Communications software performance prediction

Gill Waters, Peter Linington, David Akehurst and Andrew Symes

Computing Laboratory
University of Kent at Canterbury
CT2 7NF, UK

e-mail: A.G.Waters@ukc.ac.uk

Abstract

Software development can be costly and it is important that confidence in a software system be established as early as possible in the design process. Where the software supports communication services, it is essential that the resultant system will operate within certain performance constraints (e.g. response time).

This paper gives an overview of work in progress on a collaborative project sponsored by BT which aims to offer performance predictions at an early stage in the software design process. The Permabase architecture enables object-oriented software designs to be combined with descriptions of the network configuration and workload as a basis for the input to a simulation model which can predict aspects of the performance of the system. The prototype implementation of the architecture uses a combination of linked design and simulation tools.

1 Introduction

Software development and maintenance costs have for some years far outweighed those for hardware. The more complex the system, the greater the investment in its design and maintenance. Contributing to this are some of the intangible properties of software: it is hard to prove that it will work correctly until put in to practice and is subject to problems of poor or incomplete design or incorrect or incomplete implementation. For companies relying heavily on software, good design practice is essential.

Most software design procedures will include looking briefly at performance requirements and making estimates of the likely time to achieve results. Where a software system is designed to run on a single computing platform and the software does not require answers in real time, e.g. for payroll or invoice processing, it is a reasonably simple matter to estimate performance. This may be based on the number of lines of code or on measurements of an existing system before enhancement. Predicted problems may have a reasonably obvious solution, for example to invest in a faster processor.

Where the software supports distributed applications which communicate across networks there is frequently a requirement for it to satisfy real-time constraints - in many cases these will be soft real-time requirements such as the time taken to respond to customer requests to an on-line server or a database query. In others, deadlines must be met, especially where safety is involved, for example in control of aircraft or in railway signalling.

Software for distributed applications which is otherwise well-designed but does not meet such performance requirements is of limited use. The aim of the Permabase project is to enable performance predictions to take place early in the life cycle of communications application software design, so that over-ambitious plans can be modified or suitable decisions can be made about equipment or network provisioning. In other words, by putting performance prediction at an early stage, the design team should have increased confidence that investing effort in the implementation will be rewarded. Ideally as such tools become available, they would become an integral part of the development process.

The project, which is not constrained to looking at specific computing platforms or networks or at a specific application, has defined a generalised architecture through which applications designers can ask "What if?" questions about their software designs.

The Permabase project is a collaborative project which started in late 1995 and is led and funded by BT. The majority of the work is being carried out at BT Laboratories and the University of Kent. ERA Technology have also been involved, bringing their simulation experience to the project. The goals of the project are to produce a proof of concept pre-production implementation of a performance predictor, which would be suitable for reasonably wide application in organisations such as BT, for whom software is a major part of their business.

The rest of the paper gives more details on progress to date. In the following section a brief overview of related work is given. Section 3 elaborates on the requirements which have fed in to the the Permabase design. At this stage in the project, we have arrived at an initial prototype for Permabase, section 4 describes the concepts and architecture of this phase and discusses our experience of implementing and testing the prototype.

In producing the prototype, we have inevitably made simplifying decisions and taken some short cuts. In the next phase of the project we shall learn from these experiences and develop a new version which will more nearly satisfy the diverse requirements discussed in Section 3. The final section of the paper looks briefly at what we need to do to achieve this.

2 Related work

Although there are many models for communication systems and their components and some for analysing software performance, we are not aware of any which integrate all aspects of the system and performance to the extent that we do in this project. An Esprit project on an Integrated Modelling Support Environment (IMSE) completed in 1991 aimed to integrate modelling tools together and to automate the modelling process; the French company Simulog offered commercial versions of some of the IMSE tools. An object-oriented tool (PROTAB) for the modelling and prototyping of distributed systems was discussed in [1].

An industry perspective on the integration of performance prediction by members of the BT Permabase team [9] describes current practice and outlines a set of requirements for performance tools for the future. Our concepts of the integration of software performance with software analysis and design are also advocated in [10]. Further ideas on this, based on finding pathways through the logic represented by means of Object-Oriented constructs are described in [8]. The object-oriented design approaches described by Booch [3] is key to our representation of the application and the the treatment of use-case scenarios discussed in [5] has also been helpful.

3 Requirements for the performance prediction tool

In order to discuss the requirements, we first look at some example applications which will illustrate the potentially diverse situations to which such predictions can be applied. These are abstract examples for the purposes of discussion. In the project we shall be applying our ideas to software development projects within BT as case studies but these are not reported in this paper.

Our examples are an interactive video library retrieval system, a voice-conferencing system and information retrieval where the service is provided by a distributed server and cache arrangement. In general, the requirements will be to model applications which are highly distributed, interactive and may involve multimedia.

A video library system would potentially have a large number of customers each equipped with a set-top box. A high bandwidth network would be needed to deliver the video in real time, though the system may rely on some smoothing buffering at the receivers. Users would expect to be able to search for videos and retrieve them within an acceptable response time and would expect more rapid response to play, pause or rewind the video. The requirement for the Permabase system would be to capture the distributed software design both in the customer's equipment and in the video store and search machines. The expected user behaviour would need to be identified and modelled (e.g. frequency of scanning to find a new video, frequency of pauses and restarts etc.). The network and computing platforms would also have to be identified and modelled. Questions which designers may wish to ask include: "Given a systems with fixed parameters, what would be the average and maxim um response times?" "What is the lowest rate customer link which would satisfy video delivery and customer requests?" "How would an upgrade in the network affect the number of users or the response time?"

For the voice-conferencing system, similar information on software design and partitioning, network and workload would be needed. Here, typical speech Quality of Service characteristics would be needed, regular bit-rate and low delays to eliminate echos, low variability of delay etc. Performance prediction would relate to these QoS characteristics and questions might include: "In this situation, is one conference server sufficient?", "What is the maximum number of users within the QoS constraints?" etc.

For the cache/server example the nature of the system design would need to be captured at user, cache and server together with the interactions between them. Questions such as "What is the average response time for a certain size of cache?", "For what increase in population should a new cache be added?"

From these three examples, it can be seen that to offer a satisfactory prediction, a number of aspects of the design must be known in detail. Each example then has its own requirements for performance answers but the aspects which must be captured have common elements, e.g. descriptions of network topology and computing platform type and speed and a way of describing the load on the system. From these the system would combine the information and input it into a performance model which would then return the required results.

The types of questions which might be asked are likely to be refined during the life-cycle. Simple assumptions early on could lead to approximate results which would ensure sufficient confidence to carry on to a more detailed design to be evaluated by more in-depth performance models which might lead to refinement of the design.

To enable the system to be used for a wide variety of applications, the performance prediction environment must be acceptable to system developers who are not Performance Engineering experts. It should hide from them the complexity of the interlinked model components and of the performance models and yet should be flexible to use. To this end, we have tried to relate it to established design methods, notations and rules. Another key requirement is that results should be made available as quickly as possible. Of course this may not be very easy to achieve with a large simulation model. The procedure must be understandable by offering the users a view of the models in which the results are explicable to them.

In order to support these diverse applications it is necessary to offer an environment which is modular and reusable, e.g. to have component models for say PC or workstations which can be plugged in to the execution environment. The more modular the system, the easier it is to reuse models for different applications.

Our system must also support the abstract design and the progression of that design in to the elaboration of its components, so that it can focus on different aspects of the design at different stages. Since an Object-Oriented approach is the most useful one for modern distributed systems design, this is what we have chosen as a basis for the design representation in our system. Hierarchical design techniques also allow us to introduce increasing detail.

In the next section, we describe how the initial architecture has been designed in response to these requirements and a first version implemented.

4 Permabase architecture and initial prototype

From the previous sections, we can see that in order to model distributed software applications we need to capture the characteristics of the system in three domains: the application logic, the execution environment (consisting of both computing platforms and network topology) and the expected workload which will drive the systems as shown in Figure 1.

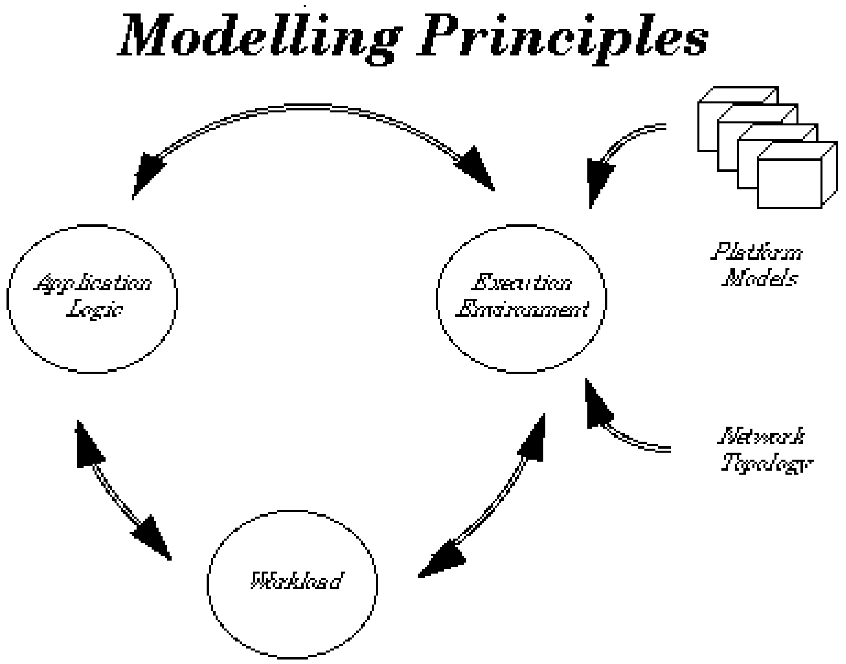

Figure 1:

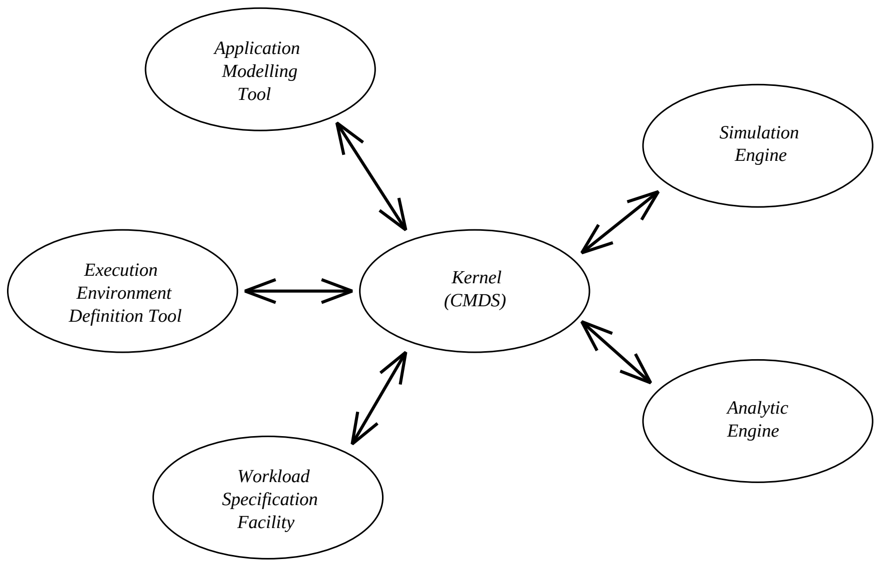

Figure 2: High level architecture

Our high level architecture (Figure 2) shows how the various components are linked together by a kernel which we call the Composite Modelling Data Structure (CMDS). The CMDS is a repository for all the relevant information from the models from the application modelling tool, the execution environment definition tool and workload specification tool and acts as a source of procedures, constraints and parameters for the performance model which may be carried out by simulation or analytical techniques.

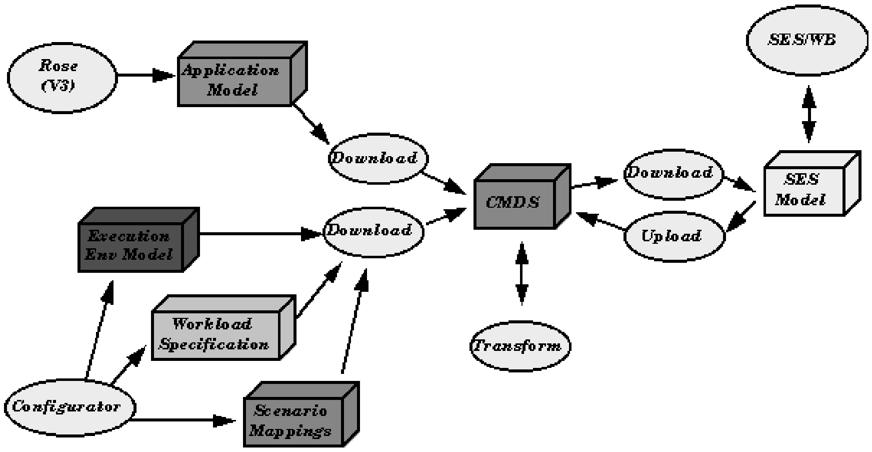

Figure 3: Expanded architecture

Figure 3 shows the choice of tools in the architecture. Tools have been chosen both for their flexibility and potential acceptability by software developers within BT. Object-oriented design is captured by the Rational Rose design tool [4]. Where necessary ROSE diagrams are annotated to offer sufficient modelling information. The key classes and objects are then translated in to the form in which they are stored in the CMDS. The execution and workload are both captured using BT's own Configurator tool [6] which we again extended to offer sufficient modelling information to be downloaded into the CMDS.

The information in the CMDS is then input to the SES Workbench simulation package [7] The simulation model has a number of simplifying assumptions, but allows simple applications described using the input tools to be analysed. Results are returned to the CMDS and can then be viewed by the user.

Ideally, the design process should proceed iteratively so that the results of an early performance evaluation captured from the component models can be fed through to modify or refine the system. These modifications can then be incorporated in the performance model to predict the likely effect of the changes. See Figure 4.

4.1 System prototype

An initial prototype was produced to test that complete performance models could be produced. The prototype was tested on two example systems. The first, elaborated below, was the client-cache-server model discussed in Section 3. The second was based on a network monitoring project and is not discussed further here. An independent performance model was devised for each of the target systems in order to validate the results.

For the cache-server model, the basic scenario modelled was a page request from a client to a server with distributed caching. The performance question asked was the end-to-end response time for user transactions.

The high level class diagram for the cache server model (not shown) represents three classes: one each for the client, cache and server. To provide the necessary events for our models, we can start by identifying use-cases which relate to the application. Typical examples for this application are request-page, replace-page etc. Taking the request-page use-case as an example,

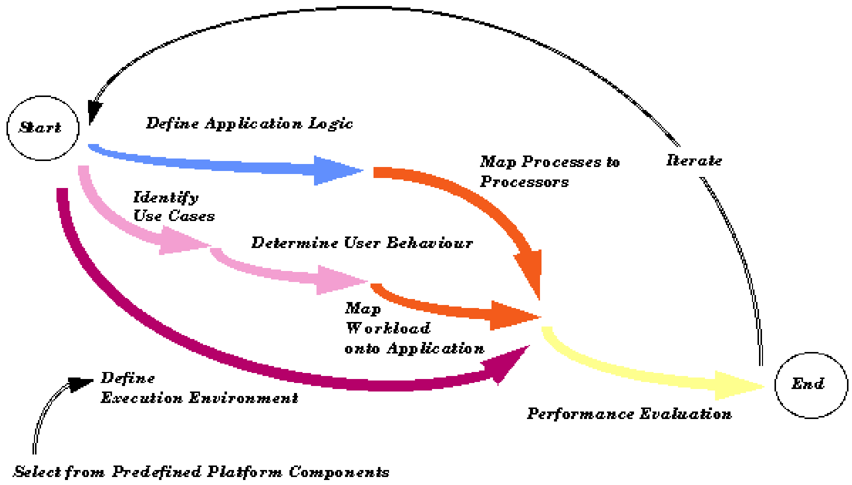

Figure 4: Iterative analysis and design process

the user's request may result in different interactions between system components depending on whether the page exists in the cache and how fresh it is.

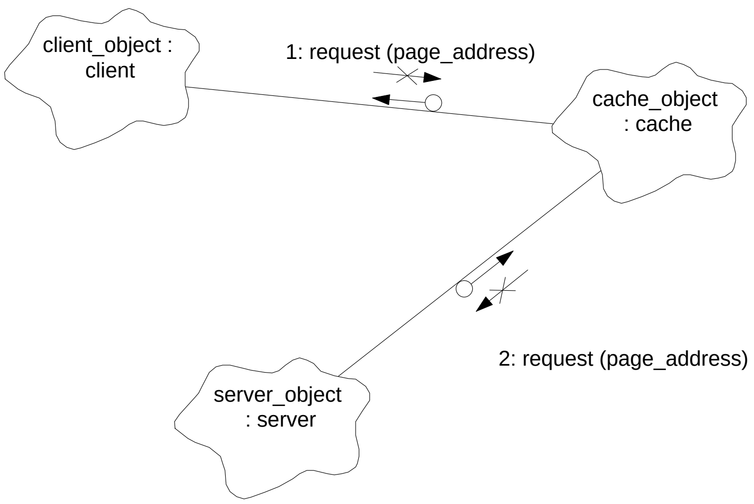

Figure 5: Scenario for requesting an uncached page

Figure 5 shows a ROSE Scenario diagram for "request page" where the page is not yet cached. The objects in the diagram are instances of the client, cache and server classes. The request for a page is sent to the cache object as a synchronous message. (The client waits for a reply.) The cache in turn sends a message to the server object to get the page. The server's reply contains the page which can then be returned by the cache to the client.

The messages need to be annotated with the length of the parameters passed, so that the performance model can quantify the time taken to cross any networks. Such annotation is available as standard in the Rose tool. Method calls on an object must also be annotated with the time they will take to invoke (as a number of work units) on the computing platform. This is added as extra textual information to the scenario diagram.

Figure 6 shows a Scenario diagram for a request for a fresh page. In this case, the response

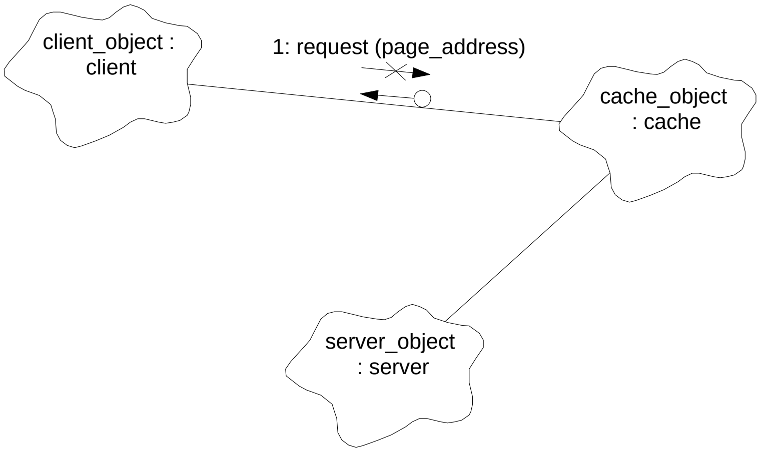

Figure 6: Scenario for requesting a fresh page

will be quicker as the cache object does not need to communicate with the server object. Further scenarios would deal with the cache checking the age of a page and sending a message to the server if necessary to ensure that it has the latest version.

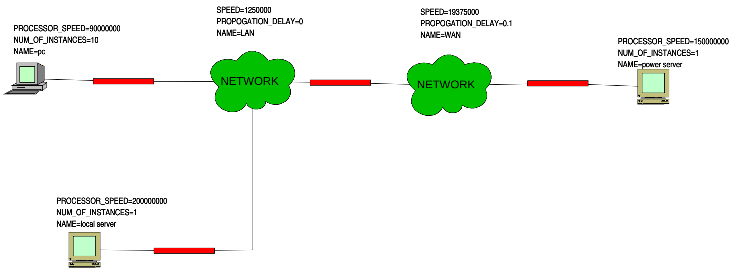

Figure 7: Execution environment

Figure 7 shows the execution environment for the cache-server application. Here, ten PCs are attached to a single LAN which also has a server. The LAN is in turn connected to a WAN leading to a power server. Again, textual annotations capture the relevant parameters about the PCs, the servers and the network.

Figure 8 shows the workload model. Two types of users are modelled, distinguished by instances of the name attached to the client class. The distribution of their request operations are quantified at this stage. Now that the system has input from the three main modelling components, the information needs to be "fused" together to specify how model components are mapped to each other. In the example shown, the workload Power_User would be instantiated as an object class Client on one of the PCs. There is no workload on the local server which is instantiated as an object class Cache on the local server, because all of its work is done in response to messages resulting from the client object's load.

The information in the CMDS about the fused model must now be translated into a SES workbench simulation model. This is done in two stages. The first produces an intermediate file format which reorganises the design into a format suitable for the development of a Discrete Event Simulation model. This is then processed into SES/query language which provides the low-

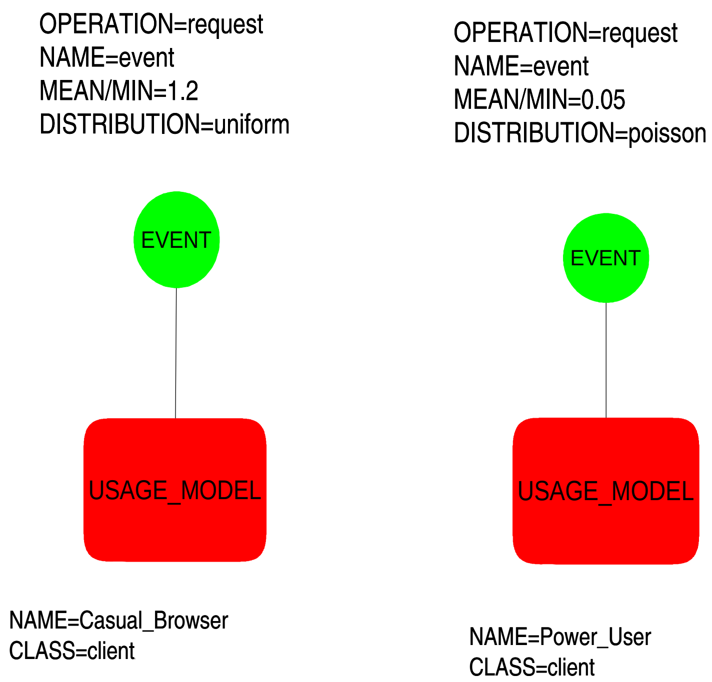

Figure 8: Workload

level functionality required to build an SES model. The model comprises workload, application and execution environment elements which are arranged for suitable hierarchical decomposition. For example, a class submodel will make reference to the execution environment as a client request event occurs at a client object. The execution environment will then make reference to other submodels related to the computer platform and the network. The SES model is thus built up in a modular fashion to form a complete model representing the designed system.

4.2 Testing

All stages of the above process were checked as part of the test procedures. The results produced by the system were then checked against an independently generated model for the same system using the same parameters. For the case shown, these results were in agreement within an acceptable margin of error.

5 Experience with the prototype

Our experience in putting together the initial prototype demonstrated, first, that it is possible to link the various aspects of applications design with the environment in which the system is expected to run and to produce useful performance evaluation models. This was achieved despite the obvious complexity of gathering information from a variety of tools to feed into another quite different tool in the form of the simulation package.

There were also a number of limitations with the prototype. First, the time taken to go through the various stages was several minutes even for the simple case being modelled. Secondly, because the prototype was deliberately kept simple, it was capable of working on a very limited set of applications.

None of the tools used contained all the facilities we would have wished for; this is to be expected as we were using general purpose tools. In particular, in order to get an accurate view of state information in the application, we had to define conventions for use of the Rose tool which included the user providing state diagrams at the design stage. We shall continue to minimise restrictions on the approach taken by the systems designer within the constraint of getting a valid model for the performance evaluation. The new Unified Modelling Language [2] is likely to help us with this when comprehensive tools become available.

Work is now progressing to the next phase of the project which will address these and other issues. Our vision was of an environment which could offer combinations of simulation and analytical techniques where each was available and appropriate. It is certainly our intention to keep the actual performance modelling as modular as possible so that this can be done. Other objectives are better checking for consistency across the models and an improved user interface, which more closely matches our requirement for answers to "What if" questions.

We shall also be augmenting the models so that they are capable of modelling high speed networks such as ATM so that the package will be suitable for the wide variety of applications which are likely to be designed for these networks in the near future. Such models will not necessarily be at cell level. Another addition will be the provision of stream models as the real-time delivery component of multimedia applications.

6 Acknowledgements

We would like to thank BT for funding the project and the members of the project team for their input: Peter Utton, Gino Martin and Brian Hill of BT and Adrian Johnson and Gerald Moran of ERA. We would also like to thank SES for supplying the Workbench licence.

References

[1] M. Baldassari, G. Bruno, and A Castella. Protob: an object-oriented case tool for modelling and prototyping distributed systems. *Software Practice and Experience*, pages 823–844, August 1991.

[2] G. Booch, I. Jacobsen, and J. Rumbaugh. The Unified Modeling Language for Object-Oriented Development, Document set version 1.0. *http://www.rational.com/uml*, 1997.

[3] Gradie Booch. *Object-oriented analysis and design with applications.* Benjamin Cummings, 1994.

[4] Rational Software Corporation. *Rational Rose Application Notes, Software Release 3.01.*

[5] I. Jacobsen et. al. *Object-Oriented Software Engineering - a Use Case driven approach, 4th edition.* Addison Wesley.

[6] Ian Pennock. *Configurator - User Guide, Issue 1.* N&S, AA&T, 1996.

[7] Scientific and Engineering Software Inc. *SES/Workbench release 3.1.*

[8] Connie Smith. *Performance engineering of Software Systems.* Addison Wesley, 1990.

[9] Peter Utton and Brian Hill. Performance prediction: an industry perspective. In *Performance Tools 97, the 9th International Conference on Modelling Techniques an Tools for Computer Performance Evaluation.* Springer, Lecture Notes in Computer Science 1245, 1997.

[10] L. Williams and C.U. Smith. Information requirements for software performance engineering. In *Quantitative Evaluation of Computing and Communication Systems*, pages 86–101. Springer, Lecture Notes in Computer Science 977, 1995.