Software performance design environments

Catherine Canevet, Stephen Gilmore,
Jane Hillston and Perdita Stevens
Laboratory for Foundations of Computer Science
The University of Edinburgh, Scotland
{ccanevet, stg, jeh, pxs}@dcs.ed.ac.uk

June 12, 2002

Abstract

We describe a software toolset which allows UML modellers to annotate their models with performance information. An equivalent performance model is extracted from the UML, solved, and the results reflected back to the UML level. Used in this way, our toolset gives a high-level approach to software performance modelling where the benefits of the performance modelling process are achieved without significant additional notational burden.

The Unified Modelling Language (UML) [1] is an effective and popular notation which is used to capture high-level designs for software systems. However, one aspect of software system design which is not typically captured in a UML document is a record of the likely (or desired) rate of performance of the major activities of the system. When this information is not included in the initial UML description of a system it increases the likelihood that the performance of the software system being developed will not be considered until very late in the software development process. At this stage, errors in the design will be very costly to repair and will require significant re-engineering.

Our aims in this paper are twofold:

1. to show how UML models enhanced with performance information can be mapped onto an existing performance modelling notation, Performance Evaluation Process Algebra (PEPA) [2]; and

2. to show how the results of the performance analysis of the PEPA models which are produced by this process can be presented to the UML modeller in the terms of their model.

The work reported here forms an early part of the DEGAS project. DEGAS stands for Design Environment for Global ApplicationS, and the project's overall goal is to make sophisticated formal analysis techniques available to designers of global applications – that is, of sofware systems that run on wireless networks and may involve mobile code – in a way which is congruent with their normal way of working. Performance prediction has been identified as an important

area where such formal analysis techniques might be able to make a significant contribution to the quicker design of better applications.

Designers in the DEGAS partner companies (Motorola and OMNYS) and throughout the industry have adopted UML as their main software design notation. UML is a diagrammatic notation for recording the design of systems, especially object-oriented software systems. A UML *model* is represented by a collection of diagrams describing parts of the system from different points of view; there are seven main diagram types. For example, there will typically be a *static structure diagram* (or *class diagram*) describing the classes and interfaces in the system and their static relationships (inheritance, dependency, etc.) State diagrams, a variant on Harel state charts, can be used to record the dynamic behaviour of particular classes. Interaction diagrams, such as sequence diagrams, are used to illustrate the way objects of different classes interact in a particular scenario. In this work we concentrate on state diagrams, which provide a behavioural description in automata-theoretic terms which is also familiar from process algebras and protocol specifications.

Our aim in working with UML in the performance modelling process is to introduce the benefits of performance analysis with process algebras without the complexities and conceptual challenges which are normally associated with formal description techniques such as these. To this end, we deploy the PEPA stochastic process algebra as an intermediate language in the performance analysis process. The UML modeller can compose models and solve these for performance results without needing to understand the PEPA language, its formal definition or even how their model is represented in PEPA. At the same time, we avoid requiring the designer to develop a model specifically for performance analysis; instead, we work directly with the UML model which is being developed for other purposes.

Structure of this paper: In the next section we discuss modelling with UML and PEPA. In Section 2 we describe how performance measures such as utilisation can be obtained directly from our results. In Section 3 we discuss the software architecture of our tool set and describe the four principal software packages which are involved. In Section 4 we present a simple example. In Section 5 we discuss related work on UML, PEPA and performance modelling. We present our conclusions in Section 6.

1 Modelling with UML and PEPA

We bring the UML and PEPA notations together by forming a bridge between two existing applications which support these languages, the ArgoUML modelling tool [3] and the PEPA Workbench [4]. UML designs which are built using ArgoUML can be exported as XML Metadata Interchange documents (XMI) [5] as a standard part of the ArgoUML tool. The XMI format is used to represent UML models when exchanging them with other tools, as here, and facilitates the analysis and manipulation of UML models using standard XML tools [6]. We have developed an application which automatically converts the XMI documents generated by ArgoUML into the input file format of the PEPA Workbench. Figure 1 shows screenshots of two ArgoUML designs of simple communicating state machines together with the equivalent descriptions in the

PEPA stochastic process algebra. With a slight abuse of notation we show the rates of the transitions as UML actions. Thus $a/rate(r)$ is used to represent the information that the activity a is performed at rate r which is not the usual *event/actions* syntax for arc adornments in UML state diagrams.

When it is provided with an input PEPA model the PEPA Workbench explores the model to generate its full state space. This state space is used to form a CTMC representation of the system which is solved to find its steady-state probability distribution. As is usual with interleaving models of concurrent systems, the size of the state space of the system as a whole is bounded by the product of the state spaces of the individual PEPA components which are composed in parallel. Simply presenting this large probability vector back to the UML modeller as the result of the analysis would be unlikely to provide any insights into the long-run operation of the model, or hotspots or bottlenecks in the system. For this reason we look for an alternative means of communicating performance measures.

2 Performance measures

The most general way to describe performance measures is to build a reward structure on the model. However, associating locations in the equilibrium probability distribution with syntactic states of the model exposes details of the representation such as orderings of components in the PEPA system description. Such an approach would generally require the UML modeller to face much of the complexity of working directly with Markov Chains. Higher-level description languages for specifying performance measures exist, such as PML$_\mu$ [7] and CSL [8], but these notations would be formidable for a typical UML developer to use.

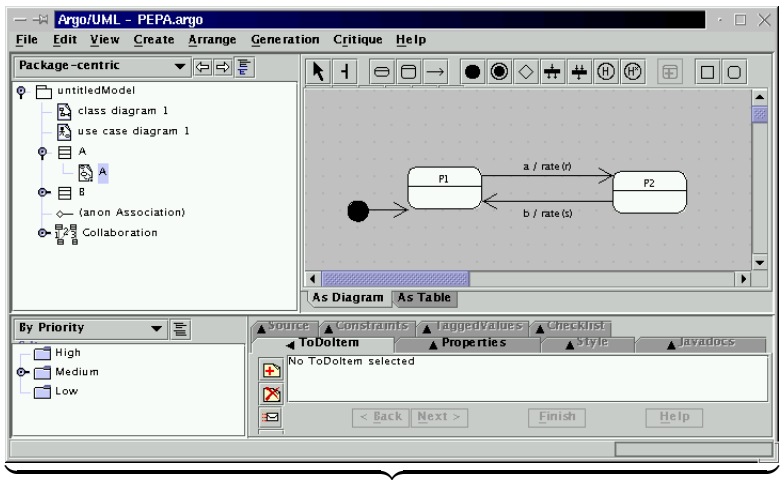
For this reason we aggregate the state space of the system over the local states of each PEPA component. This has two beneficial effects:

1. it avoids the need for any descriptions of state-space representations, whether high-level or low-level; and

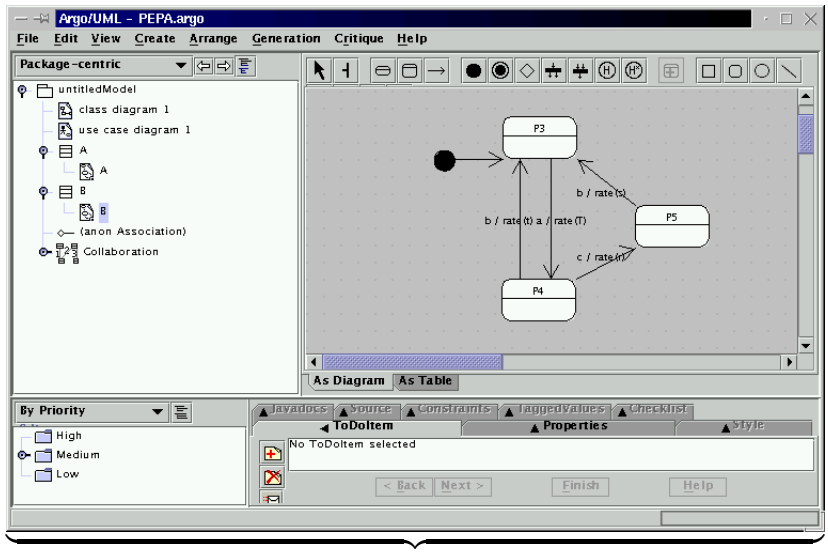
2. instead of working with a large set of very small numbers the modeller works with a small set of numbers which are orders of magnitude larger.

Our approach to specifying performance measures is to define UML components which expose the configurations of interest in the model. Behaviourally, such components may be redundant, but they are necessary for expressing performance measures over the model. Typically such components will specify that they passively witness activites which have been performed and change state in order to reflect this information. By programming such components carefully it is possible that they do not increase the state space of the underlying Markov Chain but allow the modeller to observe that some sequence of activities has happened, and to learn the probability of this. We have used this approach previously [9].

We illustrate the use of this method with a resource example and a queue example. One performance measure which is typically of interest is the calculation of the utilisation of resources in the system. To do this the modeller need only express the resource as a simple component as described below and the



$$P_1 = (a, r).P_2; \quad P_2 = (b, s).P_1$$



$$P_3 = (a, \top).P_4; \quad P_4 = (b, t).P_3 + (c, r)P_5; \quad P_5 = (b, s).P_3$$

Figure 1: Screenshots of UML designs in ArgoUML with PEPA equivalents

utilisation can be directly read from the model solution as the percentage of time that the component spends in the *Busy* state. This points to resources which are under-utilised, or over-utilised.

$$/* \text{ An idle resource can be acquired } */$$
$$Idle \stackrel{def}{=} (acquire, r_1).Busy$$

$$/* \text{ A busy resource can be released } */$$
$$Busy \stackrel{def}{=} (release, r_2).Idle$$

Taking the idea a little further, a finite-capacity M/M/n queue can be specified in PEPA with a list of component definitions ending with the following one.

$$/* \text{ no arrivals are allowed when the queue is full } */$$
$$Queue_n \stackrel{def}{=} (serve, \mu).Queue_{n-1}$$

The percentage of time that the queue will be full can be read off directly from the updated UML description. This points to the possibility of clients sending requests faster than they can be serviced by a server.

3 Software architecture

In this section we describe the architecture of our application. We have built on two existing software tools, ArgoUML and the PEPA Workbench. We have used ArgoUML with no modifications, so (up to minor XMI version differences) it could be replaced by any other XMI-capable UML tool. We modified the PEPA Workbench to make communication between the two tools easier. We begin by first describing these two tools for the benefit of readers who are not familiar with them. The architecture of the system is summarised in Figure 2.

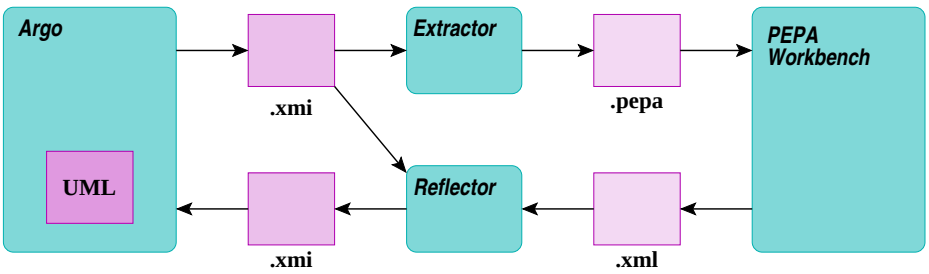


Figure 2: Software architecture of the tool

3.1 ArgoUML

ArgoUML is a modelling tool which supports software developers who are producing software designs using UML. It provides many features which are familiar from existing CASE tools. Examples of these are editors for graphical notations such as class diagrams and state diagrams.

In addition, one of the distinctive features of ArgoUML is that it provides good support for the cognitive aspects of design, including supporting informal note-taking on "To Do" lists and other creativity aids. In all, it provides a professional and thoughtfully-engineered UML development platform.

As with most modern UML tools, ArgoUML exports UML models in the XMI document format, and loads saved models from the same XMI format. This provides the import and export formats for our other tools. The XMI document written by ArgoUML is read by our Extractor tool. The same XMI document and the results from processing by the PEPA Workbench (in XML format) are read by the Reflector tool to provide an updated input document which is loaded by ArgoUML.

3.2 The PEPA Workbench

The PEPA stochastic process algebra is supported by a range of tools including the PEPA Workbench [4], the Möbius Modeling Framework [10] and the PRISM probabilistic symbolic model checker [11]. We have used the PEPA Workbench so far in this work but the design of our companion Extractor and Reflector tools is general-purpose so that it would be possible to adapt our work to use either Möbius or PRISM instead. Both Möbius and PRISM offer capabilities which the PEPA Workbench does not. Möbius supports multi-paradigm modelling where PEPA models are combined with SANs or ball-and-bucket models as used by MARCA. PRISM provides probabilistic symbolic model checking allowing models to be checked against CSL formulae. Both of these tools could be valuable in our ongoing work but an engineering challenge would remain to allow the UML modeller to access their powerful capabilities without first needing to master their technical foundations.

The PEPA Workbench exists in two distinct versions. The first version is an experimental research tool which is coded in the functional programming language Standard ML [12]. The second is a re-implementation of this in the Java programming language. These are known as "the ML edition" and "the Java edition" respectively.

We adapted the Java edition of the PEPA Workbench to interoperate with our Extractor and Reflector tools. The Java edition provides a graphical user interface to assist the PEPA modeller in working with models, accessing tools such as the state finder tool, the simulator or the walkabout utility and choosing between a range of steady-state and transient solvers and a range of output formats. It would be impractical to use this user interface together with the ArgoUML application so we added a command-line interface to the PEPA Workbench, allowing it to be configured with a range of command-line switches. One of these switches requests the Workbench to aggregate the performance results of the model over the local states of each PEPA component. Each such component is a simple sequential state machine which corresponds directly to a UML state diagram. Each of these has a very small state space relative to the state space of the model as a whole, making the result set much more compact. Of course this compactness is necessarily achieved at the expense of losing information about particular states of the global state space.

The PEPA Workbench processes an input PEPA model which the Extractor generates from an input UML model in XMI format. It writes its results as an XML document which is processed by the Reflector tool.

The PEPA Web page at `http://www.dcs.ed.ac.uk/pepa` is the download site for the PEPA Workbench and supporting software and papers.

3.3 Extractor

The Extractor application is programmed in the Python programming language. It reads in the XMI file generated when saving a UML model with ArgoUML (or a similar tool) and returns the corresponding input file for the PEPA Workbench.

We use the Minidom XML parser to parse the XMI file. Once we have converted the XMI file stream into a DOM object, we can then access the individual tags by name. The document is represented as a tree structure. When processing the XMI file we look for all the elements that we will need to provide in the PEPA model which we produce as output. We discuss these by element type in turn below.

- For each `statemachine` in the file, we identify its `context` (in order to match the classifier role later on) and its initial state, and then for each state, its identifier and its name.
- For each `transition`, we find its `workload` (the UML event), its `rate` (shown as a UML action), and its source and target state.
- For each `collaboration`, we identify for each classifier role, the name of the synchroniser, and for each `base`, the identifier of the classifier role. Using the identifier of the context, we will be able to find the initial state of the corresponding `statemachine`.

Now, we have all we need in order to write a PEPA Workbench input. First, we print each rate. Then, we print for each state machine a defining PEPA expression of the following form: `#source=(workload,rate).target+...` For each state, if it is a source, we find its transitions with the correct workload, rate and target.

Taking the information from a UML collaboration diagram as shown in Figure 3 we can assemble a collaboration line (e.g. `P1<a,b>P3`). We look for the initial states using the matching of context and classifier role, and then look for the activity names in the synchronisation set.

3.4 Reflector

The Reflector takes as its parameter the original XMI file and the XML file that contains all the results from the PEPA Workbench. It returns a modified XMI file, which when loaded in ArgoUML will show the modified model.

We use Minidom to parse the original XMI file and also to build the modified XMI file by creating the branches and leaves of the document. The `xml.dom.minidom` module supports a very simple interface for adding new XML tags and data to an XML document.

We parse the XMI file corresponding to the UML model and the XML file corresponding to the PEPA Workbench results. When processing the PEPA Workbench results file, for each state, we look for the probability tag and identify its value. Then we modify our original XMI file where for each `statemachine`, for each `state`, we add the corresponding probability.

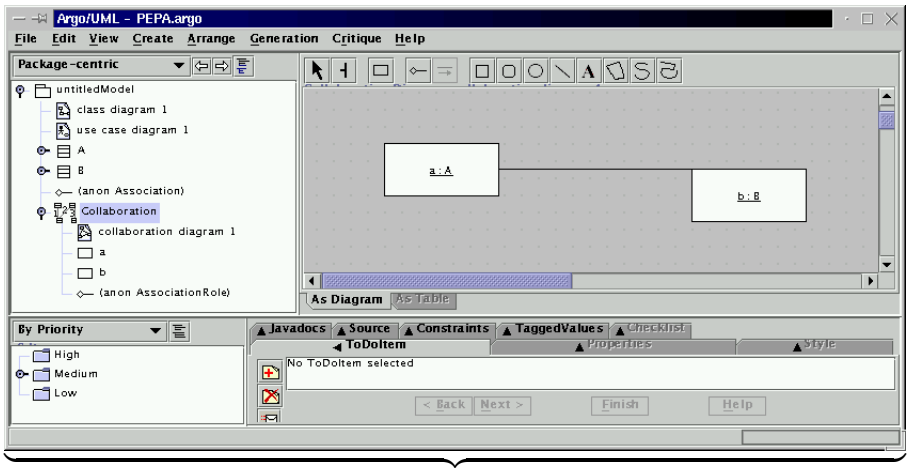


Figure 3: UML collaboration diagram

4 A simple example

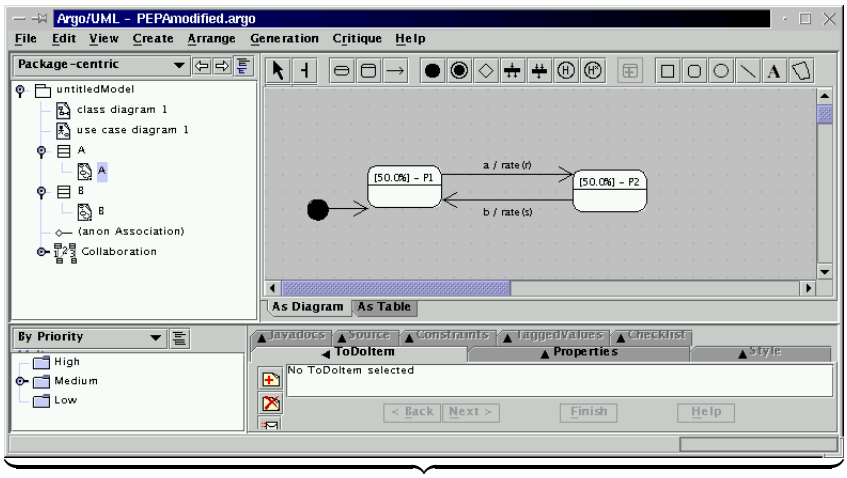
We demonstrate our method on a simple generic example model. The model is formed by a composition of a two-state component and a three-state component. To make this generic example more concrete, the two-state component might represent a client which requests a service and receives a reply and the three-state component might represent a proxy server which sometimes replies directly but at other times connects to another server before replying.

Figure 1 shows the original UML model used for our example. When this UML model is saved in ArgoUML, an XMI file is generated. If we use the Extractor with this XMI file, we obtain the PEPA model which is shown in Figure 4 in the concrete syntax of the PEPA Workbench. This is used as the

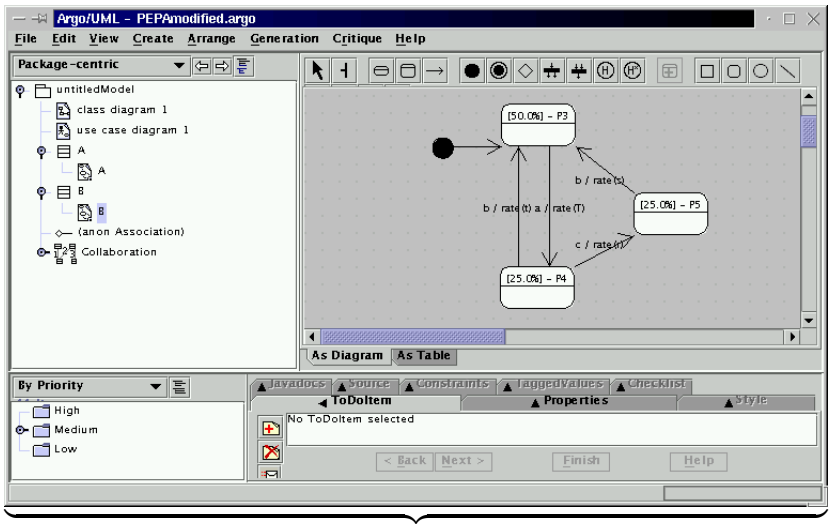
```
%r = 2.0;
%s = 2.0;
%t = 2.0;
#P4 = (c,r).P5 + (b,t).P3;
#P5 = (b,s).P3;
#P3 = (a,infty).P4;
#P1 = (a,r).P2;
#P2 = (b,s).P1;
P1 < a,b > P3
```

Figure 4: PEPA model generated by the Extractor tool

input for the PEPA Workbench which then produces its results in an XML file. We can now use the Reflector to modify the original XMI file, and make the probability of each state appear on the UML diagrams (see Figure 5).



$P_1 : \ 50.0\%, \quad P_2 : \ 50.0\%$



$P_3 : \ 50.0\%, \quad P_4 : \ 25.0\%, \quad P_5 : \ 25.0\%$

Figure 5: Screenshots of ArgoUML incorporating PEPA results

5 Related work

Ours is not the first work on using UML with stochastic process algebras, nor even the first on using UML with PEPA. Pooley [13] previously discussed generating PEPA models from a combination of sequence diagrams, collaboration diagrams, and a combined collaboration/state diagram. His method differs from ours in that more types of diagram are used and there seems to be no automatic procedure used to generate the PEPA model from the UML. In contrast, our method is automatic and can be re-run after changes to the UML model in order to generate and solve an equivalent updated PEPA model.

Mitton and Holton undertake an alternative mapping between PEPA and UML statecharts [15] but again their method does not appear to be automated.

In another paper Thomas, Munro, King and Pooley combine PEPA models with graphical notations for visualising derivation graphs, PEPA component interfaces and other aspects of the system under study [16]. This work provides an interesting insight into the PEPA modelling language for modellers who are not familiar with the notation. Their approach is supported by a prototype tool. Our contribution here differs in that we are using a standard and widely-understood modelling language (UML) instead of more specialised, but necessarily less well-understood bespoke graphical notations.

A work closely related to ours in spirit, if not in detail, is recent work by Petriu and Shen which maps UML models via XMI into layered queueing network (LQN) performance models [17]. This mapping is in one direction only, so that UML models are mapped into LQN models but there appears to be no mapping of the performance results obtained from the LQN model back up to the UML level. Thus their tool appears to be similar to our Extractor tool, combined with the PEPA Workbench, but without an equivalent of our Reflector tool.

6 Conclusions

We have presented a method of deriving performance information from UML models. Our method is unusual in that it greatly reduces the amount of additional notation and concepts which need to be understood by the modeller when compared to working directly with stochastic process algebras, Petri nets, queueing networks or other traditional performance modelling formalisms. The method is supported by a tool set which comprises some existing modelling tools (ArgoUML and the PEPA Workbench) and other translators which we have written to connect them (the Extractor and the Reflector). The translators which interconnect the applications are general-purpose and can be adapted to work with other modelling tools.

We have applied our approach to a range of small examples. We plan to investigate its usefulness when applied to larger examples.

We have used a bespoke method of adding performance information to UML models here. We also plan to investigate more standard ways of representing the performance information in UML. This will probably use the Schedulability Performance and Time profile [18], as used by Petriu and Shen, depending on the availability of appropriate UML tool support.

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

References

[1] Object Management Group. Unified Modeling Language, v1.4, March 2001. OMG document number: formal/2001-09-67.

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

[3] Tigris.org project. ArgoUML: A modelling tool for design using UML. Web page and documentation at `http://argouml.tigris.org/`, 2002.

[4] 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.

[5] Object Management Group. OMG-XML Metadata Interchange (XMI) Specification, v1.1, November 2000. OMG document number: formal/00-11-02.

[6] P. Stevens. Small-scale XMI programming: a revolution in UML tool use? *Journal of Automated Software Engineering*, 2002. Accepted for publication.

[7] G. Clark, S. Gilmore, J. Hillston, and M. Ribaudo. Exploiting modal logic to express performance measures. In B.R. Haverkort, H.C. Bohnenkamp, and C.U. Smith, editors, *Computer Performance Evaluation: Modelling Techniques and Tools, Proceedings of the 11th International Conference*, number 1786 in LNCS, pages 211–227, Schaumburg, Illinois, USA, March 2000. Springer-Verlag.

[8] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Verifying continuous time Markov chains. In *Computer-Aided Verification*, volume 1102 of *LNCS*, pages 169–276. Springer-Verlag, 1996.

[9] S. Gilmore and J. Hillston. Feature interaction in PEPA. In C. Priami, editor, *Proceedings of the Sixth Annual Workshop on Process Algebra and Performance Modelling*, pages 17–26, Nice, France, September 1998. Università degli studi di Verona.

[10] 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.

[11] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In Field and Harrison [19], pages 200–204.

[12] R. Milner, M. Tofte, R. Harper, and D. MacQueen. *The Definition of Standard ML: Revised 1996*. The MIT Press, 1996.

[13] R. Pooley. Using UML to derive stochastic process algebra models. In Davies and Bradley [14], pages 23–33.

[14] N. Davies and J. Bradley, editors. *Proceedings of the Fifteenth UK Performance Engineering Workshop*, Department of Computer Science, The University of Bristol, July 1999.

[15] P. Mitton and R. Holton. PEPA performability modelling using UML statecharts. In Thomas and Bradley [20], pages 19–33.

[16] N. Thomas, M. Munro, P. King, and R. Pooley. Visualisation for model comprehension. In Thomas and Bradley [20], pages 47–58.

[17] D.C. Petriu and H. Shen. Applying the UML performance profile: Graph grammar-based derivation of LQN models from UML specifications. In Field and Harrison [19], pages 159–177.

[18] B. Selic, A. Moore, M. Woodside, B. Watson, M. Bjorkander, M. Gerhardt, and D. Petriu. Response to the OMG RFP for Schedulability, Performance, and Time, revised, June 2001. OMG document number: ad/2001-06-14.

[19] A.J. Field and P.G. Harrison, 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, London, UK, April 2002. Springer-Verlag.

[20] N. Thomas and J. Bradley, editors. *Proceedings of the Sixteenth UK Performance Engineering Workshop*, Department of Computer Science, The University of Durham, July 2000.