# The Unified Modeling Language and Performance Engineering

Rob Pooley and Peter King
Department of Computing and Electrical Engineering
Heriot-Watt University
Riccarton
Edinburgh EH14 4AS
Scotland
rjp@cee.hw.ac.uk

**Abstract**

The use of performance analysis and prediction techniques by software designers and software engineers is at best inconsistent and at worst simply does not happen. This is principally because these techniques are seen as separate and difficult to apply. Work on Software Performance Engineering, initiated by Smith, has sought to bridge the gap, but has had limited success. With the emergence of a widely accepted standard for software design – the Unified Modeling Language – the time seems ripe for an attempt to integrate performance directly into this process, by exploiting the notation directly. This paper reviews past work in Software Performance Engineering, summarizes the Unified Modeling Language and presents encouraging result from merging these two techniques.

## 1 Motivation

To encourage designers to incorporate performance analysis, work such as that by Smith on Software Performance Engineering [25] and by Beilner and others on HIT[4] has built new performance evaluation frameworks aimed at designers. At the same time others have added performance annotations to existing design notations and provided routes for analyzing the resulting models. Most success is likely to come from those projects which attempt to build performance analysis directly into accepted design methods. The integration of performance estimation with the system design process is the goal of the work described in this paper.

The emergence of a widely accepted new standard for object oriented software design, known as the Unified Modeling Language (UML) [23, 28] has made it possible to focus on a single notation. Since that notation includes both static and dynamic aspects of systems it is very well suited to generating performance results, although additional information on timings and branching probabilities is needed. Here we consider some examples of using UML for performance prediction.

The rest of this paper is structured as follows.

In section 1 previous work in performance engineering is reviewed Section 2 outlines the UML itself, concentrating mostly on the dynamic modelling aspects of the language. Section 3 considers how to exploit the UML for performance analysis. This includes examples of direct use of UML within Section 4 considers where this work might lead and the benefits that might follow if it is successful.

# 2 Past work in software performance engineering

Connie Smith [25] first used the term "performance engineering" to describe the application of performance evaluation techniques to software systems. She presented the case for an approach which brought performance evaluation and design together. For non-specialists, the field of performance analysis is often either seen as the nuts-and-bolts problems of benchmarking and capacity planning or as the arcane theories of queueing analysis and stochastic modelling. Smith's thesis is that it can and should be an integral part of practical software engineering. Indeed she goes further and claims that without performance methods software engineering is incomplete.

Despite fairly general agreement that Smith's analysis is correct, no common approach has emerged.

## 2.1 Past approaches 1 - Creating new formalisms

One group of projects started by building performance analysis frameworks aimed at software engineers (or more generally at computer system engineers, where software is a vital aspect of the systems). This approach could be defended in the situation where designers had not settled on a general approach. However, they have not been widely adopted in a world where more mature system design techniques have become more common.

Here we consider two excellent examples of this approach.

**Software Performance Engineering** This was Smith's original methodology, where the emphasis is on formulating models in an appropriate form and capturing the desired aspects of behaviour. The question is not so much how far it succeeds in these objectives, but how likely it was to be adopted by practising software engineers. It was not presented as part of any existing design approach, although clear ideas on how to incorporate it into standard methodologies were given. Thus at the level of the practising software engineer, it has considerable merit.

The only problem is that the underlying modelling concept, the Information Processing Graph (IPG), is not already in use by software engineers.

**HIT** HIT [4, 11] was specifically built to support modelling of computer systems in a hierarchical manner, based on a layered machine view of such systems. This allows modularization of models, corresponding to components within layers of the real system. Such a view gives a form of description which is very natural for the types of systems considered.

HIT deploys a range of analytic and simulation solution techniques, without the user needing to perform translations into separate modelling languages. Although HIT was developed as a stand-alone formalism, it shows that performance evaluation of a sophisticated kind is possible starting from such a view of design.

## 2.2 Past approaches 2 - Extending formal notations

Several workers report on combining performance modelling with existing design formalisms such as SDL, LOTOS and RM-ODP.These have shown the feasibility of such approaches.

**SDL** SDL is a widely used specification language, particularly for real time systems [5].

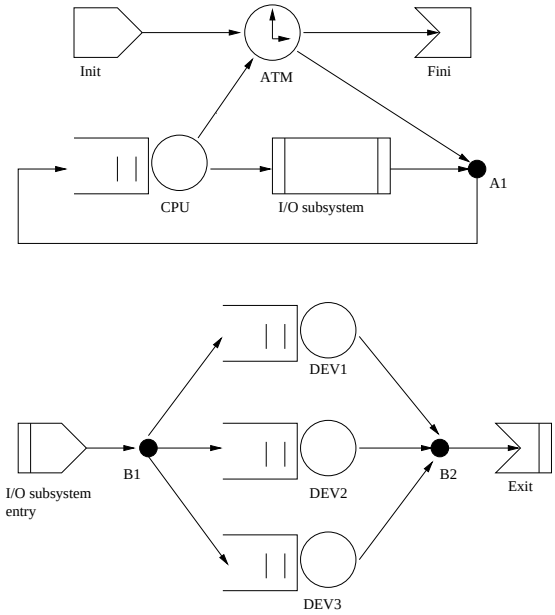
1. At Universität Erlangen-Nürnberg queueing network analysis of SDL models was shown to be feasible.

2. Work at BNR Europe [26], showed that it was possible to generate simulation models from SDL specifications, in a form suitable for solution by a simulation package.

3. Work on the use of SDL as a basis for integrating behavioural and performance modelling is also underway at Universität Dortmund [3, 8].

   This includes its integration within the framework of the HIT toolset [15].

Figure 1: An IPG for SPE




**From C.U. Smith "Performance Engineering of Software Systems", p241**

**LOTOS** LOTOS [6] is the CCITT recommended protocol specification language, based on the principles of process algebras, combining features of Milner's Calculus of Communicating Systems (CCS)[21] and Hoare's Communicating Sequential Processes (CSP) [16].

1. Stochastic LOTOS [24], which modifies the semantics of LOTOS to allow performance information to be represented and performance results computed. Useful results have been demonstrated, but the formal power of the specification is lost.

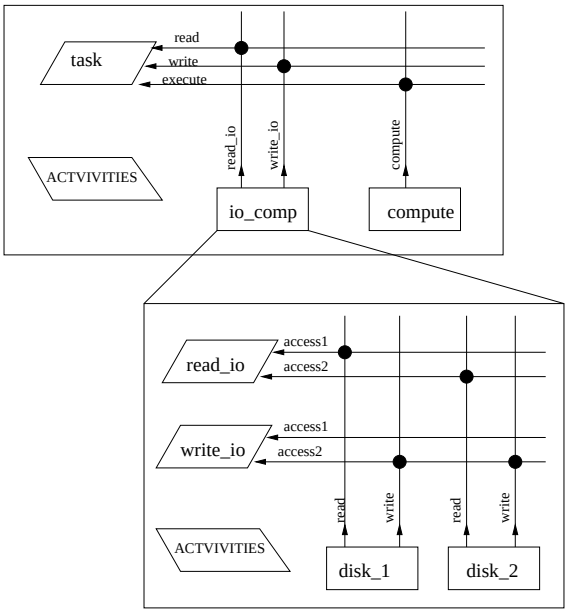
2. ESPRIT project COMPLEMENT used annotated LOTOS specifications [1]. Modelling starts with standard LOTOS specifications, so the formal semantics can be used, and then adds performance annotations before generating QNAP code. Rules for this derivation were proposed and results shown.

**RM-ODP** In Hills, Rolia and Serazzi's work [9] they use the ISO Reference Model for Distributed Processing (RM-ODP)[18] as a starting point in building performance models for Distributed Software Process Architectures.

## 2.3 The present

In the last two years, the software design community has embraced a move towards a new modelling language, intended to provide a common vocabulary for software based systems. This Unified Modelling Language (UML) [22] is gaining widespread acceptance. It is important for performance engineers to consider how it might serve their ends and how to try to influence the notation's definition to better support performance considerations.

Figure 2: A HIT diagram




# 3 Object oriented performance engineering

Recently Smith and Williams have presented ideas on how to incorporate Performance Engineering into the object oriented software design process [27]. This has developed into a use case driven approach, using concepts now incorporated into the UML, such as use cases and sequence diagrams. The design is still translated into IPGs before the model is solved, however.

Smith's original ideas remain influential, however, and this new work is also important. It continues to influence the development of the ideas presented in this paper. Here we look more directly at exploiting the UML, however.

# 4 The Unified Modelling Language

The Unified Modelling Language (UML) [22] is a graphically based notation, which is being developed by the Object Management Group as a standard means of describing software oriented designs. It contains several different types of diagram, which allow different aspects and properties of a system design to be expressed. Diagrams must be supplemented by textual and other descriptions to produce complete models. For example, a use case is really the description of what lies inside the ovals of a use case diagram, rather than just the diagram itself.

For a fuller account of UML see [23, 28].

## 4.1 Use case diagrams

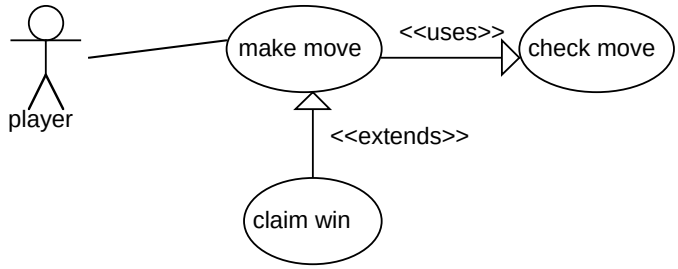
In UML the most radical addition to the first generation of object oriented modelling approaches is the *use case diagram*. This shows a system in terms of the external users of the system, known as actors and usually shown as "stick people". An actor may be a system, not necessarily a person.

Each actor is shown as participating in one or more use cases, shown as ovals linked to the participating actors.

A use case is some high level activity or capability of the system.

Use case diagrams are significant in showing external systems and users that initiate functions.

Figure 3: A use case diagram




### 4.2 Static design diagrams

UML is an object oriented design formalism. Thus the core of the language is the *class diagram*, where the classes of object which form the building blocks of systems are described. These will be familiar to anyone who has used object oriented methods, with each class having

**a name** , possibly accompanied by a stereotype and/or a set of properties for this class.

**a set of attributes** – representing internal data within objects

**a set of operations** – representing methods or functions supported by objects of that class.

These are shown in three compartments within the basic class rectangle. Only the name is compulsory.

## 5 Class relationships

Important relationships between classes of objects are shown by

**associations,** which indicate that these classes are linked by a particular aspect of the design; associations are very general; composition and aggregation are special forms of association,

**dependencies,** which show that one class cannot function without something provided by another,

**refinements,** a form of specialisation, where one thing is a more detailed version of another,

**generalisations,** one class is a generalisation (super class), the other is a specialisation (sub class).

For the purposes of this paper, we merely assume that classes and objects exist as the fundamental units of description within a design. In particular, classes encapsulate behaviour, which can be described by a state machine description.

### 5.1 Packages

Collections of classes and other model elements may be grouped into *packages*, representing modules or libraries. These are not considered in detail here, but they might be exploited in partitioning models, possibly according to coupling of objects within systems.

Figure 4: Class diagram

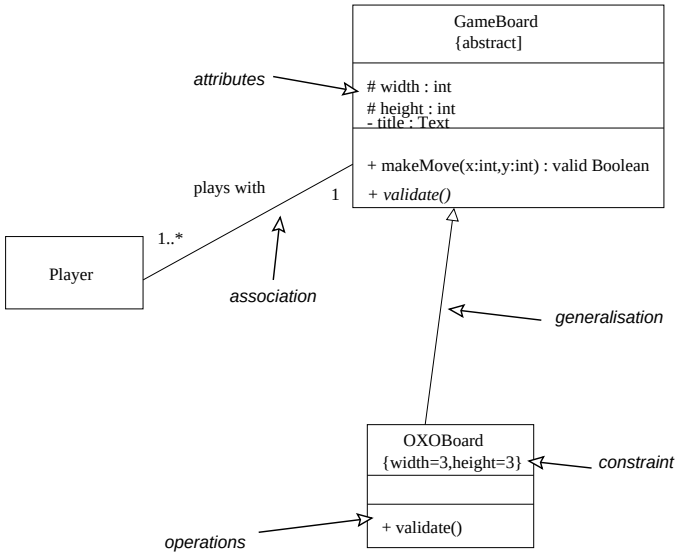
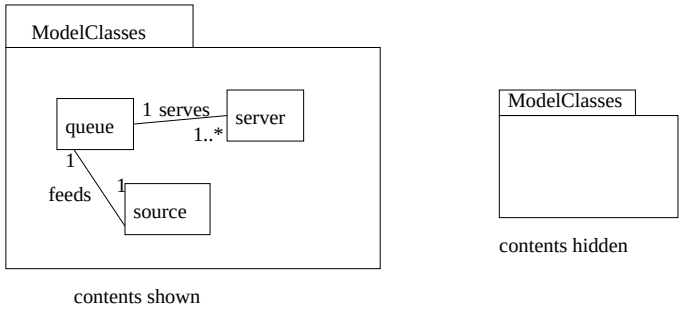



Figure 5: Package diagrams




## 5.2 Interaction diagrams

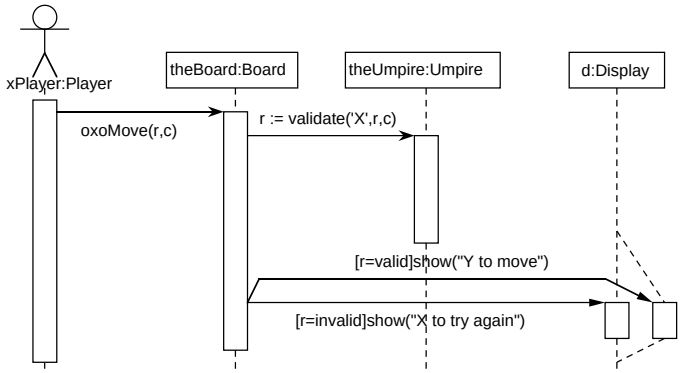
UML defines two forms of diagram to represent the way that messages are passed between objects in a system.

These two forms of interaction diagram can be generated automatically from each other in the CASE tools which are available to support UML.

**Sequence diagrams** Sequence diagrams are based on message sequence charts [17]. Objects are shown as boxes with dashed lines extending vertically below them. As these lines move down the page, they represent the passing of time. Arrows across the page show the sequences of messages passed between objects as time passes. Periods of activity by an object may be shown as a bar overlying its dashed life line.

Each sequence diagram represents one or more routes through the unfolding of a use case (high level) or of an operation in a class (low level). If a single route is shown, one particular set of conditions is being assumed. Such a set of conditions is termed a *scenario*. Conditional behaviour, represented by placing a guard such as `[r=invalid]` on a message, can be used

Figure 6: A synchronous sequence diagram




to show more than one possibility or even all possibilities, but this may be very complicated to represent.

Figure 7: An asynchronous sequence diagram

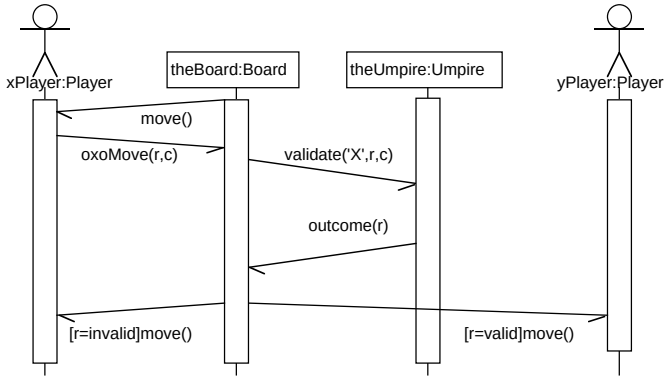



Figure 7 shows asynchronous message passing. This represents true concurrency.

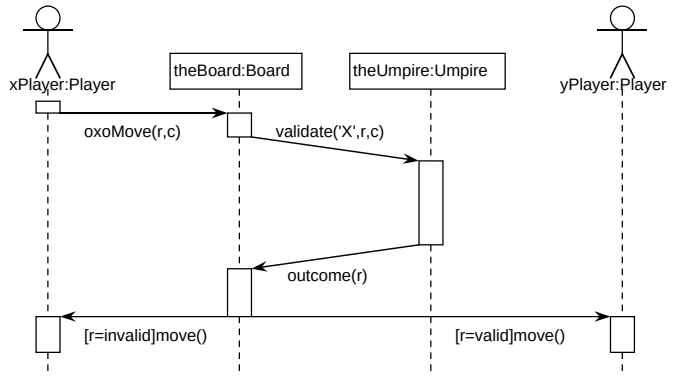
Figure 8 shows flat message passing. When a message is sent, the sender yields control, but, unlike asynchronous messages, has to wait to be restarted before it can proceed. This represents co-routines or lightweight threads.

**Collaboration diagrams** Here time is not represented explicitly. Instead the emphasis is on showing which objects communicate with which others. Messages are numbered to show the order in which they should happen. Nesting can be shown as a *decimal* numbering scheme. Partial orders, representing concurrency, can be shown by using names instead of numbers at the appropriate level.

# 6 State and activity diagrams

UML defines state diagrams, which allow a class to be defined in terms of the states it can be in and the events which cause it to move between states.

Figure 8: A flat sequence diagram




They are essentially Harel statecharts [12]. These are fairly conventional state transition diagrams except for the following:

- a state may indicate that the object is engaged in some activity;
- transitions between states can be due to messages or to changes in certain conditions or to a combination of these;
- states can be nested within super-states.

# 7 Activity diagrams

Activity diagrams are an extension of state diagrams, allowing synchronisations to be shown explicitly. A synchronisation is something like a Petri net transition. All the incoming arcs must be satisfied before progress beyond the synchronisation is permitted.

Activity diagrams have similarities to PERT charts and to flowcharts. They are especially useful in showing workflow type systems.

## 7.1 Implementation diagrams

UML defines two models which describe how your system is implemented.

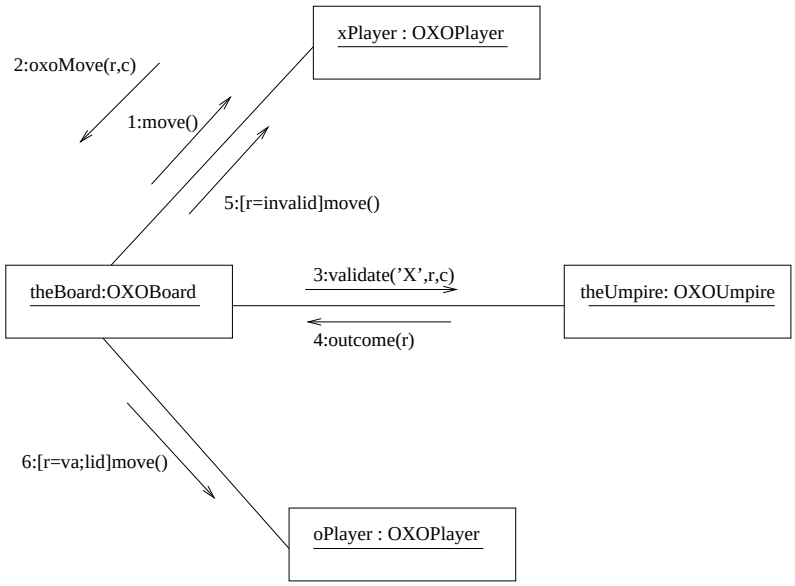
- The *component model* shows the dependencies between parts of the code,
- The *deployment model* shows the structure of the system at run-time: which parts run on which processors and how the hardware is configured to provide necessary resources.

A component diagram can show dependencies of various sorts among such components. The dependency is shown by a dashed arrow between components. Stereotypes may be used to define the type of dependency. The component which the arrow leaves is said to depend on the component to which it goes.

**Components at runtime** In the example there are two examples of service dependencies. In a program running on a single machine these are calling dependencies between functions. Here they are messages being passed or remote method invocations. Component diagrams do not show instances of components, merely the dependencies which apply between all components of one type and all components of another.

Instances are shown in deployment diagrams.

Figure 9: A collaboration diagram




**Deploying components** The deployment diagram shows what runs where. It may simply show the physical machines and devices, but that is only useful if you are building hardware systems without regard to what they will run.

We start by considering the physical system, made up of processors, and devices such as disk controllers which provide services to other hardware elements. Unbroken lines between boxes represent physical connections between machines. (Strictly speaking these are associations, like those in class diagrams.) These may represent cables, local area networks, modems and phone lines or whatever. A link can be given a name and a stereotype, so that it is clear both what sort of link (stereotype) we have and what is the name of a particular link. Further details may be given as properties within the specification of the link.

**Deploying components** When we add the software components, we show how the system is to operate at runtime. We can also show dynamic behaviour, where either data or code can migrate from machine to machine. Data may also be allowed to migrate between components on the same machine.

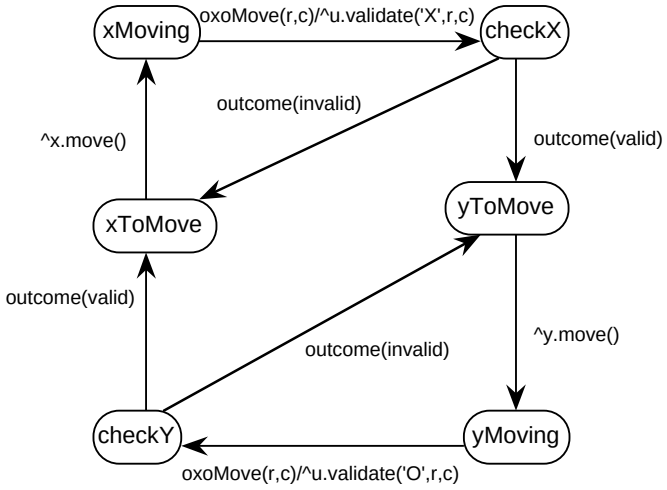
# 8 Using UML for performance

We now consider some ideas for exploiting UML designs for performance modelling. These now include both earlier simulation methods [20] and new queueing network modelling approaches. We start by considering each form of description in turn. We then consider an approach combining different approaches.

## 8.1 Use case and workloads

The actors in use case diagrams should represent all external stimuli to the system being modelled. There is considerable disagreement about which external systems should be shown here, but at least all initiators of activities within the system should be represented. It seems logical, therefore, to use actors as the basis for defining workloads in the system. Each actor does not correspond to

Figure 10: A state diagram




a single person or system, but rather to a (set of) rôle(s) played by one or more people or systems. Thus each actor may represent just one part of the workload for one part of a system.

## 8.2 Exploiting implementation diagrams

Implementation diagrams provide the final mapping onto computing and storage devices. They are essential in defining contention and quantifying resources which are available. One approach is to define a correspondence to an underlying queueing model.

Components in such a model can be mapped onto servers. Links become queues with finite service rates.

We consider two simple examples.

## 8.3 Example 1 - Auto-Teller Machine example

The simple model of an ATM till, shown in Figure 15, can be modelled as an open queueing network, as shown in Figure 16.

## 8.4 Standard elements for mapping to queues

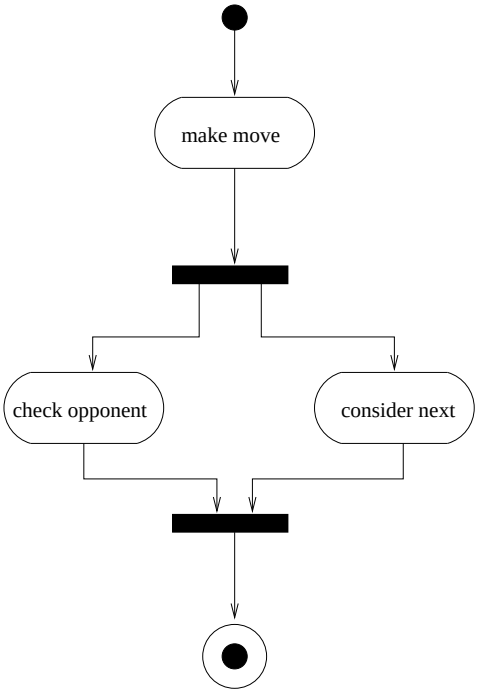
We can identify three simple building blocks for construction of queueing networks.

1. Actors – supplying a workload
2. Components – providing services
3. Links – joining components

The actor can be modelled as a general open network, as shown in figure 17, with:

- Incoming jobs at a defined rate
- Think time before jobs return
- Departing jobs with a given probability;

Figure 11: An activity diagram




or omit returning jobs and remain an open network; or omit incoming and departing jobs and become a closed network if all other elements do the same.

Incoming links from other queues may be as many as required. Outgoing links may also be as many as needed. There will be no completed jobs in a closed network. Repeated jobs are optional.

The link has two flows entering its queue and two leaving its server. This represents half duplex behaviour. Simplex links have one input and one output flow. Full duplex links require two independent queues.

**Multiple Auto Tellers**

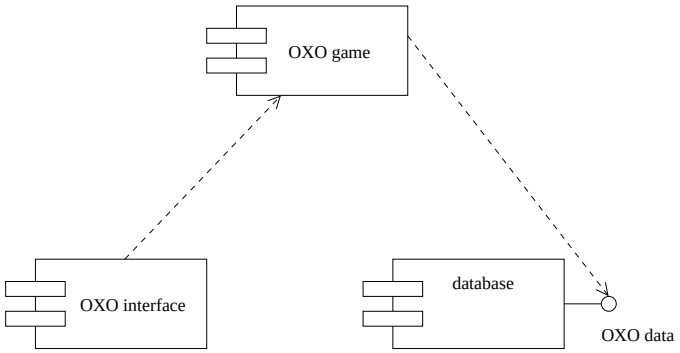
Here we add an extra till to our model. The same building blocks still work.

## 8.5 Solving the Queueing Network Model

Using the building blocks that we have defined above, we can derive a queueing network model of the ATM system. Under appropriate restrictions, most importantly the requirement that service demands at link building blocks are exponentially distributed, the queueing network satisfies the requirements of product form, and the performance can be calculated cheaply[2, 7]. Product form networks have the property that the measures of performance such as mean queue lengths, response times, etc. are insensitive to the higher moments of distributions of service times, and depend only on the mean service times. Furthermore, they do not depend on the order in which service centres are visited nor on the number of times a service centre is visited, but only on the total demand that a customer places on a service centre.

In order to demonstrate the validity of our approach, we construct a queueing network model of the ATM model, using parameters taken from Smith's book[25, Page 188]. The CPU demand of a transaction is 0.00710 seconds, and the ATM interacts with the user 14 times, each time the

Figure 12: A component diagram showing runtime dependencies




user taking on average 1 second to respond (eg to type the next number of the PIN, or to select a transaction). We include a small amount of processing at the ATM itself ( 0.002 seconds per interaction) , and also use of a communications link, taking 0.05 seconds of transmission time in each direction, on average per transaction. It will be seen that even if the system is otherwise completely idle, a transaction will take 14.11 seconds. The main component is the user's think time, which is unaffected by the presence of other users on the CPU.

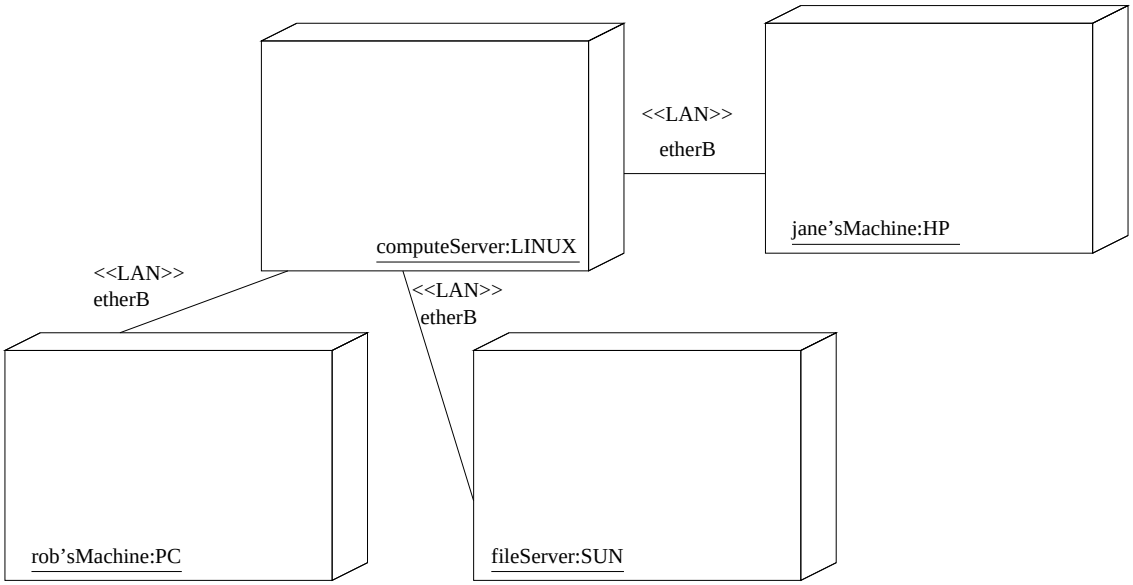
Using this simple model, a quantitative analysis can be done using any of a number of standard queueing network packages. PDQ[10] was used. First, the arrival rate of transactions was varied. The ultimate limit of throughput of the system is 20 jobs per second, representing the communication link being totally saturated. The results are in Table 1. It can be seen that even when the communication links are 95% saturated, the mean response time is only just over 16 seconds. In fact, this model requires careful interpretation, because it allows multiple responses to be outstanding from the ATM, which is clearly nonsense. The best way to interpret it is to examine the mean queue lengths and determine the mean number of jobs in the system. This is approximately 40 in this case, and can be interpreted as representing 40 ATM machines, all sharing a single communications link to the CPU, and each presenting a customer every two seconds.

| Arrival Rate | Response time |
|---|---|
| 10 | 14.2302 |
| 11 | 14.2079 |
| 12 | 14.2580 |
| 13 | 14.2938 |
| 14 | 14.3415 |
| 15 | 14.4082 |
| 16 | 14.5083 |
| 17 | 14.6750 |
| 18 | 15.0084 |
| 19 | 16.0085 |

Table 1: Response time as function of arrival rate

It is clearly unrealistic for that many ATMs to share a single communications link, so we concentrate on arrival rates which correspond to 1 customer arriving every 15 seconds or more. Although our model gives answers for higher arrival rates, they are not feasible in reality, representing multiple transactions sharing the same ATM in an interleaved fashion. Although it is not essential, we assume that there is only a single ATM attached to the communication link. In

Figure 13: Just the hardware




order to give some idea of the load that other users will be placing on the system, a workload at the CPU representing all the other ATMs was used. No modeling of how this workload reached the CPU was done, so it could in fact represent any form of background load. The domination of the user's think time over all other components of the response time can be seen, because even when new customers arrive at the ATM every 16.67 seconds, and there are 2,200 ATMs being serviced, the mean response time to a customer is still 14.21 seconds. This corresponds to a CPU utilization of 93%. The results are seen in Table 2.

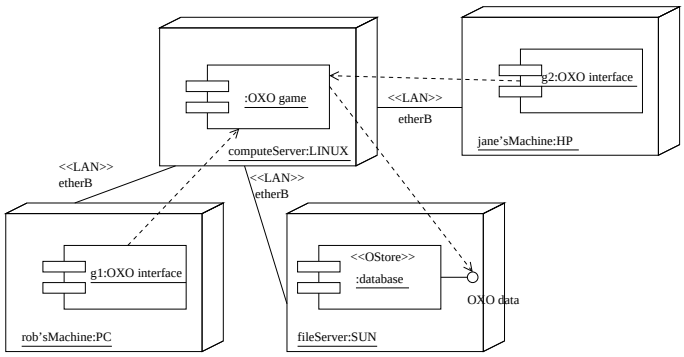
| ATMs | Response Time | CPU Utilization |
|---|---|---|
| 400 | 14.11 | 17 |
| 600 | 14.11 | 25 |
| 800 | 14.11 | 34 |
| 1000 | 14.11 | 43 |
| 1200 | 14.12 | 51 |
| 1400 | 14.12 | 60 |
| 1600 | 14.12 | 68 |
| 1800 | 14.13 | 77 |
| 2000 | 14.15 | 85 |
| 2200 | 14.21 | 94 |

Table 2: Response time versus no of ATMs Inter arrival time 16.67 secs)

If we now consider the ATM and query server as a single entity giving exponentially distributed service, then the system can be viewed as an $M/M/1$ queue. The response time of the ATM/CPU system is treated as the service time of the queue of customers which are waiting to use the ATM, then the queue of customers will contain on average 5.5 customers if they arrive on average once every 16.67 seconds.

Although the values used in this example were to a certain extent guessed, they certainly indicate that the CPU will be able to service a large number of ATM machines without the users noticing a significant degradation in service. More complex models could include the effect of disk

Figure 14: A deployment diagram with the software




I/O at the query server, and lead to more accurate predictions of performance.

## 8.6 Direct simulation of sequence diagrams

The general approach being adopted in our work is that we wish to use UML designs as directly as possible in performance evaluation. We might identify sequence diagrams as having the potential to generate and display useful information relating to performance. As already noted, time is represented in sequence diagrams by distance downwards vertically. This is usually just an intuitive guide to timing issues, allowing orderings to be seen quickly in straightforward cases. There are, however, many designs, notably those involving iteration within sequential behaviour, where this breaks down. We have decided that sequence diagrams are, therefore, more suited to being a display format than to being a detailed behavioural specification format.

### Sequence diagrams as traces

Timing information can be added by labelling messages with relative timing constraints, using nested numbering of messages to describe exact orderings. The result is still only really useful as a documentation aid in all but simple cases. Our initial prototype simulation tool [19] encoded the information about time intervals between messages, along with object names, in a driver file. This allowed for probabilistic timings, such as negative exponential inter-arrival time distributions. It was then used to drive a simple discrete event simulator, which generated an animated sequence diagram as a trace of events, as shown in Figure 21.

Random timings allowed realistic performance prediction and testing of assumptions about the possible ordering of messages.

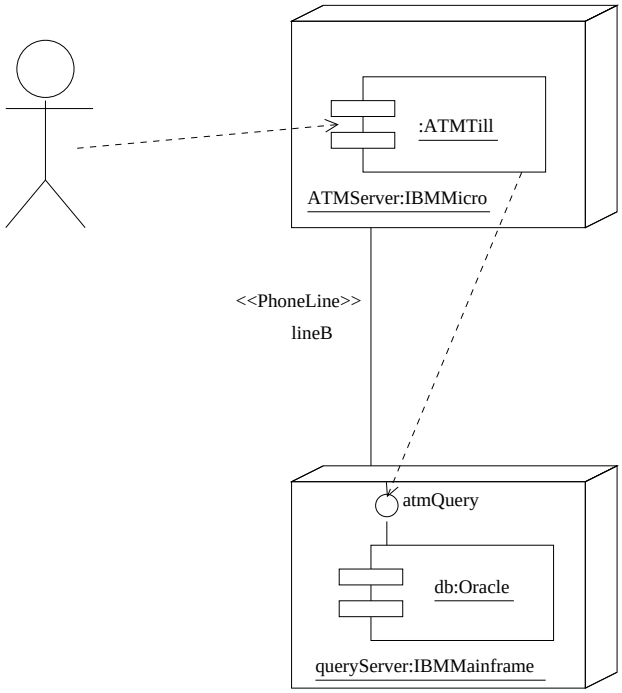
## 8.7 Exploiting collaboration diagrams

Collaboration diagrams may represent a useful complement to sequence diagrams. They are particularly useful in allowing animation of the behaviour of long running models, where the interest is on average, steady state behaviour rather than short traces of activity. They are also similar, at a more detailed level, to deployment diagrams. Similar mappings onto queueing models may be defined. They may also be combined with state diagrams as described below.

## 8.8 Exploiting state diagrams

State diagrams offer a link directly to Markovian modelling. This is very tempting, but may prove more difficult in practice, since it is not clear that transition rates are easy to obtain for systems. It is an area still requiring a lot of work.

Figure 15: Deployment diagram of a simple Auto-Teller Machine




### 8.9 Direct simulation of UML models

The combination of collaboration diagrams with state diagrams, by showing state machines within objects is the most practical basis for simulation modelling using UML. A simple example of such combination is shown in figure 22. A prototype class library in Java, to simulate such diagrams, has been built [13].

### 8.10 Exploiting activity diagrams

Activity diagrams will look vaguely familiar to anyone who has used Petri nets. They are certainly capable of expressing concurrent behaviour with explicit synchronisation. Work is currently underway to evaluate their usefulness for performance modelling and their relationship to Petri nets. They may also have a relationship to process algebras.

## 9 The future

This paper has presented the UML in terms of its individual diagram types and their potential for use in performance analysis. Since many software engineers are already adopting the UML, it seems pointless to object that it is not optimal for our purposes. Instead, we must seek to adapt our methods to its structure and the information it can give us.

### 9.1 Progress

Some preliminary work has shown that there are many aspects of UML which are quite suitable for performance modelling. In particular we have:

- found a mapping from deployment diagrams to queueing models;

Figure 16: Open queueing model of Teller Machine

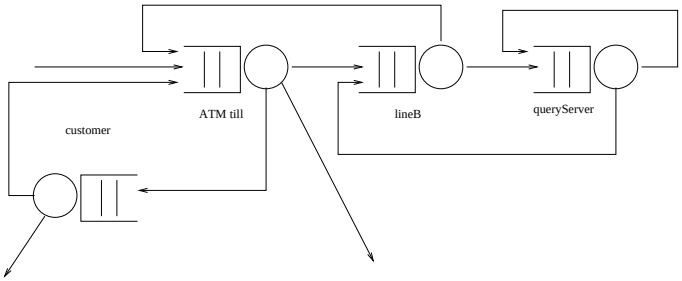
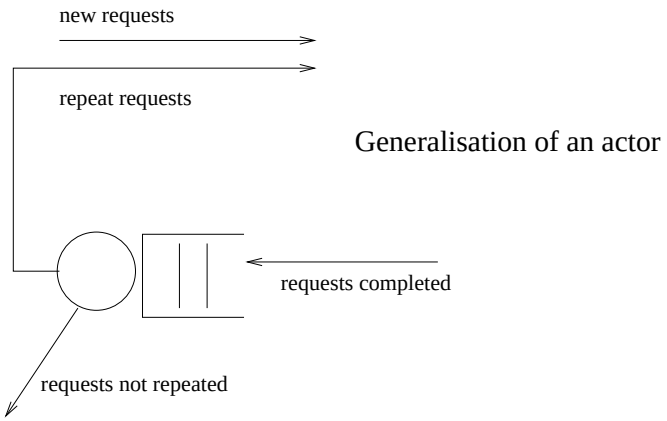



Figure 17: Actor building block




- successfully built a simulator around the sequence diagram notation;
- built a simulation library around collaborations with state machines.

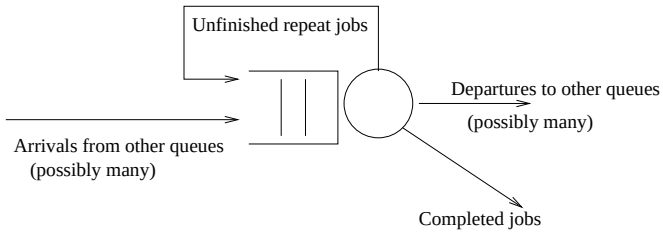
There have been shown to be several other potential developments and these are being followed up enthusiastically.

## 9.2 The challenge

A greater challenge that adding separate tools to parts of the UML lies in basing an integrated performance engineering approach on the whole notation. This will not be straightforward, especially as parts of the UML semantics seem poorly defined for our purposes. There is enough similarity to existing integrated performance modelling approaches to make us optimistic, however. In particular the use of collaboration diagrams with embedded state machines seems very promising. The extension of this to incorporate collaborations within deployments seems the ultimate goal.

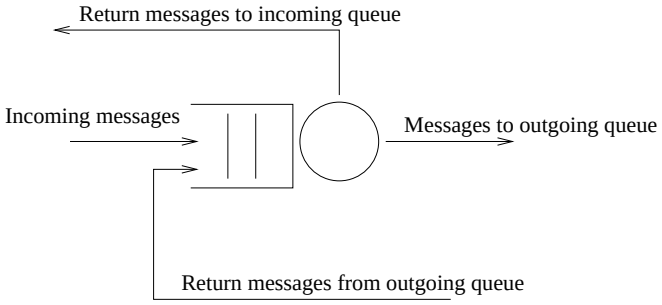
If we accept that a UML based approach is likely to be useful, we must also engage in the formulation of the language, to ensure that our concerns are dealt with. At the moment the notation is still being refined and there is scope for our views to have a real influence. This requires a serious commitment on the part of performance engineers to help shape UML.

Figure 18: Component building block




Generalisation of a component

Figure 19: Link building block




Generalisation of a link

# 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. Baskett, K.M. Chandy, R.R. Muntz, and F. Palacios-Gomez. Open, closed and mixed networks of queues with different classes of customers. *Journal of the ACM*, 22(2):248–260, 1975.

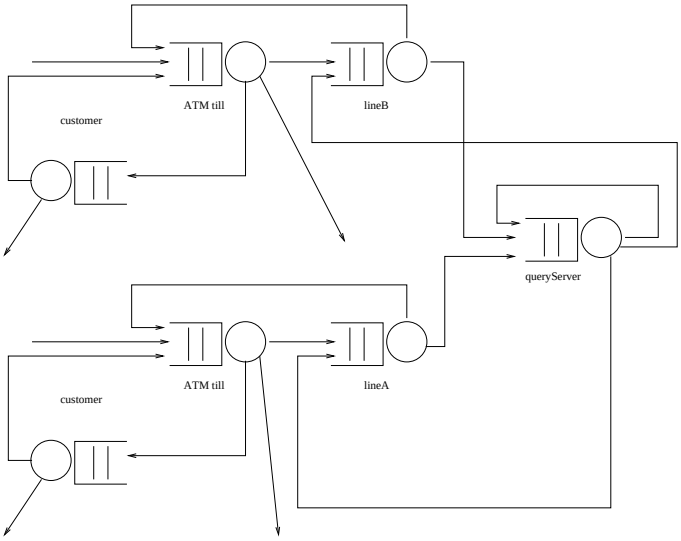
[3] F. Bause and P. Buchholz. Protocol analysis using a timed version of SDL. In *Proceedings of the 3rd Int. Conf. on Formal Description Techniques (FORTE '90)*. Springer, 1991.

[4] H. Beilner and F.J. Stewing. Concepts and techniques of the performance modelling tool HIT. In *Proceedings of the European Simulation Multiconference*. SCS Europe, 1987.

[5] F. Belina and D. Hogrefe. The CCITT-specification and description language SDL. *Computers Networks and ISDN Systems*, 16(4):311–341, 1988.

[6] T. Bolognese and E. Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN Systems*, 14(1):25–59(60), 1987.

Figure 20: Queuing network for multiple Auto-Teller Machines




[7] A.E. Conway and N.D. Georganas. *Queueing Networks — Exact Computational Algorithms.* MIT Press, Cambridge, Massachusetts, 1989.

[8] 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.

[9] Jerome Rolia Greg Hills and Giuseppe Serazzi. Performance engineering of distributed software process architectures. In H. Beilner and F. Bause, editors, *Computer Performance Evaluation - Modelling Techniques and Tools, 8th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, number 977 in LNCS, pages 357–371. Springer, 1995. Heidelberg.

[10] N.J. Gunther. *The Practical Performance Analyst.* McGraw-Hill, New York, NY, 1998.

[11] J. Mäter H. Beilner and C. Wysocki. The hierarchical evaluation tool HIT. In Haring and Kotsis [14], pages 3–6. Vienna.

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

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

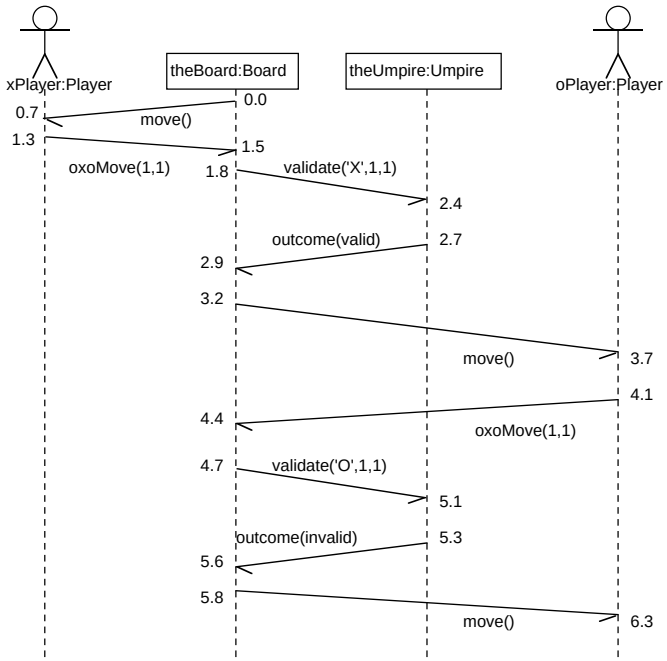
[14] G. Haring and G. Kotsis, editors. *Computer Performance Evaluation - Modelling Techniques and Tools, 7th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, Vienna, May 1994. Springer-Verlag. Vienna.

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

[16] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

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

Figure 21: A trace from the simulator





[18] ISO/IEC JTC1/SC21/WG7. Reference for open distributed processing. Technical Report N885, ISO, 1993.

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

[20] 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.

[21] Robin Milner. *Communication and Concurrency*. Number ISBN 0-13-115007-3 in International Series in Computer Science. Prentice Hall, 1989.

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

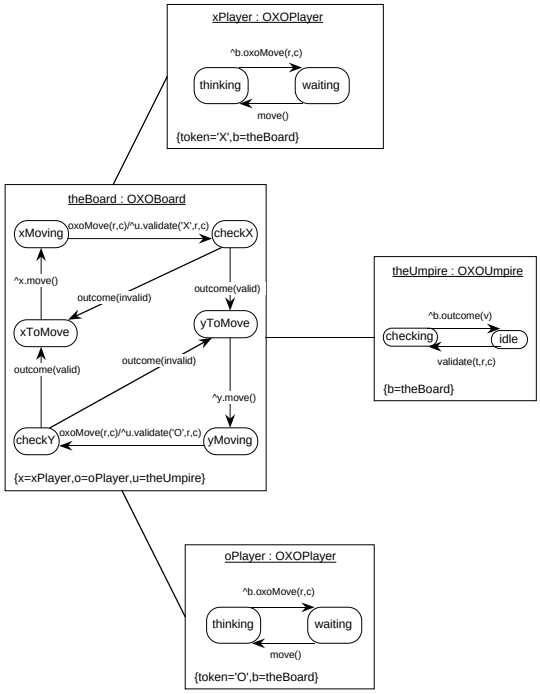
[23] Rob Pooley and Perdita Stevens. *Component Based Software Engineering with UML*. Addison-Wesley, November 1998.

[24] 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.

[25] Connie U. Smith. *Performance Engineering of Software Systems*. Addison-Wesley, Reading, Massachusetts, 1990.

[26] N.M. Macintyre S.R. Kershaw and N. Xenios. A toolset to support the performance evaluation of systems described in SDL. In Haring and Kotsis [14], pages 23–26. Vienna.

Figure 22: States embedded in collaborations




[27] Lloyd G. Williams and Connie U. Smith. Information requirements for software performance engineering. In H. Beilner and F. Bause, editors, *Computer Performance Evaluation - Modelling Techniques and Tools, 8th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, number 977 in LNCS, pages 86–101. Springer, 1995. Heidelberg.

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