Predictive Performance Analysis for Distributed Systems – PERMABASE position

David Akehurst and Gill Waters
Computing Laboratory,
University of Kent at Canterbury,
CT2 7NF, UK
Tel: +44 1227 827697
Fax: +44 1227 762811
Email: {D.H.Akehurst, A.G.Waters}@ukc.ac.uk

Peter Utton and Gino Martin
BT Labs,
Martlesham Heath,
Ipswich, IP5 3RE, UK
Tel: +44 1473 647740
Fax: +44 1473 642299
Email: {peter.utton, gino.martin}@bt.com

Abstract

The PERMBASE project has provided a prototype environment that proves the feasibility of automatically generating performance models directly from system designs. The prototype uses UML as the input specification language, performs some transformations and basic model checking on the design and generates a performance model that is supported by a discrete event simulation engine. This position paper gives a brief introduction to the project, and defines the future aims and resultant research related to the project conclusions.

1. Introduction

The PERMABASE project ([1],[2],[3]) aimed to provide performance feedback as part of the object-oriented design process for distributed systems. PERMABASE hides the expertise needed to create a performance model (simulation or analytic) from the system designer, by automatically generating the performance model from the design model. The feedback provided by the performance model allows the designer to change or confirm design decisions throughout the system design process, with an understanding of the implications of those design decisions on the overall system behaviour.

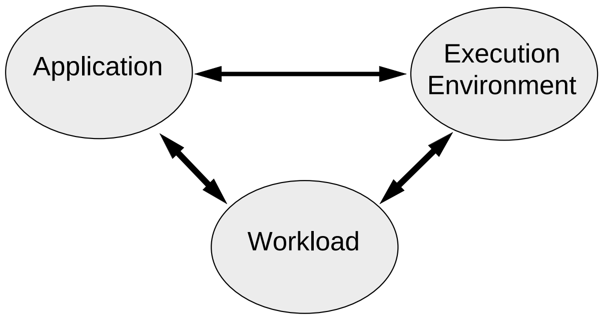


Figure 1 – PERMABASE domains of interest

The project identified three domains of interest within the specification of a distributed system (Figure 1). Each of these areas must be specified to give sufficient information such that a performance model could be created. These separate domains are defined as follows:

- Workload Specification – The specification of the "work force" driving the system. These may be human operators, other systems, or simply a model of "miscellaneous" other work being carried out on shared system components;
- Application Specification – The specification of the system logic. Primarily this is assumed to be software, but it may be hard-wired (firmware) or hardware logic;

- Execution Environment Specification – The specification of the physical environment of processors, networks, and other resources that the system operates over.

The intention was that each of these specifications should be described in an appropriate (standard) notation, and by using a suitable CASE tool for that notation. The details entered into the CASE tools form the source of the specifications to be used by the automatic generation process.

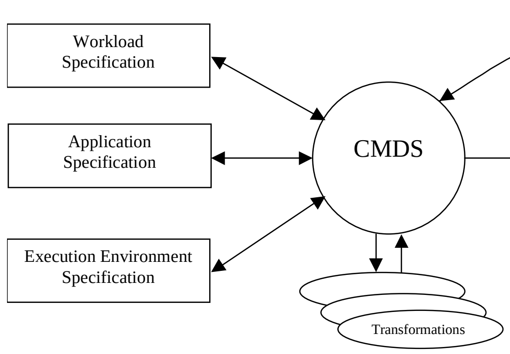


Figure 2 – Initial PERMABASE Architecture

The generation process consists of three stages (as shown in Figure 2):

- Composition of the information from the three specifications into a "Composite Model Data Structure" (CMDS). This involves the mapping and translation from the representation of the specification in the visualisation input tools into the concepts defined in the CMDS.
- Transformations on the CMDS to check for inconsistencies, and to refine the model. Such transformations include transposing the input representation of the system behaviour (in the form of multiple interaction diagrams) into a class centric representation, more suited to the technique used for performance model generation.
- Translation from the CMDS into the performance model. This involves the definition of a mapping from the (meta-level) CMDS concepts to the components of a particular performance model engine, and the subsequent algorithm for translating instances of the CMDS (specific design models) into a performance model.

The decision was taken (based on BT software design practice), to focus on the use of object-oriented designs for the application specification, but there did not appear to be any standard for the specification of the information required in the workload or execution environment specifications. BT practice for object-oriented software design was, when the project started, to use the Booch notation and method (or a slight adaptation of it, [17]), supported by the Rational Rose modelling tool ([6]).

Initially the project looked to this notation for the facility to specify the workload and execution environment information, but the notation did not appear to address workload specification at all, and physical environment specification was limited[1]. Due to this lack of a standard notation, the project adopted a BT in-house tool that is used for capturing network and system design information. There is no specific notation that the tool supports, it simply allows for the

[1] Some of those limits were imposed by the lack of support by the Rational Rose tool, rather than by the notation itself.

specification of types of node, attributes of the nodes, and connections between the instances of nodes.

As the project evolved it became apparent that these three specifications did not contain all the required information. Although the three specification areas contained between them the specification of all the components, it was realised that the inter-connectivity between components from each area is of primary concern. An initial thought was to show in the Execution Environment Specification, the connections to the components defined in the other areas, but this caused the specification to become cluttered and invalidated the definition (see above) of this specification area – it no longer defined only execution environment information.

The need to specify the interconnections of components from the three specification areas led to the introduction of a fourth specification area – the system scenario specification – which alters the architecture to that shown in Figure 3.

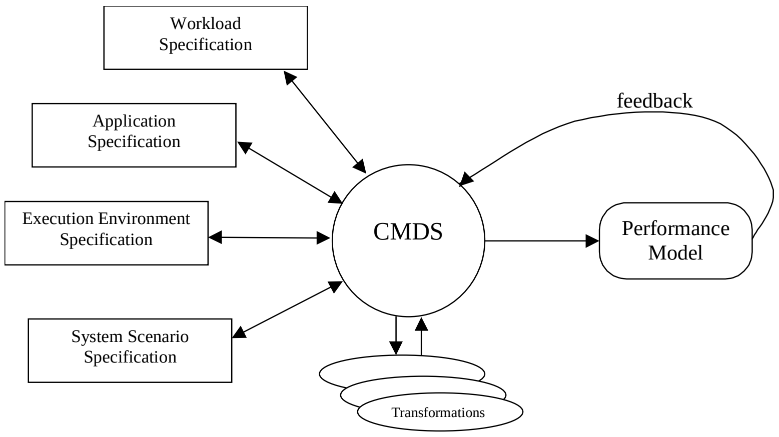

Workload Specification
Application Specification
Execution Environment Specification
System Scenario Specification
CMDS
feedback
Performance Model
Transformations

Figure 3 – PERMABASE Architecture

The existence of the system scenario specification led to the redefinition of the other three specification areas, to be *types* of component. This redefinition occurred in part, for efficiency, so that there did not need to be repetition of the specification of parts of the system, and in part to draw the whole specification style closer to the object-oriented design pattern of templates and instances. Hence, the four specification areas, required for the definition of a distributed system (for performance analysis purposes), evolved to:

- Workload Specification – The specification of the *types of* "work force" *component* driving the system;
- Application Specification – The specification of the *types of* system logic *component*;
- Execution Environment Specification – The specification of the *types of component* to be used in the physical environment;
- System Scenario Specification – The specification of a particular (distributed) system instance, in terms of the instances of the types (or classes) of component defined in the other three specifications and how they are connected.

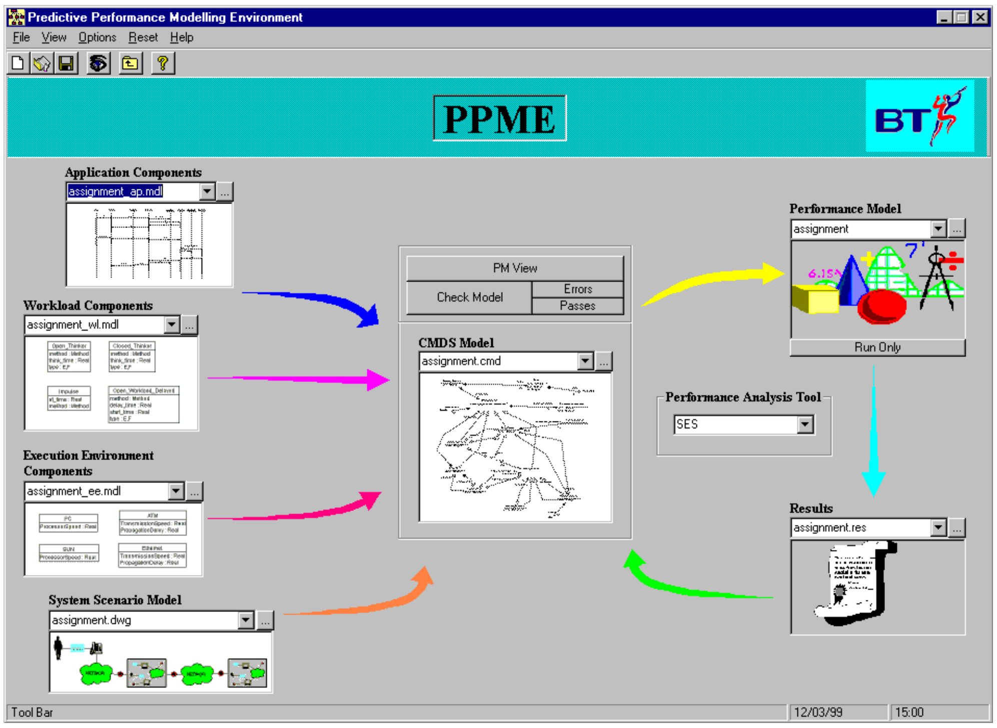
During the lifetime of the project, the UML ([4]) emerged as the leading (standard) notation for the specification of object-oriented systems. Consequently, it was adopted by PERMABASE to replace the Booch notation, due to the extra features and improved usability it offered, and because BT were beginning to adopt it as their standard practice. The use of UML within the project caused a re-assessment of the methods used to describe the source specifications. In particular the extensibility feature of UML opened up the possibility of using it as a standard notation for all (four) specification areas.

In practice the project used UML for the three ‘component type’ specification areas – Workload, Application and Execution Environment – the specification of the system scenario was left to the BT in-house tool, as the initial versions of UML did not contain adequate facilities to specify this information. Subsequent revisions to UML, and changes to the way system scenario specifications are defined, enable a mixture of UML class, object, and deployment diagrams to be used, however this was not implemented during the life-time of the PERMABSE project.

The project ideas were implemented in the form of three prototypes:

- The first used the Booch notation and the Rational Rose tool for input of the Application Specification, and bespoke notations supported by the BT in-house tool for the other specifications. The behaviour of the application components was specified using state machines. The performance model engine used was a discrete event simulation (DES) engine (SES/Workbench, [7]);
- The second prototype was based on the UML notation, supported by a new version of Rational Rose. The architecture evolved to that shown in Figure 3 above, and made more use of the UML notation. Feedback from BT on the basis of their use of the first prototype indicated that they would prefer to specify behaviour using Interaction (Sequence or Collaboration) Diagrams, so this became the supported technique. Performance model generation and execution proved to be time consuming using the DES engine, and a faster mechanism was required, particularly for simple designs. This led to the use in the second prototype of a tool built at UKC, based on Coloured Petri-nets ([8]), and incorporating some facilities for time and resource usage. This performance engine proved to be much faster for building and executing models, however, it did not provide support for the dynamic binding or dynamic object creation functionality required by some applications;
- The third prototype used the same input techniques (with minor evolutionary improvements), but added some model checking algorithms to be performed over the CMDS, and used an improved version of the DES performance engine, which supported dynamic functionality, and used faster transformation and execution techniques than the first prototype. The DES solution could not match the Petri-net one in terms of speed, but it did provide the functionality required for the main test case study.

The prototypes have been constructed using multiple input tools and different performance engines. To aid their use and make operation of the performance generation process work smoothly a GUI management environment (Figure 4) has been created. This environment interfaces to each of the tools used by the prototype, and hides the cross platform connections required for interfacing to the DES engine.


Predictive Performance Modelling Environment
File View Options Reset Help
PPME
BT
PM View
Check Model
Errors
Passes
Run Only
Tool Bar

Application Components: assignment_ap.mdl
Workload Components: assignment_wl.mdl
Execution Environment Components: assignment_ee.mdl
System Scenario Model: assignment.dwg
CMDS Model: assignment.cmd
Performance Model: assignment
Performance Analysis Tool: SES
Results: assignment.res
12/03/99 15:00

Figure 4 - PPME GUI

2. Experience of prototype use

The prototype has been used as part of a number of case studies of BT systems. The main case study (described in [3]) was of such a size that the original design decisions to ignore usability and performance of the prototype itself have been sorely tested. The case study system involved a high work-rate of several thousand events per hour, processed by a network of over three thousand processing nodes. Comparisons between the results produced by handcrafted performance models and those produced by the auto-generated models for the main case study have revealed differences of less than 1% in general.

The main disadvantage found with the automatically generated model is that it took significantly longer to execute, and hence produce results. This is believed to be a problem common to automatic generation in general, and has certainly been shared with the compiler community. Compiled code used to be renowned for being slower than hand coded assembler, though as can be seen with modern-day compilers the problems have been largely eliminated. It is believed that these techniques could be applied to the automatic performance model generation to aid the improvement of the generated model's execution speed.

In our work, performance models have been enthusiastically received by case study customers and have been seen as valuable investments for the future. They have been considered as tools to help manage the risk within the development process and assist with the evaluation of design decisions or even assess the validity of business cases for future system enhancements.

Customers do not need a lot of convincing about the value of performance modelling. However the biggest hurdle to be overcome in making the PERMABASE vision a reality is convincing developers to change their methods of working. It is human nature for people to take the line: "Well, I don't do that at the moment, so why do I need to do it in the future?" Convincing designers to document designs with enough information in them is the main the sticking point. However well performance analysis is integrated within the development process, it still demands somewhat richer source models than are generally prepared today. Of course, better documented designs are likely to have benefits on a number of fronts, not just on performance.

Within the macro design lifecycle, model development and evaluation within the PPME proceeds through an iterative micro-cycle of three phases: *model construction* (assembly of source models and download to CMDS and thence the solution engine); *solution* (solving the performance model to produce performance measures); and *problem resolution* (debugging the source models). The current prototype provides plenty of 'syntactic' and cross-reference checks during the construction phase, but is lacking in semantic checks that might help during the problem resolution phase. Design model errors we have experienced include:

- Typos in input values, which although syntactically valid, can produce order of magnitude numerical errors in results, which can be tricky to track down;
- Consistency of units, (bits/bytes, seconds/minutes/hours);
- Incorrect degree of multi-threading allowed within objects, i.e. the number of concurrently active threads of execution allowed;
- General application logic errors.

Of all of these, logic errors are the most insidious and difficult to track down. For example, an obscure logic error within our main case study concerned a mismatch between the processes which allocated and de-allocated operators to calls. The handling of concurrent requests was not properly specified, with the result that the pool of operators was eventually exhausted at which time the system rapidly overloaded.

Step by step execution of the behaviour supported by model animation is generally the tool to resolve these problems. We have used the facilities in SES/Workbench to successfully resolve design model problems, however animation of the design models should undoubtedly be the goal.

We have not totally eliminated the need for some performance modelling experience. For example to specify a caller population and workload within the main case study system, many different approaches were possible, including:

- many million telephones each with a single (closed) workload instance;
- one representative telephone with many million (closed) workload instances;
- one telephone with one workload instance (infinitely multi-threaded) and an (open) arrival generator;
- an open source creating dynamic 'call behaviour' objects.

The validity of these options is somewhat dependent on the platform model chosen for the telephone but clearly some system representations are going to be more computationally tractable to evaluate than others. A performance modeller can apply their knowledge and experience to ensure that efficient models are built and will take into account the 'what-if' questions that may be asked.

The current prototype solves models by simulation. Investigations into hybrid approaches (i.e. combining analytic and simulation techniques) have identified alternate solution strategies that provide benefits in certain circumstances.

3. Related Work

Ours is not the only approach to addressing performance within the development lifecycle. Rational advocates an approach whereby an executable version of the target application is generated early on in the development process (possibly in a stripped down form). This can then be driven in a test environment at anything up to full system loads and measurements taken. This is often described as simulation although it is somewhat different in concept from the simulation we have discussed above. Both approaches can genuinely claim to be simulating the *system* but our approach is more model-based and a key difference is that we simulate the environment as well as the application. The big benefit of the model-based approach is that you do not need the proper execution environment available to get results or to try out alternatives.

Of course, both approaches are actually complementary. Generating early executable code, which can be used as the basis for test measurements, can provide an invaluable source of parameter values for models. The final execution environment need not be available (or even decided on). Measurements taken in a test environment can be used to characterise application resource usage before different ‘what if’ scenarios are evaluated in a model. The benefits are lower costs and quicker results.

We are not the only ones interested in UML from a performance perspective. At the First International Workshop on Software and Performance (WOSP ‘98, [5]), the consensus view seemed to be that system specification languages needed to cover performance issues and should in the first instance be based on UML. For example [9] and [10] both use UML as part of their performance model specification, and in particular a tutorial by Pooley ([11]) discusses UML and performance engineering. As follow up from WOSP, Professor Rob Pooley, of Heriot-Watt University in the UK, is leading a working group on this subject. OMG, the standards body for UML has put out a Request For Proposals on performance in UML ([18]), so this looks to be a topic with a rising profile.

Other research in the area of performance modelling of distributed systems includes:

- Jonkers et. al. ([14]) describe a system with a similar purpose to the PERMABASE system. Their approach identifies two domains, the entity domain and the behaviour domain, and is based on a bespoke modelling and design language – Architectural Modelling Box (AMB), a language developed to meet design prerequisites as well as performance modelling formalisms. Their system identifies most of the same concepts as identified in PERMABASE, but uses bespoke notations and tools for design specification.
- The AUTOFOCUS project ([15] and others), although not specifically aimed at performance engineering, is another similar project to PERMABASE, which provides simulation, consistency checking, and code generation for a distributed system specification. Provision for the system specification is split into four views or description techniques, System Structure Diagrams, Data Types Definitions, State Transition Diagrams, and Extended Event Traces. Their System Structure Diagram attempts to describe the same area of the system specification addressed by this paper, but it is directed towards embedded systems and describing hardware bespoke to a particular system. The PERMABASE approach is more flexible, allowing also for the specification of general hardware components used for supporting a variety of software applications.

- Woodside in [13], who's three-view technique model divides up the specification using a similar architecture to the PERMABASE approach. However, the three views are tailored towards performance modelling specification techniques, we require a method more oriented towards system design, though this work could provide the basis for an alternative solution engine.
- A project based at the University of Helsinki in Finland ([16]), similarly to PERMABASE, focuses on generating performance models from UML designs. This is the most complete work that addresses the same requirements as PERMABASE, and appears to be quite successful. The project has built a prototype, which lacks the integration with a graphical design tool, present in PERMABASE. The solution engine, however, does appear to be at least as good as solution engines used within PERMABASE, and some collaboration or integration between the two projects should achieve some spectacular results.

4. Conclusion

We have shown that through the use of existing CASE tools and design notations, and with appropriate additional tool support, it is possible to incorporate performance analysis into the system design process. The performance model and its automatic generation are hidden from the designer, giving the benefit of feedback regarding the performance of the system, without the need for specific performance analysis training.

The current PPME prototype demonstrates that the aims of the project are feasible although it is fair to say it does not yet totally eliminate the need for performance modelling experience. Choice of abstraction within source models remains a key modelling decision. If we are to eventually embed performance modellers' 'intelligence' into software design tools then, apart from the facilities described in this paper, we will need transforms which can generate alternate model representations and algorithms which detect equivalence and score models for ease of evaluation.

BT is now looking to downstream findings from this project via commercial tools that can be rolled out within its development community whilst spin-off research projects look likely at UKC and Heriot-Watt Universities. A successful product would need to offer a number of features to be accepted by developers and so avoid becoming 'shelfware'. These should include:

- Fast solution times - users need to be able to explore alternate 'what-if' scenarios quickly;
- Comprehensive error checking for consistency and completeness of source models (with diagnostics that pinpoint the offending source model construct) - there is potential to bring significant added value to the design process by identifying design anomalies and omissions at an early stage;
- Effective debugging facilities including ideally animation at the source model level - this also has the potential as a value add tool to enable better communication of design models to others;
- A comprehensive library of component models for standard network and platform types;
- An open framework to enable import of existing models to grow the library.

There are a number of upcoming projects at UKC that will address some of the issues arising from the PERMABASE work. These include:

- The development of more sophisticated model checking and consistency techniques and tools, integrating approaches developed in the formal methods community with the system design and modelling techniques to provide advanced CASE tool functionality;
- Development of the system design and modelling techniques to improve the approaches used to specify of distributed and multi-media aspects of systems;
- Improvement of the definition and interpretation of the UML – linked with the OMG standardisation of the language;
- Continued investigation into the automatic provision and use of performance statistics as part of the design process.

5. References

[1] Peter Utton, Brian Hill; *Performance Prediction: an Industry Perspective;* Computer Performance Evaluation, Proceedings of the 9th International Conference on Modelling Techniques and Tools (Lecture Notes in Computer Science 1245), St Malo; pp.1-5; June 1997.

[2] Gill Waters, Peter Linington, David Akehurst, Andrew Symes; *Communications Software Performance Prediction;* 13th UK Workshop on Performance Engineering of Computer and Telecommunication Systems, Ilkley; pp.38/1-38/9; July 1997.

[3] Peter Utton, Gino Martin; *Further Experiences with Software Performance Modelling;* Proceedings of the First International Workshop on Software and Performance, WOSP 98, Santa Fe; pp.14-15; Oct 1998.

[4] Joint Submission - Rational Software, Microsoft, Hewlett-Packard, Oracle, Texas Instuments, MCI Systemhouse, Unisys, ICON Computing, IntelliCorp; *The Unified Modeling Language, version 1.1;* OMG, Technology Adoptions, ad/97-08-02 to ad/97-08-09; November 1997.

[5] *Proceedings of the First International Workshop on Software & Performance WOSP '98, Santa Fe;* Published ACM; Oct 1998.

[6] *The Rational Rose product;* http://www.rational.com/products/rose/index.jtmpl; Mar 1999.

[7] *The SES/Workbench product;* http://www.ses.com/Workbench/; Mar 1999.

[8] Peter Linington; *RiscSim - A Simulator for Object-Based Systems;* UK Simulation conference, Cambridge; April 1999.

[9] Jianli Xu, Juha Kuusela; *Modeling Execution Architecture of Software System Using Colored Petri Nets;* Proceedings of the First International Workshop on Software and Performance, WOSP '98; pp.70; 1998.

[10] Lloyd G. Williams, Connie U. Smith; *Performance Evaluation of Software Architectures;* Proceedings of the First International Workshop on Software and Performance, WOSP '98; pp.164; 1998.

[11] Rob Pooley; *The Unified Modelling Language and Performance Engineering;* Tutorial at First International Workshop on Software and Performance, WOSP '98; 1998.

[12] Robin Harwood; Object Oriented Development Method; British Telecom; 1998.

[13] C.M. Woodside; *A Three-View Model for Performance Engineering of Concurrent Software;* IEEE Trans. On Software Engineering, Vol. 21, No. 9; Sept. 1995; pp. 754-767.

[14] H. Jonkers, W. Janssen, A. Verschut and E. Wierstra; *A unified framework for design and performance analysis of distributed systems;* Proc. of the 3rd Annual IEEE Int. Computer Performance and Dependability Symposium (IPDS'98); Sept. 1998; pp. 109-118.

[15] Franz Huber, Sascha Molterer, Andreas Rausch, Bernhard Schätz, Marc Sihling, Oscar Slotosch; *Tool supported Specification and Simulation of Distributed Systems;* Proceedings International Symposium on Software Engineering for Parallel and Distributed Systems; 1998; pp. 155-164.

[16] Pekka Kähkipuro; *UML Based Performance Modeling Framework for Object-Oriented Distributed Systems;* Proceedings of <<UML>> '99 - The Unified Modeling Language: Beyond the Standard; October 1999; pp 356-371.

[17] Grady Booch; *Object-Orinted Analysis and Design with Applications, second edition;* The Benjamin/Cummings Publishing Company, Inc.; 1994.

[18] OMG; *UML Profile for Scheduling, Performance, and Time RFP;* ad/99-03-13; April 1999.