA-to-PRISM compiler implements a static check to detect active/active synchronisation and to fault any model which uses it.

This method of working with PEPA models requires a significant degree of expertise on the part of the modeller, because errors in evaluation can occur right from the PEPA parser through PEPA-to-PRISM and PRISM down to CUDD [26], the BDD library which provides MTBDD data structures and algorithms to PRISM. Even if errors do not occur in translation still to achieve the best performance from the solver it is necessary to know how to configure both PRISM and CUDD which means that this method is best suited to experienced modellers only.

3.3 PEPA, IPC and Dnamaca

A recent development in the PEPA tools is Jeremy Bradley's ipc (The Imperial PEPA Compiler). The ipc tool translates an input PEPA model into the Petri net notation provided by Will Knottenbelt's Dnamaca tool [27]. Translating a process algebra model into a Petri net might seem a rather strange thing to do but we can now view PEPA as a sublanguage of the PEPA nets modelling language and then translating a coloured high-level Petri net into an uncoloured classical net seems a much more familiar activity. In fact, Dnamaca would be an excellent target for PEPA nets because of its native support for priorities, which are available in PEPA nets but not in PEPA.

The ipc tool supports the PEPA language comprehensively. Apparent rates are supported, as are anonymous components. These are two advantages over the PEPA-to-PRISM compiler, and a richer class of PEPA models can be analysed by ipc/Dnamaca as a result.

In one other important respect, ipc provides more comprehensive PEPA support than comparable tools because it also translates PML_μ formulae into the Dnamaca specification language. The Dnamaca specification language is a classical Petri net logic allowing specification formulae to quantify the number of tokens in the places of the net and thereby identify states and sets of states within the reachable state space of the model.

Via ipc, the unique solution capabilities of Dnamaca become available and because of this it is possible to efficiently perform passage time analysis over PEPA models.

4 Tools for PEPA nets

The PEPA nets modelling language extends PEPA by using PEPA components as the tokens of a high-level coloured Petri net. The PEPA nets formalism is considerably newer than the PEPA stochastic process algebra and consequently tool support is less extensively developed.

4.1 The PEPA Workbench for PEPA nets

Our first tool to support PEPA nets was an extension of the ML edition of the PEPA Workbench, as might have been expected. As a high-level programming language with strong support for compile-time checking of programs, Standard ML was an excellent choice for the implementation language for the first PEPA nets tool. The PEPA nets semantics look simple on the page but they proved to be a challenge to implement both efficiently and correctly.

The changes to the PEPA Workbench involved changes to the parser and the internal data structure for representing components, and most significantly, the derivation function for next-step derivatives. This is now considerably more complex than in the PEPA case because of the interplay between transitions of tokens and firings of the net, together with the intervention of priorities on firings.

4.2 The PEPA net compiler

Because our experience with connecting PEPA to other modelling tools had been a positive one it became evident that we could continue this by compiling a PEPA net down to a PEPA model (and then analysing this with Möbius, PRISM or ipc/Dnamaca). Direct compilation from a PEPA net into the native format of the host tool would be preferable but translation into PEPA provided us with an opportunity to at least investigate some of the issues involved without all of the additional complexity of unexpected interactions with the modelling language of the host tool, which is typically less familiar to us than our own languages.

PEPA stochastic process algebra models have no notion of context, nor any capacity to move components from one context to another, and therefore cannot express the concept of dynamically varying communication structure. The task performed by the PEPA net compiler is then to remove all of the mobility from the PEPA net by making components' behaviour depend on location. This is achieved by expanding the definition of the tokens of the net replicating local state behaviours and customising these for each cell which the token may visit. (Cells are the storage areas for tokens in a PEPA net. A net may have many places and each place may have many cells.)

In the worst case—when there is only a single class of token, all tokens can travel to all of the places of the net, and the net has no static components—the resulting PEPA model is n times larger than the input PEPA net, where n is the number of cells in the net.

By chaining the PEPA nets compiler together with the PEPA compilation tools we have been able to solve PEPA net models of significant size.

4.3 DrawNET

One putative advantage of the PEPA nets modelling language over PEPA is that it has an accessible graphical syntax. As a result it seemed worthwhile to implement a graphical editor for PEPA nets. Together with Marco Gribaudo we configured the generic DrawNET tool for PEPA nets, including PEPA as a sublanguage [28].

DrawNET provides interactive consistency checking of models while they are being edited and saves models in XML format for easy parsing and conversion into other concrete syntaxes, with subsequent compilation as above.

5 Availability

The PEPA Workbench and related tools are available from the PEPA Web site at `www.dcs.ed.ac.uk/pepa`. Möbius is available from the University of Illinois at `www.crhc.uiuc.edu/PERFORM/mobius-software.html`. PRISM is available from `www.cs.bham.ac.uk/~dxp/prism/`. ipc is available from `www.doc.ic.ac.uk/ipc/`. The Dnamaca and DrawNET tools are available from their authors.

6 Conclusions

Despite the difficulty of the approach, and the possibility of errors throughout the process, linking the PEPA tools to external solvers such as PRISM and Dnamaca provided orders of magnitude improvements in solution power over the capabilities of the native PEPA tools. For this reason, this seems like a very profitable avenue for future tool development. Connecting other front-ends such as DrawNET or Argo/UML would provide a high-level diagrammatic syntax which might appeal to other users. The role of PEPA in this work is to function as an intermediate language, forming a middle ground between unrelated tools such as DrawNET and PRISM. There remains much which can still be achieved in this way and there is great potential to go forward.

Acknowledgements: Many people have contributed software or ideas or both to the development of the PEPA tools: Jeremy Bradley, Linda Brodo, Catherine Canavet, Graham Clark, Marco Gribaudo, Robert Holton, Jon Hunter, Gethin Norman, Dave Parker, Matthew Prowse, Kris Powell, Fotis Strathapolis, Nigel Thomas, Feng Wan, and others.

The work on the PEPA tools would not have been done if not for the interest of users of the PEPA language such as Howard Bowman, Robert Holton, Marta Kwiatkowska, Amani El-Rayes, Nigel Thomas, Leïla Kloul, Marina Ribaudo, and others.

The authors are supported by the DEGAS (Design Environments for Global Applications) IST-2001-32072 project funded by the FET Proactive Initiative on Global Computing.

References

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

[2] 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.

[3] Robin Milner, Mads Tofte, Robert Harper, and David MacQueen. *The Definition of Standard ML: Revised 1997.* The MIT Press, 1997.

[4] Rance Cleaveland, Joachim Parrow, and Bernhard Steffen. The concurrency workbench: A semantics-based tool for the verification of concurrent systems. *ACM Transactions on Programming Languages and Systems*, 15(1):36–72, January 1993.

[5] M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: A theorem proving environment for higher order logic.* Cambridge University Press, 1993.

[6] 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.

[7] A. El-Rayes, M. Kwiatkowska, and S. Minton. Analysing performance of lift systems in PEPA. pages 83–100, Department of Computer Science, The University of Edinburgh, September 1996.

[8] 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)*, September 1998.

[9] D.R.W. Holton. A PEPA specification of an industrial production cell. In S. Gilmore and J. Hillston, editors, *Proceedings of the Third International Workshop on Process Algebras and Performance Modelling*, pages 542–551. Special Issue of *The Computer Journal*, 38(7), December 1995.

[10] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, second edition, 1993.

[11] S. Gilmore, J. Hillston, and M. Ribaudo. PEPA nets: A structured performance modelling formalism. In T. Field, P.G. Harrison, J. Bradley, and U. Harder, editors, *Proceedings of the 12th International Conference on Modelling Tools and Techniques for Computer and Communication System Performance Evaluation*, number 2324 in Lecture Notes in Computer Science, pages 111–130, London, UK, April 2002. Springer-Verlag.

[12] 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.

[13] Robin Milner, Mads Tofte, and Robert Harper. *The Definition of Standard ML.* The MIT Press, 1990.

[14] M. Odersky and P. Wadler. Pizza into Java: Translating theory into practice. In *Proceedings of the 24th ACM Symposium on Principles of Programming Languages (POPL'97), Paris, France*, pages 146–159. ACM Press, New York (NY), USA, 1997.

[15] Gilad Bracha, Martin Odersky, David Stoutamire, and Philip Wadler. Making the future safe for the past: Adding genericity to the Java programming language. In *Proceedings of OOPSLA 98*, Vancouver, October 1998.

[16] J. Hunter. Re-evaluation of the PEPA Workbench. Master's thesis, School of Computer Science, The University of Edinburgh, September 1999.

[17] F. Wan. Interface engineering and transient analysis for the PEPA Workbench. Master's thesis, School of Computer Science, The University of Edinburgh, September 2000.

[18] Fotis Stathopoulos. Enhancing the PEPA Workbench with simulation and experimentation facilities. Master's thesis, School of Computer Science, Division of Informatics, The University of Edinburgh, 2001.

[19] C. Canevet, S. Gilmore, J. Hillston, M. Prowse, and P. Stevens. Performance modelling with UML and stochastic process algebras. *IEE Proceedings: Computers and Digital Techniques*, 150(2):107–120, March 2003.

[20] G. Clark. *Techniques for the Construction and Analysis of Algebraic Performance Models*. PhD thesis, The University of Edinburgh, 2000.

[21] Kim Guldstrand Larsen and Arne Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.

[22] G. Clark, T. Courtney, D. Daly, D. Deavours, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. Webster. The Möbius modeling tool. In *Proceedings of the 9th International Workshop on Petri Nets and Performance Models*, pages 241–250, Aachen, Germany, September 2001.

[23] G. Clark and W.H. Sanders. Implementing a stochastic process algebra within the Möbius modeling framework. In L. de Alfaro and S. Gilmore, editors, *Proceedings of the first joint PAPM-PROBMIV Workshop*, volume 2165 of *Lecture Notes in Computer Science*, pages 200–215, Aachen, Germany, September 2001. Springer-Verlag.

[24] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In T. Field, P.G. Harrison, J. Bradley, and U. Harder, editors, *Proceedings of the 12th International Conference on Modelling Tools and Techniques for Computer and Communication System Performance Evaluation*, number 2324 in Lecture Notes in Computer Science, pages 200–204, London, UK, April 2002. Springer-Verlag.

[25] S. Gilmore, J. Hillston, and D.R.W. Holton. From SPA models to programs. pages 179–198. Dipartimento di Informatica, Università di Torino, CLUT, July 1996.

[26] F. Somenzi. *CUDD: CU Decision Diagram Package*. Department of Electrical and Computer Engineering, University of Colorado at Boulder, February 2001.

[27] W.J. Knottenbelt. Generalised Markovian analysis of timed transition systems. Master's thesis, University of Cape Town, 1996.

[28] S. Gilmore and M. Gribaudo. Graphical modelling of process algebras with DrawNET. In F. Bause, editor, *Companion volume of tools papers to Proceedings of Tools'03*, September 2003. To appear.