Using UML to Derive Stochastic Process Algebra Models

Rob Pooley

Abstract

The Unified Modelling Language (UML) is an industry standard object oriented design notation, supported by the Object Management Group (OMG). It is becoming widely adopted in many areas of the software and computer systems development community. It is therefore important to consider how to manage performance analysis of such designs. This short paper shows how simple UML designs can be transformed systematically into process algebra models and so, by suitable annotation, used to provide performance estimates. It then shows how similar analysis to that which produces the continuous time Markov chain from a stochastic process algebra model may be possible for the UML model itself. It also considers whether the generation of a Petri net model is necessary or whether the techniques used in generating the continuous time Markov chain (CTMC) when analysing generalised stochastic Petri nets (GSPNs) can be applied directly to at least some UML models.

1 Introduction

It is widely accepted that performance analysis techniques have suffered a lack of acceptance in the wider software design community. Several reasons have been put forward for this, but the most compelling is the reluctance of designers to learn the specialised formalisms required by queueing network analysis and Markovian numerical solution techniques. Both stochastic Petri nets and stochastic process algebras are attempts to move closer to a designer's world view. Unfortunately, both remain rather academic and appeal to limited communities, such as real time systems designers, where there is an existing interest in more formal approaches.

There has been some success in adapting accepted formal description languages, such as SDL [2, 3] LOTOS [20], by adding timing and probabilistic annotations. The results have been used to derive simulation models [23, 22], queueing network models [1] and HIT models [6, 9].

The work described here is part of a project with a similar approach, but we ask which notation is already most widely accepted or will become so in the near future? The notation chosen is the Unified Modelling Language (UML) [19, 25], since it has been adopted by the industry body, the Object Management Group (OMG), as a draft standard, a significant effort is underway to complete and

*Department of Computing and Electrical Engineering, Heriot-Watt University, E-mail: rjp@cee.hw.ac.uk

refine this draft, several major Computer Aided Software Engineering (CASE) tool manufacturers are committed to supporting UML and many major industrial organisations are adopting UML in their design standards.

In other papers we have explored the possibilities for simulation of UML models [15], generation of simple queueing network models from UML deployment diagrams [18] and, most recently, a possible mapping from UML to stochastic Petri net models (SPNs) [16, 17], more specifically Trivedi's SPNP [5] tool's variant of SPNs. All of these showed the potential for using UML's logical and behavioural notations to define the structure of performance models This paper gives a brief explanation of how similar mappings into stochastic process algebras (SPAs) are possible for simple models.

The rest of this paper is structured as follows. Section 2 develops the UML model of a simple producer-consumer model, focusing on the dynamic and structural aspects of the system. Section 3 explores how the UML model maps into an SPA such as PEPA [10]. Section 5 discusses the generality of the approach and looks at future work to develop this approach.

2 A UML model of a producer-consumer system

Here we develop a UML model which shows how a producer-consumer system might look if based on object oriented design. We do not develop in detail all the possible UML diagrams, concentrating on those that are essential to the objective of describing the structure and behaviour of the system. We do not spend much time on the *use case* model, which is important in real software design projects, since it captures the requirements for the system. We take those very much as given.

2.1 Use case model

The users of the system are assumed to be interested in just one type of use – consume new item. In a realistic application there would be several classes of use.

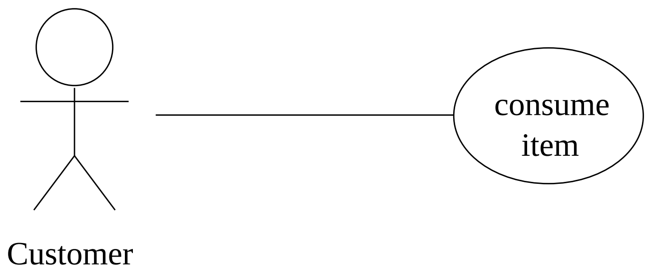
Figure 1 shows a single type of actor, the Customer, and the single use case, consume. An actor defines a rôle adopted by some external person or system. A use case defines a high level task that one or more actors wish to use the system to carry out. In the context of a performance model an actor and its association with a use case can be thought of as representing a workload for the system.

2.2 A class model

A class model defines the essential types of object available to build a system. Each class is described by a rectangle with a name. This can be refined by adding compartments below the name which list the attributes and operations contained in each instance of (object derived from) this class.

Classes are linked by lines known as *associations* which indicate that one of the classes knows about the other. The direction of this knowledge is known as the *navigability* of the association. In an implementation an association typically corresponds to one class having a reference variable of the type of the

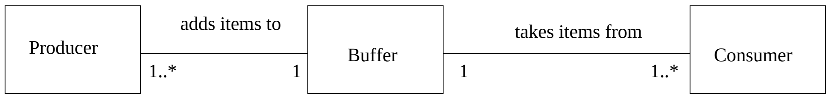
Figure 1: Use case diagram for producer-consumer



other class. Sometimes navigability has to be two way, but it is more often one way. This can be shown by adding arrow heads to the end(s) of the association.

The class model for the example is reduced to its essentials. Figure 2 shows classes for producers, buffers and consumers.

Figure 2: Class diagram for producer-consumer



2.3 Interaction diagrams

UML provides two types of *interaction* diagram – sequence diagrams and collaboration diagrams.

Sequence diagrams Figure 3 is a sequence diagram showing a sub-sequence of messages passing among objects. Such a partial sequence, representing a single possibility selected from those possible, is known as a scenario. Such diagrams are particularly helpful when describing systems where the order of messages is particularly important.

The objects are shown at the top of their *lifelines*. Time is assumed to pass as we move down the diagram. Some objects are created during the interaction and these start lower down, where the message that causes them to be created is sent. Some cease to exist before the end of the interaction shown and their lifelines stop, with a large X, when the deleting message arrives.

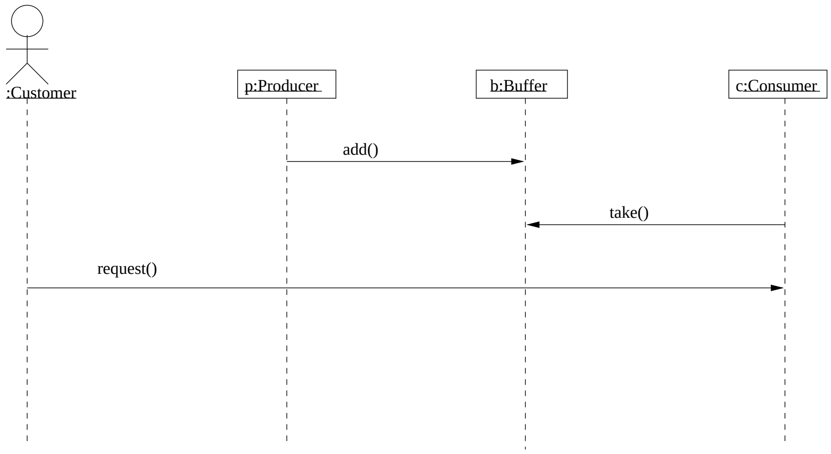
The Customer is a special kind of object, which represents an instance of the actor in the corresponding use case diagram. All objects and actors are instances and so are labelled with a class or actor name prefixed with a colon and, optionally, a name for that instance. For emphasis this whole string is underlined, unlike the name of a class in a class diagram or an actor in a use case diagram.

Messages are sent from one object or actor to another object or actor, to request the carrying out of the corresponding operation. They may have parameters attached and they may bring back result values. The messages here are shown as *asynchronous*, with a half arrow-head. Repeated sequences are shown enclosed in boxes, marked with an asterisk and the condition for the iteration to continue.

This example only shows a small part of the possibilities available in sequence diagrams, which are based on the ITU standard for message sequence charts (MSCs) [12]. They can include conditional behaviour, synchronous and flat messages and much else.

In some earlier work with Carina Kabajunga we explored the use of sequence diagrams in simulation modelling [15, 14] and the Permabase project, Funded by BT Laboatories at the University of Kent has made use of them in deriving simulation models [24]. There has also been work by researchers at Carleton University on the derivation of layered queueing network models from sequence diagrams [11].

Figure 3: Sequence diagram for production followed by consumption



Collaboration diagrams Collaborations are collections of objects, linked to show the relevant associations between their classes. The sequences of messages shown in sequence diagrams can then be super-imposed on these, with their arrows following the links. Wherever a message is to be sent, there must be a link and, by implication, an association must exist between the classes. Furthermore, the association must be navigable in the required direction.

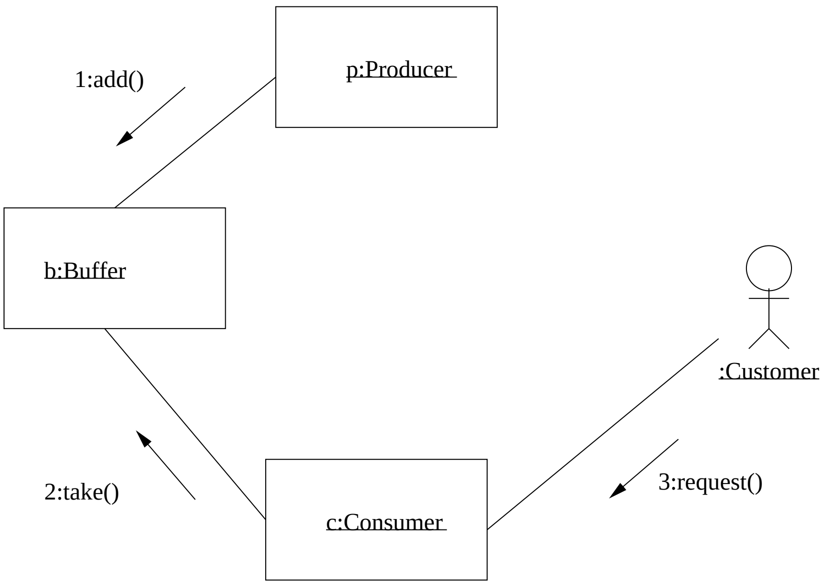
2.4 State diagrams

State charts or state diagrams are based on Harel's statechart notation [7]. They define the internal behaviour possible for an instance of a class in terms of:

states which are shown as lozenges with names and which represent a possible situation for an object of that class; the initial state of an object is shown as a black filled circle; the terminating state is a white filled circle with a smaller black filled circle inside it.

transitions which are shown as arrows between states, usually labelled with a trigger and, optionally, an action sequence;

Figure 4: Collaboration diagram for producer-consumer



triggers which represent the reason for an object to leave one state and follow the corresponding transition to another state; typically a trigger is an incoming message – shown in the usual way for messages, a change in a condition – shown as then word *when* followed by the condition enclosed in brackets – or the elapsing of time - shown by the word *after* followed by the duration in brackets;

actions which are prefixed with a forward slash and represent any actions resulting from a trigger and carried out before entering the new state; some actions involve sending messages and these are prefixed by a caret; the name of the object to which a message is sent is written in front of the message, separated from it by a dot.

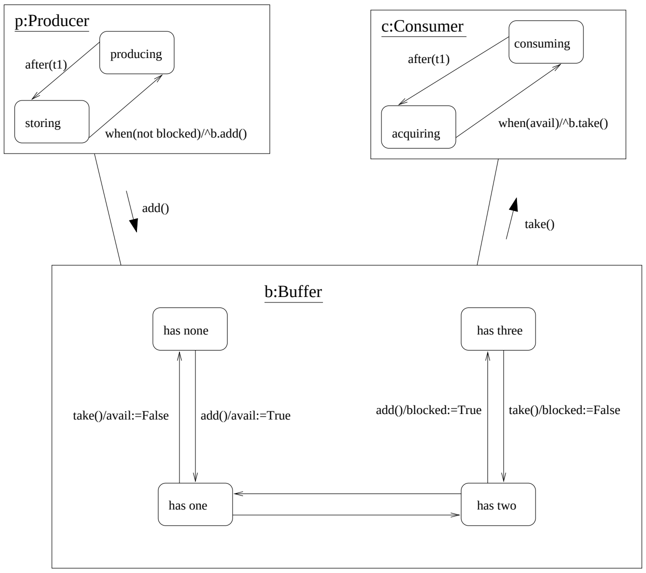
2.5 Combining collaborations and statecharts

Collaborations describe the way that objects interact externally. Statecharts describe how instances of classes behave internally, particularly in response to exrternal stimuli. In a complete design they should provide between them a full description of how the system works.

In some earlier work [8], this was developed into a system for exploring the behaviour of a system, by simulating this behaviour and providing an animated display, showing which states were active and which messages were passed when certain external stimuli were provided.

We now provide such a picture of the producer-consumer system, by inserting the statechart into each object in the collaboration. At any point in the lifetime of this system, each object must be in one, and only one, of its internal states. Each time a message is passed, it may cause a state change in the receiving object and this may cause a further message to be passed between that object

Figure 5: Combined collaboration/state diagram for producer-consumer



and another with which it has an association. The execution of the system can be viewed as the unravelling of this sequence, conditioned on the values of any input data.

The overall state of a system will be the combination of all the current internal states of its objects, plus the current values of any relevant attributes which determine the value of any when triggers plus the time currently elapsed in any after triggers, plus any messages currently sent and not responded to. Readers familiar with petri nets may sense a certainly similarity to the concept of marking and enabling in a timed net.

3 Generating a process algebra model

Generation of a process algebra model from the UML model concentrates on the combined collaboration and statechart diagram shown in Figure 5.

We begin by constructing a PEPA model of each object. The Producer has two states and we model each as an agent, calling them Producer and Blocked respectively.

The transition from Producer to Blocked requires the completion of the *produce* action, which takes time *t1*. In PEPA terms this translates to a rate of $i/t1$. The transition from Blocked to Producer requires completion of the passive action *add*. This is specified as passive so that the rate with which it is actually carried out is totally determined by the Buffer.

$$Producer \stackrel{def}{=} (produce, 1/t1).Blocked \quad (1)$$

$$Blocked \stackrel{def}{=} (add, \top).Producer \quad (2)$$

The Consumer is very similar.

$$
\begin{aligned}
Consumer &\stackrel{def}{=} (consume, 1/t2).Halted &(3)\\
Halted &\stackrel{def}{=} (take, \top).Consumer &(4)
\end{aligned}
$$

The Buffer is a simple counting process, which will not allow a *take* action if it has zero items and will not allow an *add* if it has three. It defines rates for both of these actions. The decision to define rates for *add* and *take* only in the buffer is arbitrary and in a real example they might be defined in the other processes as well or instead.

$$
\begin{aligned}
Buffer_0 &\stackrel{def}{=} (add, 1/t3).Buffer_1 &(5)\\
Buffer_1 &\stackrel{def}{=} (add, 1/t3).Buffer_2 \;+\; (take, 1/t4).Buffer_0 &(6)\\
Buffer_2 &\stackrel{def}{=} (add, 1/t3).Buffer_3 \;+\; (take, 1/t4).Buffer_1 &(7)\\
Buffer_3 &\stackrel{def}{=} (take, 1/t4).Buffer_2 &(8)\\
& &(9)
\end{aligned}
$$

To complete the model we must combine the separate processes, using the PEPA cooperation primitive. We build our model by first combining the Producer with the Buffer, forcing them to cooperate in the *add* action. PEPA's normal rule applies in determining the overall rate for this action as the minimum of the separate rates. We then combine this composite process with the Consumer, forcing cooperation in the *take* action.

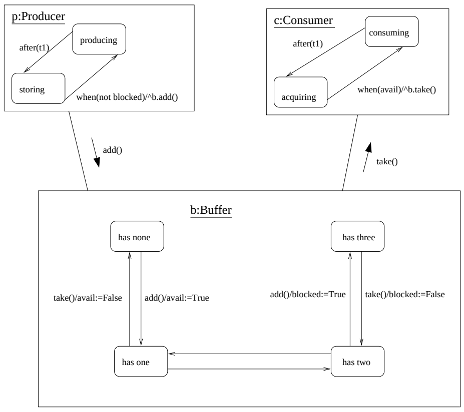
$$
ProdCons \stackrel{def}{=} (Producer \underset{add}{\bowtie} Buffer) \underset{take}{\bowtie} Consumer \qquad (10)
$$

It is important to note that the model of synchronisation used works because the messages which invoke the *add* and *take* operations are synchronous. In other words, once the Producer enters the *blocked* state and sends its *add* message, it cannot proceed until that message has been received by the Buffer. The direction of message passing implies that it is the Buffer which supports the operations *add()* and *take()*, which is a partial justification of our decision to define their rates entirely in terms of the capabilities of the Buffer. UML has two other predefined types of message - asynchronous and flat. These would need to have an appropriate means of interaction defined in PEPA before we could use them in this way.

4 Direct generation of a continuous time Markov chain

We mentioned above that it ought to be possible to go directly from the UML collaboration/state diagram to the underlying Markov chain. A full proof that this is generally possible remains to be shown. Here we show a simple producer-consumer model, where there is a buffer with capacity 1 between the producer

Figure 6: State diagrams collaborating for producer consumer model



and the consumer. This results in the behaviour shown in Figure 6. The Producer is blocked until the Consumer takes its product. The Consumer is blocked until the Producer makes its product available. Each then spends time, either producing or consuming.

At any time this system must have each of its three component objects in exactly one of its states. We call these combinations "markings" to reflect their similarity to Petri net markings. If we assume an initial marking, such as the producer producing, the buffer has none and the consumer acquiring, we can derive all the reachable markings, by following through the interactions which result. Figure 7 shows this for the example.

If we now construct a directed graph with each of these markings as a node and each edge labelled with the trigger for the change, we have a state transition diagram for the system as a whole. In this example, shown in Figure 8, we find some of these triggers are durations. If we transform these into rates, and define rates for the other triggers caused by messages, we have our Markov chain. Clearly this is only a tentative beginning to direct generation.

5 Outcome and discussion

In this short paper we have considered the possibility of mapping annotated UML models into a stochastic process algebra. For a very simple example we have built a PEPA model which can be solved to get estimates for its performance. This is very encouraging, but we must keep a careful rein on our expectations. The subset of UML features used is very restricted and fits well with PEPA. More realistic UML designs would be unlikely to be this straightforward and considerable further work is needed to develop anything approaching a robust method for direct analysis of UML models in this way.

Nevertheless, we can be encouraged by this example to investigate the approach more fully. We have also begun work on direct generation of underlying Markov chains for UML models. We believe there is every chance of constructing a flexible, multi-solver performance back-end for UML computer aided software

Figure 7: Direct "marking" of UML states in collaboration

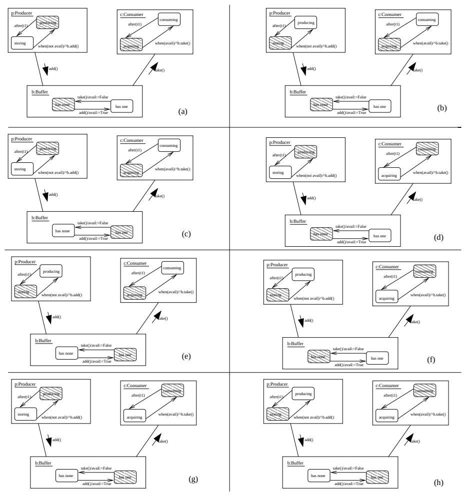
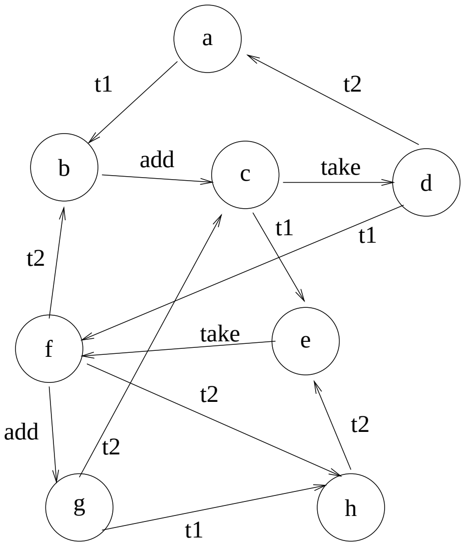


Figure 8: State diagram for reachable combinations



engineering (CASE) tools. Process algebra techniques will add to this by providing insights into compositionality of models. They may also give results in the analysis of behavioural properties of these designs.

References

[1] A. Benzekri A. Valderruten, O. Hjiej and D. Gazal. Deriving queueing networks performance models from annotated LOTOS specifications. In R. Pooley and J. Hillston, editors, *Computer Performance Evaluation - Modelling Techniques and Tools, 6th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, number 10 in Edits, pages 120–130. Edinburgh University Press, September 1992. Edinburgh.

[2] F. Bause and P. Buchholz. Protocol analysis using a timed version of SDL. In *Proceedings of the 3rd Int. Conf. on Formal Description Techniques (FORTE '90).* Springer, 1991.

[3] F. Bause and P. Buchholz. Protocol analysis using a timed version of SDL. In *Proceedings of the 3rd Int. Conf. on Formal Description Techniques (FORTE '90).* Springer, 1991.

[4] Grady Booch. *Object Oriented Analysis and Design with Applications.* Benjamin-Cummings, 2nd edition, 1994.

[5] G. Ciardo, J. Muppala, and K. Trivedi. Spnp: Stochastic petri net package. In *Proceedings of 3rd International Workshop on Petri Nets and Performance*, pages 142–151, Kyoto, Japan, 1989.

[6] D. Hogrefe E. Heck and B. Möller-Clostermann. Hierarchical performance evaluation based on formally specified communication protocols. *IEEE Trans. Comp.*, 40(4):500–513, 1991.

[7] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[8] Steven Hargreaves. Simulating UML models. Master's thesis, Department of Computer Science, University of Edinburgh, 1998.

[9] E. Heck. SDL-HIT integration. In Konlof, editor, *Method Integration: Concepts and Case Studies.* Wiley, 1992.

[10] Jane Hillston. *A Compositional Approach to Performance Modelling.* PhD thesis, Computer Science, University of Edinburgh, 1995.

[11] C. Hrischuk, J. Rolia, and C.M. Woodside. Automatic generation of a software performance model using an object orient ed prototype. In *Proceedings MASCOTS '95*, pages 399–409, 1995.

[12] ITU. *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC).* ITU-TS, Geneva, 1996.

[13] I. Jacobson, M. Christenson, P. Jonsson, and G. Overgaard. *Object Oriented Software Engineering: A Use Case Driven Approach.* Addison-Wesley, 1992.

[14] Carina Kabajunga. Support for the unified modelling language design paradigm. Master's thesis, Computer Science, University of Edinburgh, 1997.

[15] Carina Kabajunga and Rob Pooley. Simulating UML sequence diagrams. In Rob Pooley and Nigel Thomas, editors, *UK PEW 1998*, pages 198–207. UK Performance Engineering Workshop, July 1998.

[16] P.J.B. King and R.J. Pooley. Estimating the performance of UML models using Petri nets. *Computer Communication*, Submitted for publication.

[17] P.J.B. King and R.J. Pooley. Using UML to derive stochastic Petri net models. In *PNPM '99*, Submitted for publication.

[18] R.J. Pooley and P.J.B. King. The Unified Modeling Language and performance engineering. *IEE Proceedings - Software*, 146(2), March 1999. To appear.

[19] Rob Pooley and Perdita Stevens. *Component Based Software Engineering with UML*. Addison-Wesley, November 1998.

[20] N. Rico and G.v. Bochmann. Performance description and analysis for distributed systems using a variant of LOTOS. In *10th Int. IFIP Symposium on Protocol Specification, Testing and Validation*, July 1990.

[21] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorenson. *Object oriented Modelling and Design*. Prentice-Hall, 1991.

[22] N.M. Macintyre S.R. Kershaw and N. Xenios. A toolset to support the performance evaluation of systems described in SDL. In G. Haring and G. Kotsis, editors, *Computer Performance Evaluation - Modelling Techniques and Tools, 7th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 23–26, Vienna, May 1994. Springer-Verlag. Vienna.

[23] A. Valderruten, O. Hjiej, A. Benzekri, and D. Gazal. Deriving queueing networks performance models from annotated LOTOS specifications. In R. Pooley and J. Hillston, editors, *Computer Performance Evaluation - Modelling Techniques and Tools, 6th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, number 10 in Edits, pages 120–130, Edinburgh, September 1992. Edinburgh University Press. Edinburgh.

[24] G. Waters, P. Linington, D. Akehurst, and A. Symes. Communications software performance prediction. In Demetres Kouvatsos, editor, *13th UK Workshop on Performance Engineering of Computer and Telecommunication Systems*, pages 38/1–38/9, Ilkley, West Yorkshire, July 1997. BCS Performance Engineering Specialist Group.

[25] Martin Fowler with Kendall Scott. *UML Distilled*. Number ISBN 0-201-32563-2. Addison-Wesley, 1997.