Software Engineering and Performance: A Road-map

Rob Pooley
Department of Computing and Electrical Engineering,

Heriot-Watt University
Edinburgh EH14 4AS, United Kingdom
+44 131 451 3367
rjp@cee.hw.ac.uk

ABSTRACT
Software engineering has traditionally focussed on functional requirements and how to build software that has few bugs and can be easily maintained. Most design approaches include non-functional requirements among the elements of the analysis of a system, but little attention has usually been paid to how these requirements can be dealt with through the development life-cycle. Performance analysis, mostly through queueing models and simulations, has usually been used only in designing hardware devices, such as switches, or in capacity planning in the deployment of systems, again concentrating on the hardware and its ability to cope with a given workload. Work on the inclusion of performance engineering throughout the life-cycle has made relatively little impact. With the advent of more structured software design approaches, such as component based development, software architectures and object broker distributed systems, the ability to design software for performance has begun to emerge as a major challenge. This paper reviews some approaches to performance prediction and suggests how this area might develop in the next decade or so.

Keywords
software engineering, performance engineering

1 PERFORMANCE ANALYSIS, PERFORMANCE ENGINEERING

The study of the performance of computer systems and networks attempts to understand and predict their time dependent behaviour. The overall process of estimating performance is often referred to as *performance analysis*. Its integration within the engineering process is commonly referred to as *performance engineering*. As well as modelling, this involves measurement of real systems. Recent work has tried to combine estimation of a system's performance with estimates of its reliability, giving a joint measure, called *performability*.

These topics have been the subject of extensive research and their importance is widely recognised, but many software and hardware designers are reluctant to use them. They are perceived as difficult and time consuming. This need not be true, however, as powerful and user friendly tools are now available to assist. Some are described briefly below. for more details of these and others, readers should consult the references at the end of this paper especially, the Proceedings of the International Conference on Modelling Techniques and Tools for Computer Performance Evaluation [1,16,6,7] for further information. Papers of interest may also be found in the UK Performance Engineering Workshop proceedings [20,5] and the Proceedings of the First International Workshop on Software and Performance [23].

The rest of this paper is organised as follows. Section 2 introduces the state of the art in performance evaluation. Section 3 outlines the distinctive contribution of software performance engineering (SPE). Section 4 outlines significant factors arising from recent developments in software design. Section 5 looks at some current work in integrating software design and SPE. Section 6 presents an agenda for further work in this area and section 7 sums up the paper's message.

2 PERFORMANCE ENGINEERING TECHNIQUES

Performance engineering is essentially an experimental approach to predicting the likely performance of systems. It can involve building and then monitoring a typical system, under the workloads of interest, or using models which reproduce the time dependent behaviour of an unrealised system, either driven by a trace of the expected workload or by a workload represented as a set of random variables (stochastic modelling). Most of the work of interest to software engineering is probably concerned

with stochastic modelling of systems during their design, i.e. modelling abstractions of the target system. The advantages of modelling include:

- estimates are made where a system does not exist yet or is too costly to buy to monitor;
- the workloads possible with a model may not be easy to generate on a real system;
- models can generate almost any measures we wish, which monitoring may not match;
- a model can test conditions which would damage the real system.

Typical representations used for performance models include queueing networks, Petri nets (especially timed and stochastic extensions), a variety of proprietary simulation languages and notations and, most recently, timed and stochastic extensions to formal description languages and process algebras.

Models can be classified according to how they are described or how they are solved. The main solution techniques are analytic, numerical and simulation. When we solve a model we can obtain an estimate for a set of values of interest within the system being modelled, for a given set of conditions which we set for that execution. These conditions may be fixed permanently in the model or left as free variables or parameters of the model, and set at runtime. By varying the input values we explore how the outputs vary with changing conditions. This is similar to the experiments we might conduct by measuring a real system.

The failure of traditional engineering and computer science education to build in an effective introduction to notions of probability, as a key abstraction and approximation technique have tended to leave research in these areas outside the mainstream of software engineering. It is interesting to contrast the relatively low emphasis on stochastic modelling and statistical understanding in engineering with the high emphasis within management and business studies, where operations research is highly valued for its ability to contribute to efficiency.

Most engineers are happy to replace some complex behaviour with a simple delay when estimating time dependent behaviour. Most will happily use upper and lower bounds to estimate operational tolerances. Surprisingly few are capable of generalising this to use appropriate stochastic variables for such estimates.

Perhaps even worse, few seem to understand the respective roles of statistics in interpreting observations and stochastic modelling in applying such understanding to assist in abstraction.

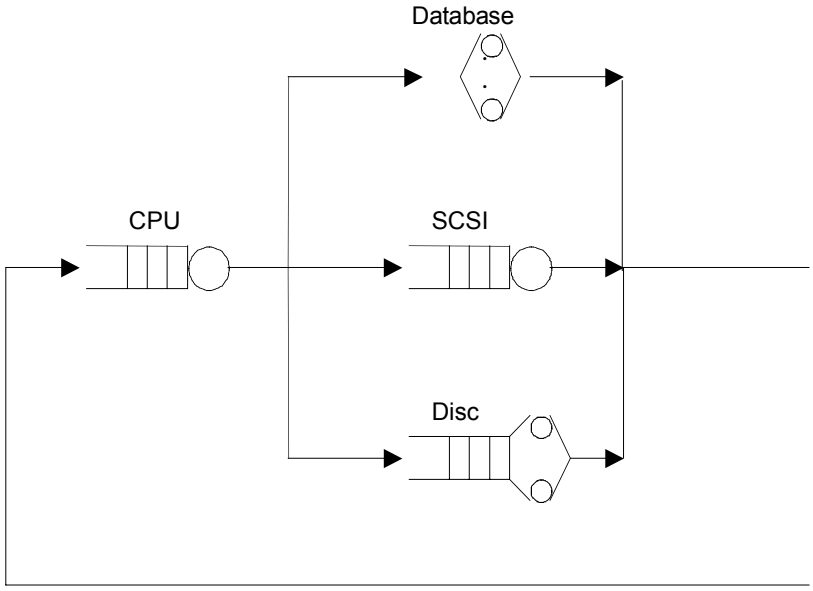

Figure 1: Typical queueing network

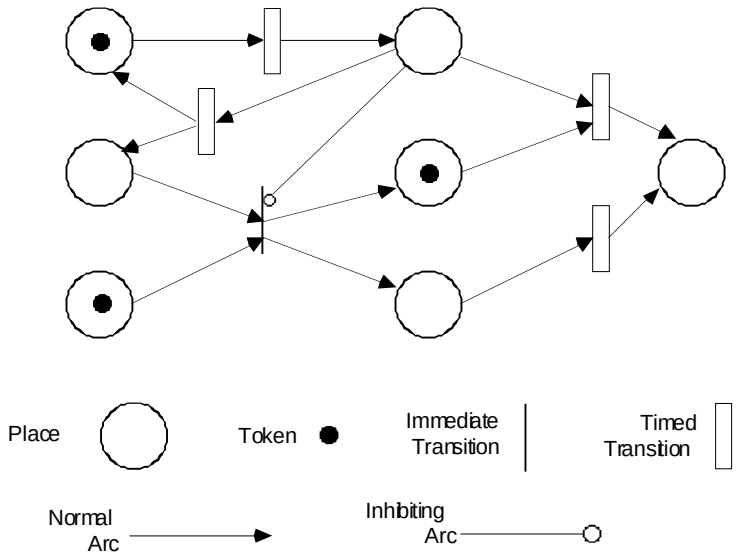

Figure 2: Typical stochastic Petri net

Simulation is the most general and versatile means of performance modelling. It has many uses, but its results are usually only approximations and the price of increased accuracy is longer execution times.

Analytical techniques provide models which can be solved symbolically for the average (steady state) behaviour of a system. Unfortunately only a very restricted set of models have such solutions. Even fewer have exact solutions.

Numerical techniques involve deriving an underlying model, typically a continuous time Markov chain, which can be solved for a given set of parameters by solving a set of simultaneous equations. These are somewhere between analytical and simulation models, being more general but slower than analytic techniques and less general but faster than simulation.

Approximate solution techniques are sometimes used for both analytic and numerical approaches. Even for approximate models, useful predictions are often possible. It is certainly usually possible to compare two alternative designs, even if the absolute results are not accurate.

3 SOFTWARE PERFORMANCE ENGINEERING

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.

Explicit efforts to build methods and notations which would appeal to software engineers and designers include Smith's Software Performance Engineering [22], which outlined a methodology for integration of performance estimation within the software engineering process, and Beilner's HIT [3], which introduced a notation based on abstract data types and layered design, from which performance estimates could be derived automatically, using a variety of solution techniques. Hughes [13] also introduced a notion of hierarchical decomposition of measures, related to hierarchical design methods. These efforts are interesting, but still require a considerable effort on the part of the designer to understand the notations used, which are related to but not the same as those commonly used in software design.

In our roadmap of performance within software engineering, shown in Figure 5, we find a continuing trend over time to adapt ideas from software engineering to the needs of stochastic modelling. This continues to branch as new influences from both stochastic modelling and software engineering enter the main highway, but then usually re-converge as software engineering itself achieves a new level of abstraction and expressiveness.

The most significant feature revealed is, arguably, the failure of performance engineering ideas to influence software engineering, however. This may have robbed software engineering of important insights, as well as preventing it from adopting performance analysis as a key component of engineering.

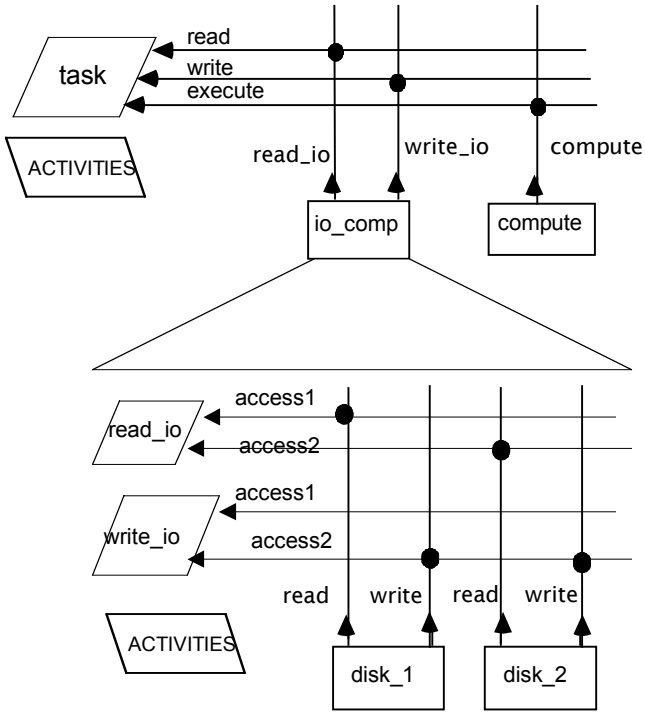

Figure 3: HIT hierarchical model

There has been greater success in adapting formal and semi-formal description languages, such as SDL [2,8] and LOTOS [21,25]. The results have been used to derive simulation models, HIT models and stochastic process algebra models. The success of this approach remains limited to the communities where these design notations are used, however. The wider community has had little help.

Other workers have constructed integrated environments, incorporating many model representations and solution techniques, in an attempt to lure designers into adopting them. Perhaps the most ambitious of these was the ESPRIT II Integrated Modelling Support Environment (IMSE) project [15]. Although this really failed it did establish the importance of hiding both the modelling techniques (where it was unsuccessful) and the details of how statistically meaningful experiments should be run (where it succeeded [9].

Figure 5 shows the early importance of statistics and stochastic modelling in a purely mathematical setting. This is rapidly overtaken by the need to allow designers to operate at levels of abstraction much closer to their design fromalisms. Since this has also driven formal methods research to move from purely mathematical and logical formalisms towards more accessible notations, we should be unsurprised to find much progress in developing performance based extensions to formal behavioural notations, such as Petri nets [4] and process algebras [10].

4 DESIGN OF SOFTWARE SYSTEMS

Software design has suffered itself from a lack of agreement on an approach and a notation which is best suited to the creation of effective applications. Fashions have moved on frequently, with functional, structured approaches being challenged by object oriented and latterly component based approaches. In some application domains formal proof has been seen as paramount, with languages such as LOTOS growing from formal concurrency research using process algebras. For the performance engineering researcher, this has meant a bewildering range of rapidly moving targets.

Figure 5 shows influences from software engineering driving performance analysis further towards performance engineering. Structured methods, objects and components and formal methods have all influenced the developments of new ways of expressing our models. This has led to a fragmented group of developments, with much duplication.

The rest of this paper takes an optimistic view of where software design is going, however. This is based on an analysis of design trends which sees convergence on three major, closely related concepts:

Component based design provides a high level abstraction of the building blocks used in system construction; it allows interfaces and connections between units of the system to be identified in a manner similar to the hardware systems where performance analysis is already well accepted; it allows investment in the understanding of component behaviour, since component reuse is a major part of this approach;

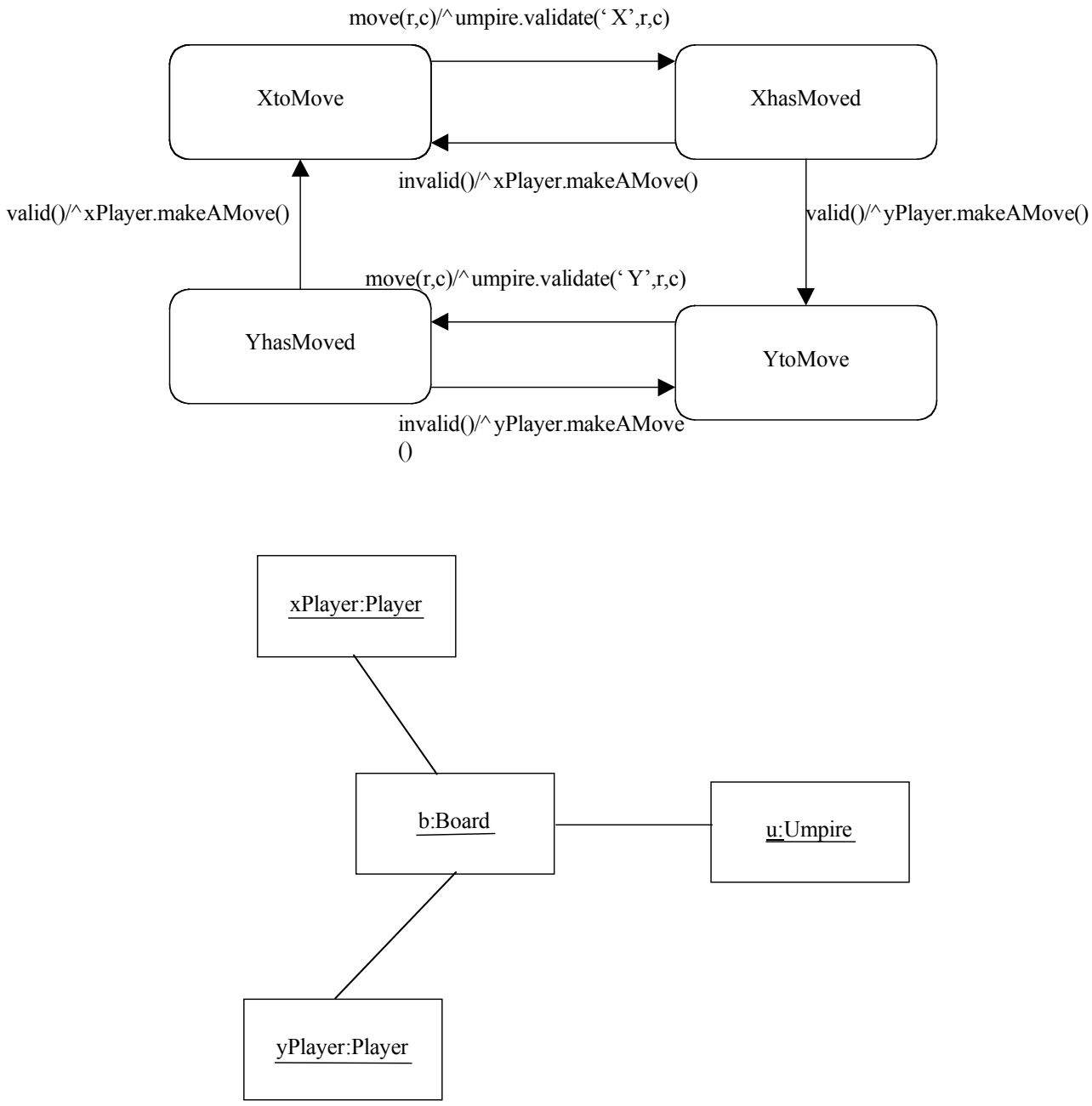

Figure 4: UML version of a Harel Statechart and a UML collaboration diagram

Architectures and patterns at all levels of design, support the view of systems as communicating components and map easily onto the way performance models define systems; design uses communicating state machines; performance modelling uses communicating stochastic processes;

UML [19,14] provides a widely accepted notation for a component based approach to design, which will become more formalised as it becomes used in critical systems; along with a widely used notation will come widely used computer assisted software engineering (CASE) tools. It is apparently possible to attach performance analysis tools to these in a relatively straightforward manner.

These ideas are presented in the context of recent research

in the next section.

5 INTEGRATING CURRENT DESIGN METHODS WITH PERFORMANCE ENGINEERING

The Unified Modelling Language (UML), is a recent attempt to merge the most widely used object oriented design notations. 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 refine this draft, with the latest version appearing in June 1999. Several major Computer Aided Software Engineering (CASE) tool manufacturers are committed to supporting UML, either as the main notation in their tool or alongside their proprietary notation, guaranteeing automatic translation. Many major industrial organisations are adopting UML in their design standards.

The first major meeting devoted exclusively to software performance was the 1st international Workshop on Software and Performance (WOSP) [23], in Santa Fe in October 1998. This concluded that the definition of an integrated means of representing performance within widely used design notations, notably UML, was an essential prerequisite to wider adoption of SPE within the software community. The recent UML '99 conference included papers on this issue [11].

To date work in this area has explored the possibilities for simulation of UML models [17,24], generation of queueing network models from UML deployment diagrams [18] and collaboration diagrams [11]. Mappings from UML to layered queueing network models (LQNs) [26], to stochastic Petri net models (SPNs) [12], more specifically Trivedi's SPNP [4] tool's variant of SPNs, and to stochastic process algebra models, particularly Hillston's PEPA [10], have been developed. All of these show the potential for using UML's logical and behavioural notations to define the structure of performance models. Others have extended the integration of UML and simulation modelling, using tools such as Rational's Rose CASE tool to input models [24].

Figure 4 shows a UML collaboration diagram for a simple two player boardgame and the statechart representing the internal behaviour of one class of object, Board, used to build this collaboration. Annotations to the statecharts expressing time are defined within UML today. Further developments in incorporating timing within message sequences and other external behaviour diagrams are being debated. This will take us to the point shown in Figure 5 where we have performance CASE tools. That is today's state of the art. Our problem is which way forward from there.

6 AN AGENDA FOR PROGRESS

There has never been a more optimistic point for expanding the use of performance engineering within the wider software engineering community. This will be accelerated by the emergence of much more elaborate systems for implementation of applications. The growing popularity of object broker architectures and World Wide Web based applications will force users to ask developers hard questions about response times and scalability.

Figure 5 presents an optimistic scenario, where we are poised at the final stage in our journey. If we accept that a fully integrated approach to software prediction within CASE based software engineering is sufficient, this is true. What is clear from all previous work is that this will not really be the end of the road. Much more will then become clear and the agenda will be extended. If we accept that we must first reach this foreseeable goal, we can create an agenda.

In some ways the agenda is already being set. The second WOSP workshop will be held in 2000 in Ottawa. This will build on a widening body of work and will aim to drive forward a set of features for performance to represented in UML models. The OMG has already launched a Revision Task Force (RTF) to look at the representation of time and performance in UML. The community supporting WOSP 2000 is aiming at influencing analysis tools within CASE products.

Beyond that point, more effective techniques for performance estimation within the context of component based designs are needed. Now that we have a large, slowly moving target, it should be possible to tailor the approaches used in analytic, numerical and simulation solutions to match systems described in this way

An interesting theme within these developments is the comparison of performance effectiveness with other software properties, such as ease of maintenance. We have to ask whether heuristics which talk in terms of coherence and coupling in software design apply also to performance goals. At first sight, it appears that these are compatible, but research is needed to establish this.

An other pointer is the growing interest among formal methods researchers in probabilistic notions of equivalence and their adoption of properties couched in probabilistic terms. While these are rarely interested in time as a stochastic variable, the gap in understanding between the two communities with an interest in introducing formal mathematical analysis into software engineering may be closing. This trend can be thought of as the replacement of traditional finite state automata as the formal underpinning of software engineering by the use of Markov chains, representing non-determinism and data dependency by probabilities and time delays by stochastic variables.

If momentum can be maintained, we can look forward to a true convergence of software engineering and performance analysis within the next decade.

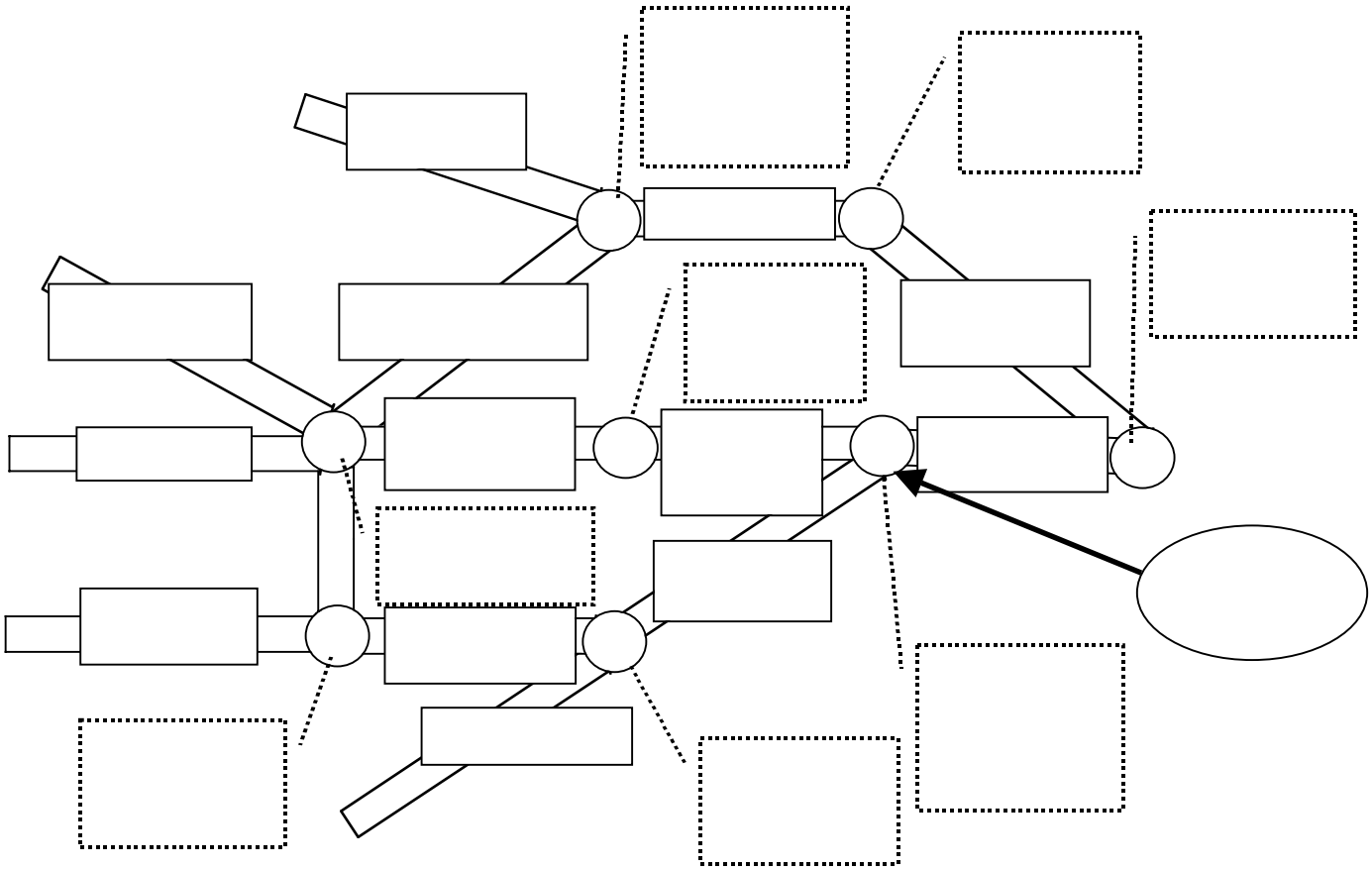

Figure 5: Software performance roadmap

7 CONCLUSIONS

Traditional performance modelling techniques can be applied to component based views of software systems. This is becoming increasingly important as object broker and related architectures become more significant in the development of new systems. The challenge is to bring the world views of software engineers and performance analysts into alignment.

For this to happen, performance analysis techniques must be embedded within design methods and tools. Although there is a long history of improvement in techniques for performance analysis, there has been less success in persuading designers to think in these terms.

As component based design is rapidly adopting UML, this must be the current focus of such work. Results so far are encouraging and it seems probable that a successful synthesis can be achieved within a decade or less. From there we enter the currently unknown territory of automation of analysis.

The overall picture can be summarised in our five key steps to achieve our current goals:

1. To create a well understood formalism, probably based on UML, allowing performance annotations to design models.

2. To create a methodology which embeds performance questions within the software lifecycle in terms of widely used approaches.

3. To integrate solution tools for performance measures transparently within extended design tools, such as object oriented CASE tools.

4. To develop ways of returning performance results from specialised tools in terms of the design models from which they were derived.

5. To integrate performance modelling measures within a performance monitoring and testing framework in a consistent manner.

We might perhaps add a desire to converge with developments in formal methods, where stochastic models are of increasing interest.

ACKNOWLEDGEMENTS

The ideas in this paper have benefitted from discussions with Jane Hillston, Heinz Beilner, Peter Hughes, Peter King, Nigel Thomas, Neil Davies and many others.. Their contribution is gratefully acknowledged.

REFERENCES

1. Balbo, G. and Serazzi, G. *Computer Performance Evaluation – Proceedings Modelling Techniques and Tools*, (Torino September 1992), Elsevier.

2. Bause F. and Buchholz, Protocol analysis using a timed version of SDL. in *Proceedings of the 3rd International Conference on Formal Description Techniques (FORTE '90)*] (Madrid, 1991), Springer.

3. Beilner, H. and Stewing, F. Concepts and techniques of the performance modelling tool HIT. in *Proceedings of the European Simulation Multiconference*, (Vienna March 1987), SCS Europe, 84-89.

4. Ciardo, G., Muppala, J., and Trivedi, K. SPNP: Stochastic Petri net package. In *Proceedings of 3rd International Workshop on Petri Nets and Performance* (Kyoto, Japan, 1989), 142--151.

5. Davies, N. and Bradley, J. Eds. *UKPEW '99, Proceedings of the Fifteenth UK Performance Engineering Workshop* (University of Bristol, July 1999). UKPEW.

6. Haring G. and Kotsis G. *Computer Performance Evaluation - Modelling Techniques and Tools, 7th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, Number 794 (Vienna, May 1994), LNCS 794, Springer-Verlag

7. Haring, G. and Wabnig, H. Eds *Short Papers and Tool Descriptions, 7th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation* (Vienna, May 1994), University of Vienna

8. Heck, E., Hogrefe, D., and Müller-Clostermann, B. Hierarchical performance evaluation based on formally communication protocols. *IEEE Trans. Comp* 40,4 (1991), 500-513.

9. Hillston, J. A tool to enhance model exploitation. In Pooley R. and King P. Eds *6th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation* (Edinburgh 1993), 131-142.

10. Hillston, J. *A Compositional Approach to Performance Modelling*, (1996) Cambridge University Press.

11. Kähkipuro, P. UML based performance modeling framework for object-oriented distributed systems. In *«UML» '99 - The Unified Modeling Language: Beyond the Standard* (October 1999), 356--371.

12. King, P. and Pooley, R. Using UML to derive stochastic Petri net models. In Davies N. and Bradley J. Eds. *UKPEW '99, Proceedings of the Fifteenth UK Performance Engineering Workshop* (Bristol, July 1999), 45-56, UKPEW.

13. Minkowitz C., Vetland V. and Hughes, P. A modular approach to system structure and Specification. In *7th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation – Tools upplement*, (Vienna 1994), University of Vienna, 83-86.

14. OMG, Unified Modeling Language 1.3, 1999 at http://www.rational.com/uml/documentation.html

15. Pooley R.J. The Integrated Modelling Support Environment. In Balbo and Serazzi Eds., *Computer Performance Evaluation – Proceedings Modelling Techniques and Tools*, (Torino September 1992), 1-16, Elsevier

16. Pooley, R. and Hillston, J. *Computer Performance Evaluation - Modelling Techniques and Tools, 6th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, Number 10 in Edits (Edinburgh 1993), Edinburgh University Press.

17. Pooley, R. and Kabajunga, C. Simulation of UML sequence diagrams. In Pooley R. and Thomas N. Eds, *UK PEW '98 - Proceedings of 14th UK Performance Engineering Workshop* (Edinburgh, July 1998), 198-207, UKPEW.

18. Pooley, R. and King, P. The Unified Modeling Language and performance engineering. *IEE Proceedings – Software*, 146, 1 (February 1999), 2-10.

19. Pooley, R. and Stevens, P. *Component Based Software Engineering with UML*, (December 1998), Addison-Wesley.

20. Pooley, R. and Thomas, N. *UKPEW '98 - Proceedings of 14th UK Performance Engineering Workshop* (July 1998), UKPEW

21. Rico, N. and Bochmann, G. Performance description and analysis for distributed systems using a variant of LOTOS. In *10th International IFIP Symposium on Protocol Specification*, Testing and Validation, (July 1990).

22. Smith, C. U. *Performance Engineering of Software Systems* (1990), Addison-Wesley.

23. Smith, C., Clements, P., and Woodside, M. Eds. *1st International Workshop on Software and Performance* (Santa Fe, October 1999), ACM.

24. Utton, P. and Martin, G. Further experiences with software performance modelling. In *WOSP '98, First International Workshop on Software and Performance* (Santa Fe, October 1998), 14-15.

25. Valderruten, A., Hjiej, O., Benzekri, A., and Gazal, D. Deriving queueing networks performance models from annotated LOTOS specifications. In *6th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation* (Edinburgh 1993), 120-130, Edinburgh University Press

26. Woodside C. M., Neilson J. E. and Majumdar S., The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software. *IEEE Trans. on Computers*, 44, 1 (January 1995), 20-39