Performance analysis of object-oriented designs for distributed systems

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

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

Subject Areas: *Software Engineering Practices, Analysis and Design Methods, Performance Analysis, Distributed Systems.*

Abstract

Performance is an important but often overlooked aspect of systems design. As a consequence, deployed software systems can fail to satisfy requirements and frequently disappoint customers and users. Successful systems are often achieved more by trial and error than through engineering practice. This paper describes a Predictive Performance Modelling Environment that enables performance measures such as response time, throughput and utilisation to be derived automatically from UML based designs for distributed systems. The results can then be exploited within the design process to guide the cost effective and timely delivery of systems that meet customer expectations.

1. Introduction

> "My computer's on a go slow today" came back the embarrassed explanation from the helpdesk operator as she apologised for keeping a customer hanging on the phone. This was the experience of one of our authors recently when he tried to book his car in for a service via a fleet management company, but how many times have you heard something similar?

Clearly incidents of this nature show how the poor performance of software-based systems can impact on users causing de-motivation, frustration and inefficient working. It can also damage customers' perceptions of a company and impact on revenue when they take their business elsewhere.

It can also cost a lot to 'retro-fit' performance into a system if it is not taken into account during the design process in the first place. Unfortunately consideration of performance issues is often left until late in the development lifecycle, when problems have already manifested themselves at system test or actually within a deployed system. If software engineering is to be legitimately claimed as an *engineering* practice then performance issues need to be considered as an integral part of the development process.

Towards this aim, performance modelling techniques need to be made more accessible to typical software engineers, without the necessity for the software engineers to become performance engineering experts. With appropriate supporting tools, development teams should be able to assess the performance implications of design decisions as they make them, and deliver systems with adequate performance in a timely manner. In so doing they will both improve customer satisfaction and control costs.

Over the last three years, research at BT Laboratories and the University of Kent has investigated the integration of performance analysis into the design process, in particular how to predict the performance of distributed object oriented systems. (Earlier results have been published in [1],[2],[3].) This research project, referred to as PERMABASE (standing for Performance Modelling of ATM Based Applications and Services), has developed an integrated process and proof of concept prototype, the Predictive Performance Modelling Environment (PPME), which will generate performance models from UML based designs.

Performance engineering is a highly skilled profession, and performance model generation requires a lot of experience, and knowledge about the technique used to analyse the model (be it an analytic mathematical model or a simulation). Current practice (assuming performance models are produced at all) is for designers to explain the system to performance experts who then code up their understanding as a performance model and (hopefully) review it with the original designers for accuracy. The scope for the introduction of errors at each stage of this process is obvious, especially as in general designers do not understand the details of the performance models and the performance modellers do not understand the details in the designer's world.

The main feature of the PPME is that it automatically generates the performance models, directly from the design specification. This avoids the need for much of the performance engineering expertise normally essential for the creation of performance models, in effect bringing the expertise transparently to the software engineering desktop.

The Unified Modelling Language (UML, [4]) is seen as a significant development and one that offers real hope for convergence within the software industry on a small(ish) set of standard notations and development practices. UML is not perfect and will undoubtedly continue to evolve, but it provides the current "best practice" notation for software system design. Hence, the prototype environment and tool set uses the UML as the notation for system design specification. Our focus has been very much on how to make performance analysis for the typical software developer a practical proposition. As a consequence, from a design perspective, there are detailed issues conceiving the use of UML for system specification as part of the integrated process; a related PERMABASE paper [5] will focus more directly on these UML issues.

The remainder of this paper will introduce the integrated design process for distributed object-oriented systems, then explain the PERMABASE concepts surrounding provision of an environment to support the process. The proof of concept PPME prototype is described, including how it has been realised using currently available tools and notations, and how they are used. Finally, we outline our experience of using the prototype on real case studies, illustrating particular problems, and conclude with a discussion of alternative approaches and issues concerned with fully realising the PERMABASE environment as a commercial package.

2. Performance analysis as part of the design process

Although people generally accept that performance is an important consideration, they invariably find excuses for why they cannot or will not do Performance Engineering (PE). Some of the common arguments are listed in Table 1, together with the standard responses from the PE community[1].

Argument	***Counter Arguments***
We'll fix it later	It will cost more later (20:1 cost of modifying code to modifying design).
(we're too busy now)	There are limits to how far you can tune a system before you hit a fundamental architectural limit.
	You risk compromising architectural principles.
We'll buy a bigger box	There may not be a big enough box if the architecture's flawed.
	You'll still have to pay for upgrade/ replacement costs.
	Smacks of a trial and error solution.
Hardware developments proceed at such a pace that we don't need to worry	Demand (workload or application functionality) always grows to exceed supply.
	Use of state of the art hardware and software technology dramatically increases the risks of performance failures because of inexperience.
We'll rely on designers intuition	It's a risk – optimum or even acceptable solutions may not be apparent particularly in a distributed environment.
	There's no single natural OO design for an application independent of how it's deployed & the interface to an object will depend on the context in which its used.
It's extra work, we don't need it	Performance effort is always needed to meet customer expectations. If it hasn't been done by the end of the development period, it will have to be done then, risking late deployment, increased costs etc.
There are too many uncertainties	Yes, there are uncertainties, but there are also techniques to combat uncertainty (e.g. best & worst case estimates, sensitivity analysis, using a lifecycle/ iterative approach)

Table 1 - Reasons not to do Performance Engineering

[1] The authors gratefully acknowledge input from Connie Smith, Lloyd Williams and Mary Hesselgrave from WOSP '98 [6] on some of these. Also this topic is covered much more fully in [7].

Rather than rejecting PE and performance modelling as something a bit strange and different or just as a specialism to be kept at arm's length, we believe that the way forward is to integrate it into software engineering, so that it becomes part of regular practice – i.e. business as usual, not just an afterthought.

A potential difficulty with this integration is that a performance modeller's viewpoint can be somewhat different from that of a traditional software designer for the following reasons:

- Performance models are essentially models of behaviour. They generally focus on different aspects to design models intended to form the basis for construction, which often concentrate on structure. For performance modelling, a *black box* approach, where the structural details are hidden, is not just acceptable, it's desirable.
- Behaviour in a performance model is often specified in terms of probabilities not just prescriptive logic, due to abstractions that simplify the system (and therefore the time taken to produce results) and possibly because it is not possible to be prescriptive (at that particular point in the lifecycle).

There are however, natural synergies, which aid the integration of performance modelling into an *iterative* object-oriented design process, namely that:

- Performance modelling is about managing uncertainty. The best available data is used to construct the initial model, which is modified and refined, as more concrete or better data becomes available. This mirrors/correlates with the system design process, which starts with high level specifications and introduces detail as the design evolves. With the performance feedback, the refinement and elaboration of details can be focused early into key areas or likely hot spots.

A general OO process involves iteration around Analysis, Design, Implementation and Test phases as illustrated in Figure 1a. When designing a distributed system, the implementation and test phases prove to be extremely expensive (if not impossible) to carry out unless the system is deployed in its delivered environment.

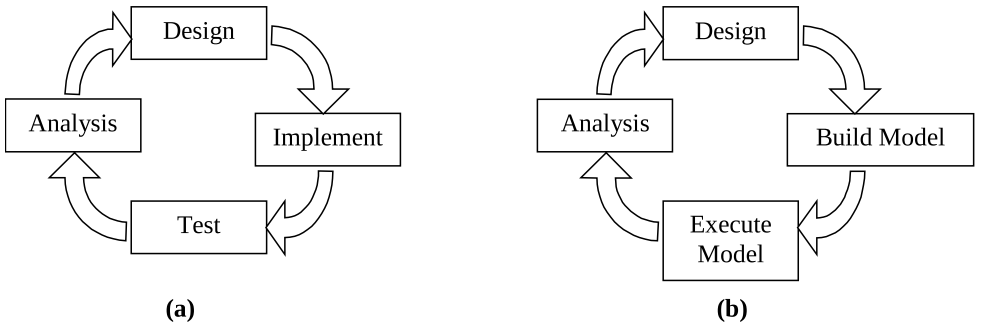


Figure 1 - A Generic design process (a) compared to a PM integrated process (b)

The idea of creating a performance model is to substitute (or augment) the feedback from testing a real implemented system with the analysis results of a performance model (Figure 1b). In fact, the performance model can give extra information, by experimenting with different configurations without the implementation costs.

3. The PERMABASE project

Requirements

To enhance the design process, we want to focus on *system* models that are rich enough to address performance issues amongst a range of other non-functional (and indeed functional) issues, yet simple enough to be tractable by users. The ideal situation would be for the performance model to be transparent to systems developers. They want the information it can provide, but should not need to understand the model itself. The PERMABASE project addresses these issues.

The main goal of the PERMABASE project has been the development of a proof of concept demonstrator that enables estimates of the performance of systems to be carried out throughout the development life cycle. An essential aspect of this is the automatic generation of performance models from system design models expressed in notations already in common use within the software development community. A future goal is to see the development of a commercial tool for performance prediction.

A support environment needs to provide the means to evaluate a system's architecture for performance implications at the early stages of design and, by following a lifecycle based approach, to act as a tool to help manage the risk inherent in the system development process. Such a modelling environment needs to be able to analyse systems for distributed applications and platforms and ultimately those which involve multimedia and run over high-speed broadband networks.

Solution concepts

A software-based system consists of application code and the physical environment in which the application runs. The performance of the system is determined by the application logic, its execution environment and the workload applied (both to the application and as background load to the execution environment). A performance model of the system requires design information from each of these specification domains (Figure 2) and, within the model, components from each domain will interact with components from the other two.

The design process (Figure 3) starts with the identification of Use Cases ([8]) within the system. The specification of the design models, for each domain, can then be carried out as three parallel activities, which proceed to be inter-linked. The final stage is the performance evaluation that feeds back results to influence the system design, which may be altered appropriately. The process iterates until satisfactory results are obtained.

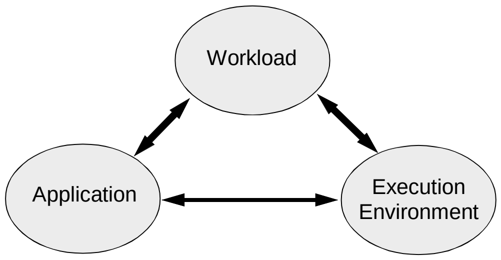


Figure 2 - The three PERMABASE specification domains

The application specification model defines the system's logical behaviour describing the precedence relations between processes carried out by the application. The user behaviour and other workload specifications characterise the order and frequency with which the actions defined in the Use Cases are carried out and the load that they generate. The Execution Environment (EE) specification model is constructed from a library of pre-defined components that characterise the physical platforms, and the networking technologies to connect them. The linkage between each of these separate domain specifications defines the interconnections between components from each domain.

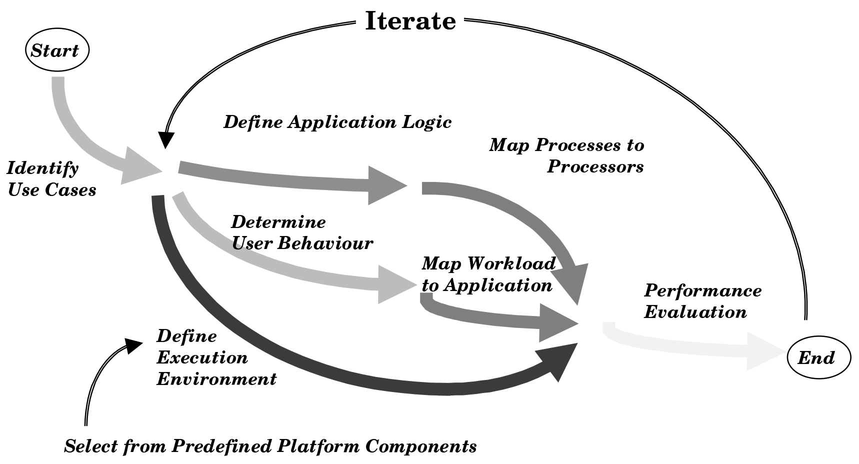

Iterate
Start
Identify Use Cases
Define Application Logic
Map Processes to Processors
Determine User Behaviour
Map Workload to Application
Performance Evaluation
End
Define Execution Environment
Select from Predefined Platform Components

Figure 3 - Design process with performance analysis

The predictions obtained from a performance evaluation are used to determine whether the system design meets its performance requirements. Any performance problems imply modifications to the system design and further iterations of the process continue until a suitable design is found. Further iterations are also necessary as design details are elaborated. In principle, more detail in the source (design) model enables more detailed and possibly more accurate performance predictions, so allowing an increasing confidence in the results over the lifecycle.

The detail tends to proceed from a high level to a low level in each of the three specification domains. For example, in the execution environment specification, networks may originate as high level ‘clouds’ with general properties for transmission time and latency and proceed towards detailed topologies including individual routers and physical network links. Workload component specifications can progress from representation as a simple workload source making requests for service at a specified rate (with variability expressed by means of a probability distribution) towards a detailed expression of user behaviour in terms of a state transition diagram. The Application specification is likely to progress from initial high level views of abstract software entities towards more detailed object models.

The performance model that is generated to assess the system performance will be evaluated by a combination of techniques, analytic and/or simulation models will be used, as appropriate to the level of detail (see table 2). The choice is dependent on the amount of information available and will produce a model with accuracy appropriate to the uncertainties associated with the current state of development of the system under consideration. This will avoid wasting effort creating, parameterising and solving unnecessarily detailed performance models. In general, it will be desirable to take a hybrid approach in which both analytic and simulation techniques are used to analyse the performance of a system.

Analytic	**Simulation**
Soluble equations, which provide metrics such as response time, throughput etc.	Enable models of detailed system behaviour.
Generally fast to compute.	Solution times can be lengthy.
Analytic models are not always possible to derive, especially for complex systems.	Complexity incurs high computational cost.
Appropriate for the early stages in the lifecycle where detail is limited.	Appropriate when more detailed results are required.

Table 2 - Comparison between analytic and simulation models

To aid the designer in understanding the results of the generated performance model, the results should be presented with respect to the original design specifications, possibly within the tools used to create the design, if they facilitate such an option.

A support environment for the design process is intended to complement current development methods and hence, it is preferable to make use of existing tools that support those methods, as this should enable it to be more readily integrated into an existing development process.

4. Prototype description

Architecture

The production of the PPME prototype required the identification of a number of viewpoints. The three specification domains define the various types of information that must be specified and provide a useful separation of concerns, but we need to determine how to enter (or show) this information from a designer's perspective. To be more definitive, what we need are a number of "viewpoints" which cover the combined specification required by the three domains.

To identify the set of viewpoints, we drew in part from the RM-ODP viewpoints ([13]) and in part from the requirements and concepts (identified above) of a distributed systems design specification tailored to enable performance model generation. This resulted in the following four viewpoints:

- Application Modelling Viewpoint – To specify the logical behaviour of the system, including software and hardware logic, and the behaviour of any other component considered to be an integral part of the system (e.g. a human operator in a directory enquiry system).
- Execution Environment Modelling Viewpoint – To specify the physical environment in which the behaviour executes, i.e. platforms, communication components, or other (bespoke) hardware. The definition of platforms includes processor, memory, operating system, plus any other platform-specific parts that support the Application components (e.g. ORB components). Communication components link the platforms, and could be high level network clouds or, in more detail, switches, routers, point-to-point links etc.
- Workload Modelling Viewpoint – To specify those components that do not form part of the system, but that interact with it, or drive it in some way. These could be users or other systems.
- The System Scenario Viewpoint – To specify the particular configuration of components that make up the particular system instance which is the subject of a performance analysis. This is in effect a "what-if?" scenario, and may be altered to consider or improve upon the performance of the system.

It is important to bear in mind that each viewpoint does not necessarily stand independently of the other viewpoints and any performance information must be considered in the context of the whole system. For example a performance bottleneck exhibited within an application process may not be the result of the way in which the application has been designed but may be the result of bottlenecks arising from the choice of execution environment.

Within the PPME, these *logical* viewpoints manifest themselves through *physical* tools, which form part of the Graphical User Interface (GUI) of the environment. A user is presented with a familiar environment of existing tools that have the necessary additions to support PERMABASE. This is seen as wholly preferable to learning an entirely new tool and set of skills.

The simplified architecture of the PPME is shown in Figure 4. The information representing each of the three specification domains is to be provided by the combination of the information specified in the four viewpoints, captured by the existing tool GUIs. Translation tools then convert the collected information into a Composite Model Data Structure (CMDS). The CMDS, together with internal transformation functions, forms a Kernel through which the flow of information, between the elements comprising the environment, can be co-ordinated. This architecture is supported and advocated by work in [14] comparing a number of different architectures for combining specifications from different tools and viewpoints.

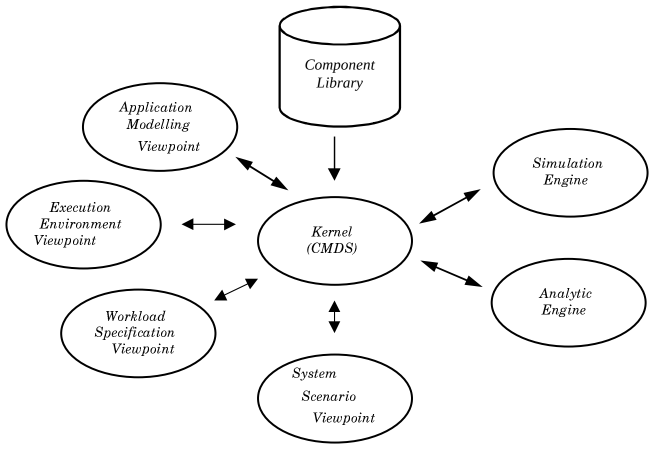

Component Library
Application Modelling Viewpoint
Simulation Engine
Execution Environment Viewpoint
Kernel (CMDS)
Analytic Engine
Workload Specification Viewpoint
System Scenario Viewpoint

Figure 4 - Simplified PPME Architecture

An additional key element is the component library that contains pre-defined component models for use in developing system level models. These can be components from any of the three specification domains, for example the Execution Environment domain can be provided with a library of different platform types.

The information in the CMDS is converted into the appropriate performance model (see Table 2 above). The two-way arrows between the CMDS and the alternative solution (simulation and analytic) engines in Figure 4 signify the upload of performance results to the CMDS after analysis of a model. The two-way arrows between CMDS and source model viewpoints signify the desire to present results back into the source domain. This enables the results to displayed to the user in association with the source model constructs to which they relate, e.g. platform utilisation in association with the relevant platform within the Execution Environment Viewpoint, response times for an object method within the Application Modelling Viewpoint.

Realisation

The latest version of the PPME is regarded as a 'second generation' prototype and addresses deficiencies encountered with earlier versions, which were too simple to cater for the needs of more complex case studies.

As a proof of concept prototype, issues such as usability and the performance of the prototype itself have not been considered as a high priority[2]. The prototype has not been designed to be a commercial product and ease of implementation has been the prime consideration.

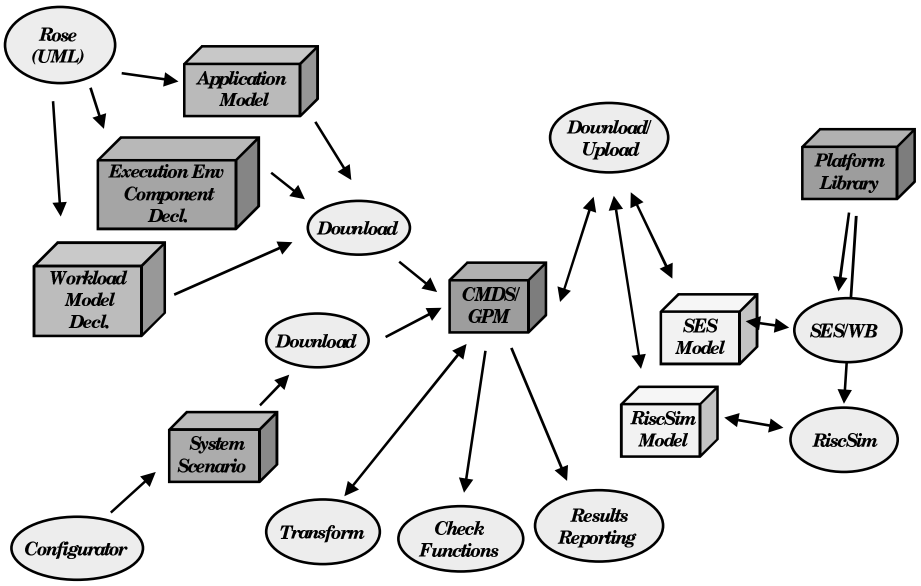


Figure 5 - KeyElements of the PPME protoype

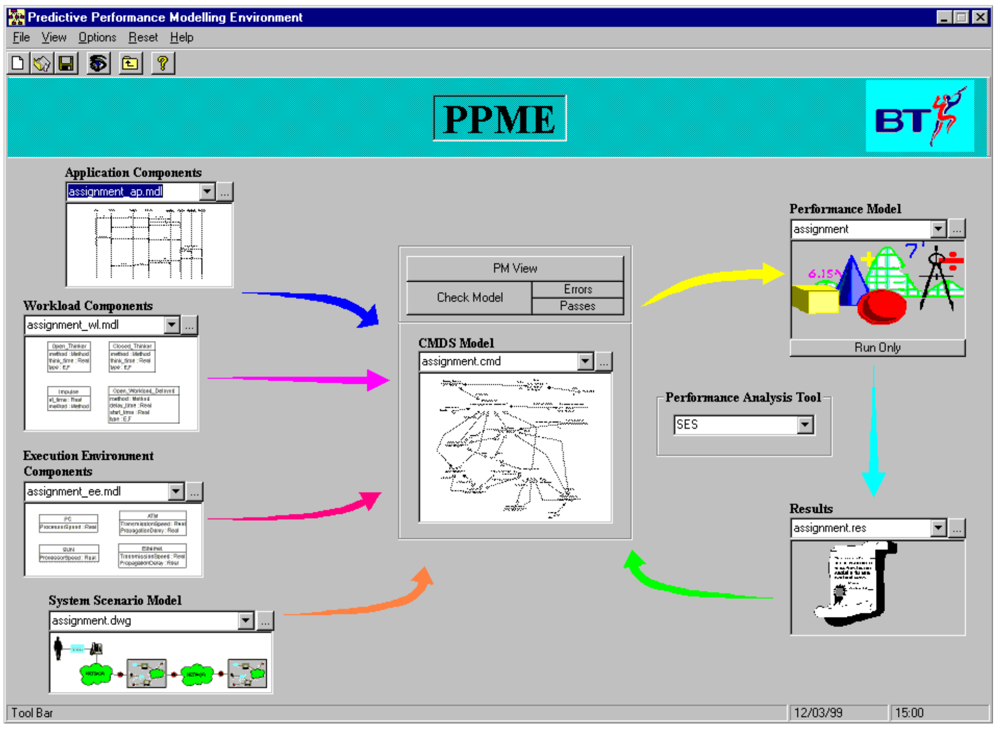


Figure 6 - PPME GUI

[2] The irony of this statement in the context of this paper is not lost on the authors.

The key components of the prototype are shown in Figure 5. The ovals represent processes or tools and the cube-like shapes represent data stores. Primarily, information flows from left to right in the picture starting from the source domains on the left-hand side and flowing into the performance analysis domain on the right-hand side. Figure 6 shows a screen shot of the PPME GUI, which is used to interface to the collection of tools that make up the environment.

We illustrate the purpose with an example based on a simplified (or early design) representation of a web like system concerning the retrieval of pages from a distant server, via an intermediate cache.

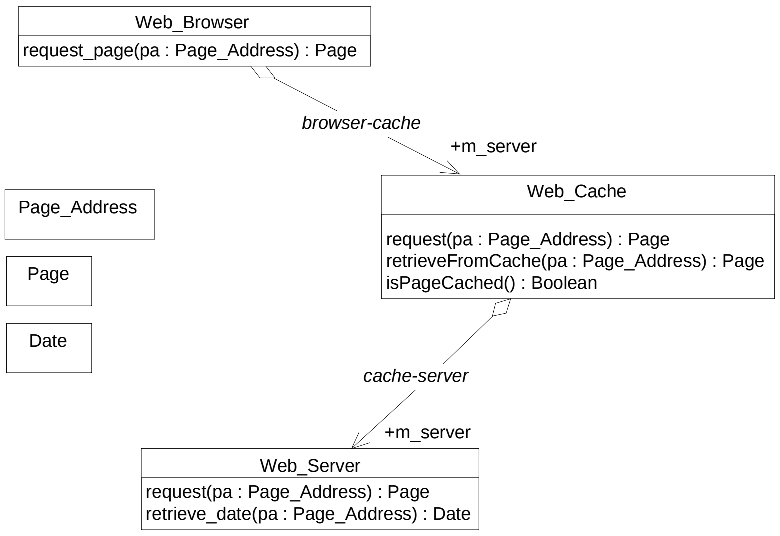

Web_Browser
request_page(pa : Page_Address) : Page
browser-cache
+m_server
Web_Cache
request(pa : Page_Address) : Page
retrieveFromCache(pa : Page_Address) : Page
isPageCached() : Boolean
Page_Address
Page
Date
cache-server
+m_server
Web_Server
request(pa : Page_Address) : Page
retrieve_date(pa : Page_Address) : Date

Figure 7 –Static Structure diagram for Web System

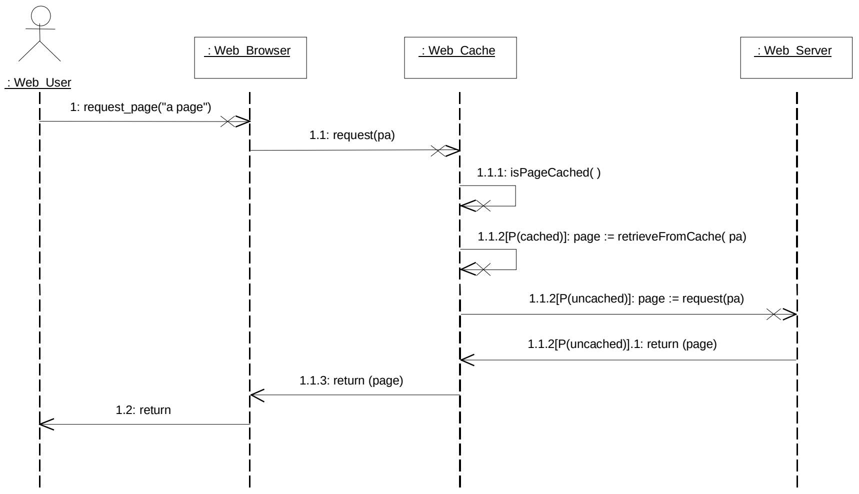
The Rational Rose OO analysis & design tool ([9]) is used to create UML diagrams. Its main purpose within the PPME is provide a model of the application logic which comprises:

- the static structure of the application in terms of classes the objects belong to and the packages defined within the application – Static Structure Diagrams (e.g. Figure 7), and
- the behaviour of objects in the application expressed in terms of interactions between objects (i.e. the method calls or message passing between objects), modelled in the source domain using Collaboration and Sequence Diagrams (e.g. Figure 8).

The combination of diagrams collectively define:

- The conditions for interactions between objects, specified either deterministically (i.e. a specific condition holds) or probabilistically (i.e. the probability that the interaction will occur, specified as a number between 0 and 1).
- The parameters passed between objects.

- Estimated resource usage, as part of class definitions by characterising the expected needs of object methods (e.g. in terms of their processing time demands) and the data size for messages (used to calculate transmission times across network links).
- Concurrency control of the behaviour defined within a class.



: Web_User
: Web_Browser
: Web_Cache
: Web_Server
1: request_page("a page")
1.1: request(pa)
1.1.1: isPageCached( )
1.1.2[P(cached)]: page := retrieveFromCache(pa)
1.1.2[P(uncached)]: page := request(pa)
1.1.2[P(uncached)].1: return (page)
1.1.3: return (page)
1.2: return

Figure 8 – A Sequence Diagram for Web System

Rose is also used to declare (re-usable) *components* to be used within any of the viewpoints (Figure 9 and Figure 10). These reside within the component library. Declarations take the form of UML class diagrams, a class with stereotype «library component», for which no behaviour is defined, is interpreted as declaring the existence of a performance model library component, which represents the (internal) behaviour of that component.

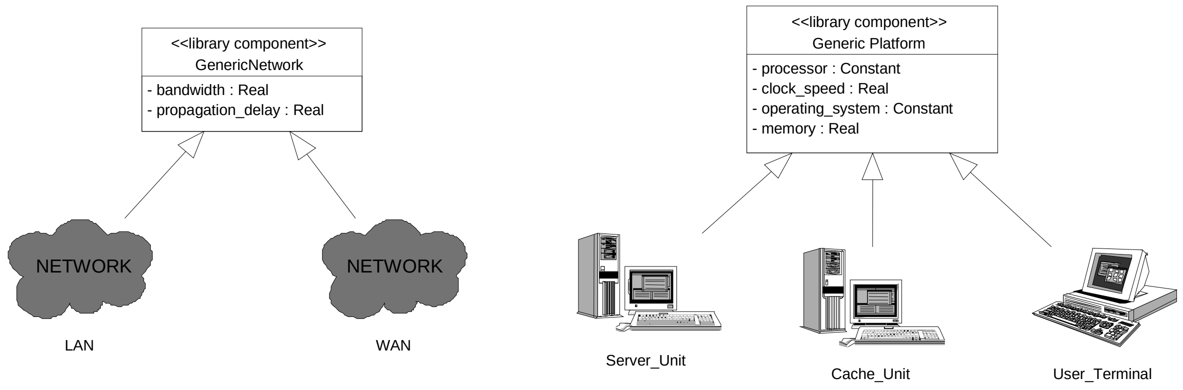


Figure 9 – Execution Environment Components for Web System

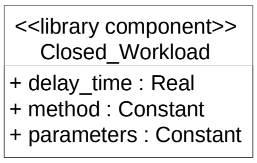


Figure 10 – Workload component for Web System

Typical components in the execution environment domain might be PCs, workstations, switches and networks of various types. Example workload components might be a simple closed workload model or an open workload model, (both workload models periodically make requests on the system, the closed workload waits for a response, the open workload does not). An example of application component models are, basic data types such as Integer and Boolean, Container types (List, Vector, Stack), specific operating system level or driver components, or bespoke components representing particularly complex behaviour that can be more efficiently modelled by hand.

Component class declarations may include attributes. If this is the case, values for these attributes are supplied via the system scenario diagram (see below) and will lead to run time parameters being supplied to the performance sub-model representing the component.

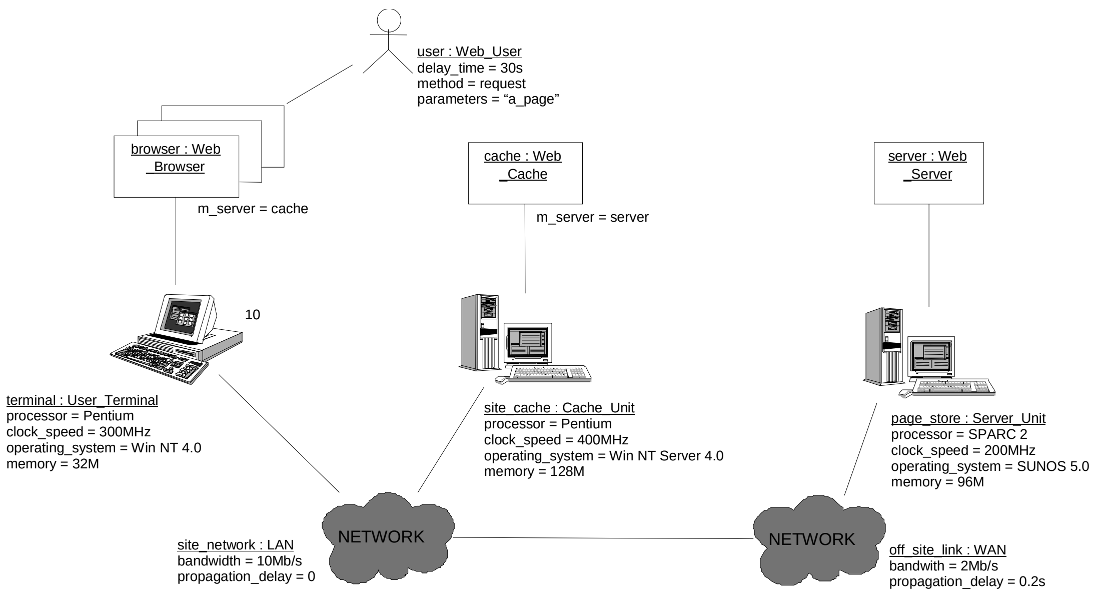


Figure 11 – Sample System Scenario Viewpoint specification

The other source tool currently used in the PPME is Configurator, developed by BT for network planning purposes. This tool is used to create the system scenario diagrams (a PERMABASE concept, e.g. Figure 11), which:

- Graphically represents the connectivity between physical elements within the system, defining the number of instances of the execution environment components, and supplying values for their attributes. (e.g. An execution environment component of type PC may have processor speed and memory declared as model attributes - values for these are defined in the system scenario diagram and used to supply parameters to the matching platform performance model in the component library.) The system scenario can be structured hierarchically by using a notion of subsystems to layer the definitions.
- Maps application objects onto the relevant host processor platforms from the execution environment.
- Connects workload components with the relevant application objects that they drive, and supplies values for the workload attributes. (e.g. Delay time may have been declared as a workload model attribute – values for this are defined in the system scenario diagram and used to supply a parameter to the matching workload performance model in the component library).

Deployment diagrams and component diagrams in UML are not currently used by the PPME. The system scenario diagram provides an alternate (richer) representation more suited to performance modelling needs. It is hoped that UML will be extended to offer the necessary capability in this area in the future although it is possible that the constructs available within collaboration diagrams are sufficiently general to enable a system scenario diagram to be constructed as a specialisation. (In actual fact Figure 11 has been drawn using a Rational Rose collaboration diagram rather than Configurator.)

Transformation

The transformation process 'downloads' the information into the CMDS, where a number of internal functions and transformations:

- create a class centric model of behaviour from the numerous collaboration diagrams, sequence diagrams, and pseudo code fragments defining the system behaviour. (This model will be described in more detail in a future paper, [12]),
- check the integrity and completeness of the system model, showing diagnostics to the user/designer,
- generate routing information for method calls across the distributed environment.

The final stage of the transformation converts the CMDS model of the system into a representation in the format required by the appropriate performance engine, pulling in any of the required library components. The performance model is executed, and the results fed back into the CMDS ready for presentation to the user/designer.

Currently, the performance model engines supported are either SES / Workbench ([10]), a sophisticated Discrete Event Simulation tool based on network queuing models from the US based company SES, or RiscSim ([11]), a simple, Petri-net based Discrete Event Simulator developed by UKC.

5. 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. Recent emphasis has been on validation of the prototype. 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.

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. As hinted at in section 2, 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 lifecycle of Figure 3, 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.

6. 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, [6]), 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 [15] and [16] both use UML as part of their performance model specification, and in particular a tutorial by Pooley ([17]) 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 is, at the time of writing, about to put out a Request For Proposals on performance in UML, so this looks to be a topic with a rising profile.

Other key messages from WOSP that resonate with the PERMABASE work were:

- The recognition of the importance of the evaluation of architecture for performance, i.e., the need for early design checks using high level models of system structure.
- The need for a shared process model covering both Software Engineering and Performance Engineering
- The importance of automated generation of performance models direct from designs.

7. 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 the ideas presented in this paper are all 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.

Thus this paper raises awareness of performance issues as an integral part of the object-oriented development process and moves the discussion into the mainstream. It is intended to encourage a debate on extending our understanding of what should be part of software engineering practice.

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

8. 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] Unified Modeling Language, version 1.1; *Rational Software Corporation et al*; September 1997; http://www.rational.com/uml and http://www.omg.org.

[5] David Akehurst; UML for Distributed Systems Performance Prediction; *in preparation for UML '99*; October 1999.

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

[7] Connie Smith; Performance Engineering of Software Systems; *Addison-Wesley*; 1990.

[8] Ivar Jacobson; Object-Oriented Software Engineering - A Use Case Driven Approach; 4th Edition; *Addison-Wesley*; 1994.

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

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

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

[12] Peter Utton, David Akehurst; Common Behaviour Notation - the key to the PPME prototype; *in prepatation for WOSP 99*; August 1999.

[13] P.F.Linington; RM-ODP: The Architecture; *In K. Raymond and L. Armstrong, editors, Open Distributed Processing II, Chapman & Hall*; pp.15-33; Feb 1995.

[14] Scott Meyers; Difficulties in Integrating Multiview Development Systems; *IEEE Software*; pp.49-57; Jan 1991.

[15] 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.

[16] 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.

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