PEPA Performability Modelling using UML statecharts

Pete Mitton*and Rob Holton †
Department of Computing
School of Informatics
University of Bradford

Abstract

UML is increasingly being used in software engineering and other fields. If experienced UML practitioners were able to easily create models that could be used for performance and functional modelling it would increase the likelihood of performance modelling being implemented at the design stage. This paper proposes a way of mapping UML models onto the stochastic process algebra PEPA and considers the advantages of expanding the mapping to other modelling paradigms.

1 Introduction

There are a number of modelling paradigms available to the performability modeller including queuing theory [1][2], stochastic petri net (SPN)[3][4] and stochastic process algebras (SPA)[5][6] [7]. An ideal modelling tool would allow the use of the most appropriate paradigm. Within the field of software

*P.R.Mitton@scm.brad.ac.uk
†D.R.W.Holton@scm.brad.ac.uk

engineering the Unified Modelling Language (UML)[8][9] is increasingly being used to model the real-time behaviour of systems [10][11][12][13]. However, the tools available do not directly produce performability data or are tied to a particular performability modelling paradigm. A tool that was able to translate to and from a UML model into queuing theory, SPN and SPA modelling paradigms would allow the most appropriate paradigm to be used. With further development the tool may also be able to automatically choose the most appropriate paradigm based on a knowledge base of the capabilities of the available tools. A side-effect of this process is that equivalence of models will be available by a formally defined translations between modelling paradigms using an intermediate UML model. For this to be computable a canonical form for each paradigm would be useful. Such a tool will also expose the assumptions made by modelling tool designers that a neophyte modeller may have misunderstood.

The UML standard includes profiles for different modelling areas [14]. Currently there are profiles within the standard for the Software Development Process and Business Modelling Process. This paper is a "work in progress" exploring how the SPA Performance Evaluation Process Algebra (PEPA) can be translated into a UML description. It explains why an approach based on UML statecharts is appropriate for performability modelling and proposes extensions required to simplify performability modelling. Consideration is also given to the representation of time within UML for performance modelling.

2 Statecharts

UML offers five forms of diagram for representing the dynamic aspects of systems. Use case diagrams are important for visualizing, specifying and documenting the behaviour of an element. Interaction diagrams model the flow of control by time ordering or message passing. An activity diagram is essentially a flowchart that shows the flow of control from activity to activity.

Statecharts show a state machine and include elements from both Moore and Mealy machines. A statechart consists of nodes that represent the state and transitions between nodes. The transitions between nodes are assumed to

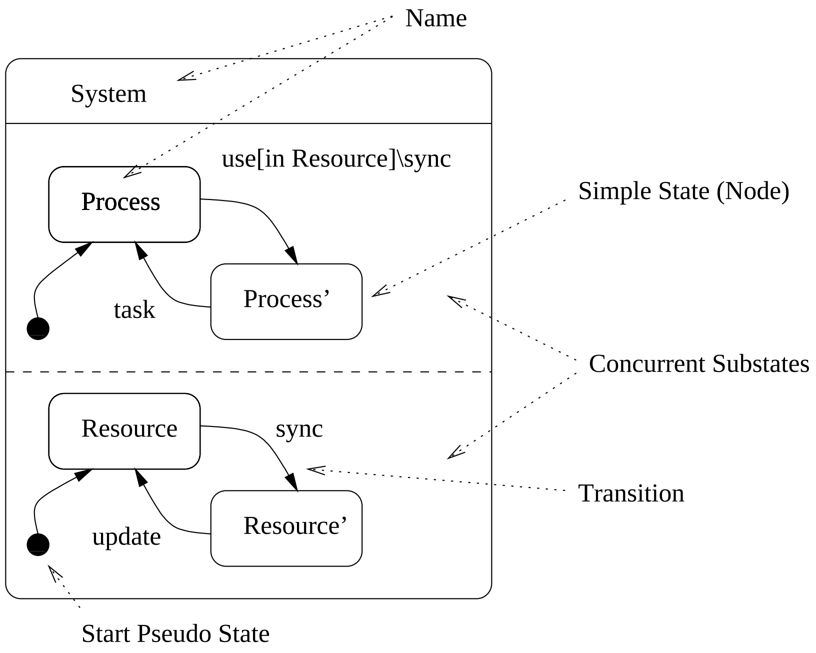

Figure 1: Simple annotated statechart

be instantaneous. A simple example of an annotated statechart is shown in Figure 2. Note that this example does not include the timing information that is essential for performability modelling.

This shows a simple model of a system that consists of a process and a resource that synchronize when the process requires the resource but are then free to operate concurrently. The horizontal dashed line through the system indicates concurrent states. Within each of these two concurrent states there are two sub-states and a pseudo-state. The pseudo-state is indicated by the filled circle and indicates the initial state for the substate. In each concurrent state one sub-state indicates that the resource is free, the other sub-state indicates that the resource is in use.

A UML transition consists of three optional parts, a event that triggers the transition, a guard condition that must be satisfied, and a sequence of actions that are taken when the transition completes. The format is

> *event-signature* '[' guard-condition ']' '/' action-expression

where *event-signature* describes an event and its arguments:

event-name '(' comma-separated-parameter-list ')'

In this example the transition from $Process$ to $Process'$ occurs when the guard condition $inResource$ is true (ie when the resource is available) and the event use is enabled. When the transition completes the action $sync$ is triggered. This action is the event that triggers the transition from $Resource$ to $Resource'$.

Activity charts are a specialized form of statechart that has most of the states as activity states and most of the transitions are triggered by completion of activities in the source state.

Statecharts provide a suitable mechanism for representing other modelling paradigms. The state machine based approach is particularly suitable for the PEPA process algebra as each state within a UML statechart model will correspond to a component within a process algebra model and transitions in a UML statechart correspond to activities in a process algebra model. However, care must be taken when considering cooperation between components/states and these will be discussed later in this paper.

It is likely that this approach can also be used for petri and queuing networks. However, work in this area is only just underway.

The statecharts used in UML are based on Harel statecharts[15][16]. There are unfortunately a number of aspects important to performability modelling that are not clearly defined ether within the UML standard or in Harel's work, of which the most important is the timing scheme. Harel has proposed two schemes, a synchronous and an asynchronous time scheme, but both Harel and the UML standard state that it is up to the tool designer to decide on the most appropriate timing scheme.

In the synchronous time scheme the system executes a single step every time unit. All external changes that have occurred since the last time step are collected and executed at the completion of the time unit. When different clocks exist for different parts of the system the first clock to complete a time unit determines the size of the step.

The asynchronous time scheme allows several several steps within a single point in time. External changes can occur at any time between steps including simultaneously. One way of implementing an asynchronous time scheme would be for an external change to create a cascade of events. Only after a stable state has been reached, as a result of the external change, would the system be able to respond to further changes.

For performability modelling, an approach combining the elements of these approaches would be most useful. To model stochastic processes a synchronous model is most appropriate where each object within the system has its own clock. The time unit of each clock corresponds to the time till the next event. In addition the completion of an event may lead to a sequence of actions that complete before the clocks of all the objects can proceed.

There are two standard forms of timed transition within UML that each take two parameters. The first parameter is the named event that enables the transition. The second parameter is the value of the delay. The two forms are the *schedule* and *timeout* events. The *schedule* event is uninteruptable and will occur, after an appropriate delay, when enabled. The *timeout* event is interruptible and will be restarted if the named event recurs. For performability purposes the second is more appropriate as it allows greater flexibility in the way that timed transitions are handled. In particular, it allows the preemption of enabled transitions when another event occurs.

There are a number of other aspects of statecharts that will be introduced, when required, in what follows.

3 PEPA Modelling Using UML Statecharts

The discussion that follows attempts to consider general performability issues. However, the discussion is currently tempered by the considerations of the PEPA process algebra. There are likely to be a number of additional considerations when other performability techniques are examined. For example, the incorporation of the multiple tokens in stochastic Petri Nets and the notion of flow within a queuing theory model.

UML is an object-oriented modelling technique that allows hierarchies of classes of objects. Each class can participate in association, generalization, dependency and constraint relationships. Each class may also contain tagged values (attributes) describing parameters and constraints upon those values for instances of the class.

For performability modelling it would be useful to incorporate the stochastic nature of events. This could be achieved by providing a subclass of state within a statechart model. However, UML provides a mechanism for creating *stereotypes* which are meta-types that are the equivalent of a new class in the UML meta-model. This is preferable as it allows the creation of a new top level class that is a specialization of a current class. The specialization may have additional constraints and/or attributes that extend the semantics of the original.

In event based modelling paradigms such as SPAs and stochastic petri nets when an event is enabled a stochastic delay occurs before the event is completed. In statechart terms the transition occurs instantaneously at the end of the delay. However, if an event completes it may have an effect on other enabled events.

In PEPA, for example, when an event completes it preempts all the other events that are enabled. Dependant on whether on not the events are sharing the same implicit resource all events that are still enabled will either restart (with resampling) or resume respectively.

It would be useful to incorporate this preemptive behaviour within our stereotype. When a transition occurs from a state within a UML statechart model an *exit* event is triggered. This event can be used to generate further events or signals. A signal is a stereotyped event which proceeds asynchronously and has optional parameters. In this case the parameters should include the resources used by the state generating the *exit* event. This allows the appropriate preemption model to be determined.

The signal event should be broadcast to all states within the model. Each state should respond appropriately to the signal event with the exception of the state generating the signal which should ignore the signal.

To enable correct behaviour (restart/resume) each state should contain information about the progress of each transition. This information should include the time elapsed since the transition was enabled and the current value of the delay time before the transition is taken. The delay time being taken from a probability distribution function.

In UML each time an event occurs any enabled *timeout* transitions are restarted (provided the guard condition is satisfied). This is compatible with PEPA models as external events do not occur and all events that will preempt the model are internally generated.

PEPA has mainly been used with exponential probability distributions, however extensions have been proposed to extend its use with other probability distributions [17]. When an exponential probability distribution is used the restart and resume behaviours are the same due to the *historyless* nature of exponential distributions.

This gives a stereotype state($P.State$) and signal relationship ($preempt$) as shown in Figure 2.

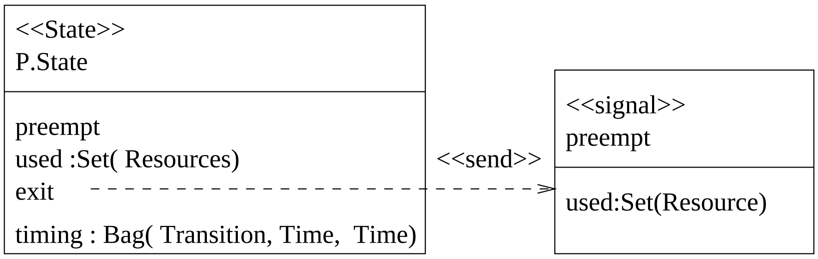

Figure 2: Performability Stereotypes

The *preempt* event is an internal transition that does not generate an *exit* event. This prevents the generation of an infinite chain of *preempt* events.

The *timing* attribute contains each timed transition from the state,the start time when the transition was enabled and the last value for the delay drawn from the probability distribution function. This delay will appear as the second parameter of the timed transition. From the last two parameters and the current time it is possible to determine the time remaining if the transition is to be preemptively resumed.

4 PEPA Operators

The PEPA process algebra has a simple syntax [5]:

$$P ::= (\alpha, r).P | P + Q | P \bowtie_L Q | P/L | A$$

where the options are the operators for prefix, choice, cooperation, hiding and constant respectively.

The term r represents the timing behaviour of the system. When the activity α is enabled the transition to the next state occurs after a time interval Δt drawn from an exponentially distributed duration with mean parameter $1/r$. Extensions to PEPA have been proposed that allow the use of other probability distributions (enter references).

To allow the proposals that follow to be as general as possible it will be assumed that the PEPA syntax allows non-exponential probability distributions although the syntax used for this discussion does not. Once the syntax extension to PEPA to allow such use is made it will then be possible to alter the proposal.

A statechart representation of the prefix operator ($P \stackrel{def}{=} (a, r).Q$) is shown in Figure 3 Note that in all the diagrams that follow, where there is more than one non-concurrent state within another state or sub-state, the start state or history state is omitted for clarity and simplicity.

Figure 3: Prefix

Prefix expressions may be chained in PEPA. Any chained expressions can be reduced to an equivalent unchained expression by introducing pseudo-states. For example, given the expression $P \stackrel{def}{=} (\alpha, r1).(\beta, r2).P$ an equivalent expression is

$$P \stackrel{def}{=} (\alpha, r1).P'$$

$$P' \stackrel{def}{=} (\beta, r2).P$$

A statechart representation of the choice operator $P \stackrel{def}{=} (a, r).Q + (b, s).R$ is shown in Figure 4

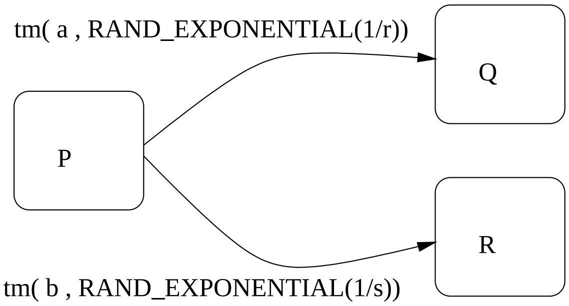

Figure 4: Choice

There are a number of cases when we consider cooperation. If the basic cooperation is $P \stackrel{def}{=} Q \underset{L}{\bowtie} R$ and Λ and Δ are the activity sets of Q and R respectively. $\Xi = \Lambda \cap L$, $\Pi = \Delta \cap L$.

Firstly the cooperation set may be empty $L = \emptyset$ (Figure 5

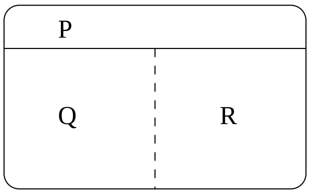

Figure 5: Cooperation - Concurrency

Secondly, the cooperation set may contain elements none of which exist in the activity sets of both cooperating components $L \cap \Lambda = \emptyset$ and $L \cap \Delta = \emptyset$. (This has the same form as the first case.) Both these examples display concurrency.

Thirdly, the cooperation set may contain elements which occur in the activity sets of both cooperating components $L \neq \emptyset$, but there are no passive components (passive components are indicated in PEPA by using the distinguished rate operator $\top$). That is the rate of any activity in $\Xi \cup \Pi$ is not $\top$. If $Q \stackrel{def}{=} (a, r1).Q'$ and $R \stackrel{def}{=} (a, r2).R'$ (Figure 6). This is an active synchronization.

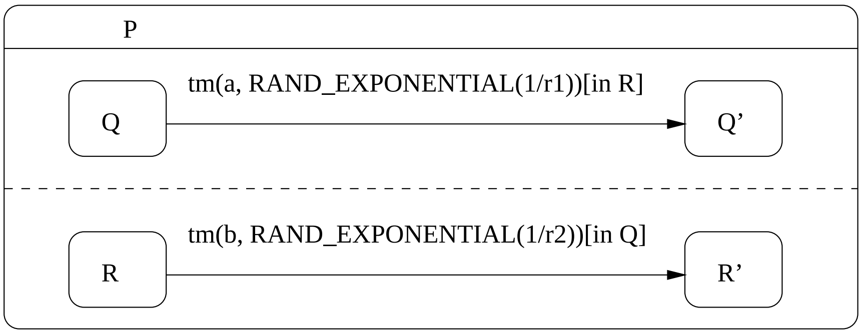

Figure 6: Cooperation - Active

Fourthly, the cooperation set may contain element which occur in the activity sets of both cooperating components $L \neq \emptyset$, but there are passive components. That is there are rates of activities in $\Xi \cap \Pi$ that are $\top$. (see Figure 7) This is a passive synchronization.

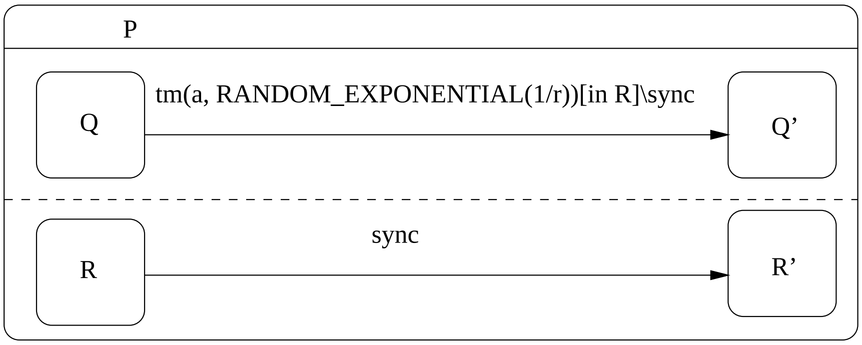

Figure 7: Cooperation - Passive

Alternatively, the *sync* states notation of UML can be used. This is illustrated in Figure 8.

This is an example of the different ways of representing the same conditions within UML. This indicates that for a modelling tool to be useful for com-

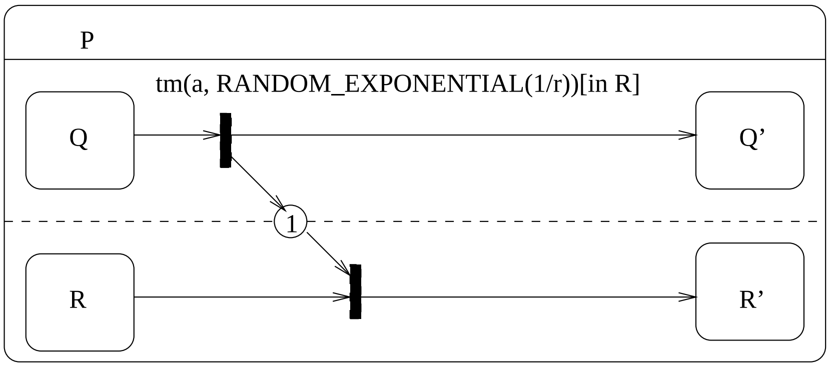

Figure 8: Cooperation - Passive using *sync* states

paring models across paradigms it should possess a canonical form for each modelling paradigm.

However, this simple mapping from PEPA to UML does not consider more complex examples where synchronization of more than two states is considered. This is the subject of the next section.

5 UML model of Processes and Resources

A process and a resource may be represented in the following form in PEPA:

$$Process \stackrel{def}{=} (use, r1).(task, r2).Process$$

$$Resource \stackrel{def}{=} (use, r3).(update, r4).Resource$$

and a system of two processes competing for the same resource can be represented by : $System \stackrel{def}{=} (Process||Process) \underset{use}{\bowtie} Resource$ (where $||$ is the PEPA shorthand expressions for concurrent operations).

If we assume that $r3$ is the distinguished rate $\top$ then the UML equivalent of the PEPA model can be constructed using the techniques of the previous section (see Figure 9). $t1, t2, t4$ are delays produced from the probability distribution function with parameters that are the reciprocals of $r1, r2, r4$ respectively.

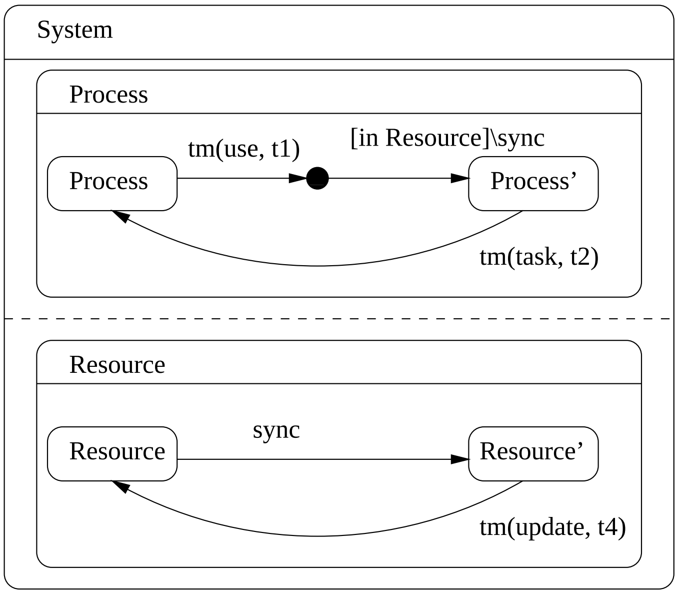

Figure 9: Simple shared resource model

A more complex example where two processes are sharing two resources is also shown (see Figure 10).

This uses a static UML branch, shown by a filled circle along a transition, to indicate that the timed transition is enabled only when any of the guard conditions following the branch are true. Normally, only one of the guards are true. However, in this case each branch may be true if the associated resource is free. The UML behaviour is that the choice of which transition is taken, and which action occurs, will be non-deterministic. This is compatible with the PEPA behaviour.

If there is a *Resource* class with a statechart the same as in the simple example then *Resource1* and *Resource2* can be instances of the class (indicated by the ':' notation in the name). However, if we have a class *Process* it can not be used in this example as there is an extra transition introduced. In general for there will be an additional transition for each resource introduced.

This suggests that syntactical 'sugar' could be introduced into the UML transition syntax that would allow reuse of an unchanged *Process* instance. The definition of this 'sugar' is under consideration.

There are similar questions to be asked about implementation of active synchronizations which are also under consideration.

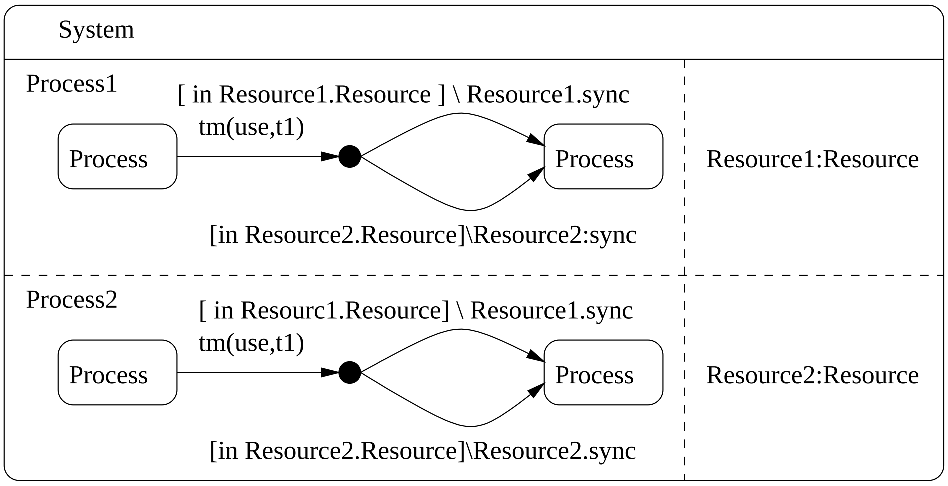

Figure 10: Complex shared resource model

6 Summary

This paper has begun the process of describing how to translate a PEPA process algebra model into an equivalent UML statechart. Particular consideration has been given to the equivalence of the timing models of the two methodologies. To ensure that the resultant UML statechart's simplicity, and thereby understandability, semantic and syntactic extensions are required. However, further work is required to consider the exact nature of these extensions. A reverse translation from UML to PEPA should also be possible as long as the timing model outlined is used.

The UML methodology is designed for complex systems. Complex case studies need to be undertaken to ensure that the translation process scales appropriately. The extensions proposed should ensure reuse of simpler components within complex systems.

The translation outlined has concentrated on the PEPA process algebra but similar translations should be possible for other modelling paradigms. Provided that a common timing model can be created, equivalence between modelling paradigms should be possible and the advantages for finding such equivalences has been considered.

A long-term aim should be to create a UML profile for performability modelling that will ensure that performance and functional characteristics are considered from the beginning of system design.

References

[1] Brian D Bunday. *Basic Queuing Theory*. Edward Arnold, 1986.

[2] Peter G Harrison and Naresh M Patel. *Performance Modelling of Communication Networks and Computer Architectures*. Addison-Wesley, 2nd edition, 1994.

[3] G Ciardo, J Muppala, and K Trivedi. Spnp: Stochastic petri net package. *Proceedings of the 3rd International Workshop on Petri Nets and Performance Models*.

[4] M Ajmone Marsan, G Balbo, G Conte, S Donatelli, and G Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. Wiley, 1995.

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

[6] S Gilmore and J Hillston. Process algebra and performance modeling. *The Computer Journal*, 38, December 1995.

[7] S. Donatelli, J. Hillston, and M. Ribaudo. A comparison of Performance Evaluation Process Algebra and Generalized Stochastic Petri Nets. In *Proc. 6th International Workshop on Petri Nets and Performance Models*, Durham, North Carolina, 1995.

[8] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Language Reference Manual*. Addison Wesley Longman, 1999.

[9] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison Wesley Longman, 1999.

[10] Bruce Powell Douglas. *Real-Time UML*. Addison Wesley Longman, 1998.

[11] Bruce Powell Douglas. *Doing Hard Time*. Addison Wesley Longman, 1999.

[12] R. Pooley. Using UML to derive stochastic process algebra models. In Davies and Bradley [18], pages 23–33.

[13] Rob Pooley and Peter King. The unified modeling language and performance engineering. In *IEE Proceedings - Software*, volume 146, pages 2–10, February 1999.

[14] http://www.rational.com/uml/documentation.html.

[15] David Harel. Statecharts: a visual formalism for complex systems. *Science Computer Programming*, 8:231–274, 1987.

[16] David Harel and Michal Politi. *Modelling Reactive Systems with Statecharts*. McGraw-Hill, 1998.

[17] P.R. Mitton and D.R.W. Holton. Extensions to PEPA to simplify the creation of component libraries. In Davies and Bradley [18], pages 209–218.

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