Lecture Notes in Computer Science 2047

Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Springer
*Berlin*
*Heidelberg*
*New York*
*Barcelona*
*Hong Kong*
*London*
*Milan*
*Paris*
*Singapore*
*Tokyo*

Reiner Dumke Claus Rautenstrauch
Andreas Schmietendorf André Scholz (Eds.)

# Performance Engineering

## State of the Art and Current Trends

Springer

Volume Editors

Reiner Dumke
Claus Rautenstrauch
André Scholz
Otto-von-Guericke-Universität Magdeburg
Postfach 4120, 39016 Magdeburg, Germany
E-mail: dumke@ivs.cs.uni-magdeburg.de
{rauten,ascholz}@iti.cs.uni-magdeburg.de

Andreas Schmietendorf
Deutsche Telekom AG, EZ Berlin
Wittestraße 30 H, 13509 Berlin
E-mail: a.schmietendorf@telekom.de

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Performance engineering : state of the art and current trends / Reiner Dumke ... (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Singapore ; Tokyo : Springer, 2001
(Lecture notes in computer science ; Vol. 2047)
ISBN 3-540-42145-9

CR Subject Classification (1998): C.4, B.8, D.2, K.6

ISSN 0302-9743
ISBN 3-540-42145-9 Springer-Verlag Berlin Heidelberg New York



Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de



Typesetting: Camera-ready by author, data conversion by PTP-Berlin, Stefan Sossna
Printed on acid-free paper SPIN: 10781462 06/3142 5 4 3 2 1 0

# Preface

The performance analysis of concrete technologies has already been discussed in a multitude of publications and conferences, but the practical application has often been neglected. An engineering procedure was comprehensively discussed for the first time during the "International Workshop on Software and Performance: WOSP 1998" in Santa Fe, NM, in 1998. Teams were formed to examine the integration of performance analysis, in particular into software engineering. Practical experiences from industry and new research approaches were discussed in these teams. Diverse national and international activities, e.g., the foundation of a working group within the German Association of Computer Science, followed.

This book continues the discussion of performance engineering methodologies. On the one hand, it is based on selected and revised contributions to conferences that took place out in 2000:

- Second International Workshop on Software and Performance – WOSP 2000, 17 – 20 September 2000 in Ottawa, Canada,
- First German Workshop on Performance Engineering within Software Development, May 17th in Darmstadt, Germany.

On the other hand, further innovative ideas were considered by a separate call for chapters. With this book we would like to illustrate the state of the art, current discussions, and development trends in the area of performance engineering.

In the first section of the book, the relation between software engineering and performance engineering is discussed. In the second section, the use of models, measures, and tools is described. Furthermore, case studies with regard to concrete technologies are discussed in the third section.

The contributions published in this book underline the international importance of this field of research. Twenty contributions were considered from Venezuela, Spain, Cyprus, Germany, Canada, USA, Finland, Sweden, and Austria.

We would like to thank all the authors as well as Springer-Verlag for the good cooperation during the preparation of the manuscript. Furthermore, our thanks are due to Mrs. Dörge for her comprehensive editorial processing.

March 2001

Reiner Dumke
Claus Rautenstrauch
Andreas Schmietendorf
André Scholz

# Historical Roots of Performance Engineering

Initially, computer systems performance analyses were primarily carried out because of limited resources. *D. E. Knuth* discussed extensive efficiency analyses of sort and search algorithms in 1973 [5].

Although the performance of modern hardware systems doubles every two years, performance analyses still play a major role in software development. This is because functional complexity and user requirements as well as the complexity of the hardware used and software technologies are increasing all the time.

Performance engineering has many historical roots. The following historical survey sketches selected periods, all of which have contributed to the basis of performance engineering.

The Danish engineer *A. E. Erlang* had developed the mathematical basis for the utilization analysis of telephone networks by 1900. He described the relationships between the functionality of communication systems and their arrival as well as service times with the help of mathematical models. The queuing theory, with which runs in hardware systems can be modeled, is based on these mathematical basics.

*A. A. Markov* developed a theory for modeling stochastical characteristics of arrival and service processes in 1907. Markov chains, as a special case of Markov processes, have offered the basis for the coverage of IT systems with the help of discrete models for performance analysis since 1950.

Furthermore, *C. A. Petri* developed an important metamodel for the specification of dynamic system characteristics – the Petri net. The basic model has been extended over the years. Time-based Petri nets in particular are a basis for performance-related analyses and simulations of IT systems.

In the 1970s a series of algorithms were developed for the evaluation of these models. Most of them were very time-consuming [2, 8]. Henceforth, research has concentrated on approximate and efficient solution procedures [1].

The operational analysis, which was developed by *J. P. Buzen* and *P. J. Denning*, provided a practicable-analytical-oriented model evaluation [3, 6]. It could be used comprehensively and was integrated into many modeling tools.

Simulation is a further measure for the evaluation of IT systems. The use of conventional programming languages was replaced by the application of specific simulation language and/or environments. The simulation system GPSS was developed back in 1961 by *G. Gordon* in order to facilitate the capacity planning of new systems.

In addition, modeling measurements are also used for the determination of performance characteristics. *J. C. Gibson* proposed one of the first measurement-oriented approaches with the "Gibson instruction mix" at the end of the 1950s[7]. Measurements and benchmarks in particular, which have been used since the mid 1970s, are based on the transfer of a synthetic workload to a hardware system. The whetstone benchmark for the measurement of the floating point performance, which was developed by *H. J. Curnow* und *B. A. Wichmann* [4], as well as the Dhrystone-Benchmark for the determination of the integer performance, which was developed by *R. Weicker* in 1984 [10], are classical benchmarks. Only single system components, e.g., processors, were considered within the first phases of using benchmarks.

Current benchmarks of vendor-independent organizations, such as the SPEC (Standard Performance Evaluation Cooperation) or TPC (Transaction Processing Council), consider complex software and hardware architectures as well as user behaviors. They are the basis for the comparison of IT systems and provide performance-related data for the creation of performance models. ISO 14756 defined the first international standardized procedure for the evaluation of IT systems with the help of benchmarks in 1999.

*C. Smith* pointed out the problematic of a performance-fixing approach at the end of the software development (fix-it-later). With software performance engineering she proposes that the performance characteristics of IT systems consisting of hardware and software be analyzed during the whole software life cycle [9].

Reiner Dumke, Claus Rautenstrauch, Andreas Schmietendorf, André Scholz

## References

1. Agrawal, S.C.: Metamodelling. A Study of Approximations in Queuing Models. Ph.D. Thesis, Purdue University of W. Lafayette, IN, 1983.
2. Basket, F., Brown, J.C., Raike, W. M.: The Management of a multi-level non-paged memory System. Proceedings of AFIPS, AFIPS Press, 1970, p. 36.
3. Buzen, J.P.: Computational Algorithms for Closed Queuing Networks with Exponential Servers. Comms. ACM 16, 1973, 9, 527-531.
4. Curnow, H.J., Wichmann, B.A.: A Synthetic Benchmark. The Computer Journal, Volume 19, Oxford University Press, 1976.
5. Knuth, D.E.: The Art of Computer Programming. Vol. 3: Sorting and Searching. Addison-Wesley: Reading/MA, 1973.
6. Denning, P.J., Buzen, J.P.: The Operational Analysis of Queuing Network Models. ACM Computing Surveys 10, 1978, 3, 225-261.
7. Gibson, J.C.: The Gibson Mix. IBM System Development Division. Poughkeepsie, NY, Technical Report. No. 00.2043.
8. Kleinrock, L.: Queuing System Volume II: Computer Applications. Wiley & Sons: New York/NY, 1976.
9. Smith, C.U.: The Evolution of Software Performance Engineering: A Survey. In Proceedings of the Fall Joint Computer Conference, November 2–6, 1986, Dallas, Texas, USA, IEEE-CS.
10. Weicker, R.P.: Dhrystone. A Synthetic Systems Programming Benchmark. Communications of the ACM 27, 1984, 10, 1013-1030.

# Aspects of Performance Engineering – An Overview

Andreas Schmietendorf and André Scholz

University of Magdeburg, Faculty of Computer Science
P.O. Box 4120, Magdeburg, 39106, Germany
schmiete@ivs.cs.uni-magdeburg.de;
ascholz@iti.cs.uni-magdeburg.de

## 1 Motivation

The efficient run of business processes depends on the support of IT systems. Delays in these systems can have fatal effects for the business. The following examples clarify the explosive nature of this problem:

The planned development budget for the luggage processing system of the Denver, Colorado airport increased decisively by about 2 billion US$ because of inadequate performance characteristics. The system was only planned for the United Airlines terminal. However, it was enlarged for all terminals of the airport within the development without considering the effects on the system's workload. The system had to manage more data and functions than any comparable system at any other airport in the world at that time. Beside a faulty project management, inadequate performance characteristics led to a delay in the opening of the airport of 16 months. A loss of 160 000 US$ per day was recorded.

An IBM information system was used for the evaluation of individual competition results at the Olympic Games in Atlanta, Georgia. The performance characteristics of the system were tested with approximately 150 users in advance. However, more than 1000 people used the system in the productive phase. The system collapsed under this workload. The matches were delayed and IBM suffered deep-cutting image losses, whose immaterial damages are hard to determine. These examples show that the evaluation of the performance characteristics of IT systems is important, especially in high-heterogeneous system environments.

However, active performance evaluations are often neglected in industry. The quality factor performance is only analyzed at the end of the software development process [4]. Then performance problems lead to costly tuning measures, the procurement of more efficient hardware, or to a redesign of the software application. Although the performance of new hardware systems is increasing all the time, particularly complex application systems, based on new technologies, e.g., multimedia data warehouse systems or distributed systems, need an explicit analysis of their performance characteristics within the development process.

These statements are also confirmed by Glass. In an extensive analysis, he identifies performance problems as the second most frequent reason for failed software projects [2].

## 2 Requirements and Aims

The performance characteristics of a system have to be considered within the whole software development process. Performance has to have the same priority as other quality factors such as functionality or maintainability.

However, a practicable development method is necessary to assure sufficient performance characteristics. Extensive and cost-intensive tuning measures, that play a major part within most development projects, can consequently be avoided. The operation of high-critical systems, e.g., complex production planning and control solutions, depends on specific performance characteristics, since inefficient system interactions are comparable with a system breakdown, because all following processes are affected. The system user's work efficiency is impaired by inadequate response times, because of frustration. Additionally, software ergonomic analyses have shown that users, who have to wait longer than five seconds for a system response, initiate new thought processes. Controlled cancellations of the new thought processes and the resumption of the old condition take time and lead to a lower productivity of the user [3, pp.326].

The development method has to determine performance characteristics early within the development process to minimize performance-entailed development risks. A structured performance analysis is necessary. This should be supported by a process model, which has to be integrated within the existing company-specific software development process. Also, the application of this method should not be isolated from the development process, since additional activities, that accompany the software development process, are usually neglected when faced with staff and temporal problems.

The fused models should not become an inefficient complex. An economical application must still be guaranteed. Expenditures should be justifiable in relationship to the total project costs [6].

The size of the expenditures is often based on empirically collected data and knowledge. The deployment of qualified employees, who have a high level of knowledge in the areas of software engineering and performance analysis, is imperative. However, methods and technologies must be shaped with the developer in mind.

## 3 The Performance Engineering Method

The described set of aims can be reached with the performance engineering method. Performance engineering can be defined as a collection of methods for the support of the performance-oriented software development of application systems throughout the entire software development process to assure an appropriate performance-related product quality. Performance engineering becomes an interface between software engineering and performance management. It is not a relaunch of long-standing performance management methods as were other engineering approaches that had already been successfully used in the area of telecommunication.

Performance engineering analyzes the expected performance characteristics of a software system in its early development phases. In the system analysis, software

developers, customers, and users define performance characteristics as service level objectives beside functional specifications.

The performance has to be determined and quantified by performance metrics. These are specific product metrics that can be derived from different system levels and perspectives. An internal and external perspective is often distinguished. Internal performance metrics refer to operation times, e.g., the number of operations per time unit needed to transfer ratios, like the number of transferred bytes per second, or to the utilization of system resources, like CPU or RAM. External performance metrics reflect the outside behavior of the software system with regard to executed functions. The metrics often refer to response times of concrete application functions and to the throughput. A mix of varied quantified metrics describes the performance characteristic of an IT system.

The negotiations of developers, customers, and users should be based on suitable and justifiable cost-performance-ratios [1]. Performance engineering already provides instruments for this phase, like rules of thumb. Unrealistic developments can be discontinued early on or can be renegotiated.

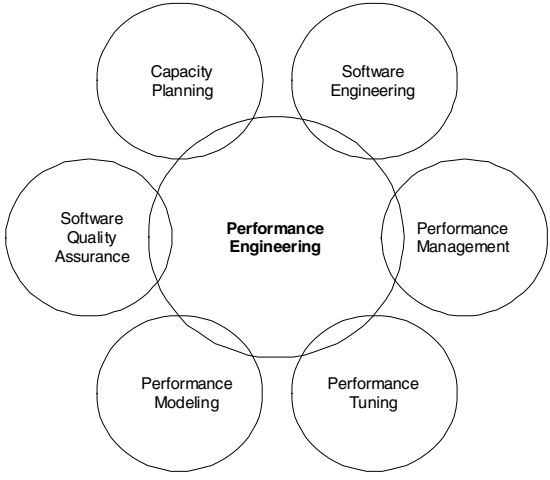



**Fig. 1.** Use of existing concepts of other disciplines

The quantified performance metrics have to be refined in the system design and implementation phase. Concrete application functions and interfaces can be evaluated at this time. The software development process should allow a cyclic verification of the performance characteristics. These will be analyzed in the respective phase of the software life cycle in the available granularity. In the first phases, estimations have to be used that are often based on rules of thumb. In further development phases, analytical and simulative models can be used. If prototypical implementations of individual program components are already available, measurements can be executed. The quantified metrics should be continuously compared with the required performance characteristics. Deviations lead to an immediate decision process. Performance engineering uses existing methods and concepts from the areas of performance management, performance modeling, software engineering, capacity planning, and performance tuning, as well as software quality assurance. It enlarges

and modifies them by performance-related analysis functions, cf. figure 1. However, the performance metrics are only as exact as the basis data of the models. In order to make a concrete and reality-based calculation, the set up of a performance database is imperative. Since a lot of performance data can be collected in the productive operation of a software system by benchmarking and monitoring, it should be assured that these data are stored in the database for further performance engineering tasks in future development projects. The quality of the performance evaluation depends decisively on the PEMM-level (Performance Engineering Maturity Model) of the development process [5].

However, critical software components are analyzed until the end of the implementation phase. The complete test environment is available within the phase of the system test. If prototypical implementations have not been used within the design and implementation phase, the fulfillment of the performance requirements can be verified with the help of load drivers, e.g., by synthetic workload, in the system test phase.

The real production system is available in the system operation phase. Often performance analyses are performed again in pilot installations over a certain time period, since the analysis is now based on real workload conditions. Thereby, an existing system concept can be modified again. The workload of the system often increases with a higher acceptance. That is why reserves should be considered within the system concept. The proposed procedure has to be adapted to domain specific environments.

Because of the scope of the tasks, specialists should support performance engineering. In further development projects, these tasks are handed over step-by-step to the software developers. Their long-term task spectrum widens. The integration can essentially be simplified if software developers come into contact with these principles during their academic education.

## References

1. Foltin, E., Schmietendorf, A.: Estimating the cost of carrying out tasks relating to performance engineering. In: Dumke, R., Abran, A.: New Approaches in Software Measurement. Lecture Notes in Computer Science LNCS 2006, Springer-Verlag Berlin Heidelberg, 2001.
2. Glass, R.: Software Runaways. Lessons learned from Massive Software Project Failures. Prentice Hall: Upper Saddle River/NJ, 1998.
3. Martin, J.: Design of Man-Computer-Dialogues, Engelwood Cliffs, NJ: Prentice Hall, Inc., 1973.
4. Rautenstrauch, C., Scholz, A.: Improving the Performance of a Database-based Information System. A Hierarchical Approach to Database Tuning. In Quincy-Bryant, J. (Ed.): Milleneal Challenges in Management Education, Cybertechnology and Leadership, San Diego/CA, 1999, pp. 153–159.
5. Schmietendorf, A., Scholz, A., Rautenstrauch, C.: Evaluating the Performance Engineering Process. In: Proceedings of the Second International Workshop on Software and Performance. WOSP2000. Ottawa, ON, 2000, ACM, pp. 89-95.
6. Scholz, A., Schmietendorf, A.: A risk-driven Performance Engineering Process Approach and its Evaluation with a Performance Engineering Maturity Model. In Bradley, J.T.; Davies, N.J.: Proceedings of the 15th Annual UK Performance Engineering Workshop. Technical Report CSTR-99-007, Research Press Int., Bristol/UK, 1999.

# Table of Contents

**Practical Experience**

# Conception of a Web-Based SPE Development Infrastructure

Reiner Dumke and Reinhard Koeppe

Otto-von-Guericke University Magdeburg, Institute for Distributed Systems,
Software Engineering Work Group,
PO box 4120, 39016 Magdeburg, Germany,
Tel.: +49-391-67-18664, Fax: +49-391-67-12810
{dumke,koeppe}@ivs.cs.uni-magdeburg.de,
http://ivs.cs.uni-magdeburg.de/sw-eng/us/

**Abstract.** In the area of software systems with their more or less restrictive performance requirements Software Performance Engineering (SPE) has gained a particular importance for the early development phases. Since the concrete performance behaviour can be determined at the implemented product only, furthermore usually in certain use cases only, there is a mixture of design activities (projection of the software model to the real hardware and software platforms) with specification and related requirements analysis activities.
Since knowledge of the performance behaviour of the real system is already required in early phases, several information sources have to be used. These sources can be the development of prototypes, the modelling based on special abstractions, the use of manufacturer information, the use of experiments or trend analyses for the system application.
This paper analyses existing models and methods for the performance-oriented system development based on a general framework, developed at Magdeburg University. The framework enables the derivation of possible assessable relations between software development components, thus guarantees respectively models an aspect-related system alignment.
As a source of information for the development process the WWW will be presented, as (external) Internet or as (internal) Intranet. The information resources used in this context with its passive and interactive characteristics lead to a software development infrastructure, shown here as a exemplary concept for the performance engineering.

**Keywords.** Performance Engineering, Software Measurement Framework, Software Development Infrastructure

## 1 Principles, Models, and Methods of the SPE

According to the „Encyclopaedia of Software Engineering“ [15] the Software Performance Engineering is defined as:

*"Software Performance Engineering (SPE) is a method for constructing software systems to meet performance objectives."*

R. Dumke et al. (Eds.): Performance Engineering, LNCS 2047, pp. 1–19, 2001.

To consider this target already in the early phases of the software development there is a set of fundamental ***principles*** and ***rules*** [15]:

**P01: Fixing-point Principle**: "For responsiveness, fixing should establish connections at the earliest feasible point in time, such that retaining the connection is cost-effective."

**P02: Locality-design Principle**: "Create actions, functions, and results that are close to physical computer product."

**P03: Processing Versus Frequency Trade-off Principle**: "Minimise the processing times frequency product."

**P04: Shared-resource Principle**: "Share resources when possible. When exclusive access is required, minimise the sum of the holding time and the scheduling time."

**P05: Parallel Processing Principle**: "Execute processing in parallel (only) when the processing speed-up offsets communication overhead and resource contention delays."

**P06: Centring Principle**: "Identify the dominant workload functions and minimise their processing."

**P07: Instrumenting Principle**: "Instrument systems as you build them to enable measurement and analysis of workload scenarios, resource requirements, and performance goal achievement."

**P08: Structuring Principle of Physical Concurrency**: "Introduce concurrency only to model physically concurrent objects or processes."

**P09: Tuning Principle**: "Reduce the mean service time of cyclic functions."

**P10: Data Structure Rule**: "Augment structures with extra information or change the structure so that it can be accessed more easily."

**P11: Store Precomputed Results Rule**: "Compute the results of expensive functions once, store the results, and satisfy subsequent requests with a table look-up."

**P12: Caching Rule**: "Store data that are accessed most often to make the cheapest to access."

**P13: Lazy Evaluation Rule**: "Postpone evaluation until an item is needed."

**P14: Packing Rule**: "Use dense storage representations to decrease storage cost by increasing the time required to store and retrieve data."

**P15: Interpreter Rule**: "Represent common sequences of operations compactly and interpret as required."

Additionally, there are models that allow to estimate the performance behaviour based on certain system descriptions. An exemplary list of these models can be seen below.

**M01: Timed flow grap**h (see in [6]) for presenting an element-related time evaluation in a program. The example of Hirvisalo is shown in figure 1.

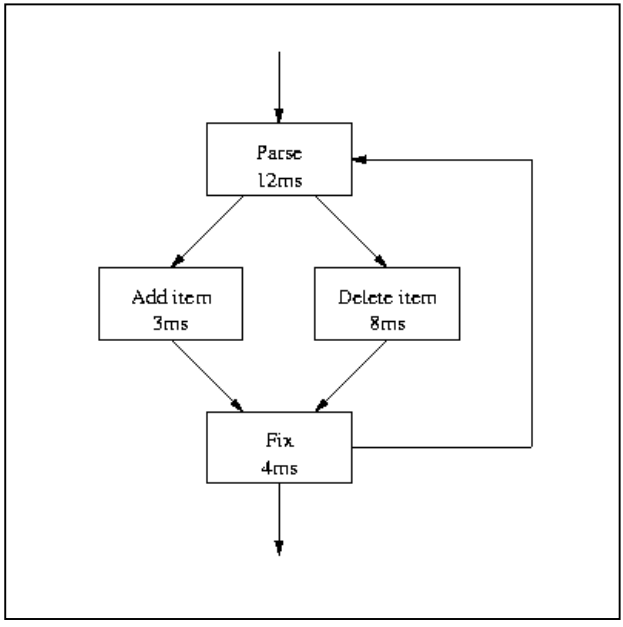



**Fig. 1.** Timed flow graph

**M02:** ***Timed petri net*** (see in [1]) for modelling the system behaviour including time-related characteristics with respect to performance aspects. Figure 2 shows a net form of Bowden.

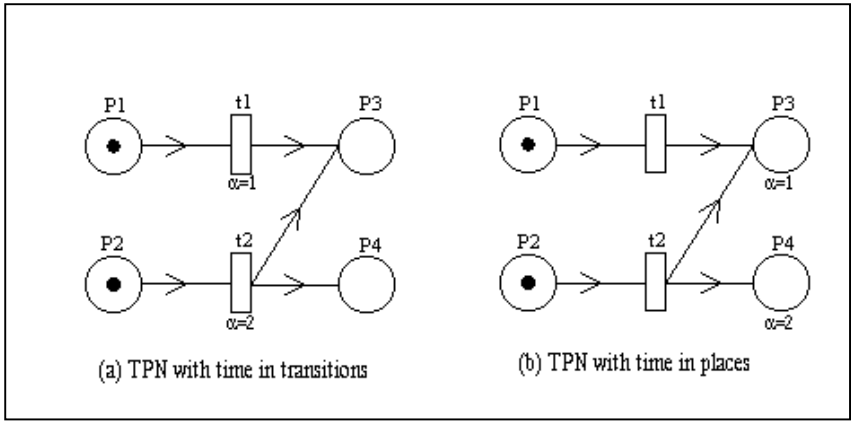



**Fig. 2.** Timed petri net

**M03:** ***Queuing network*** (see figure 3 based on [6]) for modelling waiting times in the system execution based on performance-related waiting system (TERM stands for possible service problems while saving CPU data on floppy disks).

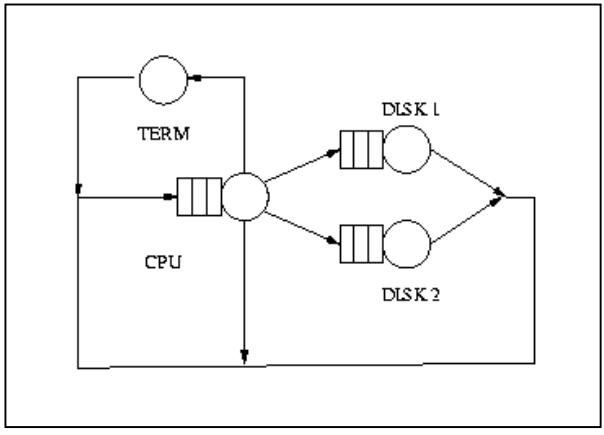



**Fig. 3.** Queuing network

**M04:** ***Extended queuing network*** (see in [6]) with new operation forms (REQ and REL) as extension, to enable the realisation of (time-related) selections. The modified example of figure 3 is given in figure 4.

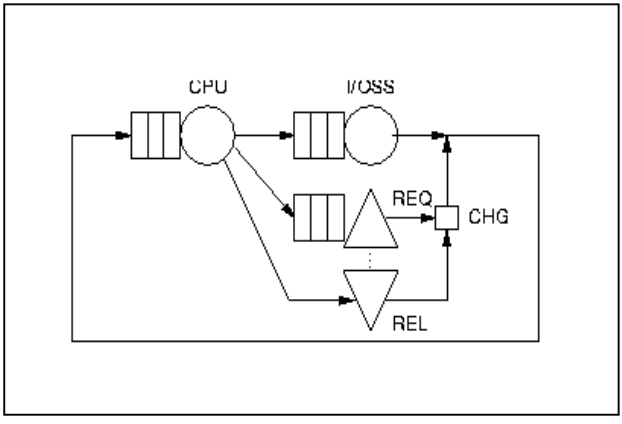



**Fig. 4.** Extended queuing network

**M05:** ***Markov state graph*** (see in [16]) for presenting state transitions (starting at an initial state), corresponding edge values represent performance aspects (see figure 5).

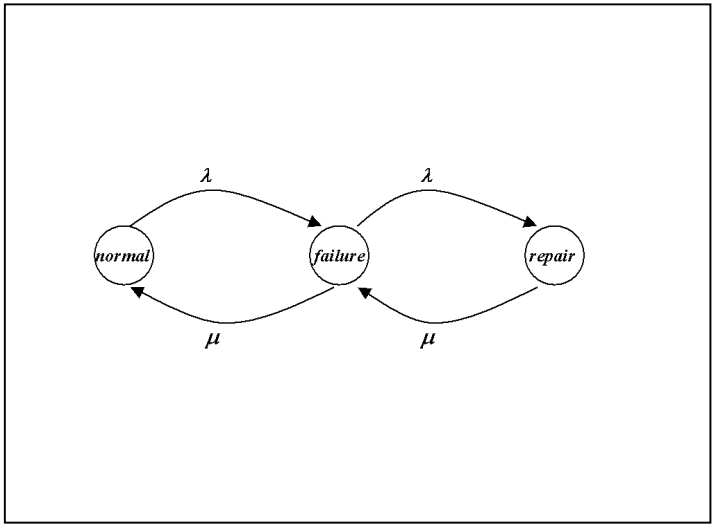



**Fig. 5.** Markov state graph

**M06:** ***State chart*** (see in [3]) based on markov state graph, additionally enables a hierarchical state representation with possible performance aspects. A simple example of a lift controlling description is shown in figure 6.

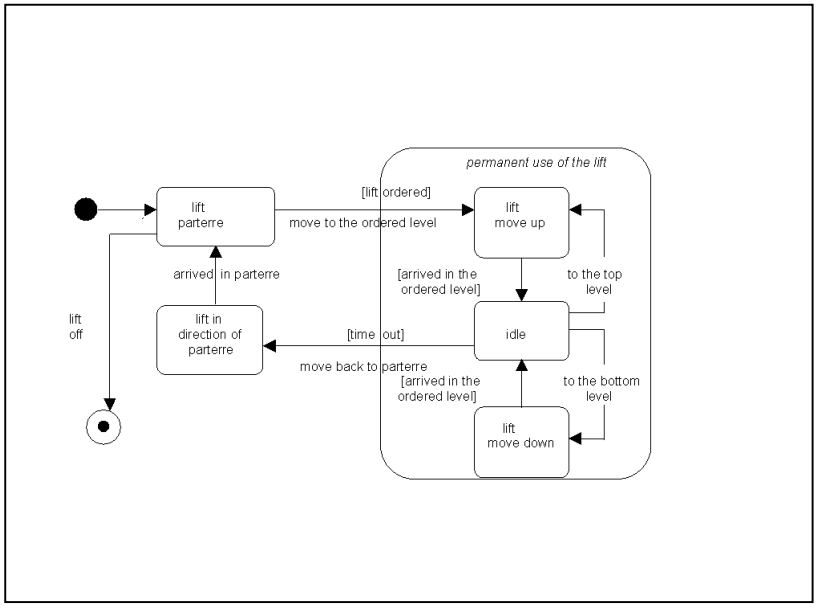



**Fig. 6.** Simple state chart

**M07:** ***SDL graph*** *(specification and design language)* (see in [3]) for modelling processes with explicit time description respectively given performance behaviour. A simple example of a SDL graph is given in figure 7.

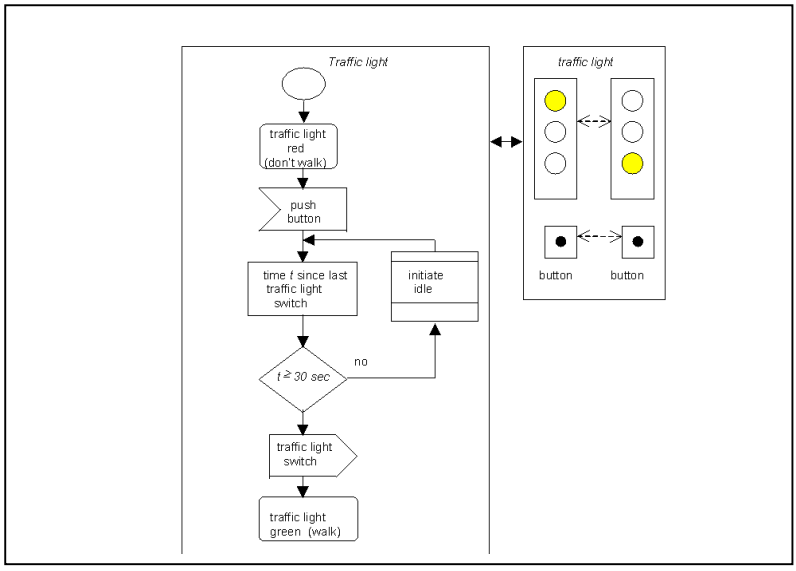



**Fig. 7.** SDL graph

**M08:** ***Sequence chart*** (see in [3]) for presenting the interaction or the communication of the involved objects with help of a lifeline, thus enabling performance statements (see figure 8).

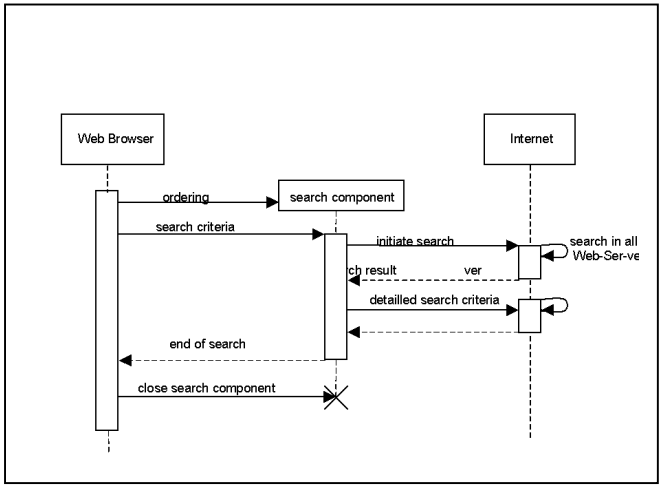



**Fig. 8.** Sequence chart

**M09:** ***Fuzzy graph*** (see in [11]) for modelling object or component relations by using fuzzy expressions (e.g. modelling of time tolerances, see an example in figure 9).

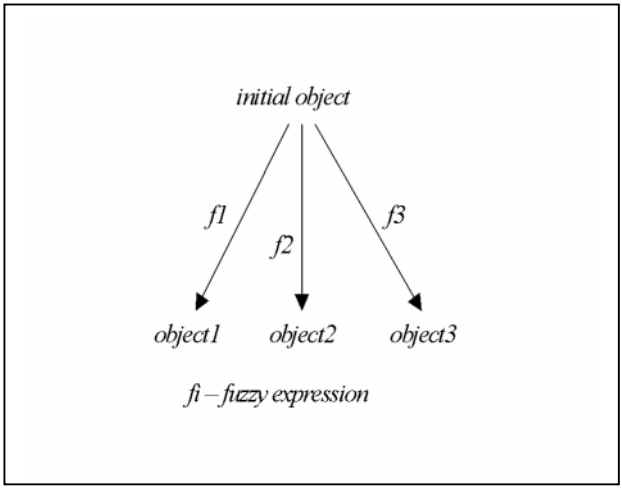



**Fig. 9.** Fuzzy graph

**M10:** ***Cache conflict graph*** (see in [10]) for modelling logical connections within programs (e.g. caused by performance-related aspects). The general form of such a conflict graph is shown in figure 10.

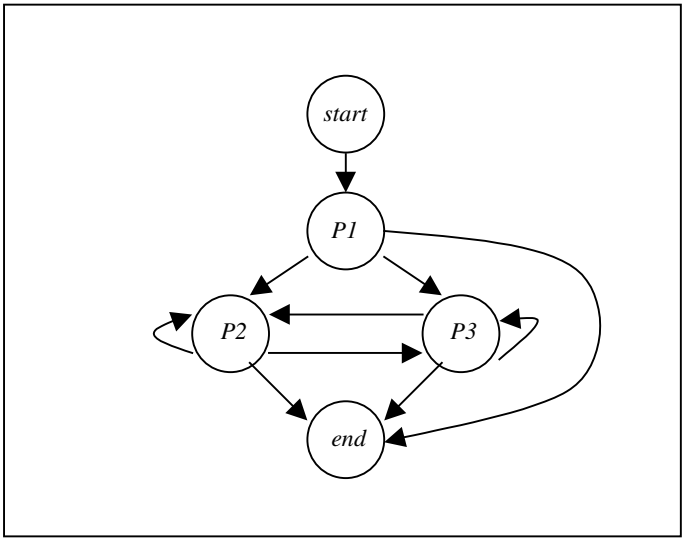



**Fig. 10.** Cache conflict graph

The next passage gives an overview about methods for guaranteeing the performance characteristics within the system development ([10], [13], [15], [17]).

**V01:** ***Simulation****:* A simulation can be realised on the basis of performance statements of (hardware) components or on the basis of the model of architectural or user-profile-related system characteristics. It gives a performance estimation. The accuracy of this estimation depends on the model and the (measured) input data.

**V02:** ***Monitoring****:* The monitoring observes systems that are operating by immediate measurements with respect to the belonging system architecture and the different measurement levels. It can be event or time oriented.

**V03:** ***Program path analysis****:* This analysis enables the evaluation of individual program paths by running and measuring the program. Performance estimations and assessments can be realised with help of supposed user profiles.

**V04:** ***Cache modelling****:* This approach enables the analyses of the widely-used cache technology. The estimation of the performance behaviour is realised with help of the behaviour of certain (cache-related) time cycles and their memory models.

**V05:** ***Pipeline modelling****:* The running program is considered as a similar sequence of running phases over the working period.

Finally, depending on the development phase in the software life cycle the methods, models and processes contain their specific characteristic as performance engineering with phase-related methods ([2], [7], [12], [15]).

**L01:** ***Capacity prediction****:* This procedure contains methods and models to predict or estimate the expected performance behaviour. It is used in the early phases of software development (especially requirements analysis and specification).

**L02:** ***Architecture-based performance evaluation****:* This evaluation includes methods related to the several levels of the system, e.g. network, client/server architecture or WWW application. In this way it is forming the basis for system selection and system design in the design phase.

**L03:** ***Code-based performance measurement****:* This method enables concrete statements of the performance behaviour in the implementation phase. Open aspects of the performance behaviour are determined with help of application and user profiles.

**L04:** ***Code-based performance tuning****:* This is a method for the performance improvement in the maintenance phase of a software system. With help of concrete user experiences the performance deficiencies can be fixed or the use opportunities and restrictions can be emphasised.

Following, the models and methods will be analysed with respect to the aspect of an integral software development.

## 2 Evaluation of the Initial Situation According to the Measurement Framework

The initial point is the measurement framework, developed in the SMLab at the University of Magdeburg. This framework enables determination and improvement of measurement and evaluation levels of all aspects taken into consideration in the

software development. As it can be seen in figure 11, the framework is embedded in a strategy and its ''core" are measurement tools (see [4]).

The CAME strategy serves for a successful embedding of a metrics tool in an IT area with its given measurement, evaluation and improvement goals. CAME stands for:

- ***Community:*** The existence of a group or team is necessary, having the motivation and the required knowledge to install a metrics program in a company. Such communities can have different experience backgrounds, e.g. the GI-Fachgruppe "Software-Messung und -Bewertung".
- ***Acceptance:*** The support of the company (top) management for installing a metrics program in an IT area is required, including the agreement to spend the required means and resources. Additionally, the management has to be clear, that not simply a couple of metrics will be introduced but also accompanying the introduction certain goals have to be defined and pursued, e.g. improvement of the product, of the software process or analysis of deficiencies.
- ***Motivation:*** It has to be guaranteed that already at the start of the installation of a metrics program respectively in the forefield, ''impressive" results can be shown to convince all participants, e.g. ''best practice" results from literature. The motivation plays an important strategic role, because it can be decisive for manner and amount of the support. Another aspect is the possibility or impossibility of a single (quality or evaluation) number, that is often required by the management.
- ***Engagement:*** Engagement contains the certainty, that a continuous effort is needed from tool usage over extensive statistical analyses to measurement data management. The installation of a measurement system including control mechanisms has a separat cost factor.

While the CAME strategy aims to establish a metrics program in an IT area successfully, the **CAME framework** consists (according to its acronyms) of four aspects respectively chronological steps. It proceeds on the assumption that there is the explicit goal to establish software measurements and stands for:

- ***Choice:*** This step deals with the selection of metrics for their application area with respect to the (empirical) criteria or characteristics to measure. As a selection technique GQM (Goal Question Metric) can be used, because it provides an orientation for a metrics selection for certain (empirical) goals. Within the CAME strategy it is already used to formulate goals and questions.
- ***Adjustment:*** This point contains the analysis of the scale characteristics of the chosen metrics, the selection of adequate limit values, and, if necessary, the calibration and adjustment for the concrete application area. Additionally, if necessary, varying measurement forms have to be taken into consideration, e.g. model-based or direct software measurement, estimations or predictions.
- ***Migration:*** This step deals with the extension and modification of the chosen metrics for a certain application area, e.g. a metric-based workflow for all development phases, for all levels of product architectures or for all versions or forms of resources.

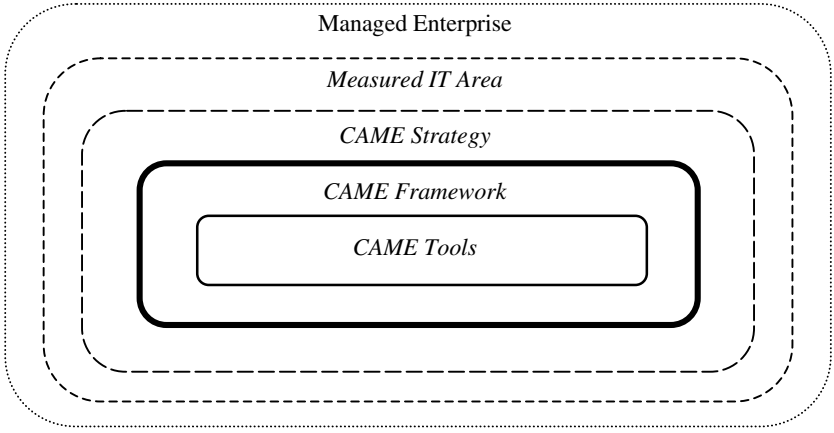



**Fig. 11.** Embedding in the SMLab Measurement Framework

- ***Efficiency:*** This point contains the automation of the metrics application with help of so-called CAME *(Computer Assisted software Measurement and Evaluation)* tools as far as the metrics allow the automation respectively as far as they are computable.

For the SPE analysis the metrics hierarchy is important, as it is defined in the first framework step. Their subdivision is derived from the dimensions of each component as shown in Figure 12. In this context, a component can be an aspect, a method or an object of software development, e.g. the software product, the process and the involved resources.

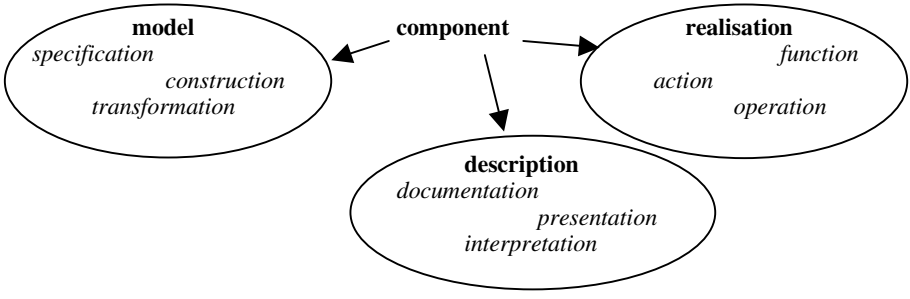



**Fig. 12.** Dimensioning of a software development component

Thus, the possible arrangement of the mentioned models, principles and methods of the performance engineering into the metrics hierarchy can be derived (see [3], [5], [9], [10]) in a manner shown in table 1.

The principles, models and methods which are applied in a particular development area will be determined by ***Choice*** or by the decision to develop and use additional methods, but this step will not be considered in this passage.

Next the framework step ***Migration*** will be discussed. Therefore, we have to analyse, if the existing respectively chooseable measurement and estimation methods for the performance behaviour already cover all aspects of the software development or show their relations correctly.

**Table 1.** Distribution of SPE models and methods

| *Measurement aspect* | *prinziple, method model, approach* | *adjustment* | *efficiency (CAME tools)* |
|---|---|---|---|
| **Product metrics** | | | |
| • ***architekture metrics*** | | | |
| - component metrics | **P09, M01** | *measurement* | SA/RT |
| - structure metrics | **P06, M03, M04, L02** | *estimation* | JMark |
| - database metrics | **P10, P12** | *modelling* | ◊ |
| • ***run-time metrics*** | | | |
| - task metrics | **P11, P13, M07, V05** | *modelling* | CrossViewPro |
| - data handling metrics | **P14, M09, M10, V04** | *modelling* | GPS4A |
| - human interface metrics | **M08** | *modelling* | Rational |
| • ***documentation metrics*** | | | |
| - manual metrics | ◊ | ◊ | ◊ |
| - development metrics | **V01, V03** | *modelling/measurement* | GPSS/ |
| - marketing metrics | ◊ | ◊ | Cinderella |
| **Process metrics** | | | |
| • ***management metrics*** | | | |
| - project metrics | ◊ | ◊ | ◊ |
| - configuration metrics | ◊ | ◊ | ◊ |
| - SQA metrics | **V02, L03** | *measurement* | CrossViewPro |
| • ***life cycle metrics*** | | | |
| - phases metrics | **P01, P15, M05, L04** | *modelling/measurement* | Cinderella |
| - milestone metrics | **L01** | *estimation* | GPSS |
| - workflow metrics | **P07** | *modelling* | UML/RT |
| • ***CASE metrics*** | | | |
| - method metrics | **P03, P05, P08, M02** | *modelling* | PetriNet |
| - paradigm metrics | **P02, M06** | *modelling* | Rational |
| - tool metrics | ◊ | ◊ | ◊ |
| **Resources metrics** | | | |
| • ***personnel metrics*** | | | |
| - skill metrics | ◊ | ◊ | ◊ |
| - communication metrics | ◊ | ◊ | ◊ |
| - produktivity metrics | ◊ | ◊ | ◊ |
| • ***software metrics*** | | | |
| - paradigm metrics | ◊ | ◊ | ◊ |
| - performance metrics | **P04, L02** | *estimation/measurement* | Microbenchmark |
| - replacement metrics | ◊ | ◊ | ◊ |
| • ***hardware metrics*** | | | |
| - reliability metrics | ◊ | ◊ | ◊ |
| - performance metrics | **L02** | *meaasurement* | Benchmarks |
| - availability metrics | ◊ | ◊ | ◊ |

From the above overview it can be seen, that possible influences of other components are not covered by the provided SPE models and methods. On the other hand this coverage is (still) impossible respectively can reduce the necessary metrics by taking advantage of certain connections. To find an adequate coverage for the SPE we relate to the Performance Engineering Maturity Model ***(PEMM)*** [14]. PEMM has five stages and can be arranged into the metrics hierarchy shown in table 2.

**Table 2.** Metrics-based characteristics of PEMM

| | ***PEMM-1*** | ***PEMM-2*** | ***PEMM-3*** | ***PEMM-4*** | ***PEMM-5*** |
|---|---|---|---|---|---|
| **Product metrics** | | | | | |
| • ***architekture metrics*** | | | | | |
| - component metrics | | | ♦ | ♦ | ♦ |
| - strtucture metrics | | | ♦ | ♦ | ♦ |
| - database metrics | | | ♦ | ♦ | ♦ |
| • ***run-time metrics*** | | | | | |
| - task metrics | ♦ | ♦ | ♦ | ♦ | ♦ |
| - data handling metrics | ♦ | ♦ | ♦ | ♦ | ♦ |
| - human interface metrics | ♦ | ♦ | ♦ | ♦ | ♦ |
| • ***documentation metrics*** | | | | | |
| - manual metrics | | | ♦ | ♦ | ♦ |
| - development metrics | | | | ♦ | ♦ |
| - marketing metrics | | | | | ♦ |
| **Process metrics** | | | | | |
| • ***management metrics*** | | | | | |
| - project metrics | | ♦ | ♦ | ♦ | ♦ |
| - configuration metrics | | | | ♦ | ♦ |
| - SQA metrics | | | | ♦ | ♦ |
| • ***life cycle metrics*** | | | | | |
| - phases metrics | | ♦ | ♦ | ♦ | ♦ |
| - milestone metrics | | | ♦ | ♦ | ♦ |
| - workflow metrics | | ♦ | ♦ | ♦ | ♦ |
| • ***CASE metrics*** | | | | | |
| - method metrics | | | ♦ | ♦ | ♦ |
| - paradigm metrics | | | | | ♦ |
| - tool metrics | | ♦ | ♦ | ♦ | ♦ |
| **Resources metrics** | | | | | |
| • ***personnel metrics*** | | | | | |
| - skill metrics | ♦ | ♦ | ♦ | ♦ | ♦ |
| - communication metrics | | ♦ | ♦ | ♦ | ♦ |
| - produktivity metrics | | | | ♦ | ♦ |
| • ***software metrics*** | | | | | |
| - paradigm metrics | | | | | ♦ |
| - performance metrics | ♦ | ♦ | ♦ | ♦ | ♦ |
| - repalcement metrics | | | | | ♦ |
| • ***hardware metrics*** | | | | | |
| - reliability metrics | | | ♦ | ♦ | ♦ |
| - performance metrics | ♦ | ♦ | ♦ | ♦ | ♦ |
| - availability metrics | | | | ♦ | ♦ |

For a complete SPE realisation with PEMM level three as minimal requirement, there are the following alternatives:

1. ***Method supplement*** to add missing methodologies of performance measurements and evaluations,
2. ***Method widening*** by using empirical connections between partial aspects of performance evaluation.

The empirical context can exemplary result from the rough empirical layer model shown in Figure 13.

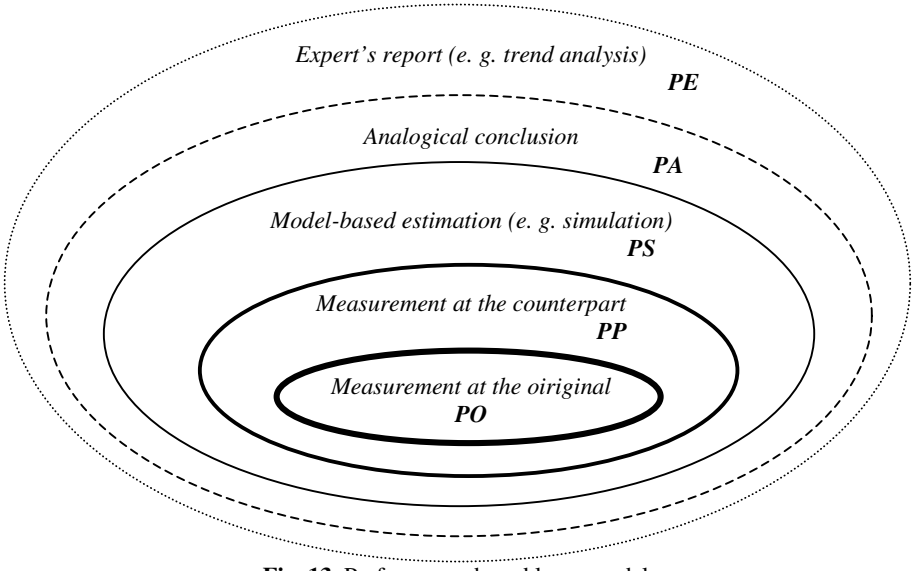



**Fig. 13.** Performance-based layer model

To make the empirical experiences accessible to the respective inner (i-1-te) level there is the universal formal context with a software system respectively a software component *k*:

$$(PerformanceBehaviour(k))_i = (CorrectionFactor \times PerformanceDetermination^{CorrectionExponent})_{i+1}$$

On the one hand, the more layers there are, the less precise the performance determination will be On the other hand, in the early development phases for partial components of a software system there are no alternatives to the given comparison statements. Examples for these transformations can be seen below.

**PP → PO:** The use of statements about the performance behaviour of a net-based application under laboratory conditions by applying monitoring as initial evaluation for the real application.

**PS → PO:** The simulation of a client/server application as initial basis for the performance evaluation of distributed systems.

**PA → PO:** The use of performance parameters of a LAN for the estimation of application characteristics in a WLAN.

**PE → PO:** The use of general trend predictions for the performance evolution for net-based platforms as basis for the estimation of the required performance of a software system to develop.

The transformations also can be performed step by step. Further time-related, multivariate and reflexive aspects will not be discussed in this passage.

A development environment for SPE has to contain required CASE tools as well as additional components to provide the missing empirical evaluations or estimations according to the future/available performance behaviour. As possible source of information the WWW can be used, where the data can be accessed as shown in table 3.

**Table 3.** Examples for information sources in the WWW

| Kinds of documents | Possibilities of information lacks |
|---|---|
| Tool descriptions for publicity of the tool vendor | Overestimation of the existing system performance |
| Scienitific paper of special performance aspects | Simplification of the system characteristics |
| special (intranet-based) or public experience factories | Only useable for special platforms or special kinds of application areas |
| Tutorials about performance engineering and performance aspects | Only remarks to a simple approach with simplificate examples |
| General descriptions of standard organisations and communities | Not situable for current used platforms or intended by political aspects |

The different forms of application of these information resources lead from a simple CASE application to a more global development environment.

## 3 A Static and Open SPE Development Environment

A first approach of a SPE development environment will be explained in this chapter. Substantial components for the software development are the CASE tools for the product development as well as the knowledge about required aspects according to the process, the resources and the corresponding application area. These aspects should concern (at least the components) of a measurement system and the corresponding standards as far as they are not integrated in CASE tools yet.

Furthermore, the information basis should be extended by experiences and trend overviews by so-called communities. Figure 14 shows the components of a SPE software development environment.

Table 4 shows exemplary links for the corresponding aspects. On this basis it is possible, to provide permanently the important external information for the development (with their mentioned faults and insufficiencies).

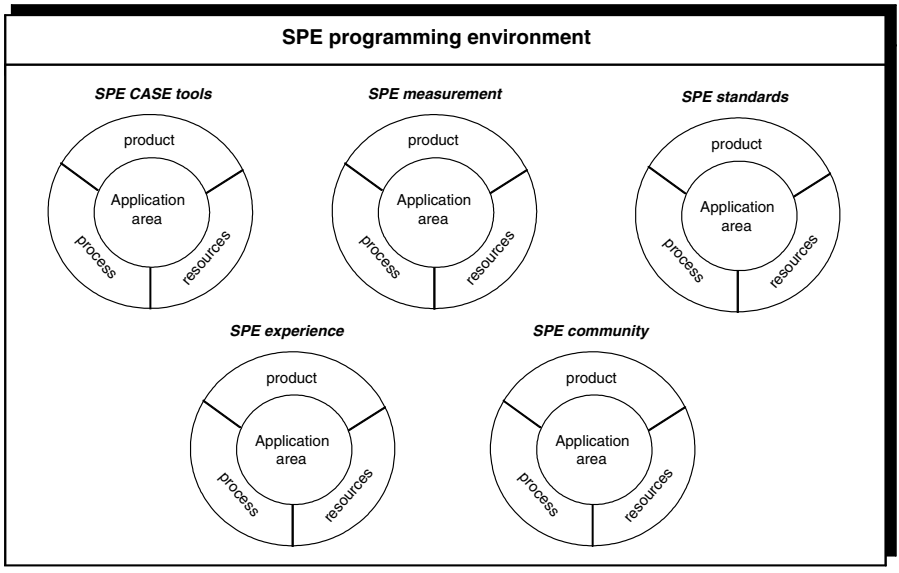



**Fig. 14.** Components of an universal software development environment

## 4 Conception of an Intelligent SPE Development Infrastructure

A simplified representation (see Figure 15) shows the new quality of a development infrastructure compared with a conventional development environment.

The stroke-dotted area characterises the classical content of development environments. For a development infrastructure, especially including external components and resources, the following environment features should be added:

- the continuous operational connection to external resources to provide information and knowledge,
- the temporary inclusion of external resources for system development, analysis and evaluation,
- the Internet-wide search for reusable components, systems or technologies,
- the temporary inclusion of external, personal resources für consulting and communication,
- the permanent information search with respect to new application areas and forms for the software systems to develop,
- the use of external resources for the task-related acquirement of knowledge respectively qualification.

Thus results a first approximation to an universal architecture of a SPE development infrastructure as shown in figure 16.

**Table 4.** Web links for a SPE development environment

**SPE CASE tools:**

***Product:*** NASA-Tool: http://techtransfer.jpl.nasa.gov/success/stories/modelcmp.html
***Process:*** TRDDC: http://www.pune.tcs.co.in/software.htm
***Resources:*** SDL Forum: http://www.sdl-forum.org/
***Application area:*** NCSA Perf: http://www.ncsa.uiuc.edu/SCD/Perf/

**SPE measurement systems:**

***Product:*** Sun tools: http://www.sun.com/books/catalog/Cockcroft/
***Process:*** NCSA SPE: http://www.ncsa.uiuc.edu/SCD/Perf/
***Resources:*** PES benchmarks: http://www.benchmarkqa.com/services.html
***Application area:*** High performance: http://www.hpc.gatech.edu/HPC-Home/hpcc.html

**SPE standards:**

***Product:*** SE standards: http://www.computer.org/cspress/catalog/bp08008/toc.htm
***Process:*** SPE book: http://www.phptr.com/ptrbooks/ptr_0136938221.html
***Resources:*** Dagstuhl report: http://www.dagstuhl.de/DATA/Reports/9738/node31.html
***Application area:*** SPE bibliography: http://peak.cs.hut.fi/research/full-bib.html

**SPE experience:**

***Product:*** CFP: http://www.cs.colorado.edu/serl/seworld/database/983.html
***Process:*** Tutorial: http://peak.cs.hut.fi/research.html
***Resources:*** Web performance: http://www.isi.edu/lsam/tools/autosearch/web_performance/19961016.html
***Application area:*** Applications: http://computer.org/proceedings/compsac/8105/81050166abs.htm

**SPE communities:**

***Product:*** SPE of Finland: http://www.prosa.fi/
***Process:*** WOSP98: http://www.sce.carleton.ca/wosp98/w98slides.html
***Resources:*** SIGMETRICS: http://www.sigmetrics.org/
***Application area:*** BSC Group England: http://www.bcs.org.uk/siggroup/sg44.htm

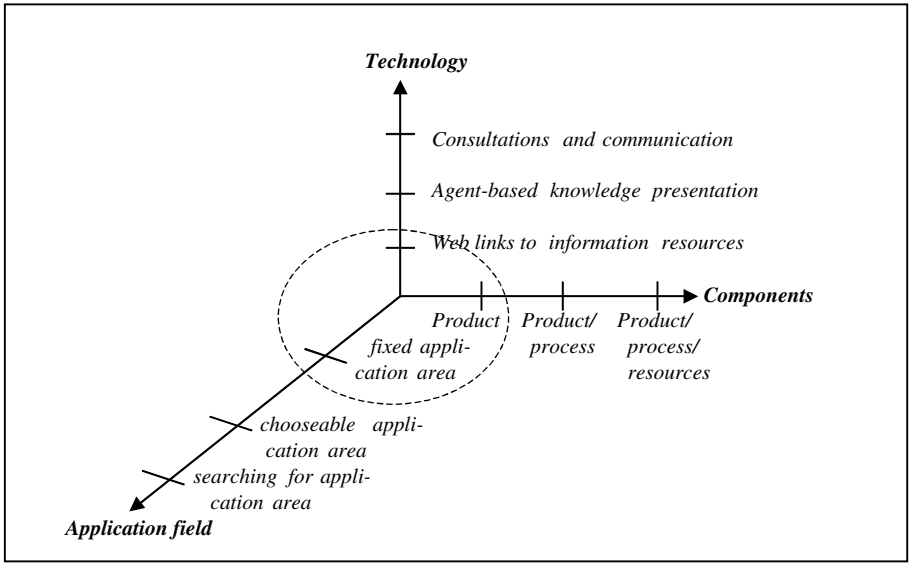



**Fig. 15.** Aspects for supporting the development

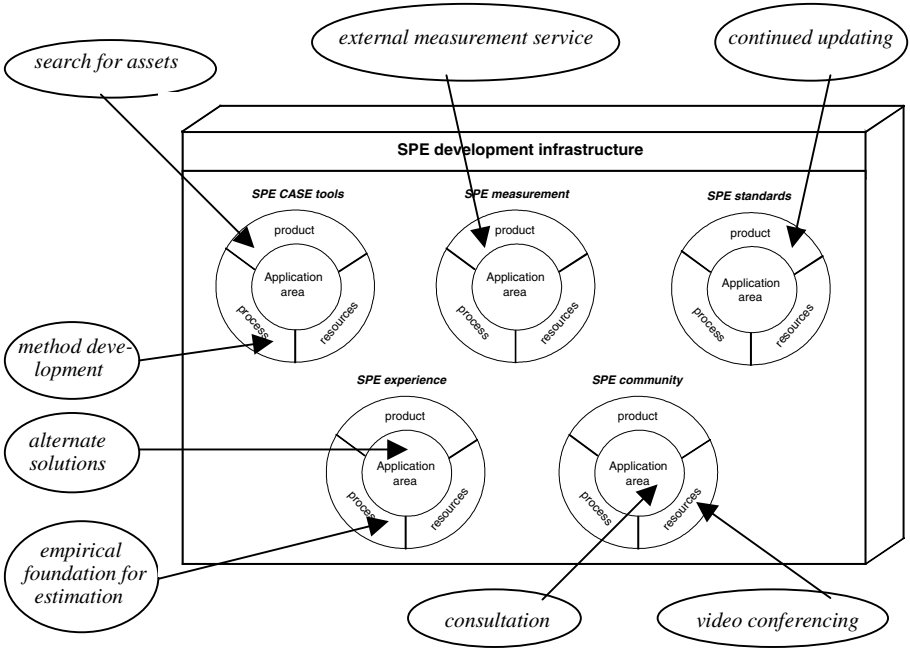



**Fig. 16.** Characteristics of a SPE development infrastructure

We call this development support as ***Infrastructure*** for ***Software Performance Engineering Environment Description,*** briefly **Infra-SPEED.** For the area of continuous information investigation and adaptation especially deliberative agents seem to be practical and efficient (see [8]). These agents e.g. could be used for the fault compensation.

## 5 Summary

This article evaluates the initial situation for the software performance engineering in a general form. The goal is a higher process level, possible to characterise the current situation with help of the Software Performance Maturity Model (PEMM). According to this goal we introduce a development infrastructure, using especially the growing information resource of the WWW.

For the implementation an agent-based solution is suggested, guaranteeing the necessary exploitation of information resources for the empirical evaluation compensation to improve the performance analysis.

## References

1. Bowden, D. J.: Modelling Time in Petri Nets. http://www.itr.unisa.edu.au/~fbowden/pprs/stomod96/
2. CSS Interactive: Capacity Planning White Papers. http://www.capacityplanning.com/whitepapers.html
3. Dumke, R.: Software Engineering – Eine Einführung für Informatiker und Ingenieure: Systeme, Erfahrungen, Methoden, Tools. Vieweg Verlag (2000)
4. Dumke, R.: Anwendungserfahrungen eines objektorientierten Measurement Framework. In: Dumke/Lehner: Software-Metriken – Entwicklungen, Werkzeuge und Anwendungsverfahren, DUV (2000) 71-93
5. Elkner, J., Dumke, R.: Konzeption und praktischer Einsatz eines Proxycache-Systems zur Erhöhung der Dienstgüte im WWW. Tagungsband der MMB'99, Trier, 22.-24. September (1999) 49-62 (siehe auch http://ivs.cs.uni-magdeburg.cs/~elkner/ webtools/)
6. Hirvisalo, V.: Software Performance Engineering. http://peak.cs.hut.fi/ research.html (1997)
7. Jahn, S.: Leistungsanalyse und Leistungssteigerung von Internet-Protokollen über ATM. Diplomarbeit, Universität Magdeburg (1997)
8. Jenning, Wooldridge: Agent Technology – Foundations, Applications, and Markets. Springer Verlag (1998)
9. Langendörfer, H.: Leistungsanalyse von Rechensystemen. Carl Hanser Verlag, 1992
10. Li,, Y.S., Malik, S.: Performance Analysis of Real-Time Embedded Software. Kluwer Academic Publ., London (1999)
11. Mori, M., Kawahara, Y.: Fuzzy Graph Rewritings. http://www.i.kyushu-u.ac.jp/~masa/fuzzy-graph/main.html
12. Schlaizer, M.: Entwicklung einer Aufbau- und Ablauforganisation für das Netzwerk-Performance-Management am Beispiel der DeTeCSM. Diplomarbeit, Universität Magdeburg (1998)
13. Schmietendorf, A.: Software Performance Engineering für verteilte Java-Applikationen. Diplomarbeit, Universität Magdeburg (1999)

14. Schmietendorf, A., Scholz, A.: The Performance Engineering Maturity Model. Metrics News, 4(1999)2, 41-51
15. Smith, C.U.: Performance Engineering. In: Maciniak, J.J. (ed.): Encyclopedia of Software Engineering. Vol. 2, John Wiley & Sons, 794-810
16. Tang, D., Hecht, X.: MEADEP and ist Application in Evaluating Dependability for Air Traffic Control Systems. Proc. of the IEEE Reliability and Maintainability Symposium, Anaheim, CA (1998)
17. Weber, T.: Analyse und Visualisierung des Einflusses ausgewählter Applikationen auf die Netzbelastung mit Hilfe des HP Open View Node Manager. Diplomarbeit, Universität Magdeburg (1996)

# Performance and Robustness Engineering and the Role of Automated Software Development

Rainer Gerlich

BSSE System and Software Engineering,
Auf dem Ruhbuehl 181, 88090 Immenstaad, Germany,
Tel. +49 (7545) 91.12.58, Fax: +49 (7545) 91.12.40
gerlich@t-online.de,
http://home.t-online.de/home/gerlich

**Abstract.** Performance engineering aims to demonstrate that the software being developed will meet the performance needs. The goal of robustness engineering is to prove that the system will work correctly in the presence or after occurence of faults or stress conditions. From this point of view robustness engineering complements performance engineering to cover the full range of conditions to which a system[1] may be exposed. Performance and robustness properties need to be continuously monitored during the development process to ensure that the system will meet the user's needs at the end. This paper will discuss aspects and problems of performance and robustness engineering. Also, it presents an approach, called „ISG" (Instantaneous System and Software Generation)[2] which allows to continuously derive performance and robustness properties from the system-under-development. In case of ISG figures are derived from the real system right from the beginning. Therefore deviations from the desired functional, performance and robustness envelope can be corrected at an early stage. The capability for getting an immediate[3] feedback from the system is obtained by the automated generation of the software from system engineering inputs. ISG builds the software by construction rules. This reduces the manual effort and allows for an immediate and representative feedback right after provision of inputs by the user. Due to automation the system can easily be instrumented on a user's demand without requiring any additional programming effort. ISG automatically stimulates the system and exposes it to stress tests and fault injection, and records coverage and performance figures. By incremental development a smooth transition from the first idea to the final version is possible at low risk. The ISG approach has been applied to the domain of real-time, distributed, fault-tolerant systems and shall be extended towards other application domains in future such as databases and graphical user interfaces.

**Keywords.** Performance engineering, robustness engineering, system validation, risk management, stress testing, fault injection, incremental development, real-time software

---

[1] A „system" as it is understood here comprises hardware and software. System development in the context of this paper addresses the software part, possibly hardware emulated by software and all aspects related to hardware-software integration. Performance and robustness engineering deal with the quality of service under nominal and non-nominal conditions of the software when executed on the system's hardware.

[2] 

[3] „Immediate" means a range of about 20 minutes up to 2 hours depending on the complexity of the system on an UltraSparc I/143 or PC 200.

R. Dumke et al. (Eds.): Performance Engineering, LNCS 2047, pp. 20–39, 2001.

# 1 Introduction

It is well known that software development bears high risks because the first feedback from the real system is available rather late in the development life cycle. The very popular V-Model (see section 4.1, Fig. 2) recommends a development approach for which the feedback from the first activities (specification, design, coding) is only based on documentation. Even if tools are applied which check the consistency and correctness of this paper work, they cannot guarantee that the system will meet all the user's needs at the end. The early conclusions on a system's properties are still based on the analysis of the documentation or the non-executed code. This analysis may be biased by wrong assumptions and misunderstanding of the written text. Consequently, as practice shows, not all problems can be identified before the system can give its own answer.

The main reason why a system does give a late answer on its properties is that a lot of manual effort is needed before a system can give this answer. To get an early feedback from the system, the current state-of-the-art solution is to reduce the required manual effort by simplification of the problem: a simplified model of the real system is built and the properties of the model are investigated assuming that such properties will be representative for the later real system. However, the key problem of this approach is the representativeness of the simplified abstract model.

ISG takes another approach. It speeds up the development process by automation. The manual effort is significantly reduced, the system is immediately available due to automated generation of (most of) the software and its stimulation for verification and validation. This way the system's complexity can be kept as it is, but the system's response time is drastically decreased.

The key issues of this paper are to discuss consequences of simplification and automation, to justify the need for performance and robustness engineering, and to compare the state-of-the-art development approach with ISG regarding risk management and manual development effort. Chapter 2 discusses means to get the desired representative „early feedback" on a system's properties, while chapter 3 addresses the validity of the results obtained.

Chapter 4 explains in more detail how a system is developed with ISG and which information is provided by automated stimulation and instrumentation.

Finally, chapter 5 gives a summary on the presented ideas and work and an outlook on future work.

# 2 Getting an Early Feedback from the System

A system's behaviour and performance depend on the chosen architecture, the distribution of the software across the architectural components, the available resources and - last but not least - on the processing of the data inputs and their conversion into outputs. The question on whether a system will meet all the user's requirements cannot be answered truely before the system can be executed. Consequently, the later a system becomes executable and the more effort is required

up to this point, the higher is the development risk. Therefore the challenge is to get an early response with little effort.

Especially, there is a high risk regarding the „non-functional“ performance and resource properties. They are already fixed by the design, but they can only be measured when the system is ready for execution. So the system should already be executable when it is still under design. To solve this conflict there are two known solutions to get an early answer: (a) the state-of-the-art approach of simplifying the system's functionality and the required manual effort, (b) the ISG approach to reduce the manual effort by automation still keeping the full functionality.

## 2.1 Fast Feedback by Simplification

The implementation of data processing algorithms is mostly considered as a very challenging task and most tools and methods concentrate on the so-called „functional“ data-processing aspects while they give less importance to „non-functional“ aspects like architecture, resources and performance or neglect them.

UML [16] is the most recent development method in the area of software engineering which collects the experience of the last decade. However, UML does not put very much emphasis on performance and the related risks, and while doing so it is completely in-line with the practice of the past.

The principal problem is that from a functional (logical) point of view a system may be considered as correct and consistent, but when running on the available architecture it may not satisfy the user's needs. Performance and resource constraints may cause serious problems so that the intended functionality cannot be provided at the end. Then a re-design is required and all the previous conclusions on correctness and completeness become obsolete. In consequence, a lot of money and time is lost.

Moreover, tuning of performance usually causes a higher functional complexity and may make the system less robust. This interdependency between functional and non-functional properties increase the development risk and make the need for an early and fast response from the system even more urgent.

The analysis of a system's performance and resource consumption cannot be done before it is really executable on the intended platform. But execution is only possible when all the code and the hardware is available. However, the provision of the code usually takes a long time according to the current manual development approaches for which the V-model is a representative method.

To escape from this problem and to minimise the development risk, methods such as „simulation“ and „rapid prototyping“ have been applied which deliver results within a short period of time at reasonable manual effort and costs. They are executed before the main development activity is started and shall allow for a prediction whether the planned processing algorithms and architecture will fulfill the needs.

Simulation is well known from other areas like electronics and mechanics and it delivers reliable results in these areas. Physical laws drive the behaviour and performance of electrical and mechanical components and they are well investigated. The computional methods for this kind of simulation are scalable. Simple systems can be simulated as well as large and complex systems.

This success has - possibly - lead to the conclusion that simulation is also well suited to perform feasibility analyses in the area of software systems in general. There

are a number of tools allowing for either analysis of behaviour like ObjectGeode [12], SDT [15] and (in former versions) STATEMATE [5]. or of performance such as OPNET [13] and SES/workbench [18] which allow to built a simplified model of the system. But no tool covers both, behaviour and performance. This makes it difficult to get a representative simulation.

Experience gained by a number of projects [1] shows that in case of a „decision-making“, event-driven software system, simulation may not predict a system's final properties. A number of examples which justify this conclusion are provided by the next section. The principal problem is discussed by section “The Problem of Representativeness”.

## 2.2 Some Counter Examples

*Ignoring Differences between Simulation and Target System.* An event-driven real-time system was investigated which consisted of a number of tasks which should read some parameters from file during the initialisation phase. SDL was used for implementation and simulation. The simulation of the SDL code could only be performed in a Unix environment and the conclusion was: the system is free of deadlocks.

Then the object code for the target system was automatically generated from the same source code for which simulation was done before. The real-time operating system for the (PC) target system was VxWorks [17]. Immediately after starting the system on the target platform it run into a deadlock. The reason was that the real-time operating system's behaviour for file I/O differs significantly from the one applied during simulation which was based on Unix. In fact, the behaviour of the I/O layer was not considered at all during simulation. The good point in this case was that this bug could immediately be identified due to the code generation capabilities of SDL. The transition from simulation to the target system was rather easy and lead to early detection of the problem.

*Ignoring Limited Resources.* A very high number of incoming messages had to be processed. Again, the simulation capabilities of SDL were applied to get an early feedback. SDL empties during simulation each input buffer before it enters the next simulation step. Therefore no problem occured due to overload. However, in the practical case the process did not get enough processor time to empty the buffer and a buffer overflow occured.

*Ignoring Processing Delays.* This example is described in detail in [4]. There are three nodes A, B and C which exchange data via a bus. Node A stimulates node B periodically to transmit data from B to C. At the end A sends a final message to C to check if the transfer was successful.

Without consideration of processing delays by the bus and the CPUs, the protocol was verified by simulation. No problem was detected. However, when the processing and bus delays were introduced it turned out that the final check from A to C arrived before the last data were sent from B to C. The problem was solved by delaying the final message from A to C by two bus periods. Hence, by ignoring of the processing time a wrong conclusion would have been obtained.

*Relying on Scalability of Performance Properties.* At the beginning of the 80's a large distributed database system should be built. The system engineers believed they could master any performance issues easily because they were applying a new promising technology of parallel computing. It was known that performance could be improved by increasing the number of computers in a range of 1 to about 3. However, to cover the peformance needs in this case they had continuously to add another computer exceeding the proven limits. Rather late they recognised that the overall performance did not increase linearly with the number of computers for arbitrary number of computers. When they were close to #10 no significant increase was possible at all. Therefore they could not meet the performance requirements at the end, and the company was close to bankruptcy. The project was canceled after about two years, and the loss (development costs and delayed service) was estimated to about 50 million DM.

*Ignoring Correlation of Events.* This is an iteration of example "Ignoring Processing Delays". Instead of considering bus slots, modelling only takes into account the transmission delay as usually done for performance modelling. Then the data are asynchronously put on the bus by node B, immediately after having received the message from node A. However, in practice, the data are transmitted synchronously by the bus slots. Hence, there is one empty bus slot at least before B puts the data on the bus due to the processing time. But no empty bus slot occurs between sending of the triggering message from A to B and provision of data by B in case of ignoring the periodic bus slots. Consequently, the probability is very low that A gets a chance to transmit the final check before B sent its final data. This also will lead to a wrong conclusion because the bug remains hidden.

To be more general: if two events, occuring with probability $p_1$ and $p_2$ during a given period, are not correlated the probability that a problem is detected which is related to both events is $p_1 * p_2$, while in practice it is $p_1$ if both events are correlated. Consequently, by wrong modeling a bug may not be detected because it occurs too rarely.

*Trying Representative Modelling in a Performance Analysis Environment.* During the HRDMS project [6] a rather complex real-time system was representatively modeled with the performance analysis tool SES/workbench. Although it supported efficient modelling the effort to built a representative modeling environment was rather high and nearly equivalent to the effort needed to build the real system.

**The Problem of Representativeness.** The conclusion on above examples is: the prediction by simulation deviates from the practical result when the underlying mechanisms are not well represented by simulation. The big problem is: either such mechanisms are unkown or ignored, not documented or confidential, or their consideration is as expensive as the real implementation.

Between simulation of software systems[4] and simulation of mechanical and electronical systems there is a principal difference: decision-making and event-driven

[4] In this respect we only address decision-making, event-driven systems such as real-time systems, distributed systems, database systems, user-front-ends

software systems behave extremely undeterministic, while physical laws are exactly known and allow for an accurate prediction.

In case of electronical systems the complexity and resource consumption may be reduced by cutting down the full spectrum of information to a limited bandwidth, because not all information is really needed. However, in case of software each piece of information is of same importance, it cannot be cut off without knowing its contents and its impact.

Consequently, the degree of representativeness of a simplified simulation of such a software system remains unknown until the results from the real system are available. Therefore no serious conclusions can be drawn from simulation results which are based on approximating analysis methods.

In the past simulation was applied to exploit intended implementation strategies and to learn about potential problems and bottlenecks. Also, it was common practice to apply simulation during the specification phase and to start from scratch again for the design phase. Obviously, in such a case no valid conclusions can be drawn on the properties of the system-under-design from the previous simulation results. The motivation of simulation only was to get a better understanding of the problem, but not to predict „guaranteed properties".

However, it is widely unknown that simulation - if understood as simplification of the problem - is not well suited to derive valid results of what a system's properties will be.

## 2.3 Immediate Feedback by Automation

ISG does simplification, too. But it does not simplify the problem, it simplifies the implementation by automation. Consequently, ISG removes all human activities between provision of system engineering inputs and reading of the evaluation report.

ISG introduces full reuse of (a) templates from which the system is built by construction rules, (b) library functions and (c) the process model for software development itself. Due to the full reuse the ISG approach is subject of continuous improvements.

No detailed knowledge on the software implementation is needed because ISG provides the infrastructure for real-time processing and/or communication automatically for the desired platform. ISG manages the generation of the executables, their distribution across a network, the collection and evaluation of results. Also, specific application software may be generated automatically.

As ISG automatically provides an executable system within a reasonable time, the system-under-development itself can be used for exploitation of its properties. Therefore the obtained results are representative for the system in its current state. Priority should be given for refinement of such parts which have been identified as potentially critical.

In case of ISG manual intervention is limited to development phases where little effort is required only: definition of functionality, architecture, performance and result analysis. The effort for other activities like testing and verification grows at higher orders or multiples of the effort needed for definition of functionality and architecture. E.g. the effort to define the nodes of a network is a linear function of the number of nodes, while the implementation of the communication channels and the recording of

the data transfer grows at second order of the number of nodes. However, this high effort is covered by automation.

Moreover, ISG simplifies verification by the chosen construction rules. The internal interfaces and the pre-run-time checks allow to reduce the total number of test cases: the total number is not the product of the subcomponent's test cases, but the sum only.

## 3 Prediction of System Properties – A Matter of Performance and Robustness Engineering

Prediction of a system's properties shall allow to reliably conclude from its properties at a certain stage of development on its final properties. The goal of performance engineering is to demonstrate that the system has the desired good properties under nominal conditions, while the goal of robustness engineering is to demonstrate that the system does not have bad properties.

Performance prediction must either confirm compliance with requirements or identify non-compliance early enough so that corrective actions can be taken with little effort by which compliance can be achieved again.

Prediction of robustness must confirm that the system can (1) continue to provide its services under non-nominal conditions and (2) return to nominal operations after being exposed to non-nominal condtions.

The more software is used the more important are its properties. While in the past software testing concentrated on the demonstration of the desired functional, behavioural or performance properties („good" properties), currently the need is increasing to prove that a system does not have undesired, „bad" properties.

The confirmation of the presence of the desired good properties and the absence of bad properties depends on the testing, verification and validation (V&V) strategy.

**Minimisation of Test Effort.** Presence of good properties and absence of bad properties can only be predicted if the system is carefully tested under nominal and non-nominal conditions. To achieve this goal the test domain must be small enough and test case execution must be sufficiently efficient so that the whole test domain can be explored within a reasonable time. Different test approaches are known.

**Minimisation of Test Effort by Ignoring the Full Test Domain.** In this case testing is optimised by taking a test approach which minmises the occurence of errors during run-time at minimum test effort. A representative of this test approach is called „testing based on an operational profile" (e.g. [11], [3]). By this approach such code is tested more deeply which is frequently used, while other code, which is only sporadically executed, is poorly tested or not tested at all.

The properties of a system tested by this strategy are not predictable. Although the non-nominal conditions and the associated bad properties may occur only sporadically, it is unclear what the system will do in such a case.

Nobody would accept that the four „operational" wheels of a car are well tested, while the spare wheel is not tested at all because it is rarely used.

**Minimisation of Test Effort by Construction Rules and Automation.** However, the test domain can be minimised by other means: (1) the test cases can be reduced by construction rules, (2) the tests can be be made more efficient by automation. This is how ISG tackles the testing and V&V problem.

A number of tools for verification of a system's behavioural properties are known, e.g. ObjectGeode, SDT and (in former versions) STATEMATE. Their verification approach is based on „exhaustive simulation" a mathematical algorithm which exploits a system's state space spawned by all the sub-spaces of its components. This leads to state explosion in practical cases [4] and does not give any benefit in practical cases.

First, to master this problem ISG takes a design approach by which the system's state space only becomes the sum of the state space of its components. This requires isolation of the components in an object-oriented manner. Second, ISG considers everything what is not explicitly allowed as erroneous. Identification of such erroneous cases forces the developer either to add the relevant case as a desired property or to avoid its occurence. Third, ISG demands to provide exception handlers for all cases not explicitly covered. This ensures that any unexpected (non-anticipated) error will properly be handled.

Due to automation and formalisation stimulation of the system on good and bad cases is possible without manual intervention This allows to exploit the reduced, but possibly still large test space in a reasonable time.

Again, ISG does a type of simplification which does not compromise the overall goal, the prediction of system properties.

### 3.1 The Role of Performance Engineering

Performance enginnering is understood as an activity which aims to exploit whether the system in its final version will meet the performance and resource constraints as given by the customer. This is a matter of (early) risk reduction in order not to fail at the end.

Performance engineering deals e.g. with response times, time jitter, CPU, network and buffer utilisation, and data profiles.. The system must be exposed to all legal conditions including stress testing over the complete lifecycle. Stress testing may cover high CPU load, extremely high data rates or data length.

### 3.2 The Role of Robustness Engineering

**The Goals of Robustness Engineering.** Robustness engineering must cover the risks imposed by anticipated or non-anticipated faults like loss of resources (processors or communication channels), permanent CPU and channel overloads and buffer overflows.
In principal, robustness engineering deals with

- fault prevention
- fault removal
- fault tolerance.

Robustness engineering is the consequent extension of performance engineering regarding risk reduction and demonstration of quality of service under extreme stress and fault conditions. It can share all the means needed for performance engineering. But it requires additional means for fault injection to demonstrate the system's robustness.

**Some Examples.** Robustness engineering shall expose the system to anticipated faults and identify unknown potential faults. It shall demonstrate that the system can survive anticipated and non-anticipated faults. Non-anticipated faults have to be converted into anticipated faults by considering for each system component the following principal malfunctions: illegal or missing outputs and high consumption of resources.

*Loss of Data.* Data may be lost (1) due to loss of a system component, e.g. when it stops its execution by a crash or by an inadvertent input, (2) by lack of resources, e.g. no space to store a message or no capacity to transmit, and (3) by faults related to the transfer, e.g. transfer to the wrong destination.

*Introduction of New Faults by Robustness Engineering.* By adding redundancies it may happen that the overall robustness does not increase, but decreases, e.g. when a number of independent redundant communication channels are provided. This extension towards higher robustness against loss of data creates a set of new potential faults: multiple copies of the same data may arrive at the destination(s) and may cause false alarms, e.g. due to multiple entries in a database. Whether corrective means make the system more robust or not must also be a matter of robustness engineering.

*Wrong Inputs.* Hardware and software faults are recognised by software either as illegal or missing inputs. An illegal input prevents processing of the received data and generation of a result. Therefore occurence of faults can be representatively investigated by two principal test activities: enforcing of loss of data and stimulation of the exception handlers.

*Protection Against Overloads.* In case of overload situations a number of unsolved problems may come up, e.g. loss of communication capabilities due to overload of the processor, loss of operational capabilities of the operating system due to lack of disk space etc.

Such situations are typical for systems like client-server systems for which the load cannot be determined in advance because the number of instances / clients is unknown and the load they are generating.

This is also true for a telephone network for which the number of users exceed significantly the network's capacity. A high overload cannot be avoided in principle. If a large number of users accesses the network at nearly the same time, like it happened at Christmas of year 2000 in Germany for SMS messages, the network must be robust enough to continue its operations at the highest possible load without crashing.

*Interdependencies between Performance and Robustness Engineering.* By robustness engineering different implementation choices may be evaluated. E.g. a strategy, which increases robustness against loss of data, may cause an overload.

Consider the problem of cyclic execution of a process. A solution could be to start the process once, and the process itself cares about generation of the next cyclic event: on reception of the actual event it requests for the next one. This is a standard solution frequently used. What is very critical in this case is that the process stops execution completely when a scheduling signal is lost. So this solution is not very robust.

Now, to get a higher probability to survive, two signals could be issued, hoping that at least one signal will arrive. However, this solution ignores the reason why a signal may be lost: the CPU overload or lack of buffer storage. But doubling of the signal rate increases the probability of an overload situation, and does not help at all, it makes the situation even worse.

A robust solution is to make the signal generation independent of the data transfer, e.g. by using a cyclic software timer. Also, this reduces the data transfer rate down to 50% of the standard solution because only the triggering event needs to be sent, but not the requests for the next event. Hence, this is the optimum solution because it is robust against loss of data - even if an event is lost, the cyclic execution does not terminate - and it reduces the data traffic, i.e. the probability that an overload situation may occur.

### 3.3 Automated Evaluation of System Properties

Due to automated stimulation and instrumentation ISG can automate the evaluation of the properties of the system-under-development as well. The templates are automatically instrumented as needed and the observed properties are recorded for automated analysis and evaluation at post-run-time.

Without manual coding information on a system's good and bad properties is continuously provided right from the beginning until the end of development. If required this information can also be collected during the operational phase. Checks on violation of desired properties are automatically performed. Currently, most of them are application-independent and can be reused by any application. If required specific checks may be added. In this case the automated evaluation can be extended by the user.

Subject of current and future work is whether and how such specific checks can be brought into a standard form so that they become application-independent and the instrumentation can be reused as it is already possible for the other checks.

An important feature is that by the automated stimulation, checking and evaluation ISG can report on violation of properties without being asked for whether the property is fulfilled or not. This is an advantage compared with e.g. model checking, where a user must ask the right question to get the information that a requirement is violated.

## 4 System Development with ISG

ISG supports an incremental development approach. Only little information is required at the beginning, an idea is sufficient. However, ISG requires a complete set of inputs and more information than other tools ask for because it automatically

generates an executable system from the provided information. The advantage is: having accepted the user inputs, ISG guarantees that the system is executable as defined by the inputs.

ISG limits the required inputs to a minimum. It only needs the names of the processes, processors, messages to be exchanged and the processes' states (only one state is allowed, but usually not meaningful). In addition, it asks for performance and resource figures such as timeout limits, periods, estimated execution times or buffer size.

Also, the generation, instrumentation, test and stimulation modes must be defined. Finally, information on the topology and the network has to be provided so that ISG knows what the platform is and where the components shall be executed.

However, not all of the required information needs to be immediately provided by a user. He/she may take default templates at the beginning and adapt the values later on when needed. This way the desired system configuration can be smoothly approached by incremental refinement always getting a feedback on the system's properties from an executable system.

Having started with a rough idea, a user can either iterate and refine the inputs in case of good results or can abandon them in case of bad results and start with a completely different approach. As only little information on the system is required to bring it into operation, no huge amount of money and time is lost.

The representativeness of the feedback depends on the actual refinement of system definition. A user may drive the refinement towards critical items according to actual results or expected criticality to get a feedback at the desired degree of representativeness.

The advantage compared with „state-of-the-art simulation" is that any effort spent to increase the degree of representativeness is not lost, but invested in the system-under-development. No duplication of the development effort occurs.

A further advantage is that there is no need to postpone execution until the target platform is fully available. The system is executable right from the beginning on any topology formed by platforms as supported by ISG. E.g. to add, remove or modify a host in a TCP/IP network only requires update of 20 bytes and to type the name of the ISG script which then does generation, distribution, execution and result evaluation automatically for the new topology.

By this feature it is possible to map the actual version of the system onto any network configuration consisting of a number of processors between only one and the planned maximum number on any of the platforms without changing the code or user inputs.

### 4.1 ISG: An Overview on the Automated Process Model of ISG

Fig. 1 gives an overview on the principles of the ISG approach. By taking high-level engineering information and by applying ISG construction rules, an executable system is generated, typically within 20 .. 120 minutes[5]. Then a feedback from the real system is available regarding the functional and non-functional properties.

[5] The generation time depends mostly on the number of process types and amounts roughly to about 1 .. 2 minutes per process type (UltraSparc I/143 or PC 200) on the average.

With ISG testing and integration can be performed in top-down manner, yielding full visibility on the properties of the actual version during the whole lifecycle (Fig. 2, right part). This is different from the V-model approach (Fig. 2. left part) for which a first feedback cannot be received before the coding phase and integration cannot be done before module testing. Hence, risks which are related to integration are identified very late, mostly too late to be able to solve a problem with little effort.

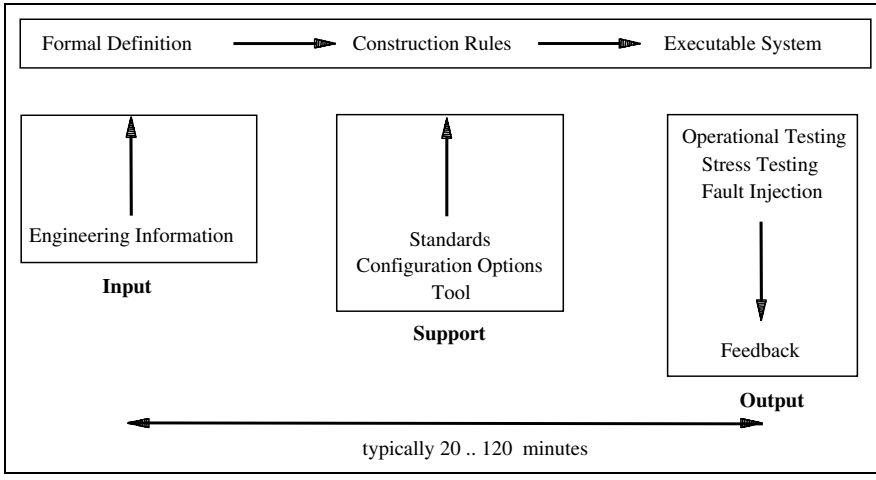



**Fig. 1.** The ISG Approach of Automated System Generation

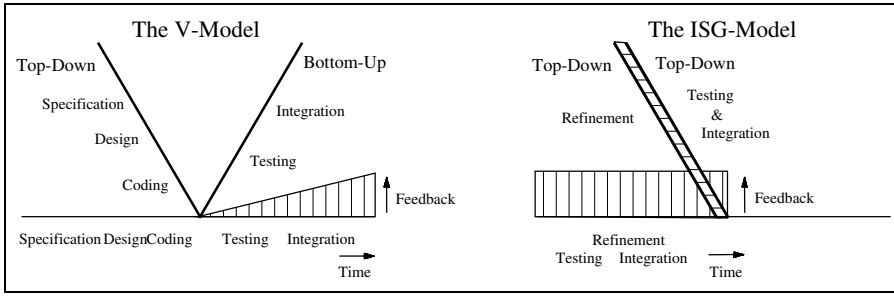



**Fig. 2.** ISG vs. the V-Model

Fig. 3 shows how the executable code is automatically generated from the user inputs. Such inputs are processed by ISG utilities, which generate the source code and the environment needed to build the executables. Having compiled the source code and built the executables, they are distributed across the network and executed. By instrumentation the information on the system's properties is produced and presented by an evaluation report after the execution. A user just has to provide the inputs (in terms of literals and figures defining the system's components and performance constraints), to start the generation process and then to wait until the windows pop up which display the evaluation report.

By each iteration ISG transforms one consistent version of the system into the next one. There are a number of entries where the generation process may be restarted. Two principal entries are shown by Fig. 3: the "major" cycle which is required after a structural change and the "minor" cycle which may be executed in case of pure re-compilation, e.g. after update of a user-provided source file. The duration of the „minor cycle“ is in the range of some minutes up to about 20 minutes depending on the number of components to be compiled.

The user inputs define a system in terms of processes, Finite State Machines (FSM), incoming and outgoing data, the user's functions to process the data (called „User-Defined Functions“, UDF), nodes (CPUs) where the activity shall be executed, logical channels and performance properties and constraints such as timeout, expected CPU consumption, periods (including time jitter), amount of transmitted data.

ISG automatically provides the infrastructure for data communication and scheduling, performance analysis, verification and validation for a given topology. The application-specific processing of the data is performed by the UDFs which can be plugged into the framework as provided by ISG. At the beginning (instrumented) UDF stubs are generated by ISG, which ensure the system's execution right from the beginning. Later on the UDF „drawers“ as provided by the user are plugged-in successively into the „shelves“ provided by ISG. A UDF may also be generated automatically with ISG.

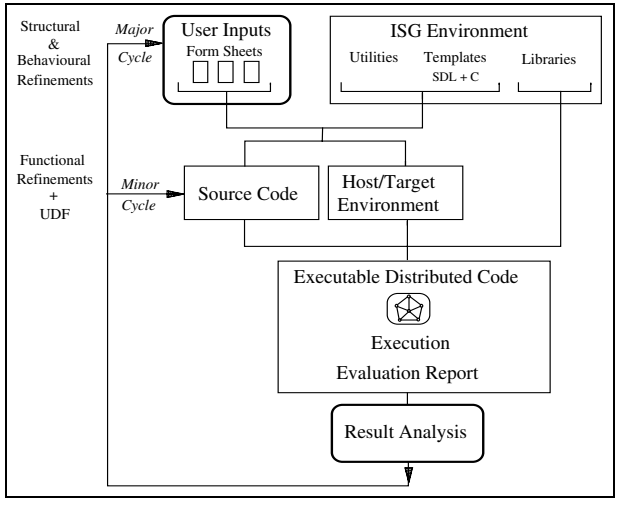



**Fig. 3.** The Automated Process Model of ISG

## 4.2 Application Areas of ISG

Basically, ISG addresses system generation from scratch by making this process more efficient and less risky. However, due to the capability to embed user-defined code it may also interface with already existing code which is plugged into the ISG-provided

skeleton as UDFs. From this point of view it may be used to port legacy software to a modern platform..

Also, a bridge can be established from another tool environment, which is already in use, to ISG in order to take benefit from its testing, reporting, verification and validation capabilities. Components which require a complex hardware environment for integration and testing on system level, can easily be integrated with ISG on pure software level, which is less expensive more flexible regarding iterations and earlier available than a hardware platform.

ISG is currently used in the technical area, e.g for the MSL project ([9], [10]) and the CRISYS project [2]. MSL is the „Material Science Laboratory“ of the International Space Station planned to fly in 2001/2002. In the CRISYS project ISG will be used for two applications acting as an integration and V&V platform for (1) a back-up power supply of a nuclear power plant and (2) a mail sorting and distribution system.

However, the concepts of ISG are not limited to the pure technical domain. The available infrastructure can be adapted with little effort to cover the needs of the IT domain, using e.g. COBOL, SQL, Java, C and C++, so that all the generation, verification and validation capabilities will be available for such applications as well. This is also true for automated generation of user-defined functions.

### 4.3 The Feedback Provided by ISG

**Performance Engineering.** The following means for performance and resource analysis are automatically built-in by ISG into the operational software system on user request.

ISG checks the consistency of the behaviour, the compliance of the achieved performance with performance requirements and the robustness of the system. It records timeouts, cycle overruns and exceeded deadlines. It compares the estimated CPU consumption with the measured one and provides a new estimation, so that a user can refine his performance estimations. Load figures are calculated for each instance of a process type

A similar calculation is done for the network utilisation. For each physical channel ISG counts the transmitted bytes which are exchanged between the CPUs and calculates the channel utilisation. When executing all the components of a distributed system on one processor only, a delay may be generated which represents the difference between communication by local channels (e.g. message queues, IPC) and remote channels such as UDP and TCP/IP.

Also, figures about the buffer utilisation are derived so that the buffers can be optimised towards their actually used size.

ISG provides information about the processing power of a certain processor type, so that it can easily be compared with other types.

Moreover, response times are calculated by ISG. A user may identify the start of a handshake between two processes: a process sends a message to another process and expects a response. To measure the response time, the sending process inserts automatically the start time in the data exchange format, and the responding process inserts the actual time when it responds. Hence, each of the processes can measure the

transmission time and the sender can calculate the overall response time. To enable this feature only a single letter which defines the tracing mode must be changed.

There are a lot of other reports available which cannot be described in detail here like: "coverage of data processing actions", "list of non-covered actions", "coverage of states", "list of non-covered states", "executed state transitions", "exception report", "error injection report", "timer report", "timing and sizing budgets", „statistics and profiles on sent and received data“, „profiles on CPU consumption“.

By timing diagrams (Fig. 4) more information on the event flow is provided (by clicking on an event the transmitted data are displayed). The data flow between the processes is shown by Message Sequence Charts [8]. A MSC gives either detailed information on the exchanged data and the involved FSM (Finite State Machine) (so that it can be used for high-level debugging) or summary information on communication by a highly compressed timescale as shown by Fig. 5.

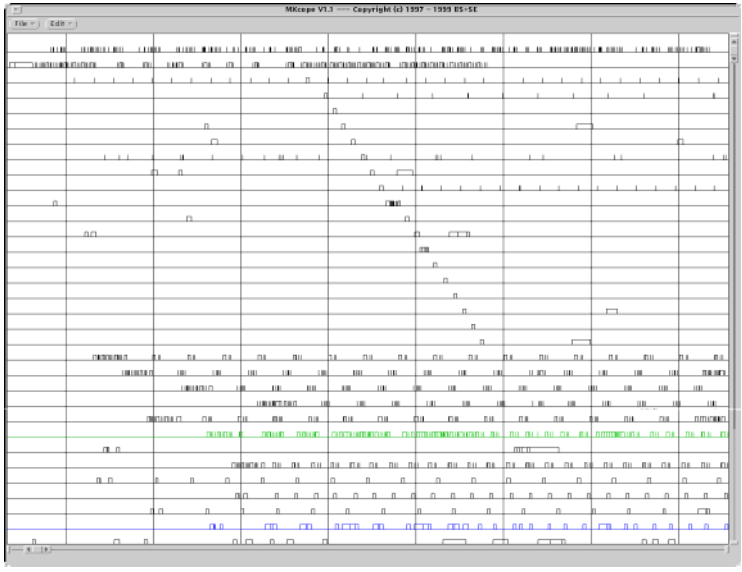


**Fig. 4.** Timing Diagram on Events

For a project a potential criticality of the timing budget was recognised by early tests. Having identified this potential bottleneck the responsible engineer introduced another strategy for data processing which saved about 20%. To introduce the more efficient strategy the structure of the database had to be changed. This was implemented within about two hours for about 600 C functions because the complete set of functions related to processing of the database contents was generated automatically by ISG.

This example shows two benefits of ISG: (1) early identification of potential risks, (2) problem solving with little effort within a short time period.

**Robustness Engineering**. ISG covers fault prevention by formal checks and automated construction of the system. Fault removal is supported by rejection of incorrect and inconsistent inputs, by automated insertion of run-time assertions, by giving a feedback on behaviour (received data, states, executed actions, exceptions)**,**

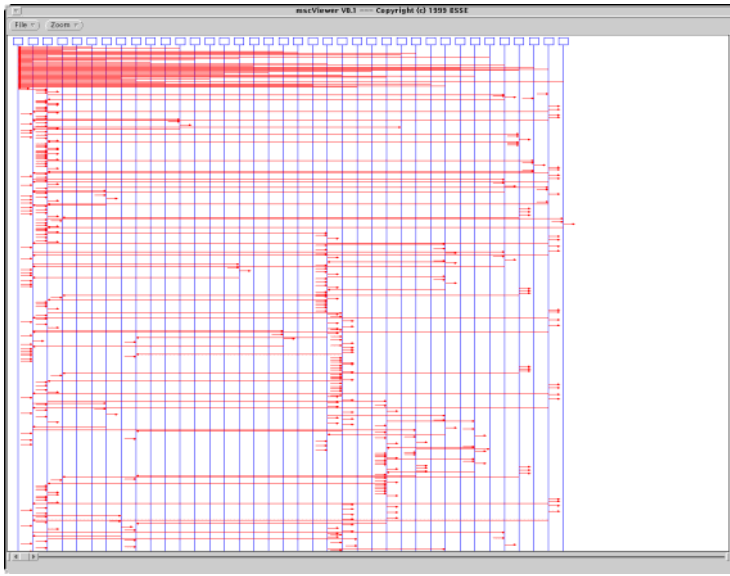


**Fig. 5.** Message Sequence Chart Describing the Overall Data Flow

performance (cycle overruns, timeout, exceeded deadlines), by evaluation and presentation of results regarding coverage analysis, resource analysis, performance analysis, by supporting fault injection, automated test stimulation, stress testing, scaling of time.

Regarding fault tolerance ISG implements exception handling on the level of the FSMs, e.g. for timeout conditions, illegal data inputs, and allows to define redundant communication channels. Management of redundant processors is supported, too. Fig. 6 shows the verification and validation concept of ISG: the automated system generation based on a formal definition of the system (bottom), the checking and reporting capabilities to identify faults (top), and system stimulation (in the middle).

Three principal test modes exist:

1. testing related to the normal operational scenario
2. stress testing
   either by stimulation of a high number of processes at a high input rate and/or
   by scaling of time
3. fault injection
   by stimulation with illegal inputs or by loss of data

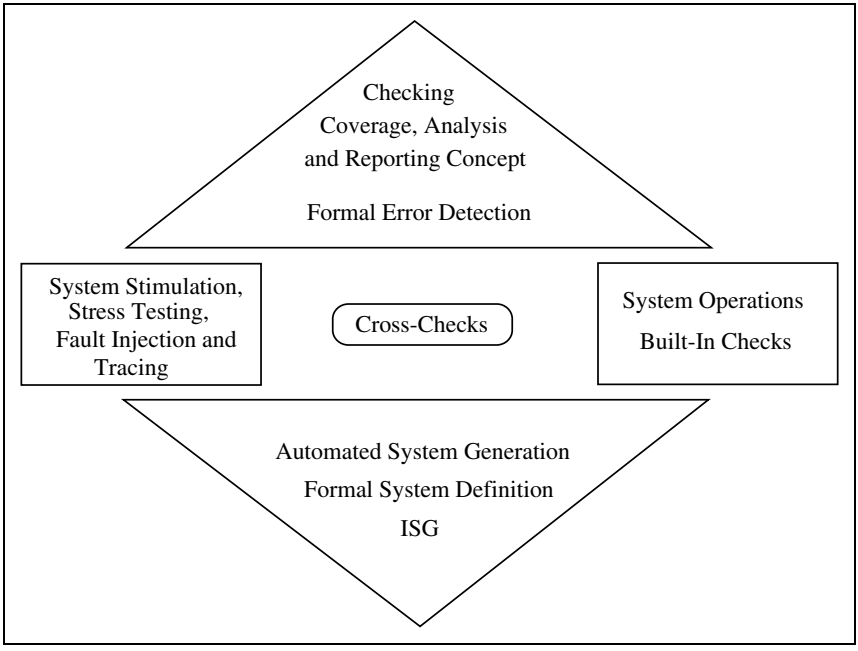



**Fig. 6.** ISG Verification and Validation Concept

Mode 1 and possibly mode 2 are a matter of performance engineering, while modes 2 and 3 are clearly a matter of robustness engineering. While the test objective of mode 1 is to prove absence of errors and the presence of the desired capabilities, the test objective of modes 2 and 3 is to create faults and to demonstrate that the system can survive such exceptional cases.

The errors which may be detected can be classified as

a. errors detected by visual checks and tool support at pre-run-time (static checks)
b. errors detected during execution due to error messages
   which cover e.g. illegal inputs to states, lack of resources like buffer overflow, exhausting of disk space and database, exceeded deadlines, cycle overruns
c. errors detected at post-run-time by coverage and performance analysis
   e.g. non-covered states, too high response times.

Starting with mode 1 (normal testing) usually a first set of simple errors will be recognised. But the removal of such errors does not imply that the system will be freee of errors.

To be prepared for such situtations it is necessary to expose the system to tests related to ISG V&V steps #2 and #3.

As it is difficult to manually generate stress situations or faults, ISG has built-in capabilities for stress testing and fault injection. For each process type a stimulation rate and the type of fault injection can be specified which makes it easy to create a

CPU or network overload or to loose data. This can be done on a high abstraction level by defining patterns like wild card options.

| Loss-Of-Data Probability | % Coverage of States | # Exceptions |
|---|---|---|
| 0 | 100 | 200 |
| 0.01 | 100 | 129 |
| 0.02 | 78.38 | 212 |
| 0.03 | 78.38 | 186 |
| 0.04 | 37.94 | 93 |
| 0.05 | 37.84 | 120 |
| 0.10 | 37.84 | 270 |
| 0.50 | 36.04 | 355 |
| 0.70 | 36.04 | 557 |
| 1.00 | 36.04 | 840 |

**Fig. 7.** Analysis of Robustness: An Example

Fig. 7 shows the results of an experiment to measure the robustness of a system against loss of data. The probability to loose data has been increased from 0 to 1. For this example the coverage of states decreases suddenly from 100% to a small value already at probabilities close to zero. This shows that the system-under-test is not very robust regarding loss of data and occurence of faults, respectively. Vice versa, if the system would be sufficiently robust such a test would document that the system fulfills the robustness requirements.

A surprising result is that the number of exceptions does not monotonically increase with the fault injection probability: by loosing data the number of system activities decreases, and consequently the chance that an exception will occur decreases as well. When the minimum coverage of states is reached then the number of exceptions increases again, because no interference between decreasing number of actions and increasing number of lost data occurs anymore. Consequently, the number of tolerated exceptions cannot be used as a measure for a system's robustness.

## 5 Conclusions and Future Work

Problems have been discussed which are related to derivation of representative system properties at an early development stage. Current practices have been analysed and an alternative approach ISG has been presented.

ISG allows to get an early and representative feedback from the system. The reduction of manual effort and development time is achieved by automated generation of the software from a minimum of system engineering inputs. This is possible because the ISG process model organises the generation such, that major or most parts can be built from reusable functions or by reusable construction rules. Therefore the system engineering information provided in terms of literals and figures can

completely control the generation process. Application-specific algorithms which cannot be generated automatically or exist already can be plugged in like drawers into the automatically generated shelves.

The automation allows for easy and inexpensive instrumentation of the code for monitoring of a system's properties. Also, the generation of the evaluation report is automated. The report provides information on functional and non-functional properties. Separate activities like behavioural or performance simulation are not needed. The response from the system is always representative, effort for refinened exploitation of a system's properties is not wasted.

Also, ISG reduces the risks because representative information can easily be derived from the system-under-development.

The need for investigation of a system's robustness, called „Robustness Engineering", and the benefit coming from automated system generation has been discussed. Robustness engineering is considered as an extension of performance engineering towards getting software which behaves deterministic under stress and fault conditions.

The current application areas of ISG are embedded and/or distributed systems in the automation domain. The current status of ISG can be considered as a first successful step which demonstrates the feasibility of automated system generation and evaluation of its properties.

Based on the experience obtained by the previous activities the concept of an automated process model shall be extended towards databases, hardware-software co-design and GUIs (Graphical User Interfaces). Currently, a GUI test tool is under development which follows above ideas. Discussions with experts in the area of databases and hardware-software co-design confirm that ISG can be extended towards these application domains.

## References

1. BSSE internal communication, unpublished
2. CRISYS: Critical Instrumentation and Control System, ESPRIT project EP 25514 (1997-2001)
3. Ehrlich, W., Prasanna, B., Stampe, J., Wu, J.: Determining the Cost of a Stop-Test Decision, IEEE Software, March (1993) 33
4. Gerlich, R.: Tuning Development of Distributed Real-time Systems with SDL and MSC: Current Experience and Future Issues, In: Cavalli, A., Sarma, A. (eds.): SDL'97 Time for testing, SDL, MSC and Trends. Proceedings of the Eighth SDL Forum, Evry, France, 23-26 September (1997) 85-100
5. Harel, L.: Statemate/Rhapsody, i-Logix, Three Riverside Drive, Andover, MA 01810, info@ilogix.com
6. HRDMS: Highly Reliable Data Management System and Simulation, ESA/ESTEC contract no. 9882/92/NL/JG(SC), Final Report, Noordwijk, The Netherlands (1994)
7. Gerlich, R.: ISG, Instantaneous System and Software Engineering, ISG User's Manual and ISG Traning Manual, Auf dem Ruhbuehl 181, D-88090 Immenstaad (1999-2000)

8. MSC: ITU-T Recommendations Z.120, Message Sequence Charts (MSC), Helsinki (1993)
9. Gerlich, R., Birk, M., Brammer, U., Ziegler, M., Lattner, K.: Automated Generation of Real-Time Software from Datasheet-based Inputs -The Process Model, the Platform and the Feedback from the MSL Project Activities, Eurospace Symposium DASIA'00 "Data Systems in Aerospace", May 22-26, 2000, Montreal, Canada, ESA (2000)
10. Birk, M., Brammer, U., Ziegler, M., Lattner, K., Gerlich, R.: Software Development for the Material Science Laboratory on ISS by Automated Generation of Real-Time Software from Datasheet-based Inputs, Eurospace Symposium DASIA'00 "Data Systems in Aerospace", May 22-26, 2000, Montreal, Canada, ESA (2000)
11. Musa, J.D.: Operational Profiles in Software-Reliability Engineering, IEEE Software, March (1993) 14-32
12. OG: ObjectGEODE, Verilog, 52, Avenue Aristide Briand; Bagneux; 92220; France, verilog@verilog.fr, since 2000 part of Telelogic, see SDT (2000)
13. OPNET: OPNET, MIL3 Inc., 3400 International Drive, NW-Washington, DC 20008, USA
14. SDL: ITU Z.100, Specification and Description Language, SDL, Geneve (1989)
15. SDT: SDT, Telelogic, Headquarters: Box 4128; S-203 12 Malmoe; Sweden. Vising address: Kungsgatan 6, info@telelogic.se
16. UML: Unified Modelling Language, http://www.rational.com/uml
17. VWKS: TORNADO / VxWorks, WindRiver Systems, Inc. 1010 Atlantic Avenue, Alameda, CA 94501-1153, USA
18. WB: SES/workbench, 4301 Westbank Dr., Bldg. A, Austin, TX 78746 USA, mktg@ses.com

# Performance Engineering of Component-Based Distributed Software Systems

Hassan Gomaa[1] and Daniel A. Menascé[2]

[1]Dept. of Information and Software Engineering
School of Information Technology and Engineering
George Mason University
4400 University Drive, Fairfax, VA 220300-4444, USA
hgomaa@gmu.edu
[2]Dept.of Computer Science
School of Information Technology and Engineering
George Mason University
4400 University Drive, Fairfax, VA 220300-4444, USA
menasce@cs.gmu.edu

**Abstract.** The ability to estimate the future performance of a large and complex distributed software system at design time can significantly reduce overall software cost and risk. This paper investigates the design and performance modeling of component interconnection patterns, which define and encapsulate the way client and server components communicate with each other. We start with UML design models of the component interconnection patterns. These designs are performance annotated using an XML-type notation. The performance-annotated UML design model is mapped to a performance model, which can be used to analyze the performance of the software architecture on various configurations.

**Keywords.** Component interconnection patterns, software architecture, XML, UML, performance model, queuing networks.

## 1 Introduction

The ability to estimate the future performance of a large and complex distributed software system at design time, and iteratively refine these estimates at development time, can significantly reduce overall software cost and risk. In previous papers, the authors have described the design and performance analysis of client/server systems [5,16,17,18].

This paper investigates component interconnection in client/server systems, in particular the design and performance modeling of component interconnection patterns, which define and encapsulate the way client and server components communicate with each other. This paper describes the component interconnection patterns of synchronous and asynchronous communication with a multi-threaded server.

We start with UML design models of the component interconnection patterns, which are then performance-annotated using an XML-based notation. The

R. Dumke et al. (Eds.): Performance Engineering, LNCS 2047, pp. 40–55, 2001.

performance-annotated UML design model is mapped to a performance model, which allows us to analyze the performance of the software architecture on various system configurations.

The rest of the paper is organized as follows. Section 2 provides a background on software architecture and describes our approach. Section 3 describes UML interconnection patterns. Section 4 describes the performance annotations of the UML-described architectures. Section 5 describes performance models of the component interconnection patterns and section 6 presents concluding remarks and future work.

## 2 Software Architecture and Performance

### 2.1 Software Architecture

The software architecture of a system is the structure of the system, which consists of software components, the externally visible properties of those components, and the interconnections among them [1,24]. Software components and their interactions can be formally specified using an architecture description language [19,23,24] to describe the software architecture. Architecture description languages (ADLs) separate the description of the overall system structure in terms of components and their interconnections from the description of the internal details of the individual components. A *component* is defined in terms of its interface, including the operations it provides and requires. A *connector* encapsulates the interconnection protocol between two or more components.

The Unified Modeling Language (UML) is a graphically-based object-oriented notation developed by the Object Management Group as a standard means of describing software designs [2,22], which is gaining widespread acceptance in the software industry. There are efforts to provide a bridge from UML to software architectures [8]. There have been some attempts to integrate ADLs with the Unified Modeling Language (UML) [21]. There have also been investigations on how software architectures can be described using the UML notation [9]. Currently, UML lacks formal semantics for architecture description and for performance modeling.

Analyzing the performance of a software architecture allows a quantifiable tradeoff analysis with the objective of minimizing risks in software designs. An example of performing architecture tradeoff analysis is given in [10]. Other approaches based on analytic performance models are described in [6,16,17,18].

### 2.2 Component Interconnection Patterns

A *design pattern* describes a recurring design problem to be solved, a solution to the problem, and the context in which that solution works [3,4]. The description is in terms of communicating objects and classes that are customized to solve a general design problem in a particular context. In this paper, a *component interconnection pattern* defines and encapsulates the way client and server components communicate with each other via connectors. In distributed applications, *component interconnection patterns* are needed to support the different types of inter-component communication,

including synchronous, asynchronous, brokered, and group communication [6,7,8,20,25].

In the UML notation, the class diagram is used to depict the static view of interconnection patterns, whereas the collaboration diagram depicts the dynamic view. This paper describes component interconnection patterns for synchronous and asynchronous communication [8].

### 2.3 Approach

A distributed software architecture is composed of a set of components and a set of connectors that can be used to connect the components. Components are black boxes that interact with other components through connectors, by exchanging messages according to the connector protocol.

The approach described in this paper, illustrated in Figure 1, starts with a UML design model of the component interconnection pattern. This design is performance-annotated using an XML-based notation. The performance annotated UML design model is then mapped to a performance model used to analyze the performance of the architecture.

## 3 Specifying Component Interconnection Patterns with UML

### 3.1 Static Modeling of Component Interconnection Patterns

UML class diagrams depict the static modeling of the component interconnection patterns and use stereotypes to differentiate among the different kinds of component and connector classes. In UML, a **stereotype** is a subclass of an existing modeling element, which is used to represent a usage distinction [22], in this case the kind of component or connector class. A stereotype is depicted using guillemets, e.g., <<s>>. We introduce the following stereotypes: <<Component>>, <<Connector>>, and <<Resource>>.

Figure 2(a) depicts a class diagram for a client/server system. A **component** is an application class that is defined in terms of its interface, which is visible to other components, and its implementation, which is hidden from other components. In this example, the client and server classes are depicted using the stereotype <<component>>. A **connector** hides the details of the interaction between components. To model the performance of a distributed software architecture, it is useful to consider a connector in terms of a client connector that communicates with a server connector via a resource. A **resource** is used to depict a part of the communication infrastructure that supports the interconnection among components or connectors and is introduced to facilitate the mapping to performance models as the presence of a resource needs to be explicitly modeled. In these patterns, the Network is a <<resource>>. Each server component has a logical 1-many association with client components. Physically, however, each client component has a 1-1 association with a client connector and each server component has a 1-1 association with a server connector. The network resource has a 1-many association with the client and server

connectors. Optionally, there may also be one or more brokers. A Broker is modeled as a component class. There may be 0,1, or more brokers in a client/server system.

Figure 2(b) shows the Client Connector, which is specialized into a Synchronous Client Connector, an Asynchronous Client Connector (which is a composite class, composed of a Client Message Input Buffer class, a Client Call Stub class and a Client Return Stub class) and a Brokered Client Connector. In Figure 2(c), the Server Connector is specialized into a Single Threaded Server Connector or Multi-Threaded Server Connector. The former is composed of a Single Threaded Server Message Buffer class and a Single Threaded Server Stub class. The Multi-Threaded Server Connector is composed of a Multi-Threaded Server Message Buffer class, a Multi-Threaded Server Stub class, a Dispatcher Buffer (all 1-1 relationships), and a Server Return Stub (1-n relationship). Components are also classified, a Client component is Synchronous or Asynchronous and a Server component is Single-Threaded or Multi-Threaded. A Multi-Threaded Server component class is composed of a Dispatcher class (1-1 relationship) and a Worker Server class (1-n relationship).

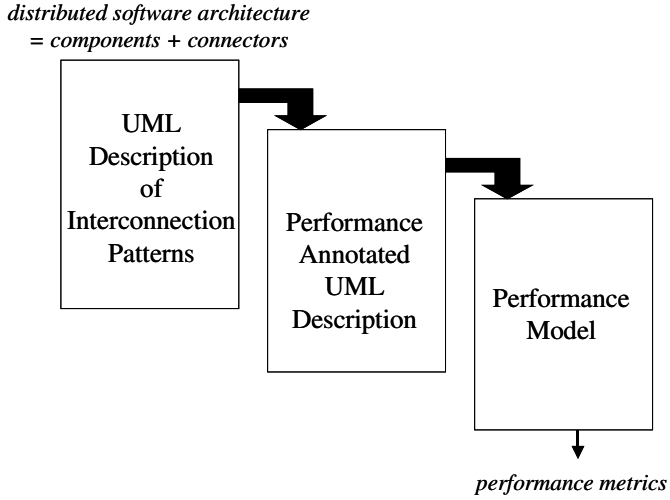



**Fig. 1.** Design and Performance Modeling Approach

### 3.2 Dynamic Modeling of Component Interconnection Patterns

UML collaboration diagrams are used to depict the dynamic interactions between the component and connector objects, i.e., instances of the classes depicted on the class diagrams. An active object has its own thread of control and executes concurrently with other objects. This is in contrast to a passive object, which does not have a thread of control. In these patterns, all components are active apart from the message buffers, which are passive monitor objects [25]. Messages are numbered on the collaboration diagrams to show the sequence of occurrence.

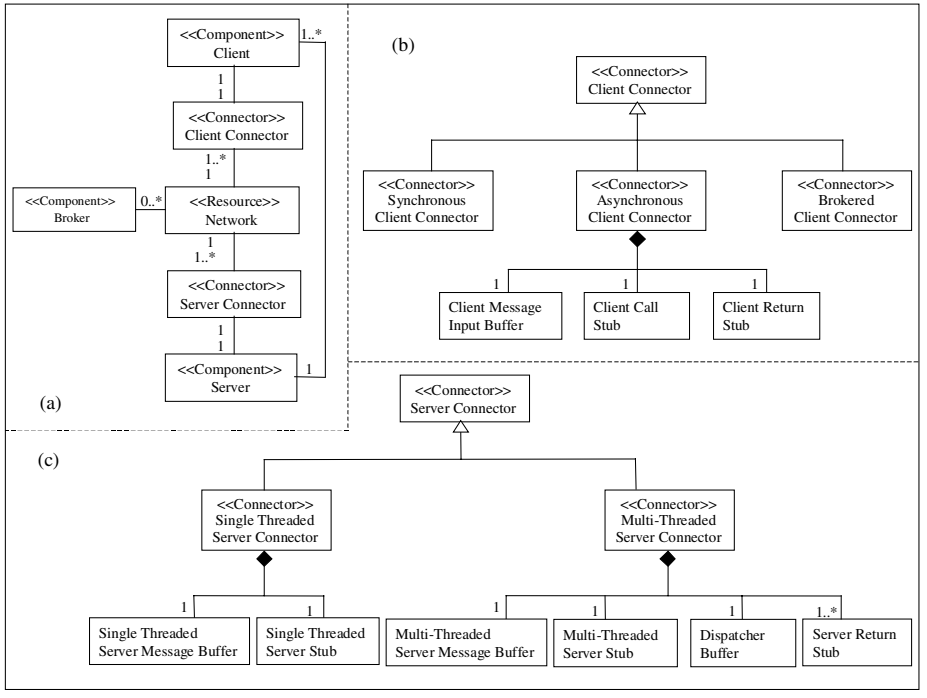



**Fig. 2.** Class Diagram for Client/Server System Connectors

Figure 3 depicts a collaboration diagram for the simplest form of client/server communication, which is a Synchronous Client component communicating with a Single Threaded Server component using static binding. The Synchronous Client component has a Synchronous Client Connector, which hides from it the details of the interconnection with the Server. The Single Threaded Server component has a Single Threaded Server Connector, which hides from it the details of the interconnection with the client.

In Figure 3, the Client Connector acts as a client stub performing the marshalling of messages and unmarshalling of responses [20]. Thus, it receives a client message, packs the message, and sends the packed message to the Server Connector (single threaded or multi-threaded). The Client Connector also unpacks the server responses for delivery to the Client component.

Figure 4 shows the more complex case of a Multi-threaded Server. Both the Multi-Threaded Server component and the Multi-Threaded Server Connector are composite objects. The clients could be either synchronous or asynchronous, as described next, and the binding could be static or dynamic.

The Server Connector receives the packed message and stores it in a message buffer. From there, the Server Stub (Single Threaded or Multi-threaded) unpacks the message. In the case of the Single Threaded Server Stub (Fig. 3), it delivers the message to the Single Threaded Server component and waits for the response. In the case of the Multi-threaded Server Stub (Fig. 4), it places the message in a Dispatcher Buffer and proceeds to unpack the next incoming message. The Dispatcher receives the message from the Dispatcher Buffer and dispatches the message to an available

Worker Server. Alternatively, the Dispatcher instantiates a Worker Server to handle the request. The Worker Server services the request and sends the response to a Server Return Stub, which packs the response and forwards it to the Client Connector.

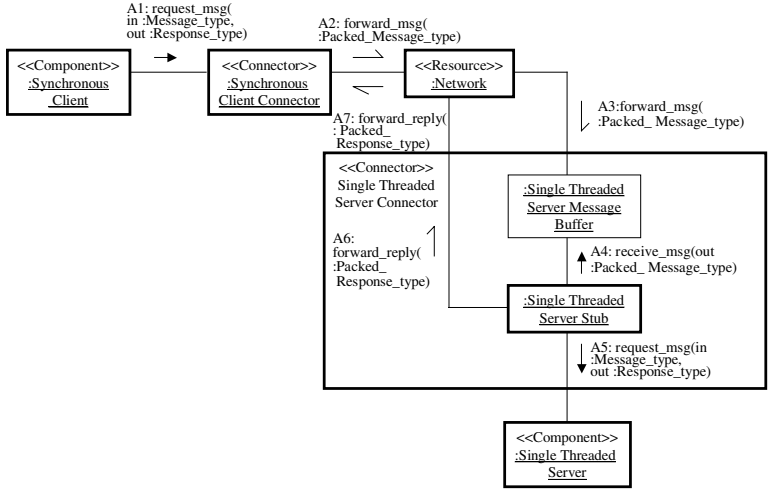



**Fig. 3.** Collaboration Diagram for Synchronous Client and Single Threaded Server

Fig. 5 shows the case of an Asynchronous Client component, which needs an Asynchronous Client Connector. The Asynchronous Client Connector is more complex than the Synchronous Client Connector as it has to support a call-back capability, whereby the Asynchronous Client component can send a message, continue processing, and later receive a server response. Thus whereas, the Synchronous Client Connector (Fig. 3) sends the message to the server and waits for the response, the Asynchronous Client Connector (Fig. 4) is capable of sending multiple client messages before a server response is received. To handle this, the Asynchronous Client Connector has two concurrent stub objects, a Client Call Stub for handling outgoing messages and a Client Return Stub for handling incoming responses. Responses are buffered in the Client Message Input Buffer, where they can be received as desired by the Asynchronous Client component.

## 4 Performance Annotations of UML-Described Architectures

This section describes the performance annotation of software architectures depicted in UML notation. The performance parameters needed to characterize the performance of the objects in UML-described architectures are given. It includes the

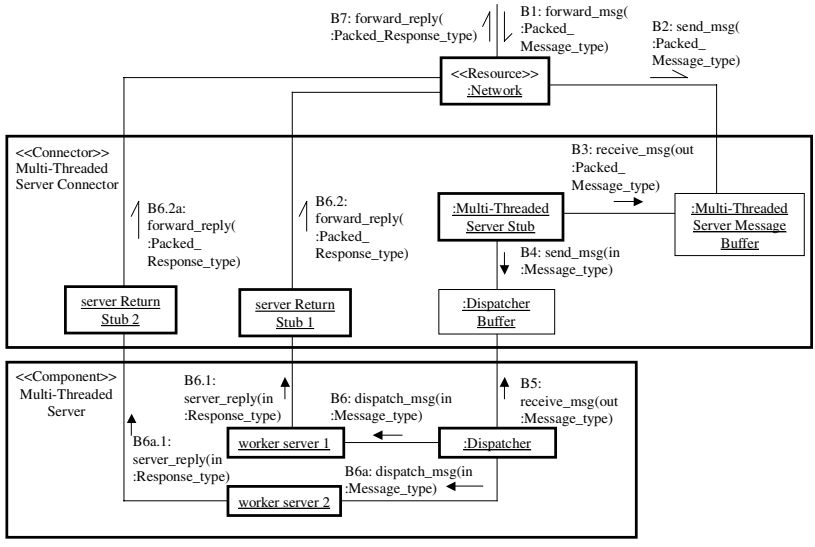



**Fig. 4.** Collaboration Diagram for Multi Threaded Server Connector

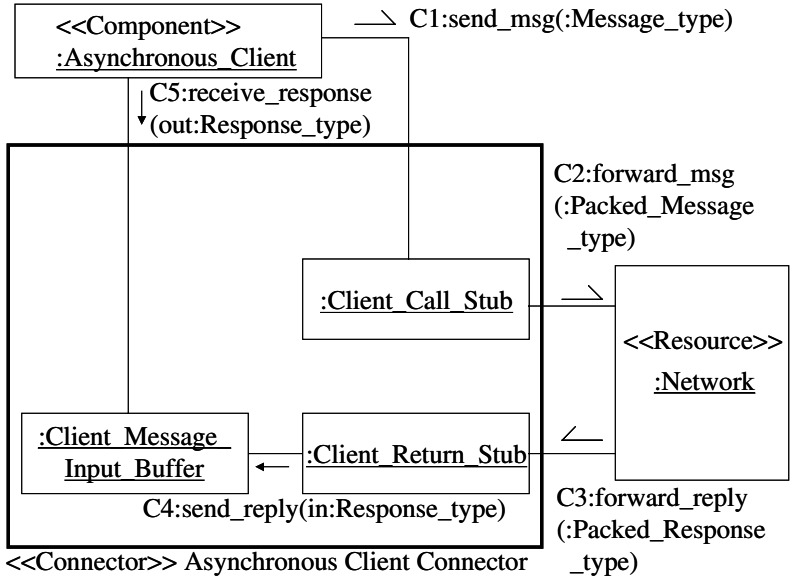



**Fig. 5.** Collaboration Diagram for Asynchronous Client Connector

performance annotations for components, connectors, and resources. The UML models are mapped to XML, which allows us to capture both the architecture and performance parameters in one notation.

### 4.1. Performance Parameters

The performance of distributed systems can be analyzed through analytic [13] or simulation models. The parameters required for these models fall into two categories:

a) Workload intensity parameters. This type of parameter measures the load placed on the various system resources. This can be measured in terms of arrival rates (e.g., requests/sec) or by the number of concurrent entities (e.g., clients) requesting work along with the rate at which each client requests work.

b) Service demand parameters. These parameters indicate the amount of each resource used on average by each type of request. Service demands can be measured in time units or in other units from which time can be derived. For example, the service demand on a communication link can be expressed by the size of a message in bits, provided we know the link capacity in bps.

The performance parameters for a component deal with two types of interface, the **port** and the **entry**. A **port** supports one-way communication. A message is sent on an output port and received on an input port. Message communication may be asynchronous or synchronous without response, as shown in Figs. 3-5. An entry supports two-way message communication, i.e., synchronous message communication with response, as shown in Figs. 3-4. Fig. 6 shows the UML objects refined to explicitly depict the input and output ports, using the Darwin notation [19], although the ROOM [23] notation could also be used. The connector shown in Fig. 6 corresponds to the client connector – resource – server connector shown in Figs. 3-5.

Figure 6 illustrates two interconnected components or objects with their input ports (i.e., places where messages arrive) and their output ports (i.e., places where messages are sent to input ports of other components). Messages can arrive at an input port from another component via a connector or from outside the system. In the latter case, one must specify the message arrival rate. In the example of Fig. 6, port *a* of Object A receives messages from the outside world at a rate of 5 messages/sec.

The figure also illustrates what happens when a message is received at an input port. For example, when a message arrives at port *b* in Object A, there is a 0.3 probability that a message will be generated at port *d* and a 0.7 probability that a message will be generated at output port *e*. In the former case, it takes an average of 1 sec to process the incoming message and generate the outgoing message. In the latter case, it takes 3 sec. The figure also shows the message sizes in bytes.

The type of connector used to interconnect components influences the performance of component based systems.

The following performance parameters are important when describing connectors:

- Client and server stub times. This is the time needed by the client and server stubs to carry out tasks such as parameter marshaling, parameter conversion, and message generation.
- Number of concurrent server threads.

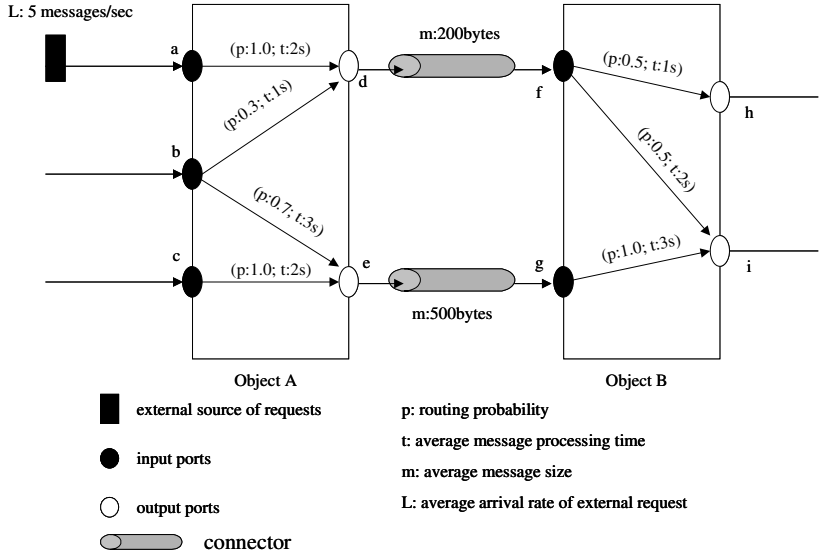



**Fig. 6.** Interconnected Components

- Maximum number of connections to be accepted by the server.
- The following parameters are important when specifying a network resource:
- Bandwidth: defines the rate at which bits are transmitted over the network.
- Latency: time it takes a bit to propagate across the network.

### 4.2 Performance Annotations of Components

From a performance perspective, an input port interface is classified into two parts: an incoming message part and an outgoing message part, which is sent via an output port. Each incoming message will require some CPU time to process it and will optionally result in an output message of a given size being generated, which is then sent over a specified output port. The arrival of a message on a given input port needs *c* milliseconds of CPU time to process, and results in a message of size *m* bytes being generated and sent on an output port *p*.

It is also possible that the input port is an external port, which receives messages directly from the external environment. In this case, it is necessary to specify the characteristics of the incoming message load in terms of the average incoming message size and arrival rate.

In the case of an entry interface, which supports two-way communication, the incoming message results in an outgoing message of a certain size being generated and sent out on the same interface. It is assumed that entries, which represent server interfaces, do not receive messages directly from the external environment.

**XML Notation for Performance Annotations of Components.** The specification, in an XML-type notation, for a component's performance parameters is given below. We use italics to indicate built-in types such as string or real and use boldface to indicate keywords. Optional specifications appear within square brackets. Specifications that may appear at least once are indicated within [ ]+. Specifications that may appear zero or more times are indicated within [ ]*.

```
<component>
  <ComponentTypeName> string
    </ComponentTypeName>
  [<port> <InputPortName> string </InputPortName>
   <PortType> external | internal </PortType>
    [<AvgIncomingMsgSize> real bytes
        </AvgIncomingMsgSize>
      <IncomingMsgArrRate> real msg/sec
        </IncomingMsgArrRate>]
    <case>
    [<probability> real </probability>
     <ProcessingTime> real sec </ProcessingTime>
     <OutgoingMsgSize> real  bytes
        </OutgoingMsgSize>
      <OutputPortName> string </OutputPortName>]+
   </case>
   </port>]*
  [<entry> <EntryName> string </EntryName>
   <ProcessingTime> real sec </ProcessingTime>
   <OutgoingMsgSize> real bytes </OutgoingMsgSize>
   </entry>]*
</component>
```

The details are as follows. There is one component declaration for each component. There is one port declaration for each input port of the given component. The **input port name** is specified. The **port type** is also specified: **external** or **internal**. An external port receives messages from the external environment. An internal port receives messages from another component. In the case of an external port, it is also necessary to specify the **average incoming message size** and the **message arrival rate**.

Performance parameters for an outgoing message are the processing time, ProcessingTime, of the incoming message, the size, OutgoingMsgSize, of the outgoing message and the port, OutputPortName, where it is sent to. If there is the possibility of more than one message being generated and sent out of a different port, then it is necessary to specify the probability of each case.

For each entry interface, the entry name EntryName is specified. The performance parameters are the processing time, ProcessingTime, of the incoming message and the outgoing message size, OutgoingMsgSize. The outgoing message is sent out on the same interface.

**XML Notation for Performance Annotations of Connectors.** There are two types of connector: Client Connector and Server Connector.

The specification in XML notation for a client connector's performance parameters is:

```
<ClientConnector>
  <ClientConnectorName> string
     </ClientConnectorName>
  <ClientConnectorType> asynchronous | synchronous
    | brokered </ClientConnectorType>
  <ClientStubTime> real sec </ClientStubTime>
</ClientConnector>
```

The Client Connector has a name as well as a ClientConnectorType, which can be asynchronous, synchronous, or brokered. The ClientStubTime is specified, which is the CPU time required to process a message and/or response.

The specification in XML notation for a server connector's performance parameters is as follows:

```
<ServerConnector>
  <ServerConnectorName> string
    </ServerConnectorName>
  <ServerStubTime> real sec </ServerStubTime>
  <NumberServerThreads> integer
    </NumberServerThreads>
  <MaxNumServerConnections> integer
    </MaxNumServerConnections>
</ServerConnector>
```

The Server Connector has a name. The server stub time, ServerStubTime, is specified, which is the CPU time required to process a message and/or response. The number of server threads is specified. For a sequential server, this value is equal to 1. For a concurrent server, this value is greater than 1. In addition, the maximum number of server connections is specified.

**XML Notation for Performance Annotations of Resources.** The specification in XML notation for a resource's performance parameters is given next. At this time, there is only one resource specified, namely the network that connects components and connectors.

The network parameters are the latency and the bandwidth.

```
<resource>
  <ResourceName> string </ResourceName>
  <latency> real msec </latency>
  <bandwidth>  real Mbps </bandwidth>
</resource>
```

# 5 Performance Models of Component Interconnection Patterns

In this section, we describe the performance models of the Component Interconnection Patterns for synchronous and asynchronous communication. These models are based on previous work in performance modeling of software systems [11,13,14,15,26].

## 5.1 Synchronous Communication – Fixed Number of Threads

Here we assume that the client sends a request to a multi-threaded server. Each request is handled by a single thread. The number of threads is fixed. All threads are assumed to be created at server initialization time. A queue of requests to be processed by the server arises. It is assumed that the server uses both CPU and I/O in order to process the request.

Figure 7 shows the performance model for a synchronous Client/Server (C/S) interaction. The number of requests within the dashed box cannot exceed the number of threads *m.* If there are more requests than available threads, the additional requests have to wait in the queue of threads.

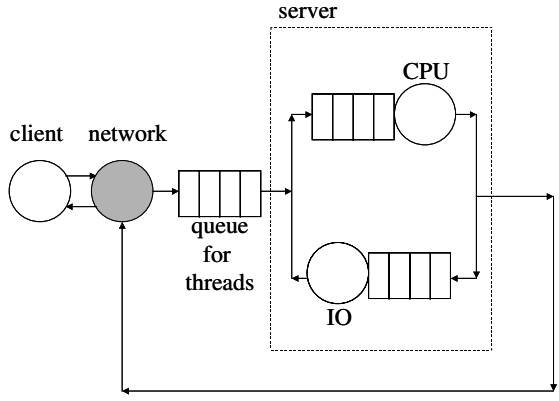



**Fig. 7.** Performance Model for Synchronous C/S Interaction with Fixed Number of Threads

The queuing model for the server consists of a hierarchical model in which the server throughput $X(n)$ for $n = 0, \ldots, m$ is obtained by solving the queuing network that represents the server (CPU and IO) using Mean Value Analysis. Then, using the decomposition principle [13], the entire server including the queue for threads is modeled using a Markov Chain model.

In a closed system, the number of clients, *N,* is fixed and the average think time, *Z,* per client has to be known. Figure 8 shows a graph of response time versus request arrival rate for different values of the number of threads. The curves show that as the number of threads increases, the response time decreases and the maximum achievable throughput increases.

### 5.2 Synchronous Communication – Dynamic Thread Pool

In the case of dynamic thread pool, each time a request is created, a new thread is created to serve the request. When the request is complete, the thread is destroyed. In this case, there is no queue for threads. The corresponding queuing model is shown in Fig. 9.

The model of Fig. 9 can be solved using a non-hierarchical queuing network model given that there is no limit in the number of threads in the system. The tradeoff in this case is between the time to wait for an existing thread and the time to create and destroy threads. Fig. 10 shows a graph that illustrates this tradeoff. The graph shows the average response time versus the number of threads in the fixed number of threads case under the assumption that the time to create and destroy a thread is one third of the time to execute the thread. As it can be seen, there is a cross over point for six threads. Below this limit, the dynamic thread case provides a better response time because of the excessive time in the thread queue. As more threads become available in the thread pool, the thread waiting time decreases in a more than linear fashion. For more than six threads, the thread creation and destruction time exceeds the thread waiting time.

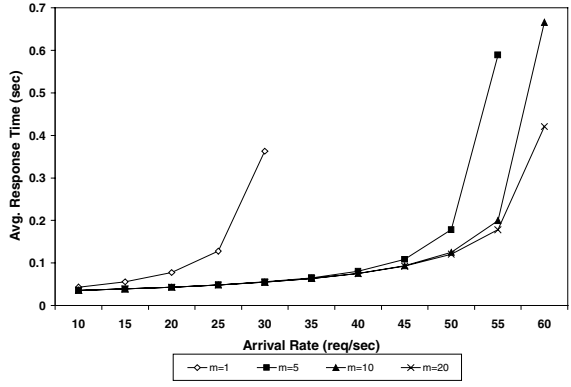



**Fig. 8.** Response Time vs. Arrival Rate for Different Values of the Number of Threads for Synchronous Communication and Fixed Number of Threads

### 5.3 Asynchronous Communication

The asynchronous communication case includes the case in which the client goes through three phases:

1. Phase 1: local processing, at the end of which a request is sent to the server. The client does not stop while waiting for the request and proceeds to phase 2. Let $t_1$ be the average local processing time at this phase.
2. Phase 2: local processing while waiting for the reply from the server. If the reply from the server does not arrive before this processing, of duration $t_2$, is complete, the client stops and waits for the reply. Otherwise, the client finishes execution of its phase 2.
3. Phase 3: the client does local processing based on the reply received from the server. Let $t_3$ be the duration of this phase.

The type of asynchronous communication discussed here can be implemented with multithreaded clients or with callbacks. To model asynchronous communication, an approximate iterative model is used.

Figure 11 shows the throughput, measured in requests/sec, versus the number of clients for different values of $t_2$. The following observations can be made. After a certain number of clients, 50 in Fig. 11, the throughput is the same for all values of $t_2$ considered. This happens because the server will be congested and the reply will always be received after $t_2$ expires. In a lighter load range, the throughput increases as $t_2$ decreases. This happens because clients are able to send requests at a higher rate.

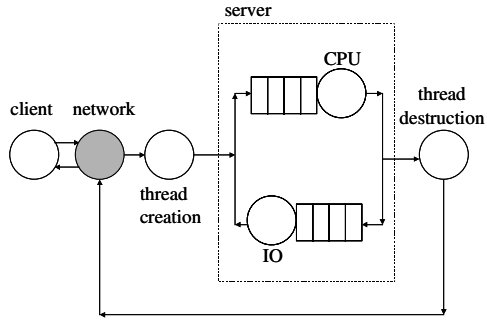



**Fig. 9.** Performance Model for Synchronous C/S Interaction with a Dynamic Thread Pool

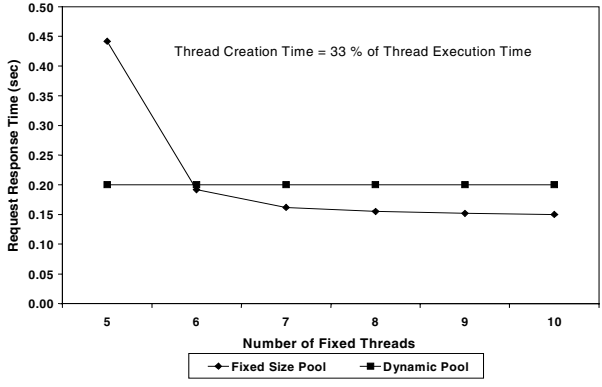



**Fig. 10.** Tradeoff Between the Fixed Number of Threads and the Dynamic Thread Pool Cases

## 6 Conclusions and Future Work

This paper described an approach for the design and performance modeling of component interconnection patterns. We start with UML design models of the component interconnection patterns. We then provide performance annotations of the UML design models. The area of performance annotation of UML design with cor-

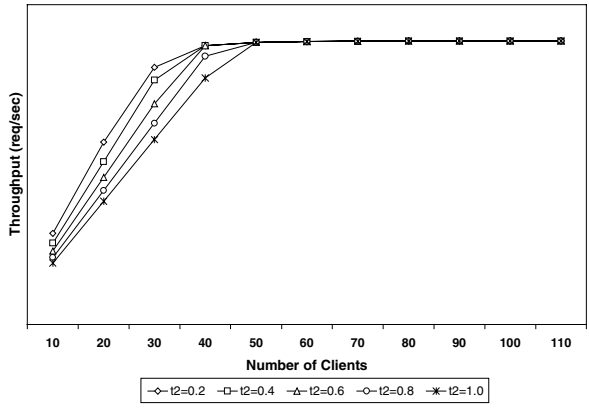



**Fig. 11.** Throughput vs. Number of Clients for Asynchronous Communication

responding performance models has attracted significant attention recently as indicated by various papers published in the 2000 Workshop on Software and Performance [12]. We then map the performance annotated UML design model to a performance model, which allows us to analyze the performance of the software architecture executing on various system configurations. This paper has described the case of synchronous and asynchronous communication with a multi-threaded server. We have also modeled component interconnection patterns for brokered communication, although there is insufficient space to describe this.

We plan to extend the work developed in this project by providing a language-based method for the design and performance analysis of component-based systems prior to their implementation. For that purpose, we plan to specify a Component-based System and Performance Description Language (CBSPDL). The goal is to specify both the architecture and performance of a large component-based distributed system in terms of new components, as well as predefined components and inter-component communication patterns that are stored in a reuse library. Thus, our goal is to expand our research in the direction of advancing the methods and software technology needed to reduce the costs and risks in the software development of large and complex distributed software systems.

**Acknowledgements.** This research was supported in part by the National Science Foundation through Grant CCR-9804113.

## References

1. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley (1998)
2. Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modeling Language User Guide. Addison Wesley, Reading MA (1999)
3. Buschmann, F. et al.: Pattern Oriented Software Architecture: A System of Patterns. Wiley (1996)
4. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley (1995)

5. Gomaa, H., Menascé, D.A.: Design and Performance Modeling of Component Interconnection Patterns for Distributed Software Architectures. Proc. 2000 Workshop Software and Performance, Ottawa, Canada, Sept. (2000)
6. Gomaa, H., Menascé, D.A., Kerschberg, L.: A Software Architectural Design Method for Large-Scale Distributed Data Intensive Information Systems. Journal of Distributed Systems Eng., Vol. 3(1996) 162-172
7. Gomaa, H.: Use Cases for Distributed Real-Time Software Architectures. Journal of Parallel and Distributed Computing Practices, June (1998)
8. Gomaa, H.: Designing Concurrent, Distributed, and Real-Time Applications with UML. Addison Wesley (2000)
9. Hofmeister, C., Nord, R.L., Soni, D.: Applied Software Architecture. Addison Wesley (2000)
10. Kazman, R., Barbacci, M., Klein, M., Carrière, S.J., Woods, S.: Experience with Performing Architecture Tradeoff Analysis. Proc. IEEE International Conf. on Software Engineering, IEEE Computer Soc. Press (1999)
11. Menascé, D.A., Almeida, V.A.F.: Client/Server System Performance Modeling. In: Haring, G., Lindemann, C., Reiser, M. (eds.): Performance Evaluation - Origins and Directions. Lecture Notes in Computer Science, Springer-Verlag (2000)
12. Menascé, D.A., Gomaa, H., Woodside, M. (eds.): Proc. of 2000 Workshop on Software and Performance. ACM Sigmetrics, Ottawa, Canada, Sept. 17-20 (2000)
13. Menascé, D.A., Almeida, V.A.F., Dowdy, L.: Capacity Planning and Performance Modeling: from mainframes to client-server systems. Prentice Hall, Upper Saddle River, New Jersey (1994)
14. Menascé, D.A., Almeida,V.A.F.: Two-level Performance Models of Client-Server Systems. Proc. 1994 Computer Measurement Group Conf., Orlando, FL, December (1994)
15. Menascé, D.A.: A Framework for Software Performance Engineering of Client/Server Systems. Proc. 1997 Computer Measurement Group Conf., Orlando, December (1997)
16. Menascé, D.A., Gomaa, H.: A Method for Design and Performance Modeling of Client/Server Systems. IEEE Tr. on Software Engineering, 26(2000)11
17. Menascé, D.A., Gomaa, H.: On a Language Based Method for Software Performance Engineering of Client/Server Systems. First International Workshop on Software Performance Eng., New Mexico, Oct. (1998)
18. Menascé, D.A., Gomaa,H., Kerschberg, L.: A Performance-Oriented Design Methodology for Large-Scale Distributed Data Intensive Information Systems. Proc. First IEEE International Conf. on Eng. of Complex Computer Systems, Southern Florida, USA, Nov. (1995)
19. Magee, J., Dulay, N., Kramer, J.: Regis: A Constructive Development Environment for Parallel and Distributed Programs. J. Distributed Systems Engineering (1994) 304-312
20. Orfali, R., Harkey, D., Edwards, J.: Essential Client/Server Survival Guide. Wiley, Third Ed. (1999)
21. Robbins, J.E., Medvidovic, N., Redmiles, D.F., Rosenblum, D.: Integrating Architectural Description Languages with a Standard Design Method. Proc. Intl. Conf. on Software Engineering, Kyoto, Japan, April (1998)
22. Rumbaugh, J., Booch, G., Jacobson, I.: The Unified Modeling Language Reference Manual. Addison Wesley, Reading, MA (1999)
23. Selic, B., Gullekson, G., Ward, P.T.: Real-Time Object-Oriented Modeling. John Wiley & Sons (1994)
24. Shaw, M., Garlan, D.: Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall (1996)
25. Magee, J., Kramer, J.: Concurrency: State Models & Java Programs. John Wiley & Sons (1999)
26. Ferscha, A., Menascé, D.A., Rolia, J., Sanders, B., Woodside, M.: Performance and Software. In: Reiser, M., Lindemann, C., Haring, G. (eds.): System Performance Evaluation: Origins and Directions. Schloss Dagstuhl Report no. 9738, September (1997)

# Conflicts and Trade-Offs between Software Performance and Maintainability

Lars Lundberg, Daniel Häggander, and Wolfgang Diestelkamp

Department of Software Engineering, Blekinge Institute of Technology,
P.O. Box 520, S-372 25 Ronneby, Sweden
{llu, dha, wdi}@ipd.hk-r.se

**Abstract.** This chapter presents experiences from five large performance-demanding industrial applications. Performance and maintainability are two prioritized qualities in all of these systems. We have identified a number of conflicts between performance and maintainability. We have also identified three major techniques for handling these conflicts. (1) By defining guidelines for obtaining acceptable performance without seriously degrading maintainability. (2) By developing implementation techniques that guarantee acceptable performance for programs that are designed for maximum maintainability. (3) By using modern execution platforms that guarantee acceptable performance without sacrificing the maintainability aspect. We conclude that the relevant performance question is not only if the system meets its performance requirements using a certain software design on a certain platform. An equally interesting question is if the system can be made more maintainable by changing the software architecture and compensating this with modern hardware and/or optimized resource allocation algorithms and techniques.

## 1 Introduction

High performance in terms of throughput and response times is always desirable, and in some real time applications it is imperative that the system responds within certain time limits. Performance is, however, not the only important quality when developing software systems; reducing the cost and time to market is equally important. It is well known that roughly 80 percent of a typical software system's cost occurs after initial deployment, in the maintenance phase (see page 32 in [1]). It is, therefore, very important to adopt design techniques that will minimize the maintenance cost, i.e., we want to obtain high maintainability.

A number of design techniques are used in order to obtain high maintainability, e.g., object-orientation, frameworks and design patterns [5]. Studies of large industrial real-time applications show that these techniques tend to generate excessive use of dynamic memory, which can degrade the performance seriously, particularly on multiprocessors [7,8].

One way of improving maintainability is to build flexible systems that can adapt to different requirements, i.e., in such cases the functionality of, and data formats used in, the system can, within certain limits, be changed without redesign. One way of

R. Dumke et al. (Eds.): Performance Engineering, LNCS 2047, pp. 56–67, 2001.

achieving this is to use meta-data organized according to the reflection architectural pattern [5]. Experiences from a flexible database system using meta-data organized in this way show that the performance penalties can be very sever, e.g., a factor of 10-30 lower performance than a traditional (inflexible) implementation [4].

The two examples above show that the ambition to build maintainable software systems can result in poor, or sometimes very poor, performance. The opposite is, however, also true, i.e., the ambition to build systems with high performance (and availability) can result in poor, or very poor, maintainability.

This means that there are situations where one has to make trade-offs and balance software performance (and availability) against maintainability and flexibility. There are, however, also situations where the conflicts can be avoided reduced or significantly.

We have looked at five large and performance demanding industrial applications. Based on our experiences we identify situations where conflicts occur. We also identify a number of techniques for handling these conflicts.

The rest of this chapter is organized in the following way. In Section 2 we give a brief description of the five applications that we have studied. Section 3 presents the different studies that we have done on the five applications and our experiences from these studies. In Section 4 we present we present different ways of handling these conflicts, i.e., our main results. Section 5 discusses our results and some related work. Our conclusions can be found in Section 6.

## 2 Industrial Cases

We have looked at five large industrial applications over a period of five years. All applications are large and performance demanding, i.e., performance and maintainability are important qualities in all applications. Four of the applications are from the telecommunication domain. The last application is a database application.

### 2.1 Billing Gateway (BGw)

The Ericsson BGw connects the operator's network with post processing systems. A post processing system can be any system interested in call data, e.g. billing systems or statistical systems (see Figure 1).

Each call generates a number of Call Data Records (CDRs). The CDRs are sent from the switches and other network elements to the BGw. After being processed, the data is sent to post processing system, e.g., a billing system that generates a customer bill. The BGw may change the format of the CDRs. It may also filter out some unbillable CDRs and separate CDRs for roaming calls from CDRs for non-roaming calls.

The BGw is written in C++ using object-oriented design techniques. The application consists of more than 100,000 lines of code. BGw is multithreaded and runs on Sun/Solaris multiprocessor servers with 1-16 processors depending on the performance requirements of the particular network operator. Scalability on multiprocessors is thus very important.

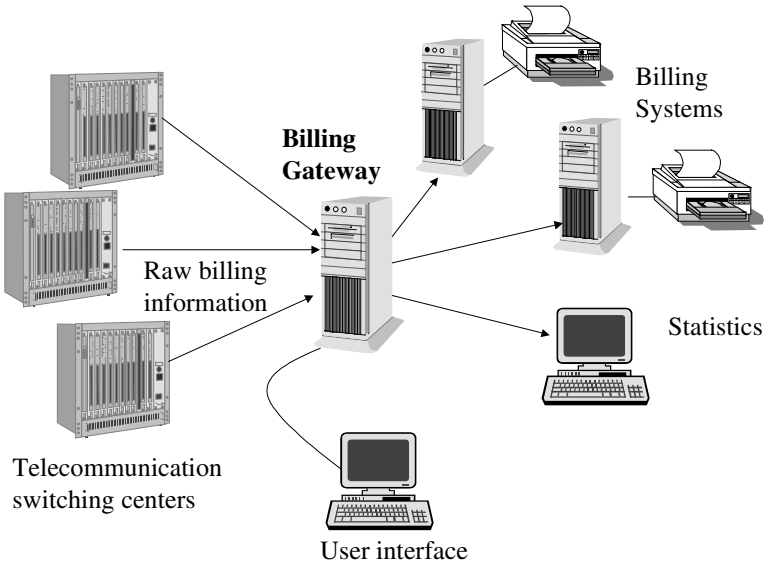



**Fig. 1.** The Billing gateway

## 2.2 Fraud Control Center (FCC)

When operators first introduce cellular telephony in an area their primary concern is capacity, coverage and signing up customers. However, as their network matures lost revenues due to fraud becomes more important. One type of fraud is cloning fraud, where the caller uses a false or stolen subscriber identity in order to make free calls or be anonymous. The Ericsson FCC is part of an anti-fraud system that combats cloning fraud with real-time analysis of network traffic.

Software in the switching centers provides real-time surveillance of suspicious activities associated with a call. The idea is to identify potential fraud calls and have them terminated. One single fraud indication is, however, not enough for call termination. The FCC makes it possible to define certain criteria that have to be fulfilled before a call is terminated, e.g., a certain minimum number of fraud indications with a certain time interval.

The FCC is written in C++ using object-oriented design techniques and Sun Solaris threads. The target platform is Sun multiprocessors.

## 2.3 Data Monitoring (DMO)

In order to detect network problems immediately, the network operators collect performance data continuously in real-time. Examples of performance data are the number of uncontrolled call terminations and the number of failed hand-over for each cell in the network. The DMO application collects performance data in real-time. If

certain parameters are outside of a user defined interval an alarm is generated. It is also possible to define an interval relative historic average values.

Two students groups were assigned the task of developing two DMO applications. The requirements where given by Ericsson. Each group consisted of four students working full time for 10 weeks. The groups had identical functional requirements. One group (DMO 1) was allowed to use a third party database system (Sybase), whereas the other group (DMO 2) had to use the standard Solaris file system. Both systems were going to run on Sun multiprocessors.

DMO1 implemented the system as one multithreaded program, and DMO 2 implemented the system as a number of Solaris processes that communicated with the database server.

### 2.4 Prepaid Service Data Point (SDP)

The SDP (Service Data Point) is a system for handling prepaid calls in telecommunication networks. The basic idea is simple; each phone number is connected to an account (many phone numbers can in some rare cases be connected to the same account), and before a call is allowed the telecommunication switch asks the SDP if there are any money left on the account. This initial message in the dialogue is called *First Interrogation*. The call is not allowed if there are no, or too little, money on the account.

If there are money on the account the call is allowed and the estimated amount of money for a certain call duration (e.g. three minutes) is reserved in a special table in the database (the account should not be reduced in advance). When the time period (e.g. three minutes) has expired, the telecommunication switch sends a request for an additional three minutes to the SDP. This type of message is called *Intermediate Interrogation*. The SDP then subtracts the previous reservation from the account and makes a new reservation for the next three minutes. This procedure is repeated every third minute until the call is completed. The telecommunication switch sends a message (*Final Report*) to the SDP when the call is completed, and at this point the SDP calculates the actual cost for the last period (which may be shorter than three minutes) and removes this amount from the account. Calls that are shorter than three minutes will only result in two messages, i.e., a First Interrogation message and a Final Report message.

The SDP is about 170,000 lines of C++ code, and the system was very hard to maintain. For performance and availability reasons, the SDP runs on two Sun multiprocessor computers. The distribution is handled in the application code, i.e., no commercial cluster software (or hardware) is used.

### 2.5 Promis Database

In 1995 the Swedish Telia, Dutch PTT and Swiss PTT started a project to implement a product database. The purpose of the system is to maintain information about products including prices, and to supply other systems (such as billing, logistics etc.) with necessary information. Because of different information layout and structure at the involved partners, a main requirement was maintainability, i.e., ease of changing

the information structure in the database. The result of this effort is the Promis database system.

Fig. 2 gives an overview over how Promis interacts with different systems. The Promis database is based on Oracle DBMS executing on a HP-UX machine. Clients for data maintenance, implemented in SQLWindows, are executed on PCs. They are connected to the server via Oracle's SQLNet. Different external systems are supplied with information either in batch jobs (run once every night) or as on-line accesses in real-time to Promis. Additionally, information from Promis is made available on the customer's intranet via a web-client, using Oracle's Webserver.

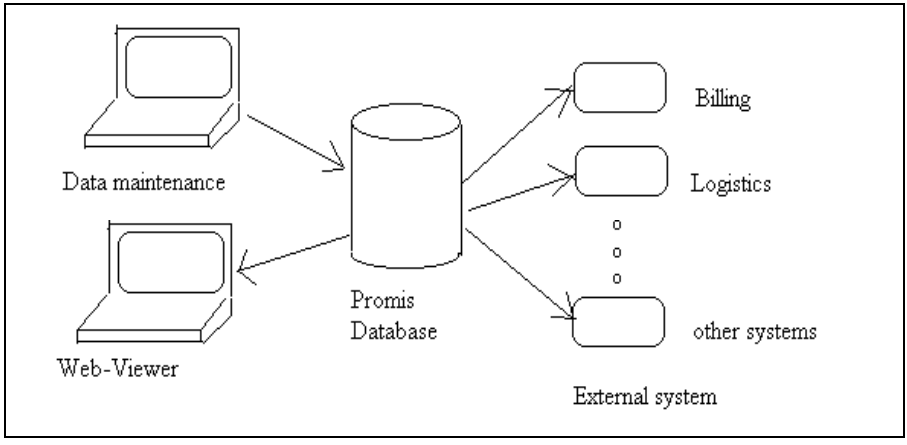



**Fig. 2.** The Promis environment

Promis uses an uncommon approach: A flexible database administration system (or structural engine) is built on top of an existing DBMS. We used a meta-data approach, i.e., the information in the database tables is interpreted via special system tables (the meta-data). One consequence of the meta-data approach is that the total number of tables is always equal to seven, independent of the actual objects stored in the database. Another, and more interesting, consequence is that user can create and store new types of objects without changing the source code. This reduces the maintainability cost.

## 3 Studies and Experiences

In this section we present the experiences and studies that we have done on the different applications. The presentation in this section summarizes the major results in a number of previous investigations. The complete report of each individual investigation is included in the reference list at the end of the chapter. The combined experiences from all of the studies, i.e., the major result in this chapter, are provided in Section 4.

### 3.1 Billing Gateway (BGw)

We did a number of multiprocessor performance evaluations on the BGw. Since the system is multithreaded and since there are a lot of parallelism (there can be up to 100 network elements connected to each BGw), we expected a good speedup. The actual speedup was, however, very disappointing [7]. In the first version the performance actually dropped when the number of processors increased because the dynamic memory management system, i.e., the global heap, was a major bottleneck.

It turned out that the excessive use of dynamic memory was caused by the designers ambition to create a flexible and maintainable system, i.e., this ambition lead to a fine grained object structure were anticipated changes are insolated within a small (sub-)object. One result of this is that each the processing of each CDR involves a large number of allocations and deallocations of dynamic memory.

It transpired that a parallel heap implementation, e.g., SmartHeap for SMPs [15], ptmalloc [6] or hoard [2], could solve the dynamic memory problem. We also evaluated application specific object pools. The application specific object pools resulted in much higher performance (roughly a factor of eight) compared to the parallel heap implementations. However, the parallel heap implementations resulted in an order of magnitude higher performance than the original implementation. The obvious drawback with the application specific object pools is that they are hard to maintain as new versions of the BGw application are developed.

We also did a study of the BGw where the system was broken up into components [12]. The reason for breaking the system into components was to decrease the maintainability cost, i.e., each change would be isolated to a component. Another effect of the component-based approach was that the system could be easily split into a number of processes that could be distributed to different computers. It turned out that the component-based prototype had much less problems with dynamic memory contention than the original (monolithic) implementation, i.e., in this case there was no conflict between maintainability and performance.

### 3.2 Fraud Control Center (FCC)

The experiences from the FCC are similar to those from the BGw [8]. The FCC had major speedup problems due to the dynamic memory management system. We initially evaluated two ways of solving the dynamic memory problem in the FCC:

- We replaced the ordinary heap with a parallel heap (ptmalloc).
- We changed the design of the FCC from one multithreaded program to a number of (single-threaded) processes.

The performance characteristics of these two solutions were quite similar, and both solutions resulted in a much better speedup compared to the original implementation.

We then did a more detailed study of the FCC and found that most of the dynamic memory problems came from a parser object in the FCC [9]. Interviews with the designers showed that the excessive use of dynamic memory was again caused by their ambition to build a maintainable and flexible system. The interviews also showed that the designers were rather happy with the result, i.e., they thought that the

parser was maintainable and could be easily adapted to new formats. The interviews also showed that the designers had not anticipated any performance problems due to their design.

We now implemented an alternative parser design. This design was very rigid (and thus small), i.e., one could not adapt it to other formats. Consequently, in order to support new formats the parser had to be completely redesigned. It turned out that the rigid parser had eight times higher performance than the original flexible parser. Even more interestingly, it turned out that the rigid parser was eight times smaller than the flexible parser. Small code size is one of the most important qualities for reducing the maintainability cost. State-of-the-art maintainability estimation techniques showed that the cost of writing a new small and rigid parser was probably less than the cost of adapting the large flexible parser to a new format. Consequently, the maintainability of the small and rigid approach was at least as good as the maintainability of the flexible approach. Consequently, in this case the conflict between performance and maintainability was based on myths and misconceptions, and could thus be avoided.

### 3.3 Data Monitoring (DMO)

It turned out that the DMO version (DMO 1) that used a commercial database system and Solaris processes for expressing parallel execution had better multiprocessor scale-up than the multithreaded version (DMO 2). Again, the reason for the poor scale up of the multithreaded version was problems with the dynamic memory. The single-processor performance of DMO 2 was, however, higher than the single-processor performance of DMO 1. The experiences from DMO also show that commercial database systems seem to be highly optimized and do not cause any performance problems on SMPs. The use of a commercial database system decreased the size of the application program and will thus surely decrease the maintainability cost.

### 3.4 Prepaid Service Data Point (SDP)

The BGw and FCC applications were originally optimized for high maintainability. The SDP is, however, highly optimized for high performance and availability. One implication of the optimization efforts was that no commercial DataBase Management Systems (DBMS) was used; instead of using a DBMS the designers implemented the database functionality as part of the application code. Another implication of the performance optimization was that the designers implemented fault tolerance, i.e., redundant distributed computers and components, as part of the application code, instead of using standard cluster solutions.

The SDP was very hard to maintain. In order to reduce the amount of code, and thus the maintenance cost and time to market for new versions of the SDP, we developed a version that used standard DBMS and cluster solutions. The code reduction was dramatic; the new version (with the same functionality) has about 15,000 lines of code, which is an order of magnitude less than the code size of the original SDP. The performance of the new version is also acceptable (it was in fact somewhat better than the performance of the original version).

In order to facilitate this kind of dramatic size reduction without jeopardizing the performance and availability requirements, we had to use high-end hardware and

software. The most spectacular example was the use of a fault tolerant computer from Sun. We did, however, develop a simple set of guidelines and a design strategy that minimized the additional hardware and software cost; the hardware and third party software cost of the new SDP is only 20-30% higher than the hardware and third party software cost of the old SDP.

### 3.5 Promise Database

The Promis database permits interesting features such as run time definition of data structures and wild searches [4]. We obtain this through the use of meta-tables, i.e., tables that contain information about the record formats defined by the user. This structure makes the system extremely flexible and easy to maintain, i.e., one does not need to modify the system in order to change the record format in the database. The price for this flexibility is decreased performance.

The cost for using the Promis approach compared to a standard implementation increases almost linearly with the number of properties, i.e., the cost for using the Promis model becomes very high for database objects with many properties. Although the cost for creating new objects is high in Promis, the biggest problem is the cost for retrieving data from the database. The reason for this is that retrieving data is much more frequent than creating new objects. The ratio between create and retrieve is about 1 : 450.

The major price for the flexibility of the Promis approach is, therefore, the cost for retrieving data objects with a large number of properties. Having identified this problem, we optimized retrieval by storing all properties associated with an object as one block. This decreases the cost for retrieval to the same order of magnitude as for ordinary database implementations (i.e. within a factor of 2-3). The disadvantage with this solution is that we introduce overhead, which consumes more disk space and increases the cost for creating and updating an object. However, since search and retrieve are the dominating operations, the optimization pays off, e.g., the slow down compared to a regular model is reduced from a slow down factor of 16.14 to a factor of 2.80 for objects with 25 properties, which is a realistic object size.

## 4 Results

In this section we present the accumulated experiences from our studies.

The experiences from the rigid versus flexible parser in the FCC and the component-based BGw show that there need not always be conflicts between maintainability and performance. The DMO experience shows that there was a conflict between single-processor performance and maintainability, but this conflicts is reduced on multiprocessors since the more maintainable version (DMO 1) has better multiprocessor scale-up. The other examples show, however, that there seem to be situations where the ambition of obtaining high maintainability results in unacceptable poor performance, and the ambition to obtain high performance results in acceptable poor maintainability.

The discussion in the previous section shows that we have investigated three different ways of handling such conflicts:

- *By defining design methods and guidelines for obtaining acceptable performance without seriously degrading maintainability.*

  Based on experience from a number of large industrial applications, we have defined a set of ten design guidelines and a process defining how to apply these guidelines to the application [10]. The process is primarily designed for migrating existing applications, which have been developed for single-processors, to multiprocessors. The process can, however, be useful also when developing new multiprocessor applications. The design guidelines are very effective, but some designers do not like the additional restriction of having a set of guidelines, and they may in some cases ignore the guidelines; failure to follow the guidelines may result in serious performance problems.

- *By developing resource allocation algorithms and implementation techniques that guarantee acceptable performance for object-oriented programs that are designed for maximum maintainability.*

  We have successfully used this approach on two kinds of programs: database programs optimized for maximum flexibility and thus also maximum maintainability (i.e. the Promis database), and object-oriented programs which tend to use excessive dynamic memory due to design optimizations with respect to maintainability.

  In the database case the performance of the flexible (but very slow) design was improved with up to almost one order of magnitude by using an alternative implementation technique. The alternative implementation technique was based on controlled redundancy; the major disadvantage of the technique was that we needed twice as much disk space.

  The dynamic memory problem could to, some extent, be reduced using an optimized dynamic memory handler, e.g., SmartHeap for SMP [15], ptmalloc [6], and hoard [2]. We have, however, found that these techniques are not always sufficient. The experiences from the application specific object pools in the BGw show that there is room for significant performance improvements. Inspired by this observation, we have developed an automatic method and that can obtain better performance than the parallel heap approaches. The performance improvement is obtained by using compile time information, which is not possible in the dynamic heap approach. The result of this line of research is a technique called Amplify, which is currently Pat. Pending.

- *By using modern execution platforms, e.g., multiprocessors and disk-arrays, that will guarantee acceptable performance without sacrificing the maintainability aspect.*

  The experiences from the SDP, FCC and DMO projects show that modern DBMS are often efficient, and it is difficult to improve performance by implementing the database functionality as part of the application program. Moreover, commercial DBMS often use multiprocessors and disk-arrays very efficiently. The effect of this is that one can often obtain the desired level of performance by buying a larger multiprocessor, which is generally a very cost-effective alternative to complicated performance optimizations in the application code. A study of the flexible Promis database application discussed above showed that data-striping (RAID 0) and range partitioning on disk-arrays can to a large

extent compensate the additional overhead caused by the flexible meta-data approach.

We have also developed guidelines on how to partition the application into different parts and then apply the high-end hardware and third party software to the parts where it is really needed. In short, the guidelines say that one should first separate the performance critical parts from the non-performance critical parts. The performance critical parts should then further be divided into a state-less part and a state-full part, where one should try to keep the state-full part as small as possible. The high-end hardware and third party software components are then concentrated to the state-full and performance critical part. These guidelines minimize the extra cost for hardware and third party software, and in the SDP application the additional cost was limited to 20-30%, which was quite acceptable considering the significant reduction of the maintainability cost.

## 5 Discussion

Others have also identified the need for design trade-offs. In the August issue of IEEE Computer the provost of Stanford John Hennessy writes "Performance – long the centerpiece – needs to share the spotlight with availability, maintainability, and other quality attributes" in his article on The Future of System Research [11]. The Architecture Trade-off Analysis Method (ATAM), proposed by Kazman et al. [14], addresses the need for making trade-offs between different quality attributes. One important difference between our guidelines and ATAM is that the ATAM work concentrates on identifying so called trade-off points, i.e., design decisions that will affect a number of quality attributes. There are no guidelines in ATAM on how to modify the software architecture.

Conflicts between quality attributes such as performance and maintainability have sometimes been attacked early in the design process, at the requirements level. Boehm and In have developed a knowledge-based tool that helps users, developers, and customers analyze requirements and identify conflicts among them [3].

Some of the dynamic memory problems were caused by the fine-grained object-oriented design. State-of-the-art design techniques are based on design pattern, and up to recently the design patterns did not consider performance and multiprocessor issues. There are a number of recent papers that identify this problem and discuss new and adapted design patterns. Douglas C. Schmidt is one of the leading researchers in this area. More information about the research activities in this filed can be found on his homepage [16].

A lot of researchers have looked at efficient implementations of dynamic memory, Wilson et al have written en excellent survey [17]. The main focus in this area has, however, been memory allocators for sequential programs. Some programming languages, e.g., Java, use garbage collection for unused dynamic memory. Extensive research has been done on garbage collection techniques, incremental and concurrent garbage collection in particular [13].

## 6 Conclusions

To sum up, we have during a period of five years studied five large industrial applications where performance and maintainability are important quality attributes. The average size of these applications is approximately 100,000 lines of code. The experiences show that there can indeed be conflicts between maintainability and performance. Some of these conflicts are based on myths and misconceptions, but some are inherent. The inherent conflicts can in some cases be handled by using modern hardware and/or optimized resource allocation algorithms and techniques. In some other cases, one must obtain a reasonable trade-off between maintainability and performance, and we have defined a set of guidelines and a simple development process for obtaining such trade-offs.

The fact that performance and maintainability are, to some extent, exchangeable puts software (and hardware) performance engineering in a new and interesting perspective. The relevant performance related question is not only if the system meets its performance requirements using a certain software design and architecture on a certain hardware and operating system platform. An equally interesting question is if the system can be made more maintainable by changing the software architecture and compensating this with modern hardware and/or optimized resource allocation algorithms and techniques.

## References

1. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley (1998)
2. Berger, E.D., Blumofe, R.D.: Hoard: A Fast Scalable, and Memory Efficient Allocator for Shared Memory Multiprocessors. http://www.hoard.org (site visited December 21 2000)
3. Boehm, B., In, H.: Identifying Quality-Requirement Conflicts. IEEE Software, March (1996)
4. Diestelkamp, W., Lundberg, L.: Performance Evaluation of a Generic Database System. International Journal of Computers and Their Applications, ISSN 1076-5204, September, 7(2000)3
5. Gamma, E., Helm, R., Johnsson, R., Vilssides: Design Patterns. Addison-Wesley (1997)
6. Gloger, W.: ptmalloc homepage. http://www.malloc.de/en/index.html (site visited December 21 2000)
7. Haggander, D., Lundberg, L.: Optimizing Dynamic Memory Management in a Multithreaded Application Executing on a Multiprocessor. In: Proceedings of the 27th International Conference on Parallel Processing, Minneapolis, USA, August (1998)
8. Haggander, D., Lundberg, L.: Memory Allocation Prevented Telecommunication Application to be Parallelized for Better Database Utilization. In: Proceedings of the 6th International Australasian Conference on Parallel and Real-Time Systems, Melbourne Australia, November (1999)
9. Haggander, D., Bengtsson, P.O., Bosch, J., Lundberg, L.: Maintainability Myth Causes Performance Problems in SMP Applications. In: Proceedings of the 6th Asian-Pacific Conference on Software Engineering, Takamatsu, Japan, December (1999)
10. Haggander, D., Lundberg, L.: A Simple Process for Migrating Server Applications to SMPs. Journal of Systems and Software, to appear (2001)
11. Hennessy, J.: The Future of System Research. IEEE Computer, August (1999)

12. Hermansson, H., Johansson, M., Lundberg, L.: A Distributed Component Architecture for a Large Telecommunication Application. In: Proceedings of the 7th Asian-Pacific Conference on Software Engineering, Singapore, December (2000)
13. Jones, R., Lins, R.: Garbage Collection: Algorithms for automatic dynamic memory management. John Wiley & Sons (1998)
14. Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., Carriere,J.: The Architecture Tradeoff Analysis Method. Tech. Report, CMU/SEI-98-TR-008
15. Microquill: SmartHeap for SMP. http://www.microquill.com/smp (site visited December 21 2000)
16. Schmidt, D.: Douglas C. Schmidt homepage. http://www.cs.wustl.edu/~schmidt (site visited December 21 2000)
17. Wilson, P., Johnstone, M., Neely, M., Boles, D.: Dynamic storage allocation: A survey and critical review. In: Proceedings of the 1995 International Workshop on Memory Management, Kinross, Scotland, Springer-Verlag (1995)

# Performance Engineering on the Basis of Performance Service Levels

Claus Rautenstrauch and André Scholz

University of Magdeburg, Institute of Business and Technical Information Systems
P. O. Box 4120, 39016 Magdeburg, Germany
{rauten|ascholz}@iti.cs.uni-magdeburg.de

**Abstract.** Insufficient performance characteristics of database-based integrated information systems, which are typically used within a business environment, can lead to extensive costs. Performance engineering assures the development of information systems with defined performance characteristics. These characteristics can be defined as requirements with the help of performance service levels. They are fixed in a service level agreement, which is the basis of negotiations between customers, users and developers.

## 1 Introduction

One of the most critical non-functional quality factor is the performance characteristic of a software system. Although Moore's law (the performance of hardware doubles every eighteen months) is still valid, insufficient performance is a continuing topic in the area of database applications. Performance can be defined as the capability of an IT-system to process a given amount of tasks in a determined time interval.

Performance can be determined and analyzed subjectively and objectively. The subjective analysis describes a performance analysis from the system user's point of view. It describes qualitatively how the performance characteristic is perceived. Because of individual expectations and stress limits, these results are related to an individual person [1, p. 3]. In order to evaluate subjective performance reasonably, results have to be aggregated to clusters. Clusters can be formed by aggregating evaluated application functions or user queries, or by which work places the requests were initiated or on which dates requests occurred.

In opposite to that, objective performance analyses are based on the use of approved measurement methods and tools. Database management systems usually offer options for an objective performance analysis, like auditing, tracing or the logging of system characteristics in dynamic performance tables in the data dictionary. Furthermore, a series of tools for the objective performance analysis is available.

R. Dumke et al. (Eds.): Performance Engineering, LNCS 2047, pp. 68-77, 2001.

**Table 1.** Subjective and objective Performance Analysis

| | | Subjective Performance | |
|---|---|---|---|
| | | Adequate | Inadequate |
| Objective Performance | Adequate | No need for Adjustments | Unrealistic Expectations |
| | Inadequate | Need for Adjustments | |

Functions with objective inadequate performance characteristics have to be adjusted primarily (cf., table 1). However, functions with subjective adequate performance characteristics but objective inadequate performance characteristics have to be modified since they can hinder other functions in execution. An extensive data run for a report for example can considerably hinder the execution of other functions if a lot of users access the same resources in parallel. However, the report itself can be subjectively uncritical.

Functions, which are subjectively inadequate but objectively adequate, are problematic from the system management's point of view. There are unrealistic user expectations. In these cases, users have to be influenced in a way that they accept a weaker performance at least temporarily.

A lot of software projects as well as productive systems fail because of insufficient performance characteristics. Most time in the past, performance was not treated as an elementary quality and service factor within the customer-provider-relationship and within the software development. The main focus was on the functional specification. Performance has the same priority as functional requirements in semi-critical and critical systems.

## 2 Database-Based Integrated Information Systems

Database-based integrated information systems (database applications) are typically used within a business environment. Databases are used as a central integration technology.

An integration can refer to different elements. Integration can focus on data, functions, processes, methods and programs. During data integration, databases are combined. They can be integrated by a link-up or by the use of a common database on the basis of a common data model and scheme. A functional integration requires intense adjustments of the tasks in addition to the relation. Functions can be integrated into an information system on the basis of a common functional model. Process integration is based on the link-up of individual processes. During method integration, methods are adjusted in its application and combined where it is appropriate. Program integration refers to a relation of individual programs and program segments.

Horizontal and vertical integrations are distinguished. A horizontal integration refers to integration along the business process chain. A vertical integration refers to an integration of administrative, dispositive as well as planning and control systems.

Integration measures overcome artificial limits in a business environment, reduce expenditures for data input and improve data quality. Redundant data storage is avoided.

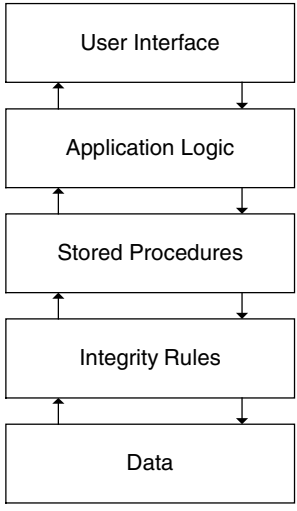



**Fig. 1.** Architecture of a database-based integrated Information System [6, p. 202]

Database-based integrated information systems refer to an integration of data and functions. They can be described by a 5-tier architecture (cf., figure 1). Every layer provides functions for the next higher layer.

All functions, which are directly available to the user, are described as the user interface. All functions, which are executed below the user interface, are described as the application logic. They support the tasks of the application. They are related to a specific client and have to be redesigned for the execution on another client. Stored procedures, integrity rules and data are stored in the database.

## 3 Performance Engineering

Up to now, performance characteristics are often only considered at the end of the software development process. If performance bottlenecks are only recognized at the end of the software development, extensive tuning activities, design modifications of the application or complete rejectings of entire software systems will be necessary.

A performance oriented software development method, like performance engineering, should be integrated in the engineering process. Performance engineering designates a collection of methods to support a performance-oriented system development of information systems along the entire software development process. Thus, adequate performance-related product qualities can be assured.

Performance engineering considers the response time as a design target throughout the whole software development process. Response time targets and analyses of the respective design structures are continuously compared. In early stages response time metrics can be quantified using estimation formulas, analytical models, simulation

models and prototypes. Existing process models have to be extended by measures of performance engineering. Performance characteristics are to be determined on the basis of design drafts, quantity frames and by the evaluation of existing data sources. Analyses are carried out cyclically after the evaluation of all available data sources.

Deviations lead to an immediate change in the software design. The system is analyzed up to the system life test. Thus, performance engineering guarantees an adequate response time behavior of the productive system.

Performance requirements are to be defined under the consideration of cost and time budgets in the system analysis phase. The system is also configured and sized on the basis of workload profiles in this phase. Data and functions are modeled in the system design phase. Data modeling refers to the set up of a target and distribution model, which can also be denormalized. Functional modeling refers to the assignment of functions to a layer of the database-based integrated information system. Functions can be scheduled for a temporal and local distribution. Database access costs are to be estimated with regard to the selectivity and use of indexes. Efficient algorithms should replace unfavorable procedures. Database accesses are analyzed in the implementation phase. Accesses can be redesigned or reformulated. Further on, the middleware, e.g., a database server, has to be optimized, for example by adjusting the physical storage structure and the memory management. Performance characteristics are verified in the test phase. In case of insufficient performance characteristics, tuning measures have to be performed.

One of the most important parts of this process is the determination of the required performance characteristics. Service level agreements are an instrument for the aggregation of performance requirements.

# 4 Performance Service Levels

A performance oriented system development requires a concrete specification of performance requirements. The use of the service level agreement concept to structure performance requirements is proposed in this contribution.

## 4.1 The Service Level Agreement Concept

Companies are faced with the fact that they have to design the costs structure of their IT services more transparent. Furthermore they have to compare their own cost rates with the market average [2, p. 302]. Small and mid-size companies have to outsource IT services, e.g., the development and maintenance of an application system, the preparation and the processing of mass E-Mails or a periodic data backup, to external service provider [4, p. 7]; [3, p. 5]. A written list of a specific set of IT services is collected in an IT service catalog.

Performance service levels allow a detailed specification of performance requirements. A sufficient specification of performance requirements is the basis for a comprehensive performance analysis of an information system. The performance characteristic can be quantified by performance metrics. A performance service level contains a set of lower bonds of quantified performance metrics [7, p. 252]. The more

performance metrics are used, the better the performance requirements are represented.

Performance services levels are fixed in a service level agreement (SLA). A SLA describes an agreement between an IT service provider and a customer about performance and corresponding service costs. It specifies the performance characteristics and other software quality factors in addition to functional requirements. The agreement is ratified as a compromise between the representatives of the customer, the management, the user and the developer. It is important to find a compromise solution since the performance requirements of the customer do not often correspond to the performance expectations of the user.

Internal and external SLA can be distinguished [5, p. 11]. Internal SLA are not based on a legal declaration, but the IT department has offer the IT services at negotiated performance levels. External SLA are a contractual frameworks between several companies. An external company offers the IT service.

### 4.2 Process Model

The structured development of a SLA requires a process model, which is described in the following.

**Services and Performance Service Levels.** The customer determines all services, e.g., the operation of an information system, in the first phase of the process (cf., figure 2). Services are functionally specified. In addition to the functional description, every service has to be concretized with time and spatial characteristics. The complete description of all services is collected in a service catalog.

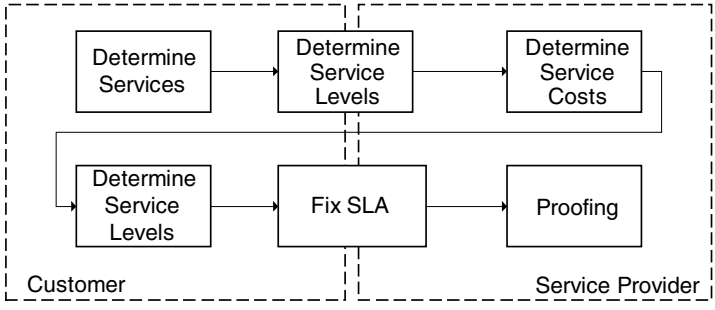



**Fig. 2.** Process Model for the Development of a SLA

The service catalog is handed over to the service provider in the second phase. He determines performance service levels for every service in consultation with the customer. The performance characteristics can be specified with the help of performance metrics.

Performance metrics are specific product metrics, which can be determined on different system layers and from different perspectives. There are a lot of classification schemes of performance metrics, e.g.:

- *Perspective:* Metrics can be determined from different perspectives, e.g., from the perspective of the service provider, which can lead to metrics like mean server utilization. Beyond it, they can be determined, e.g., from the perspective of the user, which can lead to metrics like mean response time.
- *Granularity:* Metrics can be oriented on different system granularities. Thus, they can focus on single system components, e.g., a CPU, which can lead to metrics like CPU processing time. On the other side they can focus on a whole system environment, which can lead to metrics like system response time.
- *Classical Subdivisions:* Performance metrics can be defined by their classical subdivisions, distinguishing three metric subgroups: throughput, utilization and time consumption. Thus, metrics also can be determined with regard to throughput, e.g., a transaction rate metric, to utilization, e.g., a CPU utilization metric, as well as to time consumption, e.g., a response time metric.
- *Internal / External Performance:* Another differentiation only distinguishes between internal and external performance. External performance means the time consumption of a user process including all necessary basis system processes. On the other hand, internal performance focuses on single IT services like a database but is less important, because when considering integrated information systems, processes of database-based information systems induce a lot of basis process with a versatile use of resources. Thus, in this case the informative value of internal performance metrics is low. For that reason, internal performance metrics don't play a major role within the performance specification of a SLA.

A couple of service levels due to different operation states can be defined. Therefore, an instance of every performance metric has to be assigned to every service level. A couple of service levels due to different operation states can be defined. Therefore, an instance of every performance metric has to be assigned to every service level. Thus, different performance characteristics are defined in different service levels. Thus, different performance characteristics are defined in different service levels. Furthermore, a scale needs to be assigned to every defined performance metric. Four kinds of scales, which are based on each other, can be distinguished:

- *Nominal scale:* There is a simple value assignment of symbols or numbers, which act as a descriptor. Even different values don't have a different character of expression. Values cannot be ordered or added. Merely, an equalization check is possible.
- *Ordinal scale:* There is an operational defined criterion in a rank order. The transitivity postulate is fulfilled. Thus, the scale is in the position to define a rank order over the values. Then comparison operators can be used.
- *Interval scale:* Equal large distances on the scale indicate equal large differences with regard to the specific performance characteristic. Addition and subtraction operations are defined.
- *Ratio scale:* There is an absolute or empirical neutral point. All mathematical operations can be applied.

For a wise application within a SLA, at least an interval scale is necessary. In case of the sales report, three performance service levels are defined, that are specified by two performance metrics (cf., table 7). The performance metric "mean response time"

characterizes an online access on the sales report. The performance metric "page throughput" characterizes the paper output rate of the sales report.

**Table 2.** Performance Service Levels

| IT-Service: Sales Report | | |
|---|---|---|
| **Performance Service Levels** | **Performance Metrics** | |
| | x: Mean Response Time (online) in seconds | y: Page Throughput in pages per minute |
| 1 | $x < 1$ | $8 < y$ |
| 2 | $1 \leq x < 3$ | $5 < y \leq 8$ |
| 3 | $3 \leq x < 9$ | $1 < y \leq 5$ |

Performance metrics put resource requirements of respective program segments in relationship with a planned hardware system. Since the hardware configuration of the productive system is not determined at the beginning of software development, the metrics refer to a hardware reference system. The reference system corresponds to a relative performance degree of 100%. As soon as the hardware configuration can be determined in more detail, the degree of performance is adapted to the new reference system. Only a tuple (performance value, relative performance degree) identifies specified performance data.

**Service Costs**. A cost rate is assigned to every performance service level in the next phase. In general, three techniques for the determination of service costs can be distinguished.

On the one side, actual costs of hardware, software, personnel, and so on are used in calculation models to determine the service costs. The determination is simple, but the results aren't transparent and don't allow market comparisons.

Furthermore, cost rates of services can be used, containing actual and future costs on the basis of ratios and estimable developments to determine the service costs, because of the fast development and the resulting cost changes in the IT branch. Costs have to be determined on the basis of the customer's requirements on the IT system. The customer can receive a respective price offer, which corresponds to his specific requirements, with the knowledge of these costs. With the help of this cost structure we can find out in future which services can be provided at which costs.

Further on, the target-costing concept can be applied. With the help of this concept it's tried to reach a fixed price for a specific service. Thus, companies can do market research to determine prices, which customers are willing to pay for a service. The resulting cost structures determine the scope of services.

**Table 3.** Example for Performance Service Level Cost Rates

| IT Service: Sales Report | |
|---|---|
| **Performance Service Levels** | **Price in / Month** |
| 1 | 620,– |
| 2 | 410,– |
| 3 | 180,– |

The customer decides itself in a further step for one or several services levels. Beside the individual requirements and budget, the decision depends on the performance service levels and its costs. If several services levels are linked together, the costs of the IT service are computed by a mixed calculation. Are for example 5% of all service uses on service level 1, 15% of all service uses on service level 2 and 80% of all service uses on service level 3, the total costs are computed as a sum of the weighted service level partial costs:

Total Costs = 0,05 * 620 + 0,15 * 410 + 0,8 * 180 = 236,50 .

**Fixing a SLA.** A SLA needs to be fixed in a written contract. The contract consists of an extensive service specification and further significant sections. In practice, good experiences are made with SLA encompassing the following sections:

- *Preamble:* The preamble describes the contracting parties, general goals of the SLA, and a mediator, who is responsible for interpretation and further modifications of the agreement. Furthermore, it states services that are not part of the agreement, but which might be expected to be part of it to avoid later conflicts.
- *Scope and Framework:* In this section, the scope of the agreement is described, e.g., involved departments, users, hardware, operating systems, and applications. Furthermore, the duty of the customer to cooperate is stated, e.g., to provide network interfaces, special types of terminals, or necessary qualification of the users.
- *Service Descriptions:* The service descriptions enumerates the services that have to be provided as well as the respective performance service levels reflecting the customer's demand. The services itself are defined in detail in the service catalogue. For that reason, the respective version of the service catalogue becomes (implicitly) part of the SLA.
- *Reporting:* Reporting is necessary to prove that a certain service level was met. For that reason, the SLA partners have to agree upon accuracy, measuring intervals, measuring tools, measuring methods, and recording methods for each service.
- *Data Security:* In this section, the provider guarantees the adherence of laws, rules, and guidelines concerning data security. Furthermore, physical access control and the use of security software, e.g., against viruses, are regulated.
- *Prices:* Here, all prices of each tuple (Service, Service Level), given in the service description section, are determined.
- *Contractual Period:* This section determines the period of time, in which the SLA is valid. Most time a SLA is negotiated on a one to three year basis. After that most conditions will be renegotiated.
- *Termination Clause:* This section regulates the contract termination modalities. Every party is in the position to step out the contract in respective cases.
- *Contract Penalty:* If a service provider fails to provide a negotiated level of service, he has to pay a contract penalty. In this section, all possible cases, which could result from a violation of performance service levels, should be considered. It should serve as an incentive for the provider to secure a proper delivery of services, but not as a means for the customer to reduce costs subsequently.

- *Organizational Settlements:* This section determines the organizational structuring of the SLA. Official reporting and communication channels are determined. In addition, contact persons for certain problem areas are agreed upon.

Similar findings are reported in literature as well, cf. e.g., (Pantry/Griffiths 1997, pp. 30-45). As an extension, SLA should be split into a static and dynamic part to simplify its evolution over time. The static part contains preamble, scope and framework, reporting, data security, contractual period, termination clause, contract penalty, and organizational settlements. The dynamic part includes service description and prices, which can be adapted to a changed requirements and environments. Every SLA is individual to a considerable degree. That's why there is no standard form for a SLA. Every section needs to be negotiated individually and precisely.

### 4.3 Proof of Service Levels

After fixing the SLA, the meeting of service levels during operation has to be proved. A performance service level is proved if the outcome of the corresponding metrics lies within the bounds given by the service level's definition.

The proof of a performance service level is mandatory for customers since this is their only way to assure quality. Besides quality assurance, monitoring, and proving, performance service levels offer an important way for providers to improve profitability. Take e.g., a service, whose response time is constantly on a better service level than agreed. In this case, it might be possible to run it on a server with weaker performance, which in return is less expensive, and further, to preserve more expensive resources for SLA with higher requirements and maybe a better return on investment (ROI).

## 5 Outlook

Performance service levels are a powerful concept for defining performance characteristics of purchased or in-house IT services. They can be determined at defined prices. Metrics are in the position to quantify service levels. The determination of metrics has to be analyzed very carefully, because it has to be ensured that all metrics can be measured and proved while providing the service.

Performance is a critical service factor of an information system. A service provider has to ensure a defined level of performance. Therefore, it is wise to pay attention on performance not only at the operation of a system while providing a service, but also just during development or purchase of a software system itself. If a service provider intends to purchase software, benchmarking results should be analyzed and compared with the own system environment. Results should come from third party consultants or independent benchmarking organizations, because most software providers use idealized cases for their benchmarks.

## References

1. Ghanbari, M., Hughes, C.J., Sinclair, M.C., Eade, J.P.: Principles of Performance Engineering for Telecommunication and Information Systems. London (1997)
2. Inmon, W., Rudin, K., Buss, C., Sousa, R.: Data Warehouse Performance. New York, NY et al. (1999)
3. Lacity, M., Hirschheim, R.: Information Systems Outsourcing. Myths, Metaphors and Realities. Chicester et al. (1993)
4. Loh, L., Venkatraman, N.: Determinants of Information Technology Outsourcing. A Cross-Sectional Analysis. Journal of Management Information Systems 9(1992)1, 7-24
5. Pantry, S., Griffiths, P.: The complete Guide to Preparing and Implementing Service Level Agreements. London (1997)
6. Rautenstrauch, C.: Integration Engineering. Bonn et al. (1993)
7. Scholz, A., Turowski, K.: Service Level Agreements of Performance Requirements. In: Bon, J. (ed.): World Class IT Service Management Guide. Amsterdam (2000) 249-256

# Possibilities of Performance Modelling with UML

Andreas Schmietendorf[1,2] and Evgeni Dimitrov[2]

[1]Otto-von-Guericke-Universität Magdeburg, Fakultät Informatik,
Institut für Verteilte Systeme, Postfach 41 20, D-39016 Magdeburg
schmiete@ivs.cs.uni-magdeburg.de
[2]T-Nova Deutsche Telekom Innovationsgesellschaft mbH,
Entwicklungszentrum Berlin, Wittestraße 30N, D-13476 Berlin,
A.Schmietendorf | Evgeni.Dimitrov@telekom.de

**Abstract.** The performance of an information system in the sense of its efficiency in relation to time and resources is largely determined by the design selected within software development. For an engineering procedure in the implementation of this quality feature, Performance Engineering, or PE for short, has emerged as a collection of the processes needed for this plus the appropriate methods and tools. The core concept of PE is to take account of the performance of a software system right from the early phases of development. Often, however, these measures are not performed explicitly during the development of the software, with reasons given such as deadline pressure, the extreme complexity of the information system or lack of knowledge of what Performance Engineering means. The present research gives an overview of the capabilities of UML-based PE. Corresponding approaches are analyzed and the advantages of the tools available are examined.

## 1 Introduction and Motivation

Examining performance at the end of software development is a common industrial practice. If performance problems are found with the architecture selected for the information system, this then leads to time-consuming tuning measures, the more expensive use of more powerful hardware than had been proposed or, in extreme cases, to the redesign of the application. This type of subsequent improvement always involves not only costs relating to the work on improving performance but also the cost of systems not being available on time. Corresponding examples of practical performance problems and their effect in terms of costs are given in [13].

Models are needed for PE [19]:

> *"Simple conceptual models of performance are needed to make performance engineering more accessible to and usable by the software engineering community."*

As mentioned before, modelling play an important role for an effective performance engineering. The used modelling notation should be based on UML ([2]), since this language is widely established as a standard within the software development.

The present article concentrates on analyzing the possibilities of a UML-based approach to performance modelling. For this, suitable requirements will be worked

R. Dumke et al. (Eds.): Performance Engineering, LNCS 2047, pp. 78–95, 2001.

out, existing approaches analyzed and subjected to an initial evaluation, and the possibilities for tool support summarized.

## 2 Requirements for a UML-Based Approach

Object-oriented modelling offers, for the first time, the possibility for shared data and function modelling. Using the UML notation which has largely been standardized by the OMG (Object Management Group), a suitable initial basis can be provided for corresponding performance models. The aim of performance modelling is firstly to verify the effects of architecture and design solutions in terms of their performance behaviour; secondly, the performance behaviour at the user interface is of interest in terms of response time and throughput. Whilst the first requirement can be implemented without direct consideration of the hardware and software strata below, a performance analysis of the entire system requires an examination of both the hardware and the software.

When studying the methods available for software engineering, a distinction must be made between those used for the modelling of real-time systems and those for conventional systems (frequently information systems). Whilst, in the modelling of real-time systems, it has been possible for years to explicitly take into consideration the performance behaviour through the use, for example, of expanded Petri networks, an SDL[1] to which performance aspects have been added or also automat theory models, this is not possible in the area of conventional systems without some form of further action. The reason for the difficulties lies in the complexity and abstraction of the underlying layers and the interactions between the software systems, which are sometimes impossible to follow. In addition, systems of this type are generally modelled using informal languages such as UML, which also makes a transfer to performance models more difficult. In the project presented here, the following aspects of performance engineering were thus examined:

1. Direct representation of performance aspects by means of UML and transfer of effective model diagrams into corresponding performance models.
2. Expansions of UML, in order to be able to deal with performance aspects.
3. Linking UML with formal description techniques (FDTs) such as SDL, MSC2.

Even though the form described in point 2 seems to be the best for the developer, it must be assumed, on the basis of current knowledge, that only points 1 and 3 will provide approaches that can be used in engineering terms.

Our analysis considers especially the following performance aspects:

- Workload model,
- User (external) requirements to the performance behavior,
- Performance measurement points
- System (internal) temporal constraints
- Resource usage.

[1] Specification and Description Language
[2] Message Sequence Charts

Considering the previously mentioned points, the following requirements for a UML-based approach can be summarized to give a framework for performance modelling:

- UML modelling style: use of UML diagrams for performance modelling,
- Automated (where possible) generation of performance models for analyzing the performance characteristics in the model,
- Clear delimitation between different architectural aspects, such as separate representations (diagrams) for application, infrastructure and distribution aspects,
- Supporting the iterative and incremental development of software.

# 3 Approaches for Performance Engineering Using UML

## 3.1 Direct Representation of Performance Aspects Using UML

With this approach, the aim is to use UML concepts and elements, without the addition of extension mechanisms, directly for the representation of performance aspects and to interpret them accordingly. UML provides some extension mechanism that can be used for a direct representation of performance aspects. These based on constraints, stereotypes and tagged values. By the use of this extensions to particular UML elements we are able to specify, derive and record performance information. The following describes the use of major UML concepts within the framework of a performance engineering project. We deliberately rejected the use of model elements in UML that provide no additional value for the PE tasks.

**Use Case Diagrams.** *Use cases* are used to describe the working load model (user behavior). An use case diagram models, in particular, the system's external stimuli. The actors can serve here as a basis for defining the working load in the system. It is not particularly useful to represent each user of the system by an actor. Instead, the actors should be assigned a number of roles (or single roles by way of exception). This means that every actor can represent a part of the system working load.

A process based on ISO 14756 can be used for the formalized description of the working load within the framework of use case diagrams. This requires the following steps, which are roughly outlined below. In accordance with the ISO method, these define a synthetic user. User can be "human" or "machine".

- Identification of **activity types AT** and recording the number and type of user, (examples: submit, edit, newURL,...). Activity Types are defined by the execution of the same algorithm.
- Derivation of **task types TT** through assignment of ATs to task mode (dialogue or batch job) and definition of service level requirements (timeliness Function).
- Definition of **chain types CT** as fixed sequences of task types.
- Defining the percentage **occurrence frequency q** (exact in ISO: relative chain frequency) and the **user preparation time UPT** (thinking and input times) for concrete CT whilst the program is being run.
- Laying down the reference value of **mean execution time $t_{Ref}$** (response times) for concrete task types in the form of the following notation:

$$t_{Ref} \text{ for } TT_j \text{ should take place at } x_{\%} \text{ in} \le n \text{ sec. and } y_{\%} \le n+5 \text{ sec., at } x_{\%}+y_{\%}=100_{\%}$$

Use cases described in this way (save as e.g. annotation) cannot be directly used to derive performance information. This only becomes possible if they are supported through scenarios and related interaction diagrams. However, the load model described provides the ideal conditions for the production of performance models. In addition, it can also be used as an input variable for corresponding benchmark tests using load driver systems.

Furthermore, the individual Use Cases can become weighted through timing Constrains. These time restrictions may be derived from customers demand, knowledge via the application area, expert knowledge and so forth. These are roughly time requirements for the specific use case. During a later description of the use case through scenarios (representation by means of Sequence diagrams), these can be described in more detail. The evaluation of the access from actors to the use cases, can be interpreted as specification of the workload correspondingly ISO 14756.

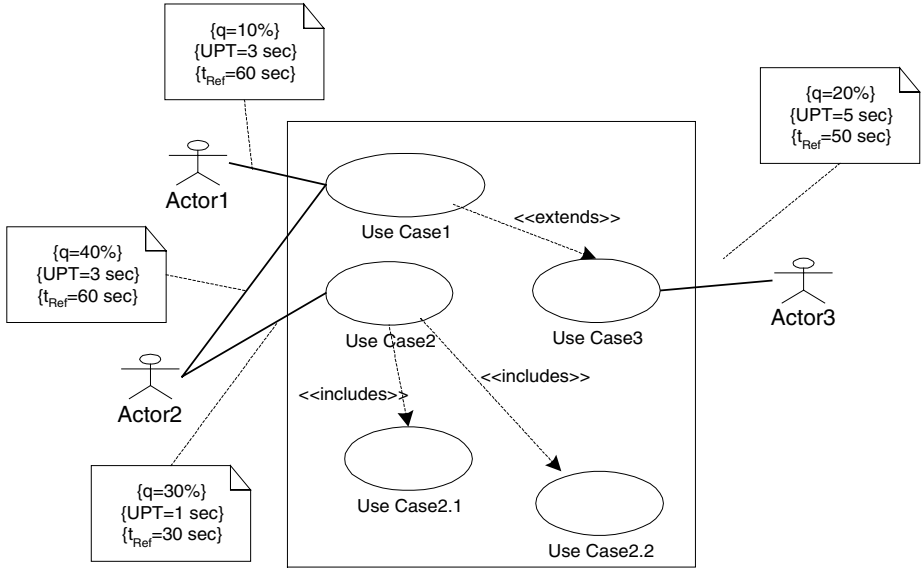



**Fig. 1.** Time weighted Use Case diagram

**Interaction Diagrams.** *Sequence diagrams* as special interaction diagrams offer sufficient potential to be able to obtain and present performance information. They represent time relations through the introduction of an explicit time axis with the time progressing from top to bottom. The vertical layout of messages in the diagram can be used to define the chronological sequence of the messages. Normally, the time axis is scaled on an ordinal basis, i.e. the size of the vertical distances is only of significance for positioning. A rationally scaled time axis is also permitted for the modelling of performance aspects and real-time processing.

Additional time information can be added by labeling the messages with relative constraints and described precisely using the encapsulated numbering of messages. The time attributes can be assigned both to the messages (horizontal lines) and also to

the actual operations (method calls) (Figure 2). In the former case, the time given is interpreted as latency, and in the latter case, it specifies the time for execution of the operation. In addition to the individual time attributes, it is also useful to set comprehensive constraints on a group of operations (Figure 2, right).

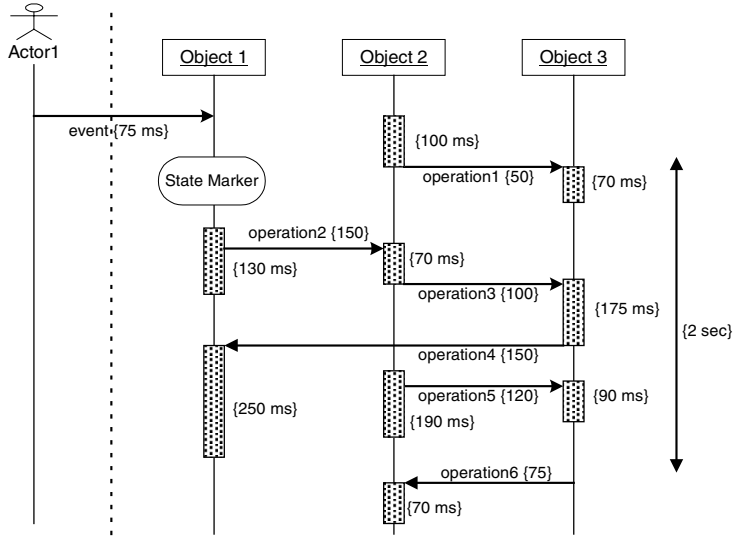



**Fig. 2.** Representation of time information in a sequence diagram

The "state marker" offers the possibility to set up a reference between sequence- and state-diagram. Consequently, the possibility exists to follow a state transition within the sequence diagram.

**State and activity Diagrams.** *State diagrams* show the relationship between events and states of a particular class and thus offer a direct link with Markov modelling. The main problem here is, however, determining transfer rates for the system – an area that does not yet have a sufficiently strong scientific basis.

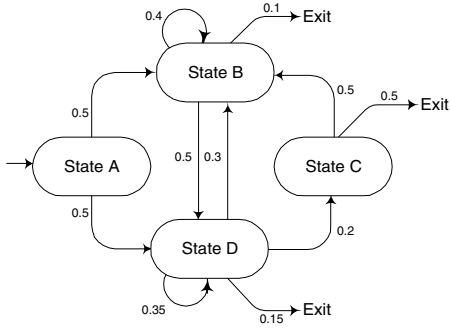



**Fig. 3.** Principe of the Customer Behavior Model Graph

An example for the practical use of state diagrams can be found in [10]. There state diagrams are used to model the workload of e-commerce systems. The application of the so-called Customer Behaviour Model Graph (CBMG) offers the possibility user demands on the system to model as a result of state transitions. A probability is assigned to each transition. For every user type (user with comparable qualities), a corresponding CBMG is to be constructed. In this case, the corresponding thinking times can also be assigned to the state transitions. With the aid of the CBMG it will be possible to derive extensive information about the user behavior. For example the average number of calls of a state within the CBMG or also the average length of a user session.

*Activity diagrams* are defined in the UML metamodel as a specialized form of state diagrams. They are basically used to describe sequences or processes that are generally used relatively early in the development process. They are similar to Petri networks and can be used to express competitive behaviour with explicit synchronization mechanisms.

*Activity diagrams can be used concretely for the following purposes*

- To the description of a use case:
- The carrying out of a use case consists of a sequence from steps. These can be represented 1: 1 in an activity diagram.
- For representation the compatibility of several use cases.
- To the description of the realization of an operation.
- To the description of business processes.

Inserted as activity diagrams at the first two cases may be able be shows Fig. 4.

However, at the moment it is not yet clear how the activity diagrams can be used directly within the scope of performance engineering and what use they will be, in concrete terms, for performance modelling.

*Combination of state and collaboration diagrams.* Combining state and collaboration diagrams forms a practical basis for the simulation of UML models. The combination is created by embedding the state diagram of an object into the collaboration diagram in which the corresponding object is represented. Suitable simulation tools are needed to be able to simulate such diagrams (prototype developments: PROSA UML Modeller [9] or [12] - a class library in Java).

**Implementation Diagrams.** *Implementation diagrams* show the software components implemented, their relationships with each other and their assignment to individual nodes in a hardware topology. They can be used to define and quantify resources. They can easily be used in a queue model. With this option, components are shown using services and relationships between components (links) using queues with a finite operating throughput. The modelling of the actors representing the work load model is carried out using an open network [12].

An other approach for the direct representation of performance related aspects by the use of native UML-elements can be found in [11].

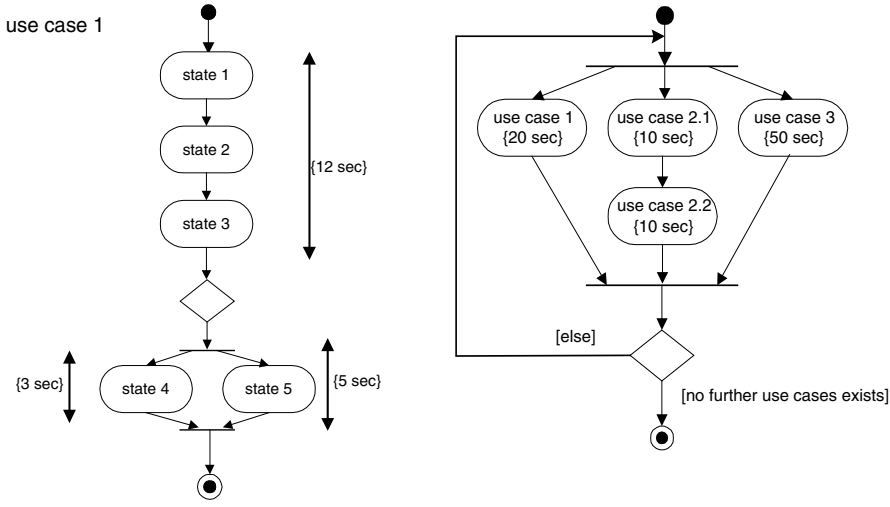



**Fig. 4.** Representation of time requirements within activity diagrams

### 3.2 Expanding UML to Be Able to Handle Performance Aspects

**Real-time UML.** A common feature of all real-time systems is correct time behaviour. The architecture of the planned software is extremely important here. A good architecture can be easily designed using the ROOM method ([14]), which has been in use for ten years. To convert the ROOM constructs to UML, the standard UML tailoring mechanisms, based on *stereotypes*, *tagged values* and *constraints,* can be used [16]. The resultant UML expansion is often described as real-time UML (UML-RT).

The three main constructs for modelling the structure (Fig.5) are:

- *Capsules*: complex physical architectural objects which interact with their environment via one or more signal-based interface objects
- *Ports*: represent the interface objects of a capsule
- *Connectors*: represent abstract views of communication channels.

Complex capsules are represented by an internal network of sub-capsules working together. These sub-capsules are, in turn, capsules in themselves and can be broken down into sub-capsules again.

Capsules and ports are modelled in UML as the stereotype of a corresponding class with the stereotype name <<capsule>> or <<port>>. A connector is modelled by an association and declared in a collaboration diagram of the capsules by the role name being given for the association.

Behaviour is modelled on the basis of:

- protocols for specifying the required behaviour at a connector and
- special state machines that are assigned to capsules.

A *Time Service* is provided to represent time aspects. A service of this type, which can be accessed via a standard port, transforms time into events which can then be treated in exactly the same way as other signal-based events.

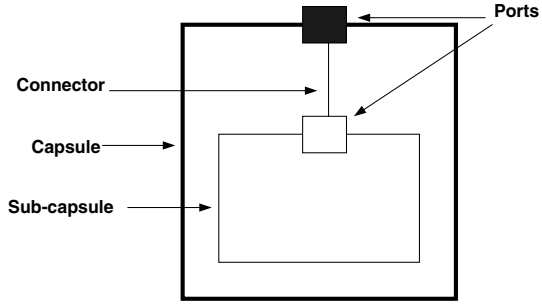



**Fig. 5.** The basic elements of UML-RT, shown as a collaboration diagram

**Expanded UML.** UML is often expanded, through the introduction of additional means of expression and constructs, with a view to achieving appropriate modelling of real-time and embedded systems ([1], [4]). Three new types of diagram are important here.

*Timing Diagram.* This type of diagram, which is widely used in electrotechnical systems, can be used to explicitly model changes in state over time. The horizontal axis shows the time and the vertical axis shows the various states of the system. The time axis is normally linear. These diagrams therefore represent a suitable supplement to the UML state diagrams in terms of the time element. Timing diagrams basically exist in two forms [4]:

- *Simple Timing* diagram – representing a single path within a state diagram;
- *Task Timing* diagram – representing the interaction of several *tasks* over time.

The most important elements of *Task Timing* diagrams, which can also be represented jointly in one diagram by means of different types of shading ([4]) are: period, deadline, time of introduction, execution time, dwell time, flat time, rise and fall time, jitter.

*Concurrency Diagram.* The collaboration diagram considers inter-object communication in a task-unrelated view. However, there are three forms of object-to-object communication in a Multi-Threaded Environment:

- intra-task,
- inter-task, intra-processor, and
- inter-task, inter-processor.

The first form can be realized by means of collaboration diagrams, but these are not suitable for modelling the other two forms. For these, *concurrency* diagrams are required ([1]). These diagrams allow the logical object architecture to be depicted in a

*multi-tasking* solution, i.e. the *task* structure of the solutions can be depicted, as can the modelling of the communication mechanisms between the *tasks*.

*System Architecture Diagram.* This type of diagram is used to give a detailed representation of the physical system structure and, in particular, for the distribution of the software over the nodes. (The UML distribution diagrams are not suitable for this, although they can show some aspects of the physical distribution.)

The most important elements of a diagram of this type are: *Processing Nodes (or Boards), Storage Nodes (Disks), Interface Devices* and *Connections*. Specific implementation characteristics of these elements, such as storage capacity, execution speed, bandwidth and further quantitative elements, can thus be taken into account.

### 3.3 Combining UML with FDTs[3] on the Example of MSC and SDL

The use of formal description languages is represented at the following at the selected example of the combination of UML with MSC/SDL. Basically, a distinction can be made between two approaches to the combination of UML with MSC/SDL:

- Link approach – some sub-models are shown in UML and others in SDL and the relationship between them is created by links.
- *Forward engineering* - translation UML→ MSC / SDL.

The link approach means using both techniques in the same phase of the development process and involves major consistency problems.

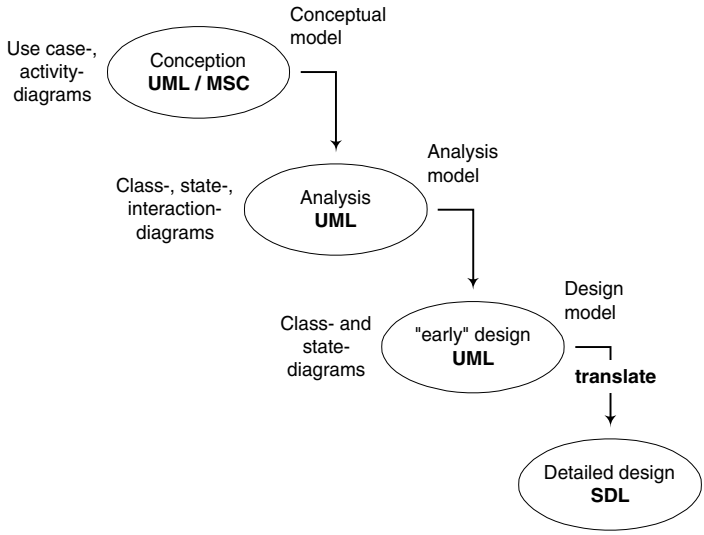



**Fig. 6.** Forward engineering for combining UML and MSC/SDL (according to [18])

[3] FDT – Formal Description Techniques

With the *forward engineering* approach (Fig. 6), UML is used in the conception and analysis phases and to some extent in "early" design. SDL, on the other hand, is stronger in the "later" design and implementation phases, incl. automatic code generation. The MSCs are used to model and simulate the use cases and interaction scenarios.

Predefined transformation rules exist for the translation of UML semantics to SDL/MSC ([18]). However, fully automatic translation is not always a good idea, since the individual classes and associations can be transformed in different ways.

In their standard form, both MSC and SDL, however, are not sufficient to derive complete performance information. A number of expansions are proposed for this, which can basically be divided into two groups:

- Expansion of the SDL/MSC syntax by the addition of further language constructs to describe real-time requirements and performance characteristics: e.g. the language QSDL ([3] or [8]),
- Annotative expansion of SDL and MSC through comment instruction ([5]).

One of the disadvantages of the first approach is the need to realize independent SDL/MSC development environments, since the commercial tools do not allow any language expansion.

The second approach is associated with a series of assumptions (cf. [5], p. 5-6) which present a problem for practical application.

The above-mentioned expansion of the SDL standard (SDL 2000) and the standardization of the UML-SDL connection by OMG will mean that the significance of combining the two techniques is certain to increase in the future.

# 4 Tool Support

## 4.1 Outline Description of the Function and Use of Tools

**Separate support of the software development process using performance tools.** In this case, tools are used more in less in isolation from the system development process. The following phases can be distinguished:

- On the basis the activities carried out within the individual phases of the software development, corresponding performance models are generated (automatically in some cases)
- The performance models are executed using performance tools
- The results achieved are transferred by means of return transformation at the software development level (within the phase in question) and interpreted accordingly.
- On the basis of the information and findings determined, corresponding modifications (changes, additions) are made to the phase results (analysis or design model, implementation code, etc.).
- During the next cycle, performance models are generated again with the changes made being taken into account.

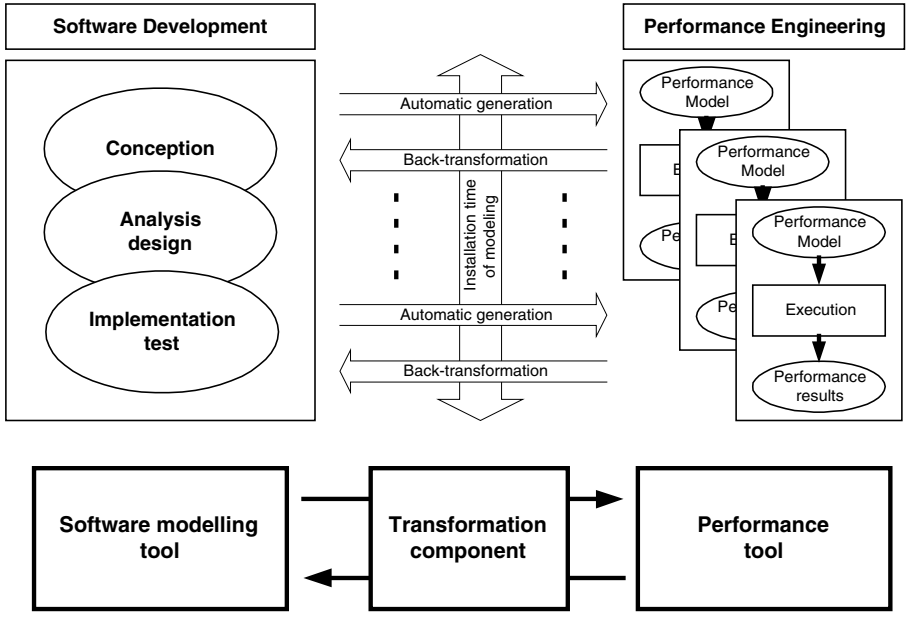



**Fig. 7.** Diagram of the software development process being supported by performance engineering

Advantages:

1. Good support when various solution methods are combined, such as, for example, analytical methods for solving the models within the conception, analysis and design phases, or simulative methods within the software implementation phase.
2. Application of models that already used successfully for the tasks of performance management, especially for performance prediction within the capacity planning. In such a way, it is possible to use extensive experiences.
3. Automatic generation of performance models (through insertion of transformation components)

Disadvantages:

1. Separation between software development ←→ performance engineering
2. Use of different representational /modelling tools:
– software development: UML (standardized!)
– PE: Execution Graphs ([17]), Angie Traces ([6]), etc.
3. The mapping of software model → performance model is not unambiguous, which means that the back-transformation, or back-interpretation of the performance results is more difficult.
4. Iterative, incremental development is not sufficiently supported.

**Expansion of Software Modelling Language (UML) by Constructs to handle Performance**. In this case, the modelling tool contains components that support a direct execution (simulation) of some UML diagrams /models. This means that it is

possible to derive performance information directly from the software model description. In addition, it is recommended that components for carrying out measurements (benchmarks, ARM[4] technology, etc.) are integrated in order to be able to derive performance information during implementation and testing.

Advantages:

1. No separation between software development and performance engineering, only one uniform notation (UML)
2. Performance information can be derived straight from the UML model.
3. Iterative, incremental development is explicitly supported

Disadvantages:

1. UML description is not formalized → the derived performance results can only be used to a limited extent. At the end of a development cycle, verification / validation of the performance results is necessary.
2. Representation of all the necessary performance aspects requires an expansion of the UML standard (new constructs, diagrams, etc.)

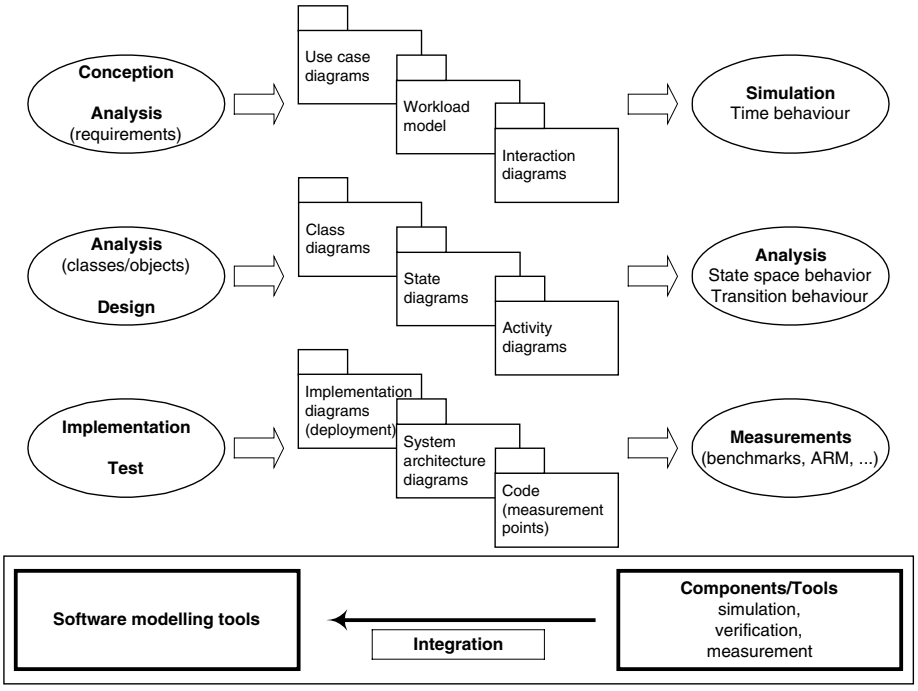



**Fig. 8.** Integration: software modelling tools and components for simulation, verification and measurement

[4] Application Response Measurement API (standard for performance instrumentation)

**Combining Software Modelling Language with other FDTs (MSC/SDL).** With the combined approach, the UML diagrams are used to be able to derive usable MSC/SDL models from them. The informal UML description is supplemented by semantic assumptions. The UML modelling components and the MSC/SDL execution and simulation components are usually integrated by means of links in a development environment.

Advantages:

1. No separation between software development and performance engineering → one development process based on UML and MSC/SDL.
2. Uses solely standardized description tools (UML, MSC, SDL)
3. Very good support of iterative and incremental development. The combination of informal description (UML) with FDTs means that both the first cycles (less concrete model information) and the later ones (more complex information) are supported well.
4. More precise results based on the use of MSC/SDL

Disadvantages:

1. Capability for several notations (UML, MSC, SDL)

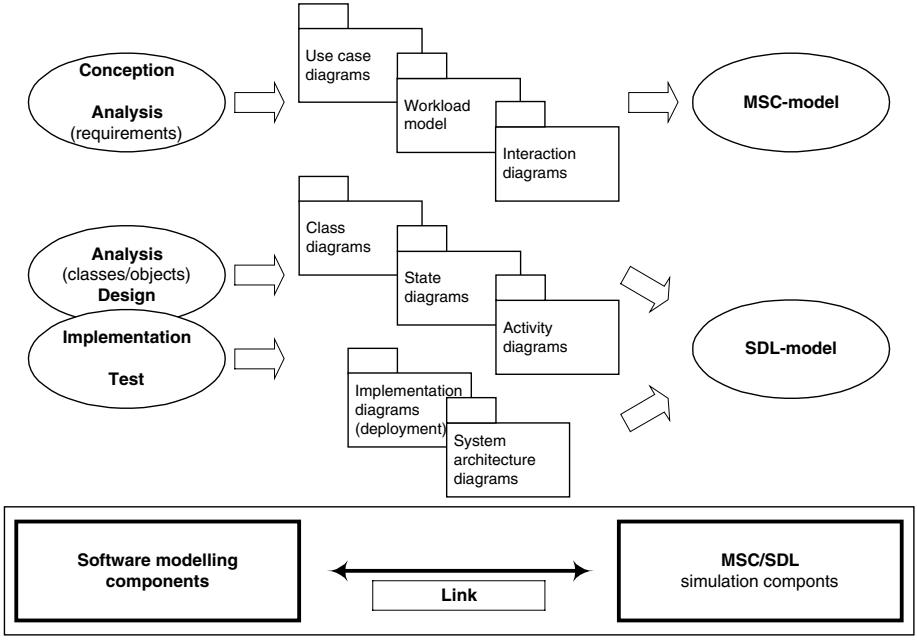



**Fig. 9.** Combining software modelling components and MSC/SDL simulation tools

### 4.2 Evaluation of Existing Tools in Table Form

Table 1 contains only those tools that allow UML-based PE. This means that the tools explicitly support one of the approaches described in section 3. Widely used UML modelling tools, such as Rational/Rose, Together, Paradigm Plus, System Architect, or StP/UML are not listed here because they do not permit any derivation of performance characteristics and information.

**Table 1.** Evaluation of selected UML-based tools (Status: I/2000)

| Tool or contact person / company | Types of diagram supported | Solution methods | model derivation from | Integration procedure model |
|---|---|---|---|---|
| PROSA / Insoft Oy | All UML diagrams | Simulation | State and collaboration diagrams | no |
| SPE.ED / L&S Comp.Techn. | Use Case, MSC and sequence diagrams | Queue model/simulation (CSIM) | Use Case, MSC and sequence diagrams | no |
| Rose-RT or ObjectTime / Rational | all UML diagrams, plus MSC | SimulationDebugging | "Angio-Traces" derived from scenarios | yes |
| Rhapsody/I-Logix | Sequence, class, use case and state diagrams | Animation simulation | Sequence and state diagrams | no |
| ARTiSAN Studio / ARTiSAN Software | all UML + *constraints*, c*oncurrency* and s*ystem architecture* -diagrams & *Timing Table* | Animation, simulation | Sequence and State diagrams | no |
| Tau 4.0 / Telelogic | all UML + ASN.1, SDL, MSC, TTCN | Animation, simulation | Sequence and state diagram SDL, MSC | yes |

| Tool or contact person / company | Types of diagram supported | Solution methods | model derivation from | Integration procedure model |
|---|---|---|---|---|
| ObjectGEODE / Verilog (Telelogic) | all UML + ASN.1, SDL, MSC | Simulation (spec. module: Performance Analyzer) | Expanded SDL, MSC | yes |

The information in the table and the evaluations undertaken are based mainly on the product manufacturers' information.

## 5 An Integrated UML Based Approach

Only the combination of UML-models with FDT's and classical performance models, like queuing-networks, supported by a prototypical benchmarking allows the definition of a efficiently performance modeling approach. A first proposal of such integrated approach will be characterized by following:

- Starting point: Extensions of the UML notation for the representation of performance related aspects. Standardisation through the OMG is required, a first proposal can be found in ([15]).
- In the early development stages: application of waiting queue- or net-based models.
- In the late development stages, if enough information about the implementation available, application of tool-based solution techniques like (BEST/1, SES,..) or the combination with FDT's (SDL/MSC).

For the solution of performance models can be use different methods: analytical, simulative or combination of both.

## 6 Comments and Conclusions

The existing UML based approaches do not satisfactorily meet the requirements defined in section 2. Carrying out PE tasks using the UML constructs that are currently standard, especially in the context of classical information systems, seems insufficient for the practical use of UML for performance modelling and evaluation. For the representation and analysis of performance aspects within the framework of UML, a structural expansion of this notation is absolutely essential.

In addition to this expansion (standardization through OMG), combining UML with formal specification techniques, such as MSC and SDL, is a promising approach, especially with regard to real-time and embedded systems. However, once again, it is essential that the proposed MSC/SDL expansions are standardized.

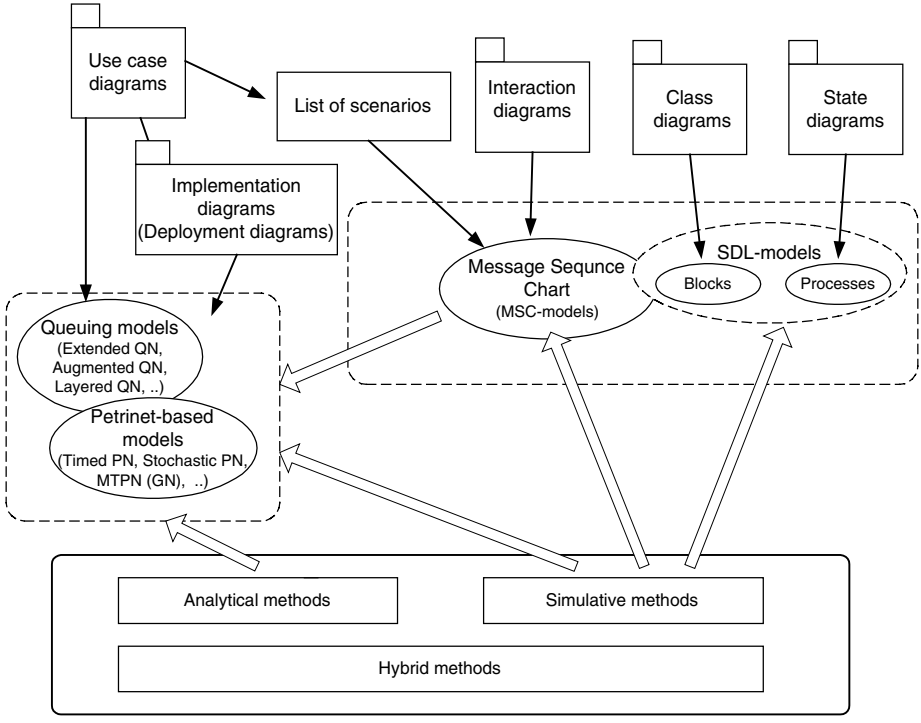



**Fig. 10.** Raw architecture of an UML-based framework

During the treatment of this study, a RFP (UML™ Profile for Scheduling, Performance, and Time) was published by the OMG. A first proposal, which especially supports the development of real time systems, can be found under [15]. The expansion suggestions especially use stereotypes and change the previous UML-Notation only marginally. Decomposition of complex scenarios, analytic examinations of performance qualities, specification of stochastic qualities, workload characterisations and expansions of the deployment diagram for the representation of the hardware architecture remains, however, unheeded. The modelling of all system layers until to the hardware was taken into account so not yet.

Tool support is extremely inadequate at the moment.

- Most UML tools that can be used in practice, such as *Rose, Together, Paradigm Plus* or *System Architect* currently offer no facilities for determining performance characteristics. This means that the tool support for Approach 1 is either limited to tools that come from the academic arena or the transfer from software to performance models must be carried out mainly manually.
- The group of tools that support expansions of UML, especially for real-time and embedded systems (Approach 2 / second section of Table 1), contains a number of professional development environments. The lack of standardization of the expansion concepts makes these more difficult to use. In addition, most of them are very specifically directed at the target area - real-time and embedded systems - and

allow the development of applications for very specific hardware platforms. These are thus less suitable for use in classic information systems.

– The tools (Approach 3/third section of Table 1) supporting a combination of UML and FDTs are the most highly developed in comparison with the other two groups of tools. They have been in existence longer, having appeared on the market as pure SDL/MSC tools at the end of 1980. In recent years, they have been successfully expanded in the direction of UML modelling.

## References


1. Artisan Software: How can I use UML to Design Object-Oriented Real-Time Systems when it has no Support for Real-Time. White Paper, (http://www.artisansw.com) (1999)
2. Booch, G., Jacobson, I.., Rumbaugh, J.: The Unified Modeling Language User Guide. The Addison-Wesley Object Technology Series, Addison-Wesley Pub Co (1998)
3. Diefenbruch, M., Hintelmann, J., Müller-Clostermann, B.: QSDL: Language and tools for performance analysis of SDL systems. In the Conference Volume for the 5th GI/ITG Specialist Discussion on "Formal description techniques for distributed systems", Kaiserslautern, June 22-23 (1995)
4. Douglass, B.P: Real-Time UML – Developing Efficient Objects for Embeded Systems. Addison-Wesley (1999)
5. Dulz, W.: Leistungsbewertung von SDL/MSC-spezifizierten Protokollsystemen. In: Swoboda, J. (ed.): Proc. of Softwaretechnik in Automation und Kommunikation (STAK'96). München, ITG Fachbericht Nr. 137, VDE-Verlag (1996 )
6. Hrischuk, C., Rolia, J., Woodside, C.M.: Automatic generation of a software performance model using an object-oriented prototype. In: International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'95) (1995) 399–409
7. ISO 14756: Information technology – Measurement and rating of performance of computer-based software systems. COMMITTEE DRAFT, Version 5g, January (1997)
8. Leblanc, P.: SDL Performance Analysis. Verilog White Paper, (http:// www.csverilog.com/download/geode.htm) (1998)
9. Lehikoinen, H.: UML Modeling and Simulation Raises Quality and Productivity. Component Computing'99, Helsinki (http://www.insoft.fi) (1999)
10. Menasce, D.A., Almeida, V.A.F., Fonseca, R., Mendes, M.A.: Resource Management Policies for E-Commerce Servers. ACM Sigmetrics, March, 27(2000)4
11. Miguel, M., Lambolais, T., Hannouz, M., Betgé-Brezetz, S., Piekarec, S.: UML Extension for the Specification and Evaluation of Latency Constraints in Architectural Models, In: Proc. of WOSP 2000, Ottawa/Canada (2000)
12. Pooley, R., King, P.: The Unified Modeling Language and Performance Engineering. IEE Proceedings – Software (1999)
13. Scholz, A., Schmietendorf, A.: Performance Engineering – Aufgaben und Inhalte (Performance Engineering - tasks and contents). In: HMD – Praxis der Wirtschaftsinformatik, Heft 213, Juni (2000)
14. Selic, B., Gullekson, G., Ward, P.: Real-Time Object-Oriented Modeling. John Wiley & Sons (1994)
15. Selic, B., Moore, A., Bjorkander, M., Gerhardt, M., Watson, B.: Response to the OMG RFP for Schedulability, Performance, and Time. Document Version 1.0 Monday, August 14 (2000)

16. Selic, B., Rumbaugh, J.: Die Verwendung der UML für die Modellierung komplexer Echtzeitsysteme. ObjektSpektrum, 4/1998 (The original English document can be found at: http://www.objecttime.com ) (1998)
17. Smith, C.: Performance Engineering of Software Systems. Reading, MA et al. (1990)
18. Verschaeve, K., Ek, A.: Three Scenarios for combining UML and SDL 96. In: Dssouli, R., Bochmann, G., Lasav, Y. (eds.): SDL'99: The Next Millennium. Elsevier Science B.V. (1999)
19. Klein, M.H.: State of the Practice Report: Problems in the Practice of Performance Engineering. Technical Report, Pittsburgh, Pennsylvania: Software Engineering Institute (1996)

# Origins of Software Performance Engineering: Highlights and Outstanding Problems

Connie U. Smith

Performance Engineering Services
PO Box 2640
Santa Fe, NM 87504
www.perfeng.com

**Abstract.** This chapter first reviews the origins of Software Performance Engineering (SPE). It provides an overview and an extensive bibliography of the early research. It then covers the fundamental elements of SPE: the data required, the software performance models and the SPE process. It concludes with a review of the current status and outstanding problems in the areas of: tools, performance models, use of SPE, principles, patterns and antipatterns for building high performance software, and SPE methods.

## 1 Introduction

Software performance engineering (SPE) is a systematic, quantitative approach to constructing software systems that meet performance objectives. In this chapter, *performance* refers to the response time or throughput as seen by the users. For real-time, or *reactive* systems, it is the time required to respond to events or the number of events processed in a time interval. Reactive systems must meet performance objectives to be correct. Other software has less stringent requirements, but responsiveness limits the amount of work processed, so it determines a system's effectiveness and the productivity of its users.

SPE provides an engineering approach to performance, avoiding the extremes of performance-driven development and "fix-it-later." SPE uses model predictions to evaluate trade-offs in software functions, hardware size, quality of results, and resource requirements.

SPE is a software-oriented approach: it focuses on architecture, design, and implementation choices. The models assist developers in controlling resource requirements by selecting architecture and design alternatives with acceptable performance characteristics. They aid in tracking performance throughout the development process and prevent problems from surfacing late in the life cycle.

SPE also provides principles, patterns, and antipatterns for creating responsive software, the data required for evaluation, procedures for obtaining performance specifications, and guidelines for the types of evaluation to be conducted at each development stage. It incorporates models for representing and predicting performance as well as a set of analysis methods [88; 100].

SPE techniques provide the following information about the new system:

R. Dumke et al. (Eds.): Performance Engineering, LNCS 2047, pp. 96–118, 2001.

- refinement and clarification of the performance requirements
- predictions of performance with precision matching the software knowledge available in the early development stage and the quality of resource usage estimates available at that time
- estimates of the sensitivity of the predictions to the accuracy of the resource usage estimates and workload intensity
- understanding of the quantitative impact of design alternatives, that is the effect of system changes on performance
- scalability of the architecture and design: the effect of future growth on performance
- identification of critical parts of the design
- identification of assumptions that, if violated, could change the assessment
- assistance for budgeting resource demands for parts of the design
- assistance in designing performance tests

This chapter first covers the evolution of SPE then it gives an overview of the SPE process and techniques. It describes the general principles for performance-oriented design, the performance patterns and antipatterns, and the quantitative techniques for predicting and analyzing performance. Finally, the conclusion reviews the status and future of SPE.

## 2 The Evolution of SPE

Performance was typically considered in the early years of computing. Knuth's early work focused on efficient data structures, algorithms, sorting and searching [49; 50]. The space and time required by programs had to be carefully managed to fit them on small machines. The hardware grew but, rather than eliminating performance problems, it made larger, more complex software feasible and programs grew into systems of programs. Software systems with strict performance requirements, such as flight control systems and other embedded systems used detailed simulation models to assess performance. Consequently creating and solving them was time-consuming, and updating the models to reflect the current state of evolving software systems was problematic. Thus, the labor-intensive modeling and assessment were cost-effective only for systems with strict performance requirements.

Authors proposed performance-oriented development approaches [11; 35; 71; 81] but most developers of non-reactive systems adopted the "fix-it-later" methodology. It advocated concentrating on software correctness, deferring performance considerations to the integration testing phase and (if performance problems were detected then) procuring additional hardware or "tuning" the software to correct them. Fix-it-later was acceptable in the 1970s, but in the 1980s the demand for computer resources increased dramatically. System complexity increased while the proportional number of developers with performance expertise decreased. This, combined with a directive to ignore performance, made the resulting performance disasters a self-fulfilling prophecy. Many of the disasters could not be corrected with hardware – platforms with the required power did not exist. Neither could they be

corrected with tuning – corrections required major design changes, and thus reimplementation. Meanwhile, technical advances led to the SPE alternative.

## 2.1 Modeling Foundations

In 1971, Buzen proposed modeling systems with queuing network models and published efficient algorithms for solving some important models [22]. In 1975, Baskett, et. al., defined a class of models that satisfy separability constraints and thus have efficient analytic solutions [6]. The models are an abstraction of the computer systems they model, so they are easier to create than general purpose simulation models. Because they are solved analytically, they can be used interactively. Since then, many advances have been made in modeling computer systems with queuing networks, faster solution techniques, and accurate approximation techniques [36; 44; 61].

Queueing network models are commonly used in capacity planning to model computer systems. A capacity planning model is constructed from specifications for the computer system configuration and measurements of resource requirements for each of the workloads modeled. The model is solved and the resulting performance metrics (response time, throughput, resource utilization, etc.) are compared to measured performance. The model is calibrated to the computer system. Then, it is used to study the effect of increases in workload and resource demands, and of configuration changes.

Initially, queueing network models were used primarily for capacity planning. For SPE they were sometimes used for feasibility analysis: request arrivals and resource requirements were estimated and the results assessed. More precise models were infeasible because the software could not be measured until it was implemented.

The second SPE modeling breakthrough was the introduction of analytical models for software [8; 15; 16; 18; 76; 89; 103]. With them, software execution is modeled, estimates of resource requirements are made, and performance metrics are calculated. Software execution models yield an approximate value for best, worst, or average resource requirements. They provide an estimate for response time; they can detect response time problems, but because they do not model resource contention they do not yield precise values for predicted response time.

The third SPE modeling breakthrough was combining the analytic software models with the queueing-network system models to more precisely model execution characteristics [13]; [82; 90]. Combined models more precisely model the execution. They also show the effect of new software on existing work and on resource utilization. They identify computer device bottlenecks and the parts of the new software with high use of bottleneck devices.

By 1980, the modeling power was established and modeling tools were available [14]; [43]; [70]. Many new tools are now available. Thus, it became cost-effective to model large software systems early in their development.

## 2.2 SPE Methods

Early experience with a large system confirmed that sufficient data could be collected early in development to predict performance bottlenecks [91]. Unfortunately, despite

the predictions, the system design was not modified to remove them and upon implementation (approximately one year later) performance in those areas was a serious problem, as predicted. The modeling problems were resolved, but that was not enough to prevent problems.

SPE methods were proposed [83] and later updated [88]. Key parts of the SPE process are methods for collecting data early in software development, and critical success factors to ensure SPE success. The methods also address compatibility with software engineering methods, what is done, when, by whom, and other organizational issues.

## 2.3 SPE Development

The 1980s and 90s brought advances in all facets of SPE. Software model advances were proposed by several authors [9; 17; 29; 69; 75; 84; 88; 92]. Martin [59] proposed data-action graphs as a representation that facilitates transformation between performance models and various software design notations. Opdahl and Sølvberg [66] integrated information system models and performance models with extended specifications. Brataas applied SPE modeling techniques to organizational workflow analysis in [19].

Rolia [72] extended the SPE models and methods to address systems of cooperating processes in a distributed and multicomputer environment. Woodside [106; 107] proposed stochastic rendezvous networks to evaluate performance of Ada systems and Woodside and coworkers incorporated the analysis techniques into a software engineering tool [20; 108]. Opdahl [65] described SPE tools interfaced with the PPP CASE tool and the IMSE environment for performance modeling – both were part of the Esprit research initiative. Lor and Berry [57] automatically generated SARA models from an arbitrary requirements specification language using a knowledge-based Design Assistant.

SPE models were extended and applied to Client/Server systems [61; 101], to distributed systems [39; 79; 97], and web applications [26; 38; 60; 78; 99]

Newer tools that incorporate features to support SPE modeling are reported by numerous authors [58; 5; 7; 63; 64; 68; 94; 96].

Extensive advances have been made in computer system performance modeling techniques. A complete list of references is beyond the scope of this chapter (for more information see other chapters in this book).

Bentley [13] proposed a set of rules for writing efficient programs. A set of formal, general principles for performance-oriented design is reported by Smith [85; 86; 88]. These principles were extended to software performance patterns and antipatterns [98; 100]. Software architects who are experts in formulating requirements and designs, and developers who are experts in data structure and algorithm selection, use intuition to develop their systems. The rules, principles, and patterns formalize that expert knowledge. Thus, the expert knowledge developed through years of experience, can be easily transferred to software developers with less experience via the performance patterns and antipatterns.

Numerous authors relay experience with SPE [1; 2; 4; 10; 12; 32; 33; 67; SMIT85b; 87] The *Proceedings of the Computer Measurement Group Conferences* contain SPE experience papers each year (see [25]).

SPE has also been adapted to embedded real-time systems. When these systems must respond to events within a specified time interval, they are called *reactive systems*. Much of the real-time systems work examines resource allocation or schedulability. Other work expresses timing requirements with assertions and proves that requirements are met. Both of those approaches are outside the scope of the software-oriented SPE approach. Many other authors propose methods for performance-oriented design but do not use SPE's quantitative approach. Levi and Agrawala [56] proposed a comprehensive system for creating real-time systems – it encompasses object-oriented descriptions, implementations, techniques, and an operating system for scheduling and execution of the resulting software. Howes [40] prescribed principles for developing efficient reactive systems. Williams and Smith [93] formalized the SPE process for real-time systems. Sholl and Kim [80] adapted the computation-structure approach to real-time systems, and LeMer [55] described a methodology and tool. Chang and coworkers [23] used petri net model extensions to evaluate real-time systems.

The SPE process, models, and tools have also been extended to object-oriented systems [47; 94; 95; 100]. Performance measurements have been used to generate performance models [41; 42; 74]. A workshop has been created to specifically address software and performance issues [111].

After the an highlighting the key elements of SPE in the next three sections, the final section mentions additional related work and discusses outstanding problems.

## 2.4 The SPE Process

The SPE process prescribes what SPE activities should be conducted during software development, and when and how to conduct them. Figure 1 depicts the SPE process; the following paragraphs explain the figure.

First, performance engineers define the specific SPE assessments for the current life cycle phase. Assessments determine whether planned software meets its *performance objectives,* such as acceptable response times, throughput thresholds, or constraints on resource requirements. A specific, quantitative objective is vital if analysts are to determine concretely whether that objective can be met. A crisp definition of the performance objectives lets developers determine the most appropriate means of achieving objectives, and avoid spending time overachieving them.

Business systems specify performance objectives in terms of *responsiveness* as seen by the system users. Reactive systems specify timing requirements for event responses or system throughput requirements. Batch systems specify the batch window and the processing steps that must complete within it. Both the response time for a task and the number of work units processed in a time interval (throughput) are measures of responsiveness. Responsiveness does not necessarily imply efficient computer resource usage. Efficiency is addressed only if critical computer resource constraints must be satisfied.

After defining the objectives, designers create the *concept* for the life cycle product. For early phases the concept is the architecture – the software requirements and the high-level plans for satisfying requirements. In subsequent phases the concept is the design, the algorithms, the code, etc. Developers apply SPEs general

principles, patterns and antipatterns (described later) for creating responsive architectures and designs.

Once the life cycle concept is formulated, analysts gather data sufficient for estimating the performance of the proposed concept. First, analysts need the projected *performance scenarios*: how the system will *typically* be used and the software components that will execute for those scenarios. In addition to the typical-usage analysis, reactive systems also examine worst-case and failure scenarios. Each perfor-

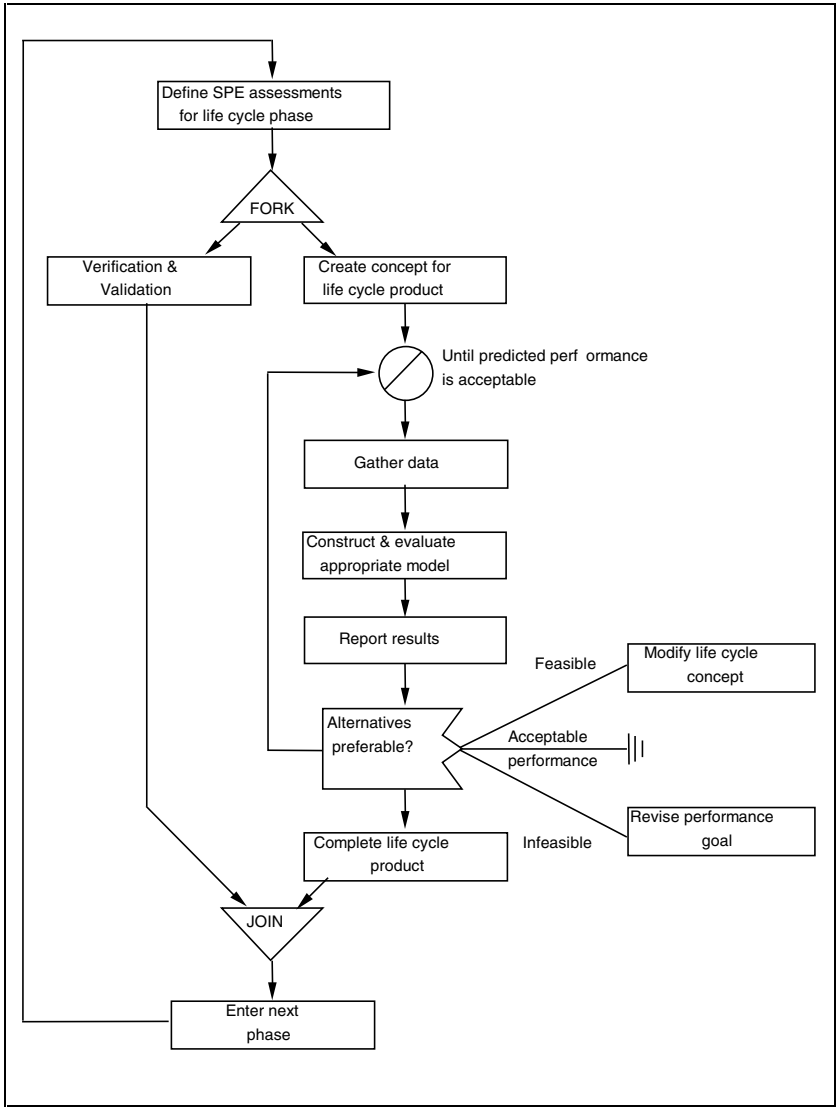



**Fig. 1.** Software Performance Engineering Process

mance scenario specifies the processing steps that execute when the scenario executes. The level of detail depends on the life cycle stage of the evaluation. Estimates of the resource usage of the processing steps complete the specifications. Section 5.1 provides more details on data requirements and techniques for gathering specifications.

Because the precision of the model results depends on the precision of the resource estimates, and because these are difficult to estimate very early in software development SPE calls for a *best and worst-case analysis strategy*. When there is high uncertainty about resource requirements, analysts use estimates of the lower and upper bound. Using them, the analysis produces an estimate of both the best and worst-case performance. If the best-case performance is unsatisfactory, they seek feasible alternatives. If the worst-case performance is satisfactory, they proceed with development. If the results are between the two extremes, models identify critical components whose resource estimates have the greatest effect, and focus turns to obtaining more precise data for them. A variety of techniques provide more precision, such as further refining the design concept and constructing more detailed models, or constructing performance benchmarks and measuring resource requirements for key elements.

An overview of the construction and evaluation of the performance models follows in Section 5.2. If the model results indicate that the performance is likely to be satisfactory, developers proceed. If not, analysts report quantitative results on the predicted performance of the original concept. If alternative strategies would improve performance, reports show the alternatives and their expected (quantitative) improvements. Developers review the findings to determine the cost-effectiveness of the alternatives. If a feasible and cost-effective alternative exists, developers modify the *concept* before the life cycle product is complete. If none is feasible as, for example, when the modifications would cause an unacceptable delivery delay, developers explicitly revise the performance objective to reflect the expected degraded performance.

Vital and on-going activities of the SPE process are to *verify* that the models represent the software execution behavior, and *validate* model predictions against performance measurements. Reports compare the model specifications for the workload, the software components that execute, and resource requirements to actual usage and software characteristics. If necessary, analysts calibrate the model to represent the system behavior. They also examine discrepancies to update the performance predictions, and to identify the reasons for differences – to prevent similar problems occurring in the future. They produce reports comparing system execution model results (response times, throughput, device utilization, etc.) to measurements. Analysts study discrepancies, identify error sources, and calibrate the model as necessary. Model verification and validation should begin early and continue throughout the life cycle. In early stages, focus is on key performance factors; prototypes or benchmarks provide more precise specifications and measurements as needed. As the software evolves, measurements serve to verify and validate the models.

This discussion outlined the steps for one design-evaluation "pass." The steps repeat throughout the life cycle. For each pass the evaluations change somewhat.

## 2.5 Guidelines for Creating Responsive Systems

Program efficiency techniques evolved first. Authors addressed program efficiency from three perspectives: efficient algorithms and data structures [49; 50]; efficient coding techniques [21; 45; 53]; and techniques for tuning existing programs to improve efficiency [13; 30; 31; 48].

Later, the techniques evolved to address large-scale *systems* of programs in early life-cycle stages when developers seek an architecture that will lead to a *system* with acceptable responsiveness [85; 86; 88]. During early stages, it is seldom the efficiency of machine resource usage that matters; it is the system *responsiveness*. Another distinction between system design and program tuning approaches is that program tuning transforms an inefficient program into a new "equivalent" program that performs the same function more efficiently. In system design, developers can transform *what* the software is to do as well as how it is to be done.

Smith [88; 100] defines nine principles: Performance Objective Principle, Instrumenting Principle, Centering Principle, Fixing Point Principle, Locality Principle, Processing Versus Frequency Principle, Shared Resources Principle, Parallel Processing Principle, and Spread-the-Load Principle. A quantitative analysis of the performance results of three of them is in Smith [85].

Both performance patterns and performance antipatterns have been proposed [98; 100]. The performance patterns present best practices for producing responsive software. They are: First Things First, Coupling, Batching, Alternate Routes, Flex-Time, and Slender Cyclic Functions. The performance antipatterns document common performance mistakes and provide solutions for them. They are: "god" Class, Excessive Dynamic Allocation, Circuitous Treasure Hunt, One-Lane Bridge, and Traffic Jam. The performance patterns and antipatterns complement and extend the performance principles. Each performance pattern is a realization of one or more of the performance principles while an antipattern violates one or more of them.

Lampson [52] also presented an excellent collection of hints for computer system design that addresses effectiveness, efficiency, and correctness. His efficiency hints are the type of folklore that has until recently been informally shared. Kopetz [51] presented principles for constructing real-time process control systems; some address responsiveness – all address performance in the more general sense.

Alter [3] and Kant [46] took a different approach; they used program optimization techniques to generate efficient programs from logical specifications. Search techniques identified the best strategy from various alternatives for choices such as data set organizations, access methods, and computation aggregations.

The principles, patterns, and antipatterns *supplement* performance assessment rather than replace it. Performance improvement has many tradeoffs – a local performance improvement may adversely affect overall performance. The quantitative methods covered in section 5 provides the data required to evaluate the net performance effect to be weighed against other aspects of correctness, feasibility, and preferability.

# 3 Quantitative Methods for SPE

The quantitative methods prescribe the data required to conduct the performance assessment; techniques for gathering the data; the performance models; techniques for adapting models to the system evolution; and techniques for verification and validation of the models.

## 3.1 Data Requirements

To create a software execution model analysts need: performance objectives, workload definitions, software execution characteristics, execution environment descriptions, and resource usage estimates. An overview of each follows.

Precise, quantitative metrics, are vital to determine concretely whether or not *performance objectives* can be met. For business applications, both interactive performance objectives and batch window objectives must be met. For interactive transactions, the objectives specify the response time or throughput required. Objective specifications define the external factors that impact objective attainment, such as the time of day, the number of concurrent users, whether the objective is an absolute maximum, a $90^{th}$ percentile, and so on. For reactive systems, timing constraints specify the maximum time between an event and the response. Some reactive systems have throughput objectives for the number of events processed in a time interval. Some have bounds on the utilization of computer resources used, such as a maximum of 50% CPU utilization.

Sometimes the requirements are clear, as in embedded systems where factors in the outer system determine deadlines for certain types of responses. In systems with human users they are rarely clear, because satisfaction is a moving target. As systems get better, users demand more of them, and hope for faster responses.

Experience in obtaining requirements is not well documented, however it seems clear that they are often not obtained in any depth, or checked for realism, consistency and completeness, and that this is the source of many performance troubles with products. One problem is that users expect the system to modify how they work, so they do not really know precisely how they will use it. Another is that while developers have a good notion of what constitutes well-defined functional requirements they are often naive about performance requirements.

Workload definitions specify the *performance scenarios* of the new software. For user interactions, scenarios (initially) specify the interactions expected to be the most frequently used. Later in the life cycle, scenarios also cover resource intensive transactions. Interaction workload definitions identify the performance scenarios and specify their workload intensity: the request arrival rates, or the number of concurrent users and the time between their requests (think time). Batch workload definitions identify the programs on the critical path, their dependencies, and the data volume to be processed. In addition to performance scenarios of typical uses, reactive systems represent scenarios of time-critical functions, and worst-case operating conditions.

Performance scenarios are usually easy to derive because many new systems replace either a manual process or a previous implementation of an automated system. It may be difficult to identify performance scenarios for revolutionary new functions or for Internet applications where the number of potential requests is unlimited and

highly variable. Recent work by Gunther uses statistical techniques for forecasting demand for Internet applications [37].

*Software processing steps* identify components of the software system to be executed for each performance scenario. The software processing steps identify: software components most likely to be invoked when users request the corresponding performance scenario; the number of times they are executed or their probability of execution; and their execution characteristics, such as the database tables used, and screens read or written. Reactive systems initially specify the likely execution paths, as well as less likely execution paths such as the worst-case path.

Software processing steps are relatively easy to identify when developed as the software evolves. It is more difficult when models are initially constructed after the software has been built, particularly for object-oriented systems. Hrishuk and Rolia developed techniques for constructing performance models from measurements of existing object-oriented systems [41; 42].

The *execution environment* defines the computer system configuration, such as the CPU, the I/O subsystem, the network elements, the operating system, the database management system, and middleware components. It provides key computer system and network devices for the underlying queueing network model and defines resource requirements for frequently used service routines.

The execution environment is usually the easiest information to obtain. Performance modeling tools automatically provide much of it, and most capacity and performance analysts are familiar with the requirements.

*Resource usage estimates* determine the amount of service required of key devices in the computer system configuration. For each software processing step in the performance scenario, analysts need: the approximate number of instructions executed (or CPU time required); the number of physical I/Os; and use of other key devices such as the network (number of messages and the amount of data), etc. For database applications, the database management system (DBMS) accounts for most of the resource usage, so the number of database calls and their characteristics are necessary. Early life cycle requirements are tentative, difficult to specify, and likely to change, so SPE uses upper and lower bound estimates to identify problem areas or software components that warrant further study to obtain more precise specifications. Later, the models study the performance sensitivity to various parameter values.

There are four sources of values for resource usage data:

- measurements of parts of the software that are already implemented, such as basic services, existing
- components, a design prototype or an earlier version [105],
- compiler output from existing code,
- demand estimates (CPU, I/O, etc.) based on designer and analyst judgment and reviews,
- "budget" figures, estimated from experience and the performance requirements, may be used as demand *objectives* for designers to meet (rather than as estimates for the code they will produce).

The sources at the top of the list are only useful for parts of the system that are actually running, so early analysis implies at least some of the numbers will be estimated.

It is seldom possible to get precise information for all these specifications early in the software's life cycle. Rather than waiting to model the system until it is available (i.e., in implementation or testing), SPE advocates gathering guesses, approximations, and bounded estimates to begin, then augmenting the models as information becomes available. There has been considerable experience with estimation using expert designer judgment [88], and it works well provided it has the participation (and encouragement) of a performance expert on the panel. Few designers are comfortable in using inexact estimates, and on their own they have a tendency to chase precision, for example by creating prototypes and measuring them. Nonetheless the estimates are useful even if they turn out later to have large errors, since the performance prediction is usually sensitive to only a few of the values.

Because one person seldom knows all the information required for the software performance models, *performance walkthroughs* provide most of the life cycle data [88] [100]. Performance walkthroughs are closely modeled after design and code walkthroughs. In addition to software specialists who contribute software plans they bring together users who contribute workload, use case, and scenario information, and technical specialists who contribute computer configuration and resource requirements of key subsystems such as DBMS and communication paths. The primary purpose of a performance walkthrough is data gathering rather than a critical review of design and implementation strategy.

### 3.2 Performance Models

Two SPE models provide the quantitative data for SPE: the *software execution model* and the *system execution model*. The software execution model represents key facets of software execution behavior. The model solution quantifies the computer resource requirements for each performance scenario. The system execution model represents computer system resources with a network of queues and servers. The model combines the performance scenarios and quantifies overall resource utilization and consequent response times of each scenario.

There are several forms of software models for SPE. The following are the most common:

- The execution graphs of [88; 100] represent the sequence of operations, including precedence, looping, choices, synchronization, and parallel execution of flows. This is representative of a large family of models such as SP [104], task graphs (used in embedded and hard-real-time systems), activity diagrams, etc. Automated tools such as *SPE•ED* [58] and HIT [7] capture the data and reduce it to formal workload parameters.
- Communicating state machines are captured in software design languages such as UML, SDL and ROOM, and in many CASE tools, to capture sequence. Some efforts have been made to capture performance data in this way [28; 109] by capturing scenarios out of the operation of the state machines, or [102] by annotating the state machine and simulating the behaviour directly.
- Petri nets and stochastic process algebras are closely related to state machines and also represent behaviour through global states and transitions. Solutions are by Markov chain analysis in the state space, or by simulation.

- Annotated code represents the behaviour by a code skeleton, with abstract elements to represent decisions and operations, and performance annotations to convey parameters. The code may be executed on a real environment, in which case we may call it a performance prototype, or by a simulator (a performance simulation) [5], or it may be reduced to formal workload parameters [62].

Component-based system assembly models, with components whose internal behaviour is well understood, has been used to create simulation models automatically in SES Strategizer [43], and to create analytic layered queueing models [73; 110].

Execution graph models are one type of software execution model. A graph represents each performance scenario. Nodes represent processing steps of the software; arcs represent control flow. The graphs are hierarchical with the lowest level containing complete information on estimated resource requirements.

A static analysis of the graphs yields mean, best- and worst-case response times of the scenarios. The static analysis makes the optimistic assumption that there are no other jobs on the host configuration competing for resources. Simple, graph-analysis algorithms provide the static analysis results [8; 15; 16; 88]. The static analysis is compared to the performance objective and any problems are resolved before proceeding to the system execution model.

An example of a *SPE•ED* software execution model is in Figure 2 [58]. It shows a simple automated teller machine interaction. The software execution model is hierarchical, the lower levels in the model are shown in the small navigation boxes at the right of the screen. Colors on the screen show the connection between a node in a software model and its details in the navigation box. The figure also shows the end-to-end response time for one ATM user, and colors show the relative time spent in each step. Analysts use results such as these to identify problem areas, then explore alternatives to correct problems by modifying the model and comparing solutions.

Next, the system execution model solution yields the following additional information:

- the effect of resource contention on response times, throughput, device utilizations and device queue lengths.
- sensitivity of performance metrics to variations in workload composition
- effect of new software on service level objectives of other existing systems
- identification of bottleneck resources
- comparative data on performance improvements to the workload demands, software changes, hardware upgrades, and various combinations of each.

To construct and evaluate the system execution model, analysts use performance modeling tools that represent the key computer resources with a network of queues. Environment specifications provide device parameters (such as CPU size and processing speed). Workload parameters and service requests come from the resource requirements computed from the software execution models. Analysts solve the model, check for reasonable results, then examine the model results. If the results show that the system fails to meet performance objectives, analysts identify bottleneck devices and identify software components that have high usage of those devices. After identifying alternatives to the software plans or the computer

configuration, analysts evaluate the alternatives by making appropriate changes to the software or system model and repeating the analysis steps.

Queueing network models are relatively lightweight and give basic analytic models which solve quickly. They have been widely used [62; 95]. However the basic forms of queueing network model are limited to systems that use one resource at a time. Extended queueing models, that describe multiple resource use, are more complex to build and take longer to solve. *Layered queueing* is a framework for extended queueing models that can be built relatively easily, and which incorporates many forms of extended queueing systems [73; 110].

Bottleneck or bounds analysis of queueing networks is even faster than a full solution and has been widely used. However, when there are many classes of users or of workloads (which is the case in most complex systems) the bounds are difficult to calculate and interpret. The difficulty with queueing models is expressiveness; they limit the behavior of the system to tasks providing service to requests in a queue. They cannot express the logic of intertask protocols, for instance.

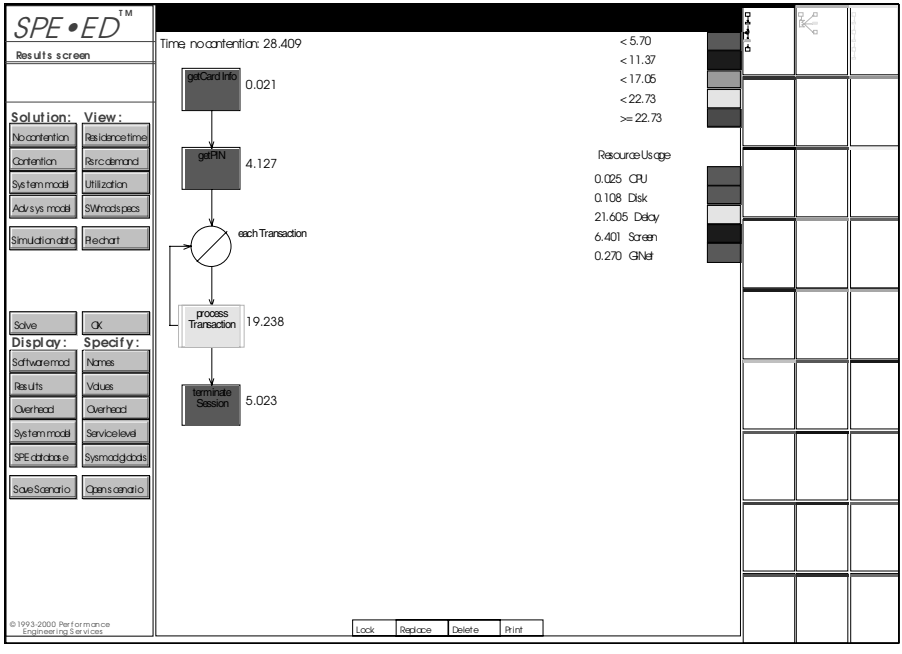



**Fig. 2**. Software Performance Model

Petri nets and other state-based models can capture logically convoluted protocol interactions that cannot be expressed at all in queueing models. They can in principle express any behavior whatever. Further, some of them, such as stochastic process algebra, include mechanisms for combining subsystems together to build systems. Their defect is that they require the storage of probabilities over a state space that may easily have millions of states, growing combinatorially with system size. They do not, and cannot scale up. However the size of system that can be solved has grown from thousands of states to millions, and interesting small designs can be analyzed this way.

Embedded and hard-real-time systems often use a completely different kind of model, a schedulability model, to verify if a schedule can be found that meets the deadlines. These are considered briefly below.

Simulation is the always-available fallback from any formulation. Some kinds of models are designed specifically for simulation, such as the performance prototype from code. Simulation models are heavyweight in the execution time needed to produce accurate results, and can give meaningless results in the hands of non-experts who do not know how to assess their precision or their accuracy. When models can be solved by both extended queueing techniques and by simulation, the simulation time may easily be three orders of magnitude greater. However simulation can always express anything that can be coded, so it never limits the modeler in that way.

System execution models based on the network of queues and servers can be solved with analytic techniques in a few seconds. Thus analysts can conduct many tradeoff studies in a short time. Analytic solution techniques generally yield utilizations within 10 percent, and response times within 30 percent of actual. Thus, they are well suited to early life cycle studies when the primary objective is to identify feasible alternatives and rule out alternatives that are unlikely to meet performance objectives. The resource usage estimates that lead to the model parameters are seldom sufficiently precise to warrant the additional time and effort required to produce more realistic models. Even reactive systems benefit from this intermediate step – it rules out serious problems before proceeding to more realistic models.

Early studies use simple SPE models. The SPE goal is to find problems with the simplest possible model. Experience shows that simple models can detect serious performance problems very early in the development process. The simple models isolate the problems and focus attention on their resolution. They are useful to evaluate many early architecture and design alternatives because they are easily formulated and quickly solved. After they serve this primary purpose, analysts augment them as the software evolves to make more realistic performance predictions. Advanced system execution models are usually appropriate when the software reaches the detail-design life cycle stage. Even when it is easy to incorporate the additional execution characteristics earlier, it is better to defer them to the advanced system execution model. It is seldom easy, however, and the time to construct, solve, and evaluate the advanced models usually does not match the input data precision early in the life cycle.

Facets of execution behavior such as usage of passive resources, complex execution environments, or tightly coupled models of software and system execution are represented in the advanced system execution model. It augments the elementary system execution model with additional types of constructs. Then procedures specify how to calculate corresponding model parameters from software models, and how to solve the advanced models. SPE methods specify “checkpoint evaluations” to identify those aspects of the execution behavior that require closer examination [88]. Simulation methods usually provide solutions for the advanced system models.

Details of these models are beyond the scope of this chapter. Introduction to system and advanced system execution models are available in books [36; 44; 54; 61] as well as other publications. The *Proceedings of the ACM SIGMETRICS Conferences* and the *Performance Evaluation Journal* report recent research results in advanced systems execution models. The *Proceedings of the Performance Petri Net Conferences* also report relevant results.

### 3.3 Verification and Validation

Another vital part of SPE is continual verification of the model specifications and validation of model predictions (V&V). It begins early, particularly when model results suggest that major changes are needed. The V&V effort matches the impact of the results and the importance of performance to the project. When performance is critical, or major software changes are indicated, analysts identify the critical components, implement or prototype them, and measure. Measurements verify resource usage and path execution specifications and validate model results.

Early V&V is important even when predicted performance is good. Performance analysts drive the resource usage specifications, and analysts tend to be optimistic about how functions will be implemented, and about their resource requirements. Resource usage of the actual system often differs significantly from the optimistic specifications.

Performance engineers interview users, designers, and programmers to confirm that usage will be as expected, and that designed and coded algorithms agree with model assumptions. They adjust models when appropriate, revise predictions, and give regular status reports. They also perform sensitivity analyses of model parameters and determine thresholds that yield acceptable performance. Then, as the software evolves and code is produced, they measure the resource usage and path executions and compare them with these thresholds to get early warning of potential problems. As software increments are deployed, measurements of the workload characteristics yield comparisons of specified scenario usage to actual and show inaccuracies or omissions. Analysts calibrate models and evaluate the effect of model changes on earlier results. As the software evolves, measurements replace resource estimates in the SPE models.

V&V is crucial to SPE. It requires the comparison of multiple sets of parameters for heavy and light loads to corresponding measurements. The model precision depends on how closely the model represents the key performance drivers. It takes vigilance to make sure they match.

## 4 Status and Future of SPE

Since computers were invented, the attitude persists that the next hardware generation will offer significant cost-performance enhancements, so it will no longer be necessary to worry about performance. There was a time, in the early 1970s, when computing power exceeded demand in most environments. The cost of achieving performance objectives, with the tools and methods of that era, made SPE uneconomical for many batch systems – its cost exceeded its savings. Today's methods and tools make SPE the appropriate choice for new systems, especially those with high visibility such as web applications.

Will tomorrow's hardware solve all performance problems and make SPE obsolete? It has not happened yet. Hardware advances merely make new software solutions feasible, so software size and sophistication offset hardware improvements. There is nothing wrong with using more powerful hardware to meet performance objectives, but SPE suggests evaluating all options early, and selecting the most

effective one. Thus hardware may be the solution, but it should be explicitly chosen – early enough to enable orderly procurement. SPE still plays a role.

The three primary elements in the evolution of SPE are the *models* for performance prediction, the *tools* that facilitate the studies, and the *methods* for applying them to systems under development. With these the *use* of SPE increases, and new design *concepts* develop that lead to high-performance systems. Future evolution in each of these five areas will change the nature of SPE but not its underlying philosophy to build performance into systems. The following paragraphs speculate on future developments in each of these areas.

### 4.1 Tools

Both research and development will produce the tools of the future. We seek better integration of the models and their analysis with software engineering CASE tools. Ideally, developers should create and evaluate their own models (rather than interfacing with performance specialists to have them built). Then software changes would automatically update prediction models. Simple models can be transparent to designers – designers could click a button while formulating designs and view automatically generated predictions. Expert systems could automatically suggest alternatives. Visual user interfaces could make analysis and reporting more effective. Software measurement tools could automatically capture, reduce, interpret, and report data at a level of detail appropriate for designers. They could automatically generate performance tests then automate the verification and validation process by comparing specifications to measurements, and predictions to actual performance, and reporting discrepancies. Each of these tools could interface with an SPE database to store evolutionary design and model data and support queries against it.

While simple versions of each of these tools are feasible with today's technology, research must establish the framework for fully functional versions. Scenarios in object-oriented CASE tools are close to software execution models, and automatic translation for them is viable, but so far it remains an unsolved problem. A key problem with the translation approach is that CASE tools currently do not collect data, such as resource requirement specifications, that is essential for performance model solutions. Further research is needed to identify viable ways of providing this information, such as with expert systems, using analogous systems, etc. Other research questions are: How should performance models integrate with program generators – should one begin with models and generate code from them, or should one create the program and let underlying models select efficient implementations, or some other combination? How can expert systems detect problems? Can software automatically determine from software execution models where instrumentation probes should be inserted? Can software automatically reduce data to appropriate levels of detail? Can software automatically generate performance tests? Each of these topics represents extensive research projects.

### 4.2 Performance Models

Performance models currently have limits in their ability to analytically solve models of tomorrow's complex environments. Analytic models of extensive parallel or

distributed environments must be handcrafted, with many analytic checkpoint evaluations tailored to the problem. More automatic solutions are desired. Analytic solutions are desirable to shorten solution times and facilitate evaluation of many alternatives. Secondly, the analytic queueing network models yield only mean value results. Analysts need to quickly and easily model transient behavior to study periodic behavior or unusual execution characteristics. For example, averaged over a 10-hour period, locking effects may be insignificant, but there may be short 1 minute intervals in which locks cause all other active jobs to "log jam," and it may take 30 minutes for the log jam to clear. Mean value results do not reflect these after-effects; transient behavior models would. Petri nets and simulation offer the desired capabilities. Another interesting problem is to adapt the modeling techniques to the "heavy-tailed distribution" in usage patterns of workloads and devices in systems [27]. Finally, as computer environments evolve, model technology must also develop. Thus, research opportunities are rich in software and computer environment models.

As with other non-functional requirements, practices for obtaining performance requirements are poorly developed. Research is needed on questions such as: tests for realism, testability, completeness and consistency of performance requirements; on methodology for capturing them, preferably in the context of a standard software notation such as UML; and on the construction of performance tests from the requirements.

Research problems in model building include a flexible expressive model for capture of behavior, automated capture of behavior from legacy components, and modeling within frameworks familiar to the designer, such as UML.

Parameter estimation is an important area, yet it is difficult to plan research that will overcome the difficulties. Systems based on re-using existing components offer an opportunity to use demand values found in advance from performance tests of their major functions, and stored in a demand database. Systems based on patterns offer a similar possibility, but if the pattern is re-implemented it has to be re-measured.

Research into modeling technology, including efficient solution techniques, is lively and broadly based. It would be interesting to be able to combine the speed of queueing techniques with state-based techniques to take advantage of the strengths of both, for analytic estimation.

### 4.3 Use of SPE

Technology transfer trends suggest that the use of SPE is likely to spread. More literature documenting SPE experiences is likely to appear. As it is applied to new, state-of-the-art software systems, new problems will be discovered that require research solutions. Future SPE applications will require skills in multiple domains and offer many new learning opportunities. As quantitative models evolve, so will the use of SPE for new problem domains. For example, models of software reliability have matured enough to be integrated with other SPE methods. Similarly, models can also support hardware-software codesign and enable software versus hardware implementation choices early in development [34]. Schmietendorf et. al. developed a "capability maturity model" to evaluate the extent of use of SPE and the maturity of its use. The results are reported in [77].

There are a number of difficulties with implementing SPE for today's systems, particularly for web applications. Extreme time pressures on the development of

systems have encouraged many software developers to bypass the traditional software development process and its corresponding steps for documenting and analyzing development plans before coding. SPE studies are most valuable when applied to the overall architecture. If the architecture definition is bypassed in the expedited process, it is not possible to conduct an SPE evaluation of the architecture. In web applications it is also difficult to predict the number of requests for particular web pages and there is much higher variability in requests than in traditional systems. It is difficult to get measurements of internal behavior across platforms in a distributed system and to tie them to the business task that is requested.

### 4.4 Concepts for Building High Performance Systems

The concepts for building high performance systems will evolve as SPE use spreads. Experience will lead to new performance patterns, antipatterns, and examples illustrating the difference between high and low- performance software that can be used to educate new software engineers. SPE quantitative techniques should be extended to build in other quality attributes, such as reliability, availability, testability, maintainability, etc. [24]. Research in these areas is challenging – for example: Do existing software metrics accurately represent quality factors? Can one develop predictive models? What design data will drive the models? What design concepts lead to improved quality?

### 4.5 SPE Methods

SPE methods should undergo significant change as its usage increases. The methods should be better integrated into the software development process, rather than an add-on activity. SPE should become better integrated into capacity planning as well. As they become integrated, many of the pragmatic techniques should be unnecessary (how to convince designers there is a serious problem, how to get data, etc.). The nature of SPE should then change. Performance walkthroughs will not be necessary for data gathering; they may only review performance during the course of regular design walkthroughs. The emphasis will change from finding and correcting design flaws to verification and validation that the system performs as expected. The process in Figure 1 currently specifies steps to be conducted by performance specialists. The methods should evolve to empower developers to conduct their own SPE studies. Additional research into automatic techniques for measuring software designs is needed, for calibrating models, and for reporting discrepancies.

Thus, the research challenges for the future are to extend the quantitative methods to model the new hardware-software developments, to extend hardware-software measurement technology to support SPE, and to develop interdisciplinary techniques to address the more general definition of performance. The challenges for future technology transfer are to integrate SPE with software engineering process and tools, to shorten the time for SPE studies, to automate the sometimes-cumbersome SPE activities, and to evolve SPE to make it easy and economical for future environments.

## References


1. Alexander, C.T.: Performance Engineering: Various Techniques and Tools. Proceedings Computer Measurement Group Conference, Las Vegas, NV (1986) 264-267
2. Alexander, W., Brice, R.: Performance Modeling in the Design Process. Proceedings National Computer Conference, Houston, TX (1982)
3. Alter, S.: A Model for Automating File and Program Design in Business Application Systems. Communications of the ACM, 22(1979)6, 345-353
4. Anderson, G.E.: The Coordinated Use of Five Performance Evaluation Methodologies. Communications of the ACM, 27(1984)2, 119-125
5. Bagrodia, R.L., Shen, C.: MIDAS: Integrated Design and Simulation of Distributed Systems. IEEE Transactions on Software Engineering, 17(1991)10, 1042-58
6. Baskett, F. et al.: Open, Closed, and Mixed Networks of Queues with Different Classes of Customers. Journal of the ACM, 22(1975)2 248-260
7. Beilner, H., Mäter,J., Wysocki, C.: The Hierarchical Evaluation Tool HIT. In: Bause, F., Beilner, H. (eds.): Performance Tools & Model Interchange Formats. vol. 581/1995, D-44221 Dortmund, Germany, Universität Dortmund, Fachbereich Informatik (1995) 6-9
8. Beizer, B.: Micro-Analysis of Computer System Performance, New York, NY, Van Nostrand Reinhold (1978)
9. Beizer, B.: Software Performance. In: Vicksa, C.R., Ramamoorthy, C.V. (eds.): Handbook of Software Engineering. New York, NY, Van Nostrand Reinhold (1984) 413-436
10. Bell, T.E. (Editor): Special Issue on Software Performance Engineering, Computer Measureent Group Transactions (1988)
11. Bell, T.E., Bixler, D.X., Dyer, M.E.: An Extendible Approach to Computer-aided Software Requirements Engineering. IEEE Transactions on Software Engineering, 3(1977)1, 49-59
12. Bell, T.E., Falk, A.M.: Performance Engineering: Some Lessons From the Trenches. Proceedings Computer Measurement Group Conference, Orlando, FL (1987) 549-552
13. Bentley, J.L.: Writing Efficient Programs, Englewood Cliffs, NJ, Prentice-Hall (1982)
14. BMC: BMC Software, 2101 City West Blvd., Houston, TX 77042, (713) 918-8800, www.bmc.com
15. Booth, T.L.: Performance Optimization of Software Systems Processing Information Sequences Modeled by Probabilistic Languages. IEEE Transactions on Software Engineering, 5(1979)1, 31-44
16. Booth, T.L.: Use of Computation Structure Models to Measure Computation Performance. Proceedings Conference on Simulation, Measurement, and Modeling of Computer Systems, Boulder, CO (1979)
17. Booth, T.L., Hart, R.O., Qin, B.: High Performance Software Design. Proceedings Hawaii International Conference on System Sciences, Honolulu, HI (1986) 41-52
18. Booth, T.L., Wiecek, C.A.: Performance Abstract Data Types as a Tool in Software Perfromance Analysis and Design. IEEE Transactions on Software Engineering, 6(1980)2, 138-151
19. Brataas, G.: Performance Engineering Method for Workflow Systems: An Integrated View of Human and Computerised Work Processes. Norwegian University of Science and Technology (1996)
20. Buhr, R.J. et al.: Software CAD: A Revolutionary Approach. IEEE Transactions on Software Engineering, 15(1989)3, 234-249
21. Bulka, D., Mayhew, D.: Efficient C++: Performance Programming Techniques. Addison Wesley Longman (2000)
22. Buzen, J.P.: Queueing Network Models of Multiprogramming. Ph.D. Dissertation, Harvard University (1971)
23. Chang, C.K. et al.: Modeling a Real-Time Multitasking System in a Timed PQ Net. IEEE Transactions on Software Engineering, 6(1989)2, 46-51

24. Chung, L. et al.: Non-Functional Requirements in Software Engineering. Boston, MA, Kluwer Academic Publishers (2000)
25. CMG: Computer Measurement Group, PO Box 1124, Turnersville, NJ 08012, (800) 436-7264, www.cmg.org
26. Crain, P., Hanson, C.: Web Application Tuning. CMG, Reno (1999)
27. Crovella, M.: Performance Evaluation with Heavy Tailed Distributions. In: Haverkort, B., Bohnenkamp, H., Smith, C.U. (eds.): Computer Performance Evaluation Modelling Techniques and Tools. Vol. 1786, Berlin, Springer (2000) 1-9
28. El-Sayed, H., Cameron, D., Woodside, C.M.: Automated Performance Modeling from Scenarios and SDL Designs of Telecom Systems. Proc. Int. Symposium on Software Engineering for Parallel and Distributed Systems (PDSE98), Kyoto, Japan (1998)
29. Estrin, G. et al.: SARA (System ARchitects' Apprentice): Modeling, Analysis, and Simulation Support for Design of Concurrent Systems. IEEE Transactions on Software Engineering, SE-12(1986)2, 293-311
30. Ferrari, D.: Computer Systems Performance Evaluation. Englewood Cliffs, NJ, Prentice-Hall (1978)
31. Ferrari, D., Serazzi, G., Zeigner, A.: Measurement and Tuning of Computer Systems. Englewood Cliffs, NJ, Prentice-Hall (1983)
32. Fox, G.: Take Practical Performance Engineering Steps Early. Proceedings Computer Measurement Group Conference, Orlando, FL (1987) 992-993
33. Fox, G.: Performance Engineering as a Part of the Development Lifecycle for Large-Scale Software Systems. Proceedings 11th International Conference on Software Engineering, Pittsburgh, PA (1989) 85-94
34. Frank, G.A., Smith, C.U., Cuadrado, J.L.: Software/Hardware Codesign with an Architecture Design and Assessment System. Proceedings Design Automation Conference, Las Vegas, NV (1985)
35. Graham, R.M., Clancy, G,.J., DeVaney, D.B.: A Software Design and Evalation System. Communications of the ACM, 16(1973)2, 110-116
36. Gunther, N.: The Practical Performance Analyst: Performance-By-Design Techniques for Distributed Systems. McGraw-Hill (1998)
37. Gunther, N.: E-Ticket Capacity Planning: Riding the E-Commerce Growth Curve. Proc. Computer Measurement Group, Orlando (2000)
38. Hanson, C., Crain, P., Wigginton, S.: User and Computer Performance Optimization. CMG, Reno (1999)
39. Hills, G., Rolia, J.A., Serazzi, G.: Performance Engineering of Distributed Software Process Architectures. Modelling Techniques and Tools for Computer Performance Evaluation, Heidelberg, Germany, Vol. 977, Springer (1995) 357-371
40. Howes, N.R.: Toward a Real-Time Ada Design Methodology. Proceedings Tri-Ada 90, Baltimore, MD (1990)
41. Hrischuk, C., Rolia, J.A., Woodside, C.M.: Automatic Generation of a Software Performance Model Using an Object-Oriented Prototype. Proceedings of the Third International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, Durham, NC (1995) 399-409
42. Hrischuk, C.E. et al.: Trace-Based Load Characterization for Generating Performance Software Models. TSE, 25(1999)1, 122-135
43. HyPerformix: Inc., 4301 West Bank Dr, Bldg A, Austin, TX 78746, (512)328-5544, www.hyperformix.com
44. Jain, R.: Art of Computer Systems Performance Analysis. New York, NY, John Wiley (1990)
45. Jalics, P.J.: Improving Performance The Easy Way. Datamation, 23(1977)4, 135-148
46. Kant, E.: Efficiency in Program Synthesis. Ann Arbor, MI, UMI Research Press (1981)
47. King,P., Pooley, R.: Derivation of Petri Net Performance Models from UML Specifications of Communications Software. In: Haverkort, B., Bohnenkamp, H., Smith, C. (eds).: Modelling Tools and Techniques. Schaumburg, IL, Vol. 1786, Springer (2000)

48. Knuth, D.E.: An Empirical Study of FORTRAN Programs. Software Practice & Experience, 1(1971)2, 105-133
49. Knuth, D.E: The Art of Computer Programming. Vol.1: Fundamental Algorithms, Third Edition, Reading, MA, Addison-Wesley (1997)
50. Knuth, D.E.: The Art of Computer Programming Vol.3: Sorting and Searching, Second Edition, Reading, MA, Addison-Wesley (1998)
51. Kopetz, H.: Design Principles of Fault Tolerant Real-Time Systems. Proceedings Hawaii International Conference on System Sciences, Honolulu, HI (1986) 53-62
52. Lampson, B.W.: Hints for Computer System Design. IEEE Software, 2(1984)1, 11-28
53. Larman, C., Guthrie, R.: Java 2 Performance and Idiom Guide. Upper Saddle River, NJ, Prentice Hall PTR (2000)
54. Lazowska, E.D. et al.: Quantitative System Performance: Computer System Analysis Using Queuing Network Models. Englewod Cliffs, NJ, Prentice-Hall, Inc. (1984)
55. LeMer, E.: MEDOC: A Methodology for Designing and Evaluating Large-Scale Real-Time Systems. Proceedings National Computer Conference, 1982, Houston, TX (1982) 263-272
56. Levi, S., Agrawala, A.K.: Real-Time System Design, New York, NY, McGraw-Hill (1990)
57. Lor, K., Berry, D.B.: Automatic Synthesis of SARA Design Models from System Requirements. IEEE Transactions on Software Engineering, 17(1991)12, 1229-1240
58. L&S Computer Technology Inc.: Performance Engineering Services Division. Austin, TX 78766, (505) 988-3811, www.perfeng.com
59. Martin, C.R.: An Integrated Software Performance Engineering Environment. Masters Thesis, Duke University (1988)
60. Menascé, D.A., Almeida, V.A.F.: Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning. Prentice Hall (2000)
61. Menascé, D.A., Almeida, V.A.F., Dowdy, L.W.: Capacity Planning and Performance Modeling. Englewood Cliffs, NJ, PTR Prentice Hall (1994)
62. Menascé, D.A., Gomaa, H.: On a Language Based Method for Software Performance Engineering of Client/Server Systems. Workshop on Software and Performance, Santa Fe, NM, ACM (1998) 63-69
63. Nichols, K.M.: Performance Tools. IEEE Software, 7(1990)3, 21-30
64. Nichols, K.M., Oman, P.: Special Issue in High Performance. IEEE Software, 8(1991)5
65. Opdahl, A.: A CASE Tool for Performance Engineering During Software Design. Proceedings Fifth Nordic Workshop on Programming Environmental Research, Tampere, Finland (1992)
66. Opdahl,A., Sølvberg, A.: Conceptual Integration of Information System and Performance Modeling. Proceedings Working Conference on Information System Concepts: Improving the Understanding (1992)
67. Paterok, M., Heller, R., deMeer, H.: Performance Evaluation of an SDL Run Time System - A Case Study. Proceedings 5th International Conference on Modeling Techniques and Tools for Computer Performance Evaluation, Torino, Italy (1991) 86-101
68. Pooley, R.: The Integrated Modeling Support Environment. Proceedings 5th International Conference on Modeling Techniques and Tools for Computer Performance Evaluation, Torino, Italy (1991) 86-101
69. Qin, B.: A Model to Predict the Average Response Time of User Programs. Performance Evaluation, Vol. 10 (1989) 93-101
70. QSP: Quantitative System Performance. 7516 34th Ave N., Seattle, WA 98117-4723
71. Riddle, W.E. et al.: Behavior Modeling During Software Design. IEEE Transactions on Software Engineering, Vol. 4 (1978)
72. Rolia, J.A.: Predicting the Performance of Software Systems. University of Toronto (1992)
73. Rolia, J.A., Sevcik, K.C.: The Method of Layers. IEEE Trans. on Software Engineering, 21(1995)8 689-700

74. Rolia, J.A., Vetland, V.: Correlating Resource Demand Information with ARM Data for Application Services. Int. Workshop on Software and Performance, Santa Fe, NM, ACM (1998)
75. Sahner, R.A., Trivedi, K.S.: Performance and Reliability Analysis Using Directed Acyclic Graphs. IEEE Transactions on Software Engineering, 13(1987)10, 1105-1114
76. Sanguinetti, J.W.: A Formal Technique for Analyzing the Performance of Complex Systems. Proceedings Performance Evaluation Users Group 14, Boston, MA (1978)
77. Schmietendorf, A., Scholz, A., Rautenstrauch, C.: Evaluating the Performance Engineering Process. Proc. Workshop on Software and Performance (WOSP2000), Ottawa, Canada (2000)
78. Sevcik, K.C., Smith, C.U., Williams, L.G.: Performance Prediction Techniques Applied to Electronic Commerce Systems. In: Kou, W., Yesha, Y. (eds.): Electronic Commerce Technology Trends: Challenges and Opportunities. IBM Press (2000)
79. Sheikh, F., Woodside, C.M.: Layered Analytic Performance Modelling of a Distributed Database System. ICDCS (1997)
80. Sholl, H., Kim, S.: An Approach to Performance Modeling as an Aid in Structuring Real-time, Distributed System Software. Proceedings Hawaii International Conference on System Sciences, Honolulu, HI (1986) 5-16
81. Sholl, H.A., Booth, T.L.: Software Performance Modeling Using Computation Structures. IEEE Transactions on Software Engineering, 1(1975)4
82. Smith, C.U.: The Prediction and Evaluation of the Performance of Software from Extended Design Specifications. Ph.D. Dissertation, University of Texas (1980)
83. Smith, C.U.: Software Performance Engineering: Proceedings Computer Measurement Group Conference XII (1981) 5-14
84. Smith, C.U.: A Methodology for Predicting the Memory Management Overhead of New Software Systems. Proceedings Hawaii International Conference on System Sciences, Honolulu, HI (1982) 200-209
85. Smith, C.U.: Independent General Principles for Constructing Responsive Software Systems. ACM Transactions on Computer Systems, 4(1986)1, 1-31
86. Smith, C.U.: Applying Synthesis Principles to Create Responsive Software Systems. IEEE Transactions on Software Engineering, 14(1988)10, 1394-1408
87. Smith, C.U.: Who Uses SPE? Computer Measurement Group Transactions (1988) 69-75
88. Smith, C.U.: Performance Engineering of Software Systems. Reading, MA, Addison-Wesley (1990)
89. Smith, C.U., Browne, J.C.: Performance Specifications and Analysis of Software Designs. Proceedings ACM Sigmetrics Conference on Simulation Measurement and Modeling of Computer Systems, Boulder, CO (1979)
90. Smith, C.U., Browne, J.C.: Aspects of Software Design Analysis: Concurrency and Blocking. Proceedings ACM Sigmetrics Conference on Simulation Measurement and Modeling of Computer Systems (1980)
91. Smith, C.U., Browne, J.C.: Performance Engineering of Software Systems: A Case Study. Proceedings National Computer Conference, Houston, TX, Vol. 15, (1982) 217-224
92. Smith, C.U., Loendorf, D.D.: Performance Analysis of Software for an MIMD Computer. Proceedings ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems, Seattle, WA (1982) 151-162
93. Smith, C.U., Williams, L.G.: Software Performance Engineering: A Case Study with Design Comparisons. IEEE Transactions on Software Engineering, 19(1993)7, 720-741
94. Smith, C.U., Williams, L.G.: Performance Engineering of Object-Oriented Systems with SPEED. In: Mari, R. et al. (eds.): Lecture Notes in Computer Science 1245: Computer Performance Evaluation. Berlin, Germany, Springer (1997) 135-154
95. Smith, C.U., Williams, L.G.: Performance Engineering Evaluation of CORBA-based Distributed Systems with SPEED. In: Puigjaner, R. (ed.): Lecture Notes in Computer Science. Berlin, Germany, Springer (1998)

96. Smith, C.U., Williams, L.G.: Performance Engineering Models of CORBA-based Distributed Object Systems. Proc. Computer Measurement Group, Anaheim (1998)
97. Smith, C.U., Williams, L.G.: Performance Models of Distributed System Architectures. Proc. Computer Measurement Group, Anaheim (1998)
98. Smith, C.U., Williams, L.G.: Software Performance Antipatterns. Workshop on Software and Performance, Ottawa, Canada, ACM (2000)
99. Smith, C.U., Williams, L.G.: SPE Models for Web Applications. Proc. Computer Measurement Group, Orlando (2000)
100. Smith, C.U., Williams, L.G.: Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software. Addison-Wesley (2001)
101. Smith, C.U., Wong, B.: SPE Evaluation of a Client/Server Application. Proc. Computer Measurement Group, Orlando, FL (1994)
102. Steppler, M: Performance Analysis of Communication Systems Formally Specified in SDL. Proc. 1st Int. Workshop on Software and Performance (WOSP98), Santa Fe, NM (1998)
103. Trivedi, K.S.: Probability and Statistics With Reliability, Queueing, and Computer Science Applications. Englewood Cliffs, NJ, Prentice-Hall (1982)
104. Vetland, V., Hughes, P., Solvberg, A.: A Composite Modelling Approach to Software Performance Measurement. Proc. ACM Sigmetrics, Santa Clara, CA (1993) 275-276
105. Vetland, V.: Measurement-Based Composite Computational Work Modelling of Software. University of Trondheim (1993)
106. Woodside, C.M.: Throughput Calculation for Basic Stochastic Rendezvous Networks. Technical Report, Carleton University, Ottawa, Canada, April (1986)
107. Woodside, C.M.: Throughput Calculation for Basic Stochastic Rendezvous Networks. Performance Evaluation, Vol. 9 (1989)
108. Woodside, C.M. et al.: The CAEDE Performance Analysis Tool," Ada Letters, XI(1991)3
109. Woodside, C.M. et al.: A Wideband Approach to Integrating Performance Prediction into a Software Design Environment. Proc. First Int. Workshop on Software and Performance (WOSP98), Santa Fe, NM (1998)
110. Woodside, C.M. et al.: The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software. IEEE Trans. Computers, 44(1995)1, 20-34
111. WOSP: Workshop on Software and Performance (WOSP98, WOSP2000). ACM Sigmetrics

# Performance Parameters and Context of Use

Chris Stary

University of Linz, Department of Business Information Systems
Communications Engineering, Freistädterstraße 315, A-4040 Linz
stary@ce.uni-linz.ac.at

**Abstract.** The acceptance of software is determined in how far users can be satisfied in the context of using the software. In this chapter context of use is defined in terms of usability-engineering parameters. A first step towards conceptual integration of software-performance engineering with usability engineering is described. Performance engineering in terms of usability evaluation is then enabled through exploiting specification diagrams, thus, providing early feedback to the involved developers.

## 1 Introduction

This chapter addresses fundamental and methodological issues. From the fundamental perspective it brings into play the context of software use. Since most of today's software applications are interactive software systems, usability-engineering principles tend to play a major role within the software development. However, the mapping of those principles to software-performance metrics still needs to be performed in a structured and consistent way, even for technically-oriented features [11]. As a consequence, we have to discuss how usability-engineering metrics can be mapped to operational performance-engineering metrics, or at least might be related to them. ***Fundamental*** relationships between performance engineering and usability engineering will be revealed in terms of principles and metrics. It turns out that performance-engineering metrics can be considered as enablers for usability-engineering metrics with respect to technology[1]. For instance, time constraints related to task accomplishment that have to be considered when designing task-conform systems (referring to the usability-engineering principle task conformance) highly correlate to response time and throughput of the software system (performance-engineering metrics).

The discussion of practices and techniques within the software development does not only require conceptual considerations, but also ***methodological*** ones, namely, how to put usability-engineering principles into practice through modelling systems and measuring performance of these systems. With that respect we start out with *task conformance* and *flexibility* (in the sense of adapting software systems to changed

[1] We consider interactive systems to be socio-technical systems, with human users, computer systems, and the organisation of work as equally important components for design and evaluation.

R. Dumke et al. (Eds.): Performance Engineering, LNCS 2047, pp. 119–130, 2001.

requirements)[2] of software systems, since both seem to become crucial for advanced system development. For instance, a system that does not provide the required functionality in terms of the most effective and efficient interaction features and data-manipulation procedures (referring to task conformance) is not accepted by users at their work place. Hence, the developers need to find the shortest navigation paths to data manipulation (in accordance to work task procedures) which is a performance goal. Design representations have to be validated against those goals. In the context of changing work requirements and with the advent of publicly accessible information systems, such as information kiosks, the flexibility of user interfaces and functionality, comprising also the relationships between content, presentation and navigation, heavily relies on proper system capabilities. It addresses issues like dynamic switching between modalities, e.g., 'Is it possible to switch from one modality to another accurately in a particular situational context?'

The integration of performance and interactive software engineering (fundamental issue) will be handled at the conceptual layer, addressing elementary clusters of both fields' principles and measures. The methodological issue will be handled at an analytic performance modelling layer, enabling the quantitative validation of usability principles. It turns out that model-driven approaches to software performance engineering based on unifying diagrammatic notations are promising candidates, in order to implement usability principles in terms of performance metrics. Notations of the mentioned type can be exploited to implement system performance based on usability. Task conformance and flexibility require particular attention, since the accuracy of user support evolving from usability parameters has to be oriented towards the conceptual architecture of software systems to become operational. Consequently, usability engineering advocates building performance into the software specifications (as opposed to tuning user interface code).

## 2 Linking Usability-Engineering Principles to Software-Performance Engineering Parameters

Initially, software-performance engineering has been a method for constructing software systems to meet responsiveness goals. The objective for software developers was to model software requirements and designs in order to be able to evaluate whether predicted performance metrics meet specified goals. If not, alternatives were processed and assessed. This procedure has been intended to continue through design and implementation, in order to develop more precise models of the software and its predicted performance.

Responsiveness referred to response time and throughput as perceived by users. It has been recognised very early [13] that both parameters are important human interface factors in all systems. With the advent of graphical user interfaces both influence the amount of tasks that can be processed and the acceptance of software features by users.

Starting out with capacity management software-performance engineering has developed to a change management program. Accurate change management focuses

[2] As can be seen from below, this selection has been made intentionally due to the history of software-performance engineering and the origin of usability-engineering techniques.

not only on resource requirement handling but also on ensuring sufficient flexibility for adaptation. As such, user acceptance comes into play again: Through change requests it might be necessary that processes supported by the system and becoming available interactively at the user interface might have to be re-arranged.

Software usability engineering is in a similar situation software-performance engineering has been in the eighties[3]. At that time software architects who were experts in formulating requirements and designs for high-performance systems used intuition to develop their systems. Today, usability engineers who are experts in specifying requirements and designs for interactive systems use intuition to develop interactive software. One striking example is the development of web applications: In web engineering fundamental interaction tasks, such as information seeking, are not commonly agreed upon [4]. However, some frameworks can be used to bridge the gap between performance engineering parameters, as clustered by Oshana [10], and commonly accepted quality parameters of usable software. The latter comprise essential usability-engineering principles, as addressed in standards, such as ISO DIS 13407 [8]. The ISO/IEC 9216 [7] quality model as given in ([2], p. 171) provides a level of detail that allows a closer look on relating usability issues to software-performance parameters. In table 1, a first step towards intertwining both engineering approaches is documented. 'X' denotes direct, '(X)' indirect relationships of cluster elements, '-' denotes no immediate association, as far as can be understood from the analysed literature.

**Table 1.** Performance engineering clusters put in relation to software quality clusters

| Performance Engineering<br><br>Software Quality | Workload | Quantitative Performance Objectives | Software Characteristics | Execution Environ-ment | Resource Requirements |
|---|---|---|---|---|---|
| Functionality | X | X | X | X | X |
| Reliability | X | X | - | - | X |
| Usability | X | - | (X) | X | (X) |
| Maintainability | X | - | - | (X) | (X) |
| Efficiency | (X) | X | (X) | X | X |
| Portability | (X) | - | (X) | X | X |

According to ISO/IEC 9126 the software quality parameters comprise several features, namely:

Functionality: accuracy, suitability, interoperability, compliance, security.
Reliability: maturity, fault tolerance, recoverability, availability.
Usability: understandability, learnability, operability.
Maintainability: analysability, changeability, stability, testability.
Efficiency: time behaviour, resource utilisation.
Portability: adaptability, installability, co-existence, conformance, replaceability.

[3] Usability parameters are considered to be 'soft' and describing quality of use. Typical examples are task conformance and flexibility or parameters related to human judgement, such as comfort.

In ISO/IEC 9126 traditional usability parameters (as addressed, e.g., in [9]), such as task conformance and flexibility, have been relocated to other clusters, such as functionality in case of task conformance, or portability and maintainability in case of flexibility (adaptability, changeability). However, developers are required to break the parameters down to external and internal measures of software ([2], p. 171). This requirement is actually in contrast to recent concepts found in software-performance engineering, such as [3], which equal human-oriented parameters with computer resources[4]. ISO/IEC 9126 requires to allocate external and internal measures to usability parameters, which clearly separates the view on software technology from that on humans, and directs the evaluation top down from usability engineering to software-performance engineering. However, there is still a lack of operational definitions to accomplish this top-down procedure. We will address exactly that issue in the following section.

Although the clusters provided by Oshana [10] mainly address traditional issues in software-performance engineering (p. 37), they provide those characteristics that enable the mapping of usability parameters to internal measures: '*Workload* or expected system use under applicable performance scenarios; *quantitative performance objectives*, in our case CPU and memory utilization and I/O bandwidth; *software characteristics*, or sequential processing steps for each performance scenario; *execution environment*, a representation of the hardware platform on which the proposed system will execute; *resource requirements*, or amounts of service required for key components; and *processing overhead*, usually obtained by benchmarking typical functions such as FFTs.' We will exemplify the top-down procedure for two major usability engineering principles: task conformance and flexibility.

## 3 Analytical Models for Usable Software

The following work is based on the finding that so far there do not exist analytical models for ***usable*** software, as they do for functional software (see, e.g., [1]). In software-performance engineering software execution can be modelled, in order to estimate resource requirements and calculate performance metrics. The definition of analytical models in software-usability engineering is still lacking, due to the lack of operational definitions of usability-engineering principles, and the different subjects of concern – users, organisation of work, and technology used for task accomplishment. In the following, a step towards analytical modelling for software-usability engineering is made, namely through giving operational algorithms that can directly be applied in the framework of performance-engineering approaches.

Figure 1 shows the steps in performance engineering when applied to interactive-systems engineering. The approach is similar to the provision of quality of service in the field of web engineering (e.g., [20]). In contrast to this bottom-up approach, namely quality of network services that finally enable quality of use at the user interfaces, we follow a top-down procedure, i.e. starting with a given understanding

[4] These approaches re-invoke the discussion about humans to be symbol-processing agents, as we had in the field of Artificial Intelligence in the last half of 1900.

of usable software and breaking down performance parameters and algorithms to implement objectives as defined by usability principles. The intention is not only to model more precisely the execution of interactive software, but also to show the effect of software on existing work situations and on resource utilisation. Finally, bottlenecks in interaction have to be identified as well as the parts of the software with high use of bottleneck elements or hindrances in task accomplishment.

In order to implement this idea we will use the experiences from model-based interactive software development as given in Stary [15] and the intense discussion on how to provide operational definitions for the implementation of usability principles (e.g., [16,17]). The TADEUS project features several models to define (executable) specifications of interactive software (the so-called application model):

- a model from the organisation at hand - the business intelligence model
- a task model - that part of the organisation the interactive computer system is developed for
- the user model – the set of user roles to be supported, both, with respect to functional roles in the organisation, and with respect to individual interaction preferences and skills
- the interaction model – providing those modalities that might be used for interactive task accomplishment.

The business intelligence model provides relevant information for the scenarios of use and the task model. The latter provides the input to specify software designs (see upper part of figure 1) and to derive execution models (see centre of the figure 1 - user and problem domain data model). For interactive system development, modalities for interaction have to provided, e.g., in terms of GUI-platform-specific specifications. They become part of the system configuration specifications (centre-left in figure 1), e.g., in terms of a common look and feel of the application, and are put into relation to tasks, data, and roles at the execution modelling level. The integrated structure and behaviour specification (application model) then provides a sound basis to check performance metrics.

The crucial part now is to define operational metrics, i.e. to define algorithms to check, whether performance criteria can be met. For example, task conformance can be measured in such an analytical way through two metrics: completeness with respect to task accomplishment (*effectiveness*), and *efficiency*. The first metrics means that at the system level there have to be both, a path to be followed for user control (i.e. navigation procedures), and mechanisms for data presentation and manipulation, in order to accomplish a given task. The second metrics is required to meet the principle task conformance in an optimised way. High efficiency in task accomplishment can only be achieved, in case the users are not burdened with navigation tasks and are enabled to concentrate on the content to be handled in the course of task accomplishment. In order to minimise the mental load the navigation paths have to be short. For instance, in TADEUS, at the specification level an algorithm checks, whether the shortest path in the interaction model has been selected, in order to accomplish the control and data-manipulation tasks.

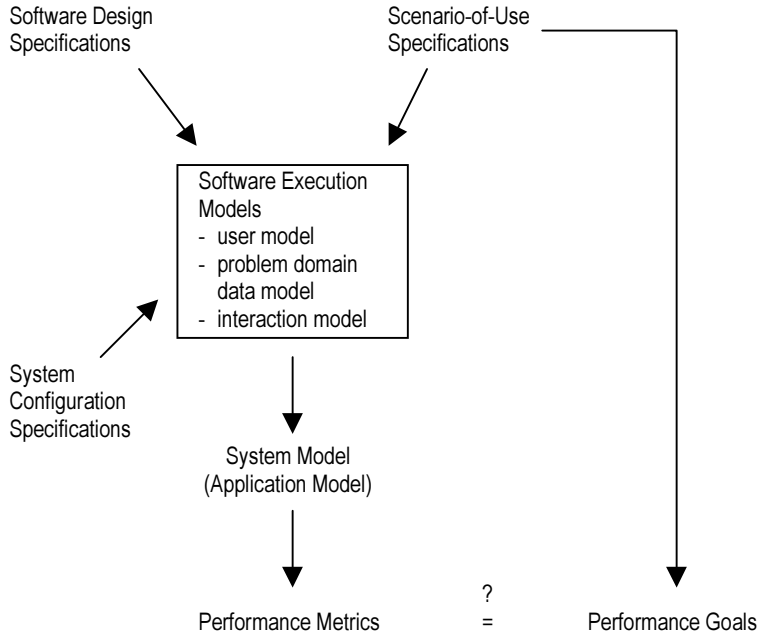



**Fig. 1.** Performance engineering methodology applied to usability engineering

The emergence of a widely accepted standard for software design, namely object-oriented specification, such as UML [12], has not only made it possible to focus on a single notation for specifying tasks, processes, interaction styles, user profiles, and problem domain data, but also to provide operational definitions or design and performance principles, such as completeness of specifications with respect to tasks [18]. Since such notations includes both static and dynamic aspects of systems. They are very well suited to generating performance results [11]. Unifying notations enable the thorough implementation of Smith's idea of formulating models in an appropriate form and capturing the desired behaviour [14]. However, in order to be adopted by practising software or usability engineers, performance engineering has to become part of a development methodology, such as partially proposed in Beilner et al. [1]. TADEUS is such an approach for the design of interactive systems. It currently uses OSA-diagrams [6] that can easily being mapped to UML. The TADEUS unifying notation provides elements for object-oriented representation throughout analysis, design, and prototyping. For performance engineering (implementing usability principles) additional semantically expressive constructs have to be provided at the specification level. Consider the case of arranging a cinema visit, as shown in Figure 2. The enriched notation allows to trace (e.g., for checking task conformance) how the task is related to data (which is a prerequisite for task accomplishment). It also allows the assignment to interaction styles, such as form-based interaction for acquiring options.

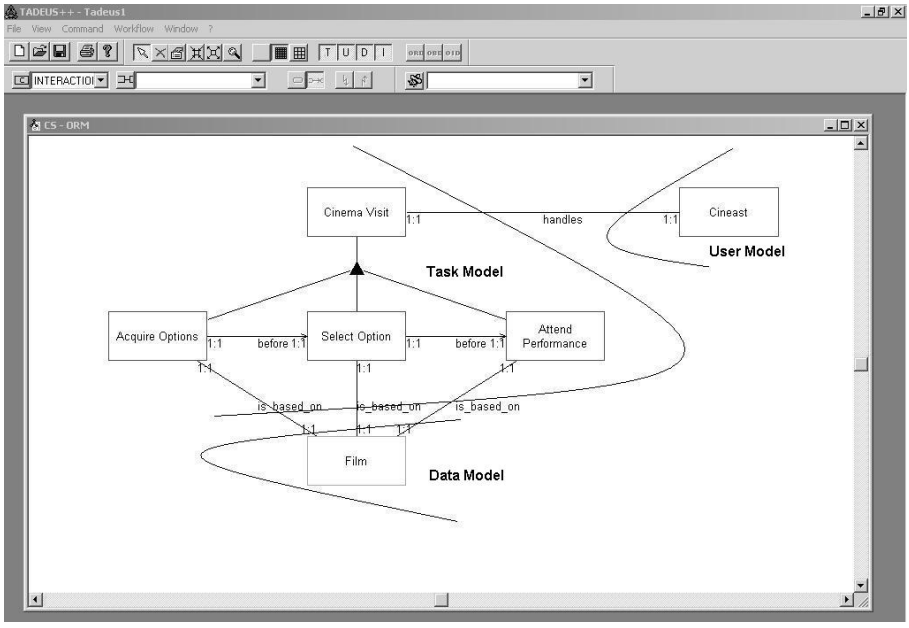



**Fig. 2.** A sample TADEUS task, role, and data model specification for arranging a cinema visit

**Table 2**. Typical activities in the course of contextual specification and semantic relationships provided in TADEUS

| SET UP | REFINE & ABSTRACT | RELATE |
|---|---|---|
| EMPLOYS<br>HAS | IS A<br>HAS PART | HANDLES, CREATES,<br>CONCERNS, INFORMS,<br>CONTROLS,<br>REQUIRES,<br>BEFORE, IS BASED ON,<br>CORRESPONDS TO<br>IS PRESENTED |

Table 2 gives an overview of the set of TADEUS relationships (grouped according different specification activities), in order to capture relationships between models and model entities. For each of the relationships, the TADEUS environment provides algorithms and allows the creation of such algorithms to process further semantic relationships [19]. The relationships between objects and/or classes are used (i) for specifying the different models of the TADEUS approach, and (ii) for the integration of these models. For instance, 'before' is a typical relationship being used within a model (in this case, the task model), whereas 'handles' is a typical relationship to connect elements of different models (in this case the user-role and the task model). In

both cases algorithms specific for each relationship are used to check the semantically correct use of the relationships.

In TADEUS, each model is described in an object-oriented way, namely through a structure diagram, i.e. an ORD (Object Relationship Diagram), and a behaviour (state/transition) diagram, i.e. an OBD (Object Behaviour Diagram). Structural relationships are expressed in TADEUS through those relationships listed in table 2, linking elements either within a model or stemming from different models. Behaviour relationships are expressed through linking OBDs at the state/transition level, either within a model or between models. Both cases result in OIDs (Object Interaction Diagrams). In Figure 3 the different types of specification are visualised in the context of model-based specification.

In order to ensure the semantically and syntactically correct use of specification elements, activities are performed both at the structure and behaviour level:

1. Checking ORDs: In TADEUS it is checked whether each relationship
   - is used correctly, i.e. whether the entities selected for applying the relationship correspond to the defined semantics, e.g., task elements have to be given in the task model to apply the 'before' relationship;
   - is complete, i.e. all the necessary information is provided to implement the relationship, e.g., checking whether there exist task-OBDs for each task being part of a 'before'-relationship.

2. Checking OBDs: It is checked whether the objects and/or classes connected by the relationship are behaving according to the semantics of this relationship, e.g., the behaviour specification of data objects corresponds to the sequence implied by 'before'.

The corresponding algorithms are designed in such way that they scan the entire set of design representations. Basically, the meaning of constructs is mapped to constraints that concern ORDs and OBDs of particular models, as well as OIDs when OBDs are synchronised. The checker indicates an exception, as long as the specifications do not completely comply to the defined semantics. For instance, the correct use of the 'before' relationship in the task model requires to meet the following constraints: (i) at the structure layer: 'Before' can only be put between task specifications, (ii) at the behaviour layer: The corresponding OBDs are traced whether all activities of task 1 (assuming task 1 'before' task 2) have been completed before the first activity of task 2 is started. Hence, there should not exist synchronisation links of OIDs interfering the completion of task 1 before task 2 is evoked.

The same mechanism can be used to ensure task conformance and flexibility. Task conformance is understood according to existing standards (e.g., ISO), namely

(a) to provide accurate functionality by the software system for accomplishing work tasks, and
(b) to burden users only with a minimum of interaction tasks at the user interface to accomplish these tasks.

In terms of specifications this understanding of task conformance means to provide for (a) task specifications (TADEUS task modelling) and refinements to procedures both, at the problem domain data level (TADEUS data modelling) and at the user

interface level (TADEUS interaction and application modelling). In order to achieve (b), [illegible] id

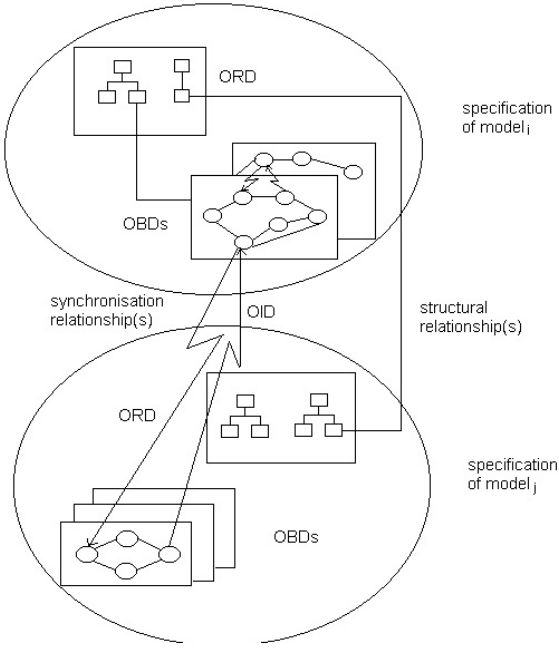



**Fig. 3.** Interplay between notation and model-based development in TADEUS

manipulation of data has to be minimised, whereas the software output has to be adjusted to the task and user characteristics. The corresponding specification is performed at the TADEUS-user, interaction and application-modelling level. Hence, the task-specific path of an OBD or OID must not contain any procedure or step that is not required for task accomplishment. It has to be minimal in the sense, that only those navigation and manipulation tasks are activated that directly contribute to the accomplishment of the addressed work task. In figure 4 a sample task-model OBD is given that provides the basis at the task level to achieve task conformance.

Consequently, in order to ensure task conformance, two different types of checks are performed in TADEUS: (i) task completeness of specification, and (ii) minimal length of paths provided for interactive task accomplishment (only in case check (i) has been successful). Check (i) ensures that the task is actually an interactive task (meaning that there exists a presentation of the task at the user interface) as well as there exists an interaction modality to perform the required interactive data manipulation to accomplish the addressed task.

Initially, the checker determines whether the task in the task-model OBD is linked to an element of an ORD of the interaction model to present the task at the user interface (through the relationship 'is-presented'). Then it checks links throughout the entire specification whether the task is linked to a data set that have to be manipulated

interactively to accomplish a task. In order to represent that relationship between tasks and problem domain data, the 'is-based-on'-relationship is used. Each interactive task

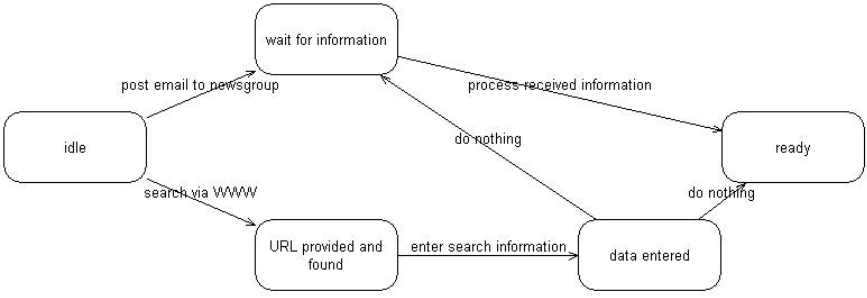



**Fig. 4.** OBD for the task Acquire Options

also requires a link to interaction-model elements which present the work-task data for interactive manipulation to the user. Finally, each task requires a link to a user role (expressed through 'handles'). Once the completeness of a task specification has been checked, each path related to that task in an OBD is checked, whether there exists a path that is not related to the data manipulation of that task. This way, the specification is minimal in the sense, that the user is provided with only those dialogs that are necessary to complete the addressed task through the assigned role. Note, that there might be different paths for a task when handled through different roles.

Flexibility is understood in several directions, although common characteristics can be identified:

(i) Flexibility with respect to the organisation of tasks
(ii) Flexibility with respect to user roles
(iii) Flexibility with respect to interaction styles
(iv) Flexibility with respect to assignments.

Flexibility in general means (a) to have more than one options to meet an objective, and (b) it has to be possible to switch between those options.

Flexibility with respect to the organisation of tasks means that a particular task might be accomplished in several ways. Hence, a specification enabling flexibility with respect to tasks contains more than one path in an OBD or ORD (implemented through 'before'-relationships) to deliver a certain output given a certain input. Flexibility with respect to user roles means that a user might have several roles and even switch between them, eventually leading to different perspectives on the same task. In TADEUS, the runtime environment for user interface prototyping allows to assign a person to different roles, hence instantiating different roles with the same person.

Flexibility with respect to interaction styles does not only require the provision of several styles based on a common set of dialog elements, as shown in figure 5 for Graphical User Interfaces and browser-based interaction, but also the availability of more than a single way to accomplish interaction tasks at the user interface, for instance direct manipulation (drag & drop) and menu-based window management,

within a particular style of interaction. The latter can be checked again at the OBD/OID-level, namely through checking whether a particular operation, such as closing a window, can be performed along different state transitions. For instance, closing a window can be enabled through a menu entry or a button located in the window bar.

Flexibility with respect to assignments involves (i) – (iii) as follows: In case a user is assigned to different tasks or changes roles, in order to accomplish a certain task, assignments of and between tasks and roles have to be changed. In case a user wants to switch between modalities or to change interaction styles, the assignment of interaction styles to data and/or tasks have to be modified. These modifications might also be caused by the software system, in case adaptation to users is performed automatically. In any case, changing assignments requires links between the different entities that are activated or de-activated at a certain point of time. It requires the existence of assignment links as well as their dynamic manipulation. Both has been provided in the TADEUS environment for user interaction, either through additional semantic relationships, e.g., 'handles' between roles and tasks (linking different models), or the runtime environment of the prototyping engine. The latter might be part of a workflow management systems supporting the adaptability to organisational and/or user needs.

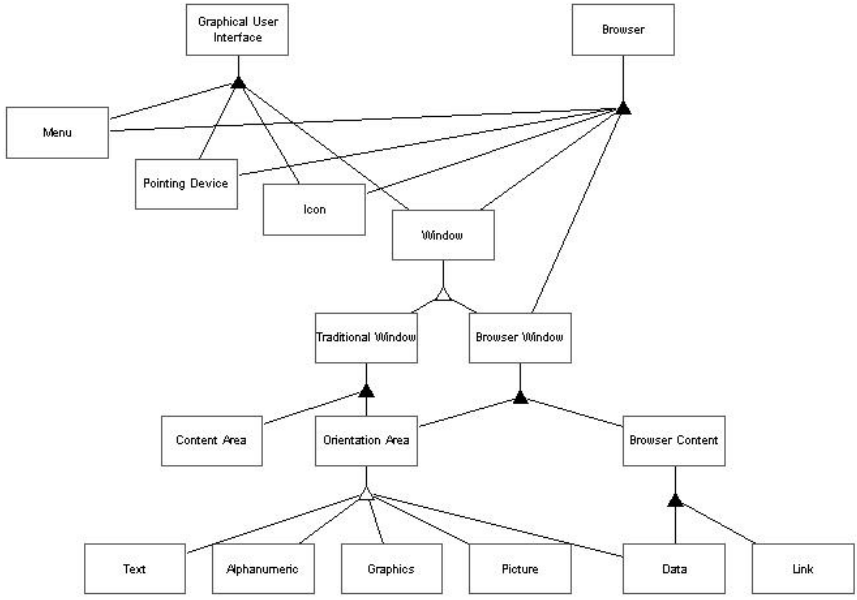



**Fig. 5.** Part of a TADEUS static interaction model (ORD) for GUI- and browser-based interaction

In this chapter we have shown

(i) that existing model-based frameworks for software development can be used to embed usability-engineering metrics into software-performance engineering concepts,

(ii) at the specification level usability-engineering principles can become operational to implement major usability principles.

This way, the context of use can be considered to be a set of performance parameters that can be measured very early in the course of software development.

## References


1. Beilner, H., Banse, E.: Computer Performance Evaluation – Modelling Techniques and Tools. LNCS, Vol. 977, Springer, Heidelberg (1995)
2. Bevan, N., Azuma, M.: Quality in Use: Incorporating Human Factors into the Software Engineering Lifecycle. In: Proceedings ISESS'97 International Software Engineering Standards Symposium, IEEE (1997) 169-179
3. Brataas, G., Hughes, P.H., Solvberg, A.: Performance Engineering of Human and Computerised Workflows. In: Olivé, A., Pastro, J.A. (eds.).: Advanced Information Systems Engineering. LNCS No. 1250, Springer, Barcelona (1997) 187-202
4. Byrne, M.D., John, B.E., Wehrle, N.S., Crow, D.C.: The Tangled Web We Wove: A Taskonomy of WWW Use. In: Proceedings CHI'99, ACM (1999) 544-551
5. Dahn, D., Laughery, K.R.: The Integrated Performance Modeling Environment – Simulating Human-System Performance. In: Proceedings Winter Simulation Conference, ACM (1997) 1141-1145
6. Embley, D.W., Kurtz, B.D., Woodfield, S.N.: Object-Oriented Systems Analysis. A Model-Driven Approach, Yourdon Press, Englewood Cliffs, New Jersey (1992)
7. ISO/IEC 9126: Software Product Evaluation - Quality Characteristics and Guidelines for their Use. International Standards Organisation, Geneva (1991)
8. ISO DIS 13407: A User-Centred Design Process for Interactive Systems. International Standards Organisation, Geneva (1997)
9. Newman, W.M., Lamming, M.G.: Interactive System Design. Addison Wesley, Wokingham (1995)
10. Oshana, R.S.: Winning Teams: Performance Engineering During Development. In: Computer, IEEE, Vol. 33 (2000) 36-44
11. Pooley, R., King, P.: The Unified Modelling Language and Performance Engineering. In: IEE Proc.-Software, February, 146(1999)1, 2-11
12. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual. Addison-Wesley, Reading, MA (1999)
13. Shneiderman, B.: Human Factors Experiments in Designing Interactive Systems. In: Computer, IEEE, December, 12(1979)12, 9-20
14. Smith, C.U.: Performance Engineering of Software Systems. Addison-Wesley, Reading (1990)
15. Stary, Ch.: TADEUS: Seamless Development of Task-Based and User-Oriented Interfaces. In: Transactions on Systems, Man, and Cybernetics, IEEE, Vol. 30 (2000) 509-525
16. Stary, Ch., Totter, A.: Cognitive and Organizational Dimensions of Task Appropriateness. In: Proceedings ECCE-8, 8th European Conference on Cognitive Ergonomics, European Society of Cognitive Ergonomics (1996) 127-131
17. Stary, Ch., Vidakis, N., Mohacsi, St., Nagelholz, M.: Workflow-Oriented Prototyping for the Development of Interactive Software. In: Proceedings IEEE 21st International Computer Software and Applications Conference COMPSAC´97 (1997) 530-535
18. Stary, Ch., Totter, A.: Re-Thinking Function Allocation in Terms of Task Conformance. In: Proceedings ALLFN'97, First International Conference on Allocation of Functions, International Ergonomics Association, Galway, IEA Press, Louisville, KY (1997) 135-142

19. Vidakis, N., Ch. Stary: Algorithmic Support in TADEUS. In: Proceedings ICCI'96, IEEE (1996)
20. Vouk, M.A., Bitzer, D.L., Klevans, L.R.L.: Workflow and End-User Quality of Service. In: Transactions on Data & Knowledge Engineering, IEEE, July/August, 11(1999)4, 673-687

# Using Load Dependent Servers to Reduce the Complexity of Large Client-Server Simulation Models

Mariela Curiel[1] and Ramon Puigjaner[2]

[1] Departamento de Computación y Tecnología de la Información,
Universidad Simón Bolívar, Apartado Postal 89000,
Caracas, 1080-A, Venezuela
mcuriel@ldc.usb.ve
http://www.ldc.usb.ve/~mcuriel
[2] Departament de C. Matemàtiques i I, Universitat de les Illes Balears,
Palma de Mallorca, E-07071, Spain
putxi@ps.uib.es

**Abstract.** The process of developing software systems that meet performance objectives has been called Software Performance Engineering (SPE). The goal of SPE techniques is to construct performance models in each phase of the software development cycle. This allows us to detect designs that are likely to exhibit unacceptable performance. Simple models are preferable in early stages of the development cycle because their solutions require less computational effort and time. Hence, our research is oriented to reduce the execution time of large client-server simulation models by replacing some of their components, by less time-consuming models: Load Dependent Servers (LDS). In this chapter we describe the construction of Load Dependent Servers that model server stations with transactional workloads. These LDS also include a representation of the main overhead sources in transactional environments: operating system and DBMS overhead. As case study we model a server station running TPC-C transactions.

## 1 Introduction

In order to develop client-server applications that meet performance objectives it is necessary to predict the system behaviour in each phase of the development cycle, i.e., to apply software performance engineering (SPE) techniques ([9], [12]). The goal of SPE techniques is to evaluate performance in each phase of the software development cycle by constructing performance models. Such models allow us to detect designs that are likely to exhibit unacceptable performance timely. Although some tools have been developed to support the construction of client-server performance models, most of them are based in the time consuming simulation technique (e.g., [10] and [11]). Simulation models are adequate in the final stages of the development cycle, but simple and less-precise models (e.g.,

R. Dumke et al. (Eds.): Performance Engineering, LNCS 2047, pp. 131–147, 2001.

analytical models) are sufficient at the beginning because many parameter values are unknown ([5],[14]). Additionally, the solution of simple models requires less computational effort and time, which is suitable for the first development cycles. Nowadays, performance analysts cannot afford to wait for days of simulation because the product development cycles are shorter. Therefore, one of the greatest challenges of future performance tools is to offer alternative methods that require less time and memory to solve the models [14]. The combination of several techniques in a single tool will enable the use of the adequate technique in the corresponding development phase.

An idea for reducing the complexity and execution time of large simulation models is to replace some of their components by less time-consuming models. The purpose is to substitute a group of elements by a synthetic element that imitates their behaviour (see Figure 1). The first element to be replaced is the server station [1]. When a LDS replaces a server station model in a complex client-server model, the simulation time is reduced because the number of states and transitions is smaller. The LDS models to be proposed in this chapter are oriented towards predicting system performance in early stages of the development cycle, i.e., in predictive studies.

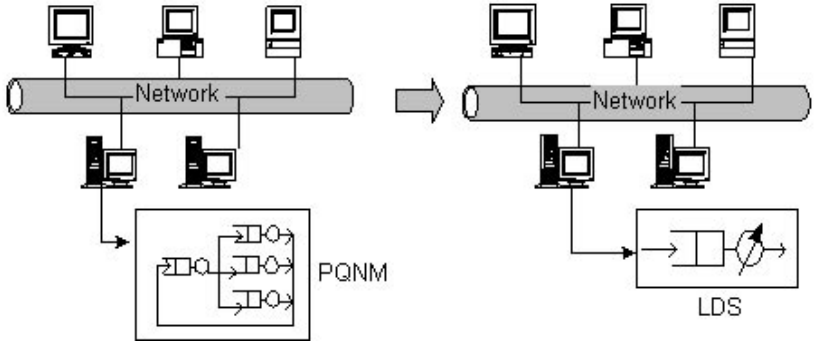



**Fig. 1.** Replacing a complex model with a LDS

When the system workload is transactional, it is advisable to represent the operating system and other overhead sources in the models. As it is remarked in [6], applications such as database management systems have significant operating system activity. The inability to include operating system overhead in models that represent transactional systems compromises their accuracy, and prevents a properly understanding of the workload behaviour. In the LDS models to be proposed, the operating system overhead and the overhead due to the DBMS (Data Base Management Subsystem) are included. We are proposing a

[1] We will refer to the server associated with the client-server architectures as server station, and the server associated to the queuing network models as service centre.

methodology for measuring and representing these overhead sources in queuing network models.

The remainder of the chapter is structured as follows: The development of the LDS is described in Section 2. Section 3 is devoted to explain the proposed method for measuring and modelling overhead. In Section 4, we describe the construction of a LDS that represents a server executing TPC-C transactions; actual server performance is compared with model predictions. In the last Section we conclude and present future work.

## 2 Load Dependent Servers

Sophisticated Queuing Network Models (QNM) can be solved using Hierarchical Modelling [8]. This method decomposes a large model into a number of smaller submodels, which are then recombined using a special type of service centre called Flow Equivalent Service Centre (FESC). When one of the smaller subsystems is to be solved, it is termed "aggregate subsystem", and the rest of the service centres, with which it interacts are collectively termed "complementary network". In the problem we are dealing with, the aggregate corresponds to the server station model to be replaced and the rest of the model represents the complementary network.

The goal is to replace the entire aggregate subsystem by a single service centre that mimics its behaviour, thus reducing the size of the network to be solved and, in consequence, the execution time. This behaviour, as viewed by the complementary network, is the flow of the customers outgoing the aggregate and incoming the complementary network. An approximation for this flow can be obtained by making the decomposability assumption that the average rate at which customers leave the aggregate depends only on the state of the aggregate, where the customer population within the aggregate defines the state. Therefore, an aggregate can be completely defined by a list of its throughputs as a function of its possible customer populations.

FESC are represented in queuing network models using Load Dependent Servers (LDS). A LDS can be thought of as a service centre whose service rate (the reciprocal of its service time) is a function of the customer population in its queue. In contrast, a queuing service centre is load independent if it has service rate $\mu$ regardless of the number of customers in its queue. An FESC for an aggregate is a Load Dependent Server with service rates $\mu_k(n)$ equal to the throughputs $X_k(n)$ of the aggregate for all populations $n$ an classes $k$ (We will discuss the method for obtaining these rates in Section 4).

The construction of QNM, and particularly LDS, involves three important activities: parameterization, calibration and validation. Next Section discusses the methodology proposed in [2] and [3] to carry these activities out when the system overhead is represented.

## 3 Obtaining Model Parameters

There are three types of input parameters in a QNM: workload, basic software and hardware parameters. The workload parameters describe the load imposed on the computer system by the jobs or transactions submitted to it. The basic software parameters describe features of the basic software -such as the operating system and the data management subsystem- that affect performance. The hardware parameters characterise the computers and the networks used to interconnect them. The main source of information for determining input parameters is the set of performance measurements collected from the observation of the real system under study.

The determination of basic software parameters is difficult. Given a fix hardware configuration, it is assumed that overhead is dependent on the number and characteristics of active processes in the system. As a result, a proper overhead characterisation requires measurements on every combination of executing processes, i.e., on all the feasible states of the workload. The difficulty of this task in complex systems leads to a search for alternatives that allow us to reduce the number of measurements without loosing significant information (e.g., experimental designs [7]). The methodology proposed in [3] indicates how to obtain the workload and basic software parameters by designing a reasonable number of experiments. It also provides a strategy to represent the overhead and calibrate QNM, in particular LDS. The methodology consists of the following 5 steps: 1) obtaining the frequent states of the workload: this allows us to discard unlikely states from the set of measurements; 2) measuring the overhead of the frequent states; 3) fitting regression models to represent the overhead behaviour; 4) introducing the overhead in the QNM and 5) calibrating the models. The next subsections go deeply into each step.

### 3.1 Obtaining the Frequent States

The objectives of this first step are to obtain the workload parameters and develop a plain[2] queuing network model that provides the frequent states of the workload.

**Workload parameters.** When the real workload is not totally known, as it is common in predictive studies, the transaction characteristics can be collected from a representative benchmark. The methodology suggests to measure transaction demands in dedicated servers. The overhead, which appears when many transactions run currently, is not considered in this step, i.e., we measure the transaction demands by executing each class of transaction in isolation. The overhead is included in step 4 of the methodology. Accounting tools available

[2] From now on we use the adjective plain for distinguishing a simple queuing network model from a model that is solved using hierarchical modelling techniques, i.e., a LDS.

in the operating system and software monitors may be used to this end. In order to validate the measured demands, we construct a plain QNM where one client station submits transactions to the modelled server station. The demands are considered valid if predicted and measured response times per transaction are similar with certain margin of error (about 15%). The hardware parameters (CPU and disks parameters) are also obtained in this phase. The workload and hardware parameters serve to feed the LDS.

**The frequent states.** If we modify some parameters in the previous model, it is possible to compute the frequent states of the workload. The necessary changes are: 1) instead of one client station, we have to introduce the expected number of client stations; 2) the percentage of transactions of each type running into the system and the thinking times should also be introduced. The solution of this new model, which can be obtained by analytical or simulation methods, provides the marginal probabilities, i.e., the probability of having $j$ customers of class $k$ in the service centre $i$. Using these probabilities and other workload characteristics -such as the maximum number of client stations and the maximum number of customer classes- it is possible to compute all the feasible states in the system and their corresponding probabilities. A state is defined as a vector $S = (n_1, n_2, \ldots, n_r)$, where each $n_k$, denotes the number of class $k$ transactions in the state. The total number of transactions in any state is the number of transactions running in the server. We take those states whose probability is greater than a predefined threshold off all the feasible system states. The states with higher probabilities are the so-called *frequent states*. The sum of the frequent states probabilities should not be lower than 0.8.

### 3.2 Estimating the Frequent States Overhead

This activity can be decomposed into three tasks:

**Defining overhead.** The adopted definition is important because it determines the measurement strategy. We identified *basic resource usage* as the resource usage that the operating system attributes to one transaction when it runs alone in the system, i.e., without the presence of other transactions. It includes the resource usage in OS system calls to serve transactions requests. The basic resource usage per transaction is measured in step 1 of the methodology. The remaining resource consumption is considered *overhead*. It includes the OS and DBMS resource usage that is not directly attributable to the transaction, when the transaction runs alone, and the consumption of transactions, OS processes, DBMS processes, and other application processes when the server load increases.

**Studying the hardware-software platform.** Before measuring, it is necessary to investigate some features of the software platform -such as the operating system and the DBMS architecture- and review the available monitors. As regards software architecture, one should lay great stress on memory management,

file system and input/output modules. This study eases the identification of overhead sources (variables to be measured) and the right interpretation of monitor data.

**Measuring.** During the measurement period, each state is reproduced in the server and executed in isolation. Choosing the interval length involves a trade-off between shortening the measurement period and obtaining reliable data of the set of transactions. During the measurement period the thinking times are set to zero to guarantee the permanence of all the state transactions in the system. What we are going to measure is the overhead in the processor(s) and disks.

### 3.3 Fitting Regression Models

The objective of this phase is to derive regression models to describe the extra resource usage in the main devices: processor and disks. A regression model allows us to predict a random variable $y$ as a function of several other variables. The mathematical relationship established between the variables can take many forms. The most commonly used relationship assumes that the dependent variable $y$ is a linear function of the independent variables $xi's$. The general equation for a linear model is: $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ..... + \beta_m x_m + \epsilon$. The estimated variable $(y)$ is called response variable, and the variables used to predict the response $(x_i)$ are called predictors. The $\beta_i's$ are the $k + 1$ fixed parameters and $\epsilon$ is the error term.

In the overhead models the response variable is the

measured overhead (extra CPU usage and extra I/O operations), and the predictors are the amount of transactions per class in the system. The overhead models tend to be more complex because interactions among the predictors cannot be ignored.

### 3.4 Representing Overhead

The fitted regression models are introduced in the QNM. There are three basic ways of representing overhead in queuing network models:

1. By introducing new classes in the model, whose customers generate overhead.
2. By distributing overhead among the original customer classes of the QNM. The following approaches could be used: a) doing a fair distribution; b) assuming that overhead is proportional to the transaction demands; or c) distributing overhead among the transactions that appear as model predictors. So far, we have not found any of these alternatives leading more quickly to the model validation.
3. By mixing the above two approaches.

As a first approximated overhead representation (what we call *base representation*) one can choose any of the above alternatives. Whatever option one chooses, it is very difficult to get the overhead representation right at the first attempt. So, the calibration phase will be often necessary.

### 3.5 The Calibration Phase

Calibration is the change of the model to force it to match the actual system [9]. When we calibrate we modify some target parameters in the performance model. Target parameters may be any input or output parameters. On the other hand, the objective performance metric is the output metric of the actual system, which will match in the performance model after a successful calibration procedure. In our calibration process the objective performance metrics are the response time and the throughputs per customer class. As target parameters we use the customer classes service demands.

The drawback in multiple class models is that the effects of some calibration techniques, which are rather known in single class models, are unknown and they can vary with the particular system. Any modification to the target parameter of one transaction can cause the desired effect in its output performance metric, but it can also provoke an undesired effect in the output performance metrics of other transactions. We propose a calibration strategy of three phases:

- *Base representation*: is the initial overhead distribution, which is done using any of the methods described in Section 3.4. Once the overhead representation is introduced, the QNM is solved and its predictions are compared to server measurements.
- *First calibration strategy*: If we do not succeed in validating the model with the base representation, we apply the first calibration strategy. In this phase we modify the CPU demands and/or the I/O operations per transaction to increase (or reduce) the response times. When we want a larger increment, we modify the I/O demands. Changes in the CPU usage produce a minor effect. Increasing the demands of heavy transactions also produces a significant increment in the response time of all the transactions. The calibration procedure continues while errors are greater than 30%.
- *Second calibration strategy*: this phase is optional and can only be applied when the data are available. The aim is to observe the goodness of our models to predict the overhead when the load increases. With this new strategy, we try to match model predictions with measurements of a more loaded server.

## 4 Calculation of the LDS Capacity

Steps 1, 2, 3 and 4 of the methodology indicate how to obtain model parameters, particularly LDS parameters. After concluding the parameterization phase, we can start the LDS construction; it implies to obtain the load dependent throughputs per class ($X_k(n)$). These are computed by solving the network for each feasible population, $n = 1, ..., M$; where $M$ is the maximum number of transactions in the frequent states. The regression model characteristics determine the resolution method: if predictor values vary during the execution, analytical techniques cannot be used. The throughput per class as a function of the number of customers ($n$) is computed as follows:

1. Using the Little's law we compute the residence time (the total time spent at the LDS) per customer class ($k$):

$$R_k(n) = \frac{n_{k,cpu}(n) + n_{k,disk_1}(n) + \ldots + n_{k,disk_D}(n)}{X_{k,cpu}(n) - (X_{k,disk_1}(n) + X_{k,disk_2}(n) + ... + X_{k,disk_D})} \quad (1)$$

   where, $R_k(n)$ is the residence time of class $k$ customers in the system; $n_{k,cpu}(n)$ is the average number of class $k$ customers in the CPU, when the multiprogramming factor is equal to $n$; $n_{k,disk_i}(n)$ is the average number of class $k$ customers in the disk $i$; $X_{k,cpu}(n)$ and $X_{k,disk_i}(n)$ are the class $k$ throughputs in the CPU, and disk $i$, respectively, as a function of $n$; $D$ is the maximum number of disks represented in the model.
2. If the residence time of a class $k$ customer is $R_k(n)$, this client leaves the network with a frequency, in average, of $\frac{1}{R_k(n)}$.
3. If the workload has batch transactions, the actual number of customers in the system is $n + n_{batch}$, where $n_{batch}$ is the number of batch transactions. So, the actual frequency of class $k$ customers is computed as indicated in equation 2.

$$X_k(n) = (\frac{1}{R_k(n)})(n + n_{batch}) \quad (2)$$

It is assumed that $X_k(n)$ is the capacity of class $k$ transactions in the LDS when there are $n$ on-line customers in the system.

The computed $X_k(n)$, $n = 1, ..., M$, are placed in a vector $(V_k)$, which stores the load dependent throughputs of class $k$ customers. The LDS will use the values in each vector (we create one vector per each customer class) to compute the service time. However, the throughputs, $X_k(n)$, cannot be placed in position $V_k(n)$ because the actual number of class $k$ customers is not $n$, but $n_k$. Thus, the right place to put $X_k(n)$ is $V_k(n_k)$. When we do not control the number of customers per class, but they are created according to certain probabilities, it could exist some $z$ values to which $V_k(z)$ is not computed, i.e., at the end of $n$ iterations there could be empty positions in $V_k$. Given that $i$ and $j$ $(i < j)$ are positions of $V_k$, whose values have been computed; empty positions between $i$ and $j$ are filled with numbers that allow us to reach the value of $X_k(j)$, starting from the value of $X_k(i)$ in a linear way.

## 5 Case Study: Modelling a Server with TPC-C Transactions

This Section describes the steps to construct a LDS that model a server running a transactional workload. A benchmark is used as workload because the actual workload is not necessarily available in predictive studies. The most popular transactional benchmark at the time this research began was the TPC-C benchmark. The parameterization and calibration tasks of the model are done according to the methodology described in 3.1. Subsections 5.3, 5.4, 5.5 and

5.6 describe implementation details. We will consider that the model is valid when errors are below or very close to 30%. This is a reasonable expectation of model accuracy in predictive studies because errors in input data are similar [5]. Lazowska et al [8] agree on this percentage when they refer to response time predictions of multiple-classes models.

### 5.1 The Workload

The TPC-C benchmark [13] is intended to model a medium complexity on-line transaction processing (OLTP) workload. It is composed of read-only and update-intensive transactions that exercise the system components associated with transactional environments. The *New Order* transaction allows us to introduce a complete order through a single database transaction. It represents a frequently executed mid-weight, read-write transaction. The *Payment* transaction updates the customer's balance. It is a read-write, lightweight transaction with high frequency of execution. The *Order Status* transaction asks for the status of the customer's last order. It represents a mid-weight, read-only transaction with low frequency of execution. The *Delivery* transaction processes 10 new (not yet delivered) orders in batch mode. Each order is completely processed within the scope of a read-write database transaction. The Delivery transaction is executed with low freq! uency. Finally, the *Stock Level* transaction determines the amount of recently sold items that have a stock level below a specified threshold. It represents a heavy read-only database transaction.

The workload of the case study is composed of a subset of TPC-C transactions: New-Order, Payment, Order Status and Delivery. Thirty-five client stations submit transactions to the server. The client stations are implemented as software processes. Results with the entire group of TPC-C transactions are reported in [3].

### 5.2 The Hardware-Software Platform

The modelled server is a SPARC, SUN Ultra 1 workstation with SOLARIS 2.5 operating system. It has an internal disk of 2 GB and an external disk of 9.2 GB locally connected. ORACLE RDBMS V. 7.3.2 manages the database, which stores data corresponding to 10 warehouses. The full database is stored on the external disk and the internal disk stores data that is managed by the operating system (e.g., swap areas).

### 5.3 Obtaining the Frequent States

**The Plain Model.** The plain queuing network model helps get the frequent states of the workload. The model was developed in the QNAP2 language. The next paragraphs describe how the model parameters are obtained. Details can be found in [1].

**Table 1.** Workload description

| TPC-C transaction | Percentage | Keying time (seconds) | Thinking time (seconds) | Logical I/O operations |
|---|---|---|---|---|
| New Order | 40 | 18 | 6 | 46 |
| Payment | 40 | 3 | 6 | 9 |
| Order Status | 10 | 2 | 5 | 12 |
| Delivery | 10 | 2 | 5 | 160 |

- *Workload Description*: because of the differences in service demands (see logical operations in Table 1), we represent each TPC-C transaction with one customer class.
- *Service Centre Description*: a single service centre is used to represent each server device: the CPU and the disks. The CPU service time is assumed exponential and class-dependent. OS monitors help to obtain the mean CPU service time (see Service demands). Disk service times are also exponentially distributed and their means can be computed with the use of data provided by the manufacturer (mean of seek time, rotational speed and data transfer rate) and disk occupation, necessary to establish the correct seek time.
- *Service Demands*: The CPU usage is measured using the Solaris accounting facilities. Obtaining the physical I/O operations per transaction is, however, a more difficult task because most monitors only show logical I/O operations. Therefore, we choose a monitor that measures operations on disks without distinguishing the source process: the *iostat* monitor. As a result, transaction demands have to be measured in an isolated system, where just the transaction of interest runs. The measurement interval enables the execution of 16 to 20 transactions of the same class. The following reasoning leads us to this account: If few transactions are executed, the execution time could be extremely short to allow monitors to catch reliable data. On the other hand, when the number of transaction is large, the measurement period increases, and as a result, the total time invested in the model parameterization. The shorter the time to construct the model, the most useful and timely the model is. Additionally, if we run many instances of the same transaction for a long interval, the last transactions may find many data in the OS cache, whereupon its demands are not realistic.

Once the transactions demands have been validated, we introduce the rest of data that define the system under study in the model, i.e., the expected number of client stations submitting transactions (35), the percentage of transactions of each type running in the system, the keying and thinking time (Table 1). The percentages and times are taken from [13]; however, the thinking times are cut by half to increase the load. The solution of this new model is processed to obtain the total workload states. We get a total of 26,978,328 states and, the 83 states with the highest probabilities (their probabilities sum 0.91) are chosen. These are the frequent states. Each frequent state has seven or less transactions.

### 5.4 Estimating the Overhead

**Identifying the Overhead Sources.** Typical overhead sources and how they could be measured or estimated are described below.

- *Extra CPU usage in kernel mode.* Most operating systems have two operation modes: privileged (or kernel) mode and user mode. The time the CPU is running in privileged mode is called *system time*. By contrast, *user time* is the time that the CPU executes in user mode. The total *system time* is composed of the time attributed to user processes when they run in kernel mode, and the time used by kernel processes.
  - *System time of user processes.* In UNIX-based systems, the system time per process includes not only the time that each process executes in privileged mode due to the invocation of system calls, but part of the time that the operating system invests in activities such as scheduling, paging and interruption handling. Since we do not consider the time invested in the execution of basic system calls (system calls that a transaction executes in an isolated system) as an overhead source, we should take it out from the total system time to obtain the extra CPU consumption in kernel mode (note that it depends on the overhead definition). A way of achieving this aim is by subtracting the expected system CPU usage (S) from the measured one. The expected system CPU usage is obtained as follows:

$$\mathrm{S} = \sum_{i=1}^{T} n_i s_i \tag{3}$$

    where $s_i$ is the measured system time of class $i$ transactions when they are executed in an isolated system (step 1 of the methodology), and $n_i$ is the number of concluded class $i$ transactions in the measurement interval where the overhead is measured. $S_i$ includes the time in system calls that we want to remove. The total system time is given by UNIX monitors like *sar*, *iostat*, *vmstat*, etc.
  - *System time of kernel processes.* Previous UNIX-based systems had a monolithic design, i.e., the OS consisted of just one process. However, in most of the modern UNIX-based systems like Solaris, some daemons work together with the kernel running in privileged mode. The CPU usage of these processes is an overhead source that can be measured using monitors like Proctool (www.sunfreeware.com) or the Solaris tracing facilities. Even another alternative is to estimate this overhead source by subtracting the total CPU consumed by user processes (system and user CPU time) from the total measured CPU usage (obtained with almost any UNIX monitor). This gives an estimate of the CPU usage due to kernel processes.
- *Extra CPU usage in user mode.* In operating systems where no OS processes execute in user mode, the main responsible for this extra CPU consumption are the application servers (for example DBMS servers). They run in user

mode and increase their resource demands with the number of clients that they have to attend. Normally, other kinds of user processes, such as DBMS clients, do not increase their CPU usage (in user mode) with the load. In microkernel designs, however, there are OS processes running in user mode, whose consumption have to be incorporated to the extra user time. The CPU usage of user processes can be measured using any class-based monitor (they display resource usage per process). Also, this overhead source can be estimated by subtracting the expected user CPU usage from the measured one. The expected user CPU usage is computed as in 3, but taking the user time per transaction, instead of the system time.

- *Extra I/O operations*. The main sources of overhead are the I/O operations on swap areas, OS metadata and DBMS metadata. Operations on data pages are considered overhead (DBMS overhead) only if the pages were previously in memory, but they have been removed because of memory conflicts. This last overhead source is difficult to measure; it can only be estimated using subtractions. Most UNIX monitors do not provide each overhead source separately. In general, they give some kind of statistics and we have to estimate the overhead by comparing the results of several monitors and making simple subtractions. What the main UNIX monitors provide is:
  - *iostat*: It displays physical I/O operations distributed per I/O devices but not per process. By observing the activity per device we cannot distinguish operations on data files from operations due to overhead (for example, operations on swap areas), unless database and OS data are in different disks. From Solaris 2.6 on this difficulty is overcome because the iostat monitor provides I/O operations per disk partition. So, it is enough to know where data partitions, partitions used as swap areas, and other types of partitions are located.
  - *sar*: It displays physical I/O operations per I/O device, the raw I/O activity, buffer cache statistics, and paging activity.
    - UNIX systems offer different methods to access data: through the file system (file system-based) or directly to disks (*raw I/O operations*). When the DBMS does not use the file system, by observing the raw operations statistics we can distinguish the I/O operations attributed to the DBMS.
    - Although *sar* command provide the *paging activity*, these statistics do not necessarily measure extra I/O operations. In recent UNIX-based operating systems (e.g., Solaris, Linux, System V and BSD 4.4) the file data pages are part of the virtual memory pages. So, paging counters measure operations on swap areas (as usual), but also measure operations on data files, which are not necessarily extra I/O operations.
    - The buffer cache stores metadata blocks. *The buffer cache statistics* (hit and miss ratio) are useful for measuring the operations on metadata.
  - *SOLARIS tracing facilities*: With these tools we can determine where (in which disk partition) each I/O operation is executed, and which

process performed it. The problem with these tools is the large amounts of storage space that traces require and the subsequent trace analysis.

In the case study, we use information about the data location on disks to distinguish between operating system overhead and DBMS overhead. Since the database is completely stored on the external disk and the swap areas are located on the internal one, all the internal disk accesses are considered overhead due to the operating system. The operations on the external disk are operations on data blocks and on metadata blocks. An estimation of the total external disk overhead (extra operations on data and operations on metadata) is obtained by subtracting the expected I/O operations from the measured ones. The expected operations, $D_t$ , are computed as follows:

$$D_t = \sum_{i=1}^{T} n_i d_i \tag{4}$$

where $d_i$ is the number of I/O operations initiated by class $i$ transactions in an isolated system, and $n_i$ is the number of concluded class $i$ transactions in the interval where the overhead is measured. Once the total external disk overhead is computed it is possible to determine the percentage of overhead due to OS metadata by observing the buffer cache statistics.

**Experiments.** In order to measure the overhead that any frequent state generates, the state is reproduced in the server for a long enough time interval. The chosen interval enables the execution of about 100 transactions, which is considered a reasonable number in this specific case study. If a state has $m$ transactions, the benchmark controller (the process that creates the simulated client stations and writes the result on files) simulates $m$ client stations. Each client station chooses only one class of transaction from the state and sends it constantly during the measurement interval. When the variance of the measurements is low, five replications per measurement are enough. Otherwise, we double the number of measurements. The number of replications is maintained low because we are looking for a solution that yields reliable results with a reduced number of measurements.

### 5.5 The Regression Models

The measured or estimated[3] overhead data is used to fit regression models. The response variables of the regression models are: the CPU overhead (CPUO), the extra I/O operations on the operating system disk or internal disk (OSDO), and the extra I/O operations on the data base disk or external disk (DBDO) per second. The predictor variables are $T$ (the total number of transactions in the state), $N$ (the number of New Order transactions), $P$ (the number of Payment transactions), $O$ (the number of Order Status transactions), and $D$ (the amount of Delivery transactions).

[3] When the overhead cannot be directly measured it is estimated.

Some data help check the fitted regression model. One of these data is the *coefficient of determination* ($R^2$), which measures the goodness of the regression. The higher the value of $R^2$, the better the regression. The $R^2$ feasible values belong to the interval [0,1] and they are often expressed as a percentage by multiplying by 100. The significance of the parameters is tested using *hypothesis testing*. To verify that the regression assumptions hold, we use an *assortment of plots* that taken together reveal the strengths and weaknesses of the model. Using these plots, for example, it is possible to determine whether or not the residuals are normally distributed and statistically independent. When the model is not appropriate, we try with transformations in the response variables, in the predictor variables or in both. The models (5) and (6) describe the CPU and database disk overhead. They explain the 84.2% and 90% of the variation respectively. In order to obtain normally distributed errors, it was necessary to apply a transformation to the response variable of the CPU overhead model. As no trend is observed in the internal disk data, a histogram is constructed to determine the data distribution. It is important to remark that the histograms are used in groups where the overhead is negligible.

$$(\text{CPUO})/\text{sec})^{0,75} = 0.027 + 0.005T + 0.044D + 0.0053DP \tag{5}$$

$$\text{DBDO}/\text{sec} = 2.4355 + 6.7552D - 0.3265DT \tag{6}$$

### 5.6 Overhead Representation and Calibration Process

As soon as we have the regression models, it is possible to conclude the parameterization of the LDS and generate it. The client stations are removed from the Load Dependent Server, only the CPU, the disks and the customer descriptions are required to construct the model. The results of each calibration are:

**Base representation.** In the preliminary representation two new customer classes introduce the OS and DBMS overhead: OSPROC and BDPROC. Each second, a customer of each new class is created. The LDS predictions using this overhead representation are shown in Table 2. We obtain errors in response times below 21%, which agrees with our objectives. However, it is possible to reduce the errors by making some modifications to the model. This leads to the first calibration strategy.

**First calibration strategy.** The response times in Table 2 suggest to increase the Payment and Order Status demands. After several iterations, where customers demands were modified, we achieve errors in the response times of the system under study (a server with 35 client stations) below 6% , and below 13.5% in the transaction throughputs. In the final representation the overhead is distributed among BDPROC customers (35%) and the four TPC-C transactions (65%). Figure 2 shows model predictions and measurements of the server

**Table 2.** Response times using the base representation

| TPC-C Transaction | Meas. (sec) | Pred. (sec) | Error (%) |
|---|---|---|---|
| New Order | 1.202 | 1.128 | 6.218 |
| Payment | 0.322 | 0.255 | 20.966 |
| Order Status | 0.297 | 0.244 | 17.998 |
| Delivery | 1.989 | 1.919 | 3.538 |

with this calibration strategy. Results with a larger number of client stations are also displayed.

**Second calibration strategy.** One can notice in Figure 2 (see results of the first calibration strategy) that prediction errors increase with the load. The higher error is 41.7%, which is associated to the Order Status transaction. In order to improve the goodness of our model when the load increases, we can apply the second calibration strategy. The strategy looks for matching the model predictions with measurements of a more loaded server (35, 40 and 45 client stations submitting transactions). The three sources of measurements are considered during the calibration process. Figure 2 also shows model predictions and measurements of the server with the second calibration strategy. As was to be expected the use of more values in the calibration process allows us to reduce the errors when the load increases. All errors in the new response time predictions are below to 30%, even for 50 client stations. The results agree with the proposed objectives. The errors in throughputs per transaction are below 15.76% So, we have achieved a model that is able to predict the throughputs and response time for the expected load (35 client stations) and for a more loaded server with reasonable errors. It is important to remark that errors in the predictions of some transactions can be reduced to the detriment of other. In this case we have chosen a combination of parameters that minimise the errors globally.

### 5.7 Execution Time

Although along this chapter we have only described the LDS results, we also develop a plain queuing model with the same parameters. The aim is to compare LDS behaviour with a model whose results are not obtained by approximations. Predictions are similar in both models, but important differences are observed in the execution time: The construction of the LDS lasts 339 seconds. This time is important during the calibration phase, but once this phase is overcome, a valid LDS is constructed just once. The LDS provides results for all the configurations (35, 40, 45 and 50 client stations) in scarcely 8 seconds, i.e., 2 seconds per configuration. On the other hand, the plain QNM provides the same type of

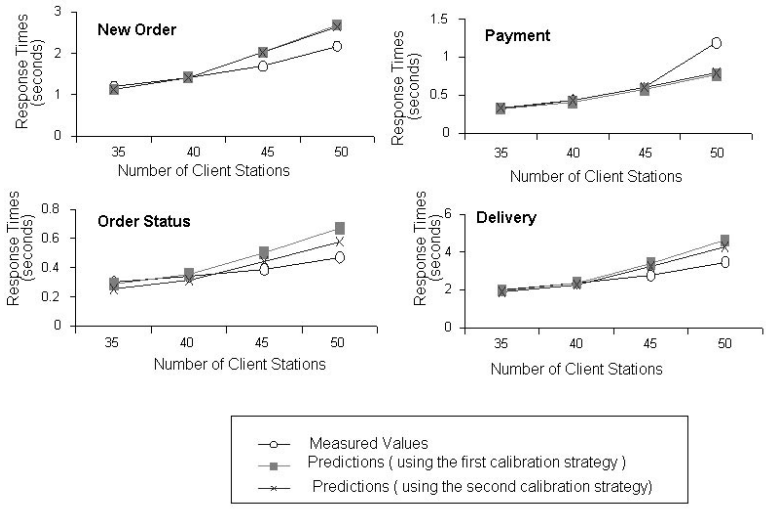



**Fig. 2.** Results with the two calibration strategies

results in 318 seconds. So, the execution time is considerably reduced with the LDS.

## 6 Conclusions and Future Work

The main contribution of this work is the idea of reducing the complexity and execution time of large simulation models by replacing some of their components by less time-consuming models: Load Dependent Servers. These LDS can be used to predict performance in early phases of the development cycle. In this chapter we have described how to construct a LDS that models a server station with a transactional workload. From the development process and the obtained results we can conclude:

1. The generation of the LDS is time-consuming when simulation techniques are applied. However, once generated, the LDS is able to provide results in very short periods of time, whatever the kind of model where it is incorporated, i.e., analytical or simulation models.
2. For the system under study, the LDS predicts the transaction response time and the transaction throughputs with acceptable accuracy. It can also predict throughput and response time behaviour satisfactorily when the load increases. The development of the model did not imply any additional effort compared to the development of a plain queuing network model.
3. Future work involves studying the impact of incorporating these load dependent servers in performance tools. Although the profits are obvious, some questions should be still answered, for example: what kind of interface would

be required to introduce the overhead models? how the LDS would interact with other model components?
4. The methodology allowed us to obtain reliable and useful overhead data for the queuing network model. The obtained results are valid just for the used hardware-software platform. Any meaningful change in it would involve a new application of the methodology. In [3] we describe how the methodology can be implemented in other hardware-software platforms. Future work involves the incorporation of the OS and DBMS overhead due to operations across the network.

## References


1. Curiel, M., Puigjaner, R.: A Measurement Methodology to Calibrate Analytical Models Including Operating System Overhead. Proceedings of the Sixth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, Montreal, July (1998) 255–260
2. Curiel, M., Puigjaner, R.: Modeling Overhead in Servers with Transactional Workloads. Proceedings of the Seventh International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems. Maryland University, USA, October (1999) 182–189
3. Curiel, M.: Un Método para la Medición del Overhead de Servidores con Cargas Transacccionales y su Representación mediante Modelos de Redes de Colas. PhD. Thesis. Balearic Islands University. July (2000).
4. Gunter, N.: The Practical Performance Analyst. Performance-by-Design Techniques for Distributed Systems. McGraw-Hill (1998).
5. Heidelberger, P., Lavenberg, S. S.: Computer Performance Evaluation Methodology. IEEE Transactions on Computers, December (1984) 1195–1220.
6. Herrod, S. A.: Using Complete Machine Simulation to Understand Computer System Behavior. PhD. Thesis. Stanford University (1998).
7. Jain, R.: The Art of Computer System Performance Analysis. John Wiley & Sons, INC. (1991).
8. Lazowska, E., Zahorhan, J., Scott, G., Sevcik, K.: Quantitative System Performance. Prentice Hall (1984).
9. Menascé, D., Almeida, V., Dowdy, L. W., Capacity Planning and Performance Modeling. From Mainframes to Client-Server Systems. Prentice Hall (1994).
10. Mok, D. S., and Funka-Lea, C. A.: An Interactive Graphical Modeling Tool for Performance and Process Simulation. Proceedings of the 1993 Winter Simulation Conference (1993) 285–293.
11. SIMULOG: MODARCH Version 3.0. Reference Manual (1997).
12. Smith, C. U.: Software Performance Engineering. Lecture Notes in Computer Science, Vol 729. Computer Performance Evaluation. Modelling Techniques and Tools. (1993) 509–536.
13. Transaction Processing Council: TPC-C Benchmark Standard Specification, Revision 3.21 (1996)
14. Trivedi, K. S., Haverkort, B. R., Rindos, A., Mainkar, V.: Techniques and Tools for Reliability and Performance Evaluation: Problems and Perspectives. Lecture Notes in Computer Science. Computer Performance Evaluation. Proceedings of 7th International Conference on Modelling Techniques and Tools. Springer-Verlag, May (1994) 1–24.

# Performance Evaluation of Mobile Agents: Issues and Approaches

Marios D. Dikaiakos and George Samaras

Department of Computer Science
University of Cyprus, CY-1678 Nicosia, Cyprus
{mdd,cssamara}@ucy.ac.cy

## 1 Introduction

With the emergence of Internet as a world-wide infrastructure for communication and information exchange, Internet-based distributed applications have gained remarkable popularity. One of the most promising approaches for developing such applications is the Java-based mobile-agent paradigm [5,8,15,39]. Mobile Agents (MA) are being used already in a variety of applications ranging from telecommunications management to Web databases, cooperative environments information-gathering systems, electronic commerce systems, intelligent distributed systems, and so on [23,24,7,9,4]. In that context, a distributed application can be considered as a dynamic group of agents working in coordination to accomplish some goal.

Mobile agents offer a number of diverse advantages in the development of distributed systems including enhanced programmability through modular, object-oriented structures and the increased flexibility provided by mobility; performance optimization for distributed operations that involve heavy network delays and/or weak connectivity; extended autonomy in terms of existing support for asynchronous execution and disconnected operations [6,8,21]. The employment of MA technologies for the development of next- generation Internet systems opens numerous research problems related to programming APIs and tools, security, fault-tolerance, design paradigms and programming techniques, communication, intelligence, scalability and performance [18].

The issue of performance, in particular, is very important in emerging Internet-systems: numerous studies show that performance of systems and applications determines to a large extent the popularity of Internet services and user-perceived Quality of Service [1,3]. Moreover, performance evaluation is crucial for performance "debugging," that is the thorough understanding of performance behavior of systems. Results from performance analyses can enhance the discovery of performance and scalability bottlenecks, the quantitative comparison of different platforms and systems, the optimization of application designs, and the extrapolation of properties of future systems.

So far, systematic performance studies and experiments have provided important insights for the design of parallel and distributed systems and the tuning of applications [13,14,17,29]. Nevertheless, the more complex a system or application is, the harder its evaluation becomes, because of the large variety of factors

R. Dumke et al. (Eds.): Performance Engineering, LNCS 2047, pp. 148–166, 2001.

that affect its performance [10]. This observation is particularly true for systems and applications running on top of gigantic platforms like Internet [1]. The performance evaluation of mobile-agent systems is even harder than the analysis of more traditional parallel and distributed systems, due to the dynamic nature and agile configuration of mobile agents.

The objective of this chapter is to investigate issues pertinent to performance engineering for mobile-agent platforms and systems. First, we describe briefly the basic characteristics of mobile agents. Then, we explore the notion of performance evaluation in the context of mobile agents and present an overview of recent approaches for performance evaluation and analysis of mobile-agent systems. These approaches include benchmarking efforts, scalability studies, analytical models, and Petri-net modeling.

Finally, we present a novel performance analysis approach that we developed to gauge quantitatively the performance characteristics of different mobile-agent platforms [12,29]. We materialize this approach as a hierarchical framework of benchmarks designed to isolate performance properties of interest, at different levels of detail [12,29]. We identify the structure and parameters of benchmarks and propose metrics that can be used to capture their properties. We present a set of proposed benchmarks and examine their behavior when implemented with commercial, Java-based, mobile-agent platforms.

## 2 Mobile Agents

In mobile applications, data may be organized as collections of objects, in which case objects become the unit of information exchange between mobile and static hosts. Objects encapsulate not only pure data but also information regarding their manipulation, such as operations for accessing them. Incorporating active computations with objects and making them mobile leads to *Mobile Agents*.

Mobile agents are processes dispatched from a source computer to accomplish a specified task [21,35,38,39]. Each mobile agent is a computation along with its own data and execution state. In this sense, the mobile agent paradigm extends the RPC communication mechanism, according to which a message is just a procedure call whereas now it is an object with state and functionality. After its submission, the mobile agent proceeds autonomously and independently of the sending client. When the agent reaches a server, it is delivered to an agent execution environment. Then, if the agent possesses necessary authentication credentials, its executable parts are started. To accomplish its task, the mobile agent can transport itself to another server, spawn new agents, or interact with other agents. Upon completion, the mobile agent delivers the results to the sending client or to another server.

By letting mobile hosts submit agents, the burden of computation is shifted from the resource-poor mobile hosts to the fixed network. Mobility is inherent in the model; mobile agents migrate not only to find the required resources but also to follow mobile clients. Finally, mobile agents provide the flexibility to adaptively shift load to and from a mobile host depending on bandwidth

and other available resources. Mobile-agent technology is suitable for wireless or dial-up environments [21,25].

Mobile agents are typically written in interpreted languages, such as Java and Tcl, and thus tend to be independent of the operating system and hardware architecture. A number of commercial and research projects have developed mobile-agent environments, such as Aglets by IBM [8], Concordia by Mitsubishi [40], Voyager by Objectspace [15], Grasshopper by IKV++ [5], Mole by the University of Stuttgart [33], etc.

## 3 Different Approaches for MA Performance Analysis

Traditionally, the performance assessment of software systems is conducted through experimentation and monitoring, simulation, modeling and combinations thereof. The more complex a system is the harder its performance evaluation becomes, dictating the employment of these techniques at various levels of abstraction. To this end, software systems are modeled as hierarchical structures of interacting modules, i.e., subsystems and objects; each module is assigned a performance model that incorporates performance and load parameters of relevance, and a description of the underlying architecture and workload [41]. Model development is performed in a "top-down" manner, starting from high-level structure and moving towards code implementation. Experimentation and/or simulation can be used at various layers of abstraction to specify the values of modeling parameters.

The development and assembly of performance models for MA systems is more complicated than for more "traditional" parallel, distributed or object-oriented software; when analyzing the performance of MA-based systems, we must take into account issues, such as:

- The absence of global time, control and state information: this makes it hard to define and determine unequivocally the condition of a particular MA-system at a particular moment.
- The complex architecture of MA platforms: simple models and metrics used for the performance characterization of typical parallel and distributed systems are not adequate for isolating performance problems of MA systems. Further investigation and definition of more complex models and metrics are necessary.
- The variety of distributed computing (software) models that are applicable to mobile-agent applications: this variety dictates the design of different experiments, tailored to the different software models of interest.
- Construction of simple and portable benchmarks for experimentation is difficult due to the diversity of operations found in MA platforms.
- The presence of mobility, which makes it hard to establish a concise and stable representation of system resources that affect MA performance, due to the dynamic nature and agile configuration of MA systems.
- The additional complexity introduced by issues that affect the performance of Java, such as interpretation versus compilation, garbage collection, etc.

A number of recent projects have addressed the issues mentioned above. One thread of work examines the relative advantages of the mobile-agent paradigm versus other distributed-computing models from a performance perspective:

For instance, in [34], Strasser and Schwehm introduce a mathematical model to compare analytically the performance of *agent migration* and *remote execution* with the more traditional approach of *remote procedure calls* (RPC), in the case where mobile agents are used for filtering information in information retrieval applications. Their performance model takes into account issues, such as network throughput, communication latency, and network load. Furthermore, the size and execution time for RPCs, the size of agent code, data, and state, the "selectivity" (filtering ratio) of an agent, etc. The authors use their model to identify situations where agent migration has performance advantages over remote procedure calls. Analytical results are corroborated with experimentation using the Mole platform [33].

A similar problem is studied by Puliafito, Riccobene, and Scarpa in [26]. The authors compare the mobile-agent, remote-evaluation and client-server models of distributed computing. To this end, they use non-Markovian Petri nets modeling, applying probability distributions to model parameters, such as request size, time for searching data in a server, processing time, size of replies and queries, code size for migrating codes, and throughput of the communication network. Petri-net analysis shows that for the scenarios examined, which are pertinent to information filtering applications, the performance advantage of mobile agents arises under certain "external" to this model factors, such as network connectivity and speed.

Mathematical modeling is used by Kotz, Jiang, Gray, Cybenko and Peterson to study a more extended mobile-agent application scenario in [20]. According to this scenario, mobile agents are used to support a data-filtering application involving many wireless clients that filter information from a large data stream arriving across a wired network from a server. The mathematical model is used to compare analytically two alternative approaches: a) The server combines and broadcasts all the data streams over the wireless channel and filtering takes place at each client site. b) Each client dispatches an agent to the server; the agent monitors and filters the data stream before sending relevant data to its corresponding client. The authors use two performance metrics for their comparison study: computation and bandwidth requirements. They conclude that the mobile agent approach trades server computation and cost for savings in network bandwidth and client computation, which is an important remark in the context of "thin" clients used in mobile-computing applications.

In addition to mathematical analysis and Petri-net modeling, there has been a range of simulation and experimental studies on mobile-agent systems. In [31], Spalink, Hartman and Gibson study the performance advantages of employing a mobile agent for conducting search within a file at the file-server's location instead of searching remotely over the network. Trace-driven simulation shows that the MA approach is advantageous when the server's CPU is not a bottleneck.

Another thread of work employs Petri-net modeling and experimentation to investigate performance properties of mobile-agent systems: For instance, General Stochastic Petri nets are used by Rana in [27] to investigate the performance properties of two agent design patterns [2], "Task" and "Interaction," and a combination thereof. The study is performed in the context of an agent-based, e-commerce application. The Petri-net models introduced are executed with a Petri-net simulator to study the performance and scalability of the underlying application. Furthermore, Petri nets are used by Rana and Stout in [28] to model the performance of multi-agent systems in a way that captures properties arising both from agent-collaboration requirements pertinent to multi-agent applications and the performance characteristics of MA systems.

Samaras, Dikaiakos, Spyrou and Liverdos in [29] propose, employ, and validate an approach to evaluate and analyze MA performance in the context of basic mobile-computing models applied in the provision of distributed database access over the Web. The proposed experimental setup is used to compare quantitatively two commercial MA platforms and to study the performance of different approaches for database access over the Web, using mobile agents. In [11,12], we extend our previous work and propose a hierarchical framework that can be used to investigate quantitatively and experimentally various aspects of mobile-agent performance. This framework is implemented as a set of benchmarks and used to study the performance and scalability of a number of commercial MA platforms. More details are given in subsequent sections. In a similar vein, Silva, Soares, Martins, Batista and Santos define and run benchmarks in [30], to evaluate the performance of some of the existing mobile agent platforms.

## 4 A Framework for Investigating MA Performance

To cope with the complexities of MA performance analysis we propose the adoption of a hierarchical approach inspired by the structure of MA-based applications. This structure is determined by:

1. The MA platform adopted to program a particular application. Mobile-agent platforms (such as [4,5,15,19,39]) are systems that provide some basic functionality supporting the mobility of objects (transportation and location services), the communication between them, security, fault-tolerance etc. Access to this functionality is granted through a platform-specific programming interface provided for the development of applications. Various MA platforms differ in terms of their functionality, programming interface and performance characteristics which are dictated by underlying implementation details.
2. The high-level abstractions representing software-design choices made by developers for a particular application. These abstractions are implemented with the programming interface of the MA platform employed.

Notably, the differences of mobile-agent platforms and the variety of application domains result to a huge space of options vis-a-vis MA-system structure. We abstract most of these options away by focusing on *Basic Elements* and *Application Kernels*.

### 4.1 Basic Elements and Application Kernels

We define as *Basic Elements* of mobile-agent platforms a set of basic abstractions that incorporate the fundamental functionalities commonly found and used in MA platforms. For the objectives of our work, the Basic Elements of MA platforms are identified from existing, "popular" implementations as follows [4,5,15, 19,39]:

- *Agents*, defined by their state, implementation (bytecode), capability of interaction with other agents/programs (interface), and a unique identifier.
- *Places*, representing the environment in which agents are created and executed. A place is characterized by the virtual machine executing the agent's bytecode (the *engine*), its network address (location), its computing resources, and any services it may host (e.g., a database gateway or a Web-search program).
- *Behaviors* of agents within and between places, which correspond to the basic functionalities of a MA platform: creating an agent at a local or remote place, dispatching an agent from one place to another, receiving an agent that arrives at some place, communicating information between agents via messages, multicasts, or messenger agents, synchronizing the processing of two agents, etc.

Basic elements of MA systems are combined into scenarios of MA-use, which we call *Application Kernels*. Application Kernels define solutions common to various problems of agent design and are defined in terms of places participating in a scenario, agents placed at or moving between these places, and interactions of agents and places (agent movements, communication, synchronization, resource use). Application Kernels correspond to widely applicable models of distributed computation on particular application domains [32], and represent widely accepted and portable approaches for addressing typical agent-design problems [2]. Typically, Application Kernels are the building blocks of larger MA applications.

Consequently, we define Application Kernels that correspond to the Client-Server (**C/S**) model of distributed computing and its extensions for mobile computing: the Client-Agent-Server model (**C/A/S**), the Client-Intercept-Server model (**C/I/S**), the Proxy-Server model, and variations thereof that use mobile agents for communication between the client and the server (**C/S-MA**, **C/A/S-MA**, **C/I/S-MA**; see Figures 1 and 2). More details on these models are given in [29,32].

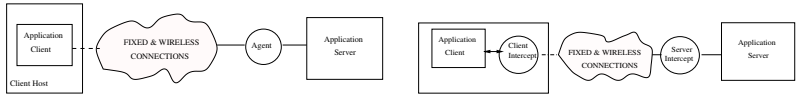



**Fig. 1.** The Client-Agent-Server and Client-Intercept-Server Models.

Additional Application Kernels correspond to the *Forwarding* and the *Meeting* agent-design patterns, defined in [2,8]. The *Forwarding* pattern "allows a given host to mechanically forward all or specific agents to another host" [2]. The *Meeting* pattern provides a way for two or more agents to initiate local interaction at a given host (see Figure 2) [2,37]. We chose the *Forwarding* and *Meeting* patterns because they can help us quantify the performance traits of agents and places in terms of their capability to re-route agents and to host inter-agent interactions.

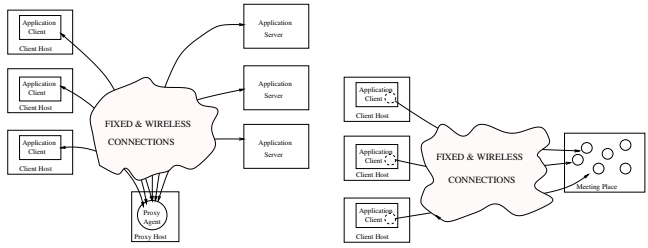



**Fig. 2.** The Proxy-Server Model and the Meeting Pattern.

### 4.2 A Hierarchical Performance Analysis Approach

To analyze the performance of mobile-agent applications, we need first to develop an approach for capturing basic performance properties of MA platforms. These properties must be defined independently of how particular mobile-agent API's are used to program and deploy applications and systems on Internet. To this end, our approach focuses on Basic Elements of MA platforms and seeks to expose the performance behavior of these functionalities: how fast they are, what is their overhead, if they become a performance bottleneck when used extensively, etc.

Having isolated the performance characteristics of basic MA elements, we focus on performance traits of Application Kernels in order to explain the performance behavior of full-blown applications, which use these kernels as building blocks. Performance traits of an Application Kernel depend on the characteristics of its constituent (basic) elements, and on how these elements are combined together and influence each other. To identify how a kernel affects application performance, we have first to isolate its basic performance properties, that is metrics capturing its performance capacity, overheads incurred by the interaction of its constituent elements, bottlenecks affecting kernel-performance, etc. For example, an Application Kernel could involve an agent residing at a place on a fixed network and providing database-connectivity services to agents arriving from remote places over wireless connections. This kernel may exist within a large digital library or e-commerce application. It may, as well, belong to the

"critical path" that determines end-to-end performance of that application. To identify how this kernel affects overall performance, we have to find out what is the overhead of transporting an agent from a remote place to a database-enabled place, connecting to the database, performing a simple query, and returning the results over a wireless connection. Interaction with the database is kept minimal because we are trying to capture the overhead of this kernel and not to investigate database behavior.

Performance analysis of Application Kernels involves the use of simple workloads seeking to discover fundamental performance properties. It is very interesting, however, to explore the performance behavior of instances of these kernels under conditions expected to occur in a real execution of a full-blown application. To this end, we can enrich the scenarios implemented by Application Kernels by extending the functionality of mobile agents and by simulating realistic workload conditions.

In view of the above remarks we propose a framework for the Hierarchical Analysis of MA-performance. Our framework consists of four layers of abstraction:

1. At a first layer, it explores and characterizes performance traits of Basic Elements of MA platforms.
2. At a second layer, it investigates implementations for popular Application Kernels upon simple workloads.
3. At a third layer, it studies Micro-Applications, that is, implementations of application kernels which realize particular functionalities of interest (e.g., database connectivity), running on realistic workloads.
4. Last but not least, at a fourth layer, our framework studies full-blown Applications running under real conditions and workloads.

Our approach has to be accompanied by proper metrics, which may differ from layer to layer, and parameters representing the particular context of each study, i.e., the processing and communication resources available and the workload applied. It should be stressed that the design of our performance analyses in each layer of our conceptual hierarchy should provide measurements and observations that can help us establish causality relationships between the conclusions from one layer of abstraction to the observations at the next layer in our performance analysis hierarchy.

### 4.3 Parameterizing Basic Elements and Application Kernels

To proceed with performance experiments, measurements and analyses, after the identification of Basic Elements and Application Kernels, we need to specify the *parameters* that define the context of our experimentation, and the *metrics* measured. Parameters determine the *workload* that drives a particular experiment, expressed as the number of invocations of some Basic Element or Application Kernel; large numbers of invocations correspond to intensive use of the element or kernel during periods of high load. Furthermore, the *resources* attached to

participating places and agents: the channels connecting places, the operating system and hardware resources of each place, and the functionality of agents and places.

The exact definition of parameters and parameter-values depend on the particular aspects under investigation. For example, to capture the intrinsic performance properties of Basic Elements, we consider agents with limited functionality and interface, which carry the minimum amount of code and data needed for their basic behaviors. These agents run within places, which are free of additional processing load from other applications. Places may correspond either to agent servers with full agent-handling functionality or to agent-enabled applets. The latter option addresses situations where agents interact with client-applications, which can be downloaded and executed in a Web browser. Participating places may belong to the same local-area network, to different local-area networks within a wide-area network, or to partly-wireless networks. Different operating systems can be considered.

Parameters become more complicated when studying application kernels. For instance, when exploring Client-Server types of models, we have to define the resources to be incorporated at the place which corresponds to the server-side of the model. Resources could range from a minimalistic program acknowledging the receipt of an incoming request, to a server with full database capabilities.

## 5 Implementation and Experimentation with Benchmarks

The hierarchical performance analysis framework presented in the previous section can be applied to study performance characteristics of different MA platforms and investigate MA-based applications. To this end, we propose three layers of benchmarks that correspond to the first three layers of the hierarchy presented in the previous Section. These benchmarks are defined as follows:

- **Micro-benchmarks:** short loops designed to isolate and measure performance properties of basic behaviors of MA systems, for typical system configurations.
- **Micro-kernels:** short, synthetic codes designed to measure and investigate the properties of Application Kernels, for typical applications and system configurations.
- **Micro-Applications:** instantiations of Application Kernels for real applications. Here, we involve places with full application functionality and employ realistic workloads complying to the *TPC-W* specification [36].

### 5.1 Experimentation

For our experiments, we focus on a set of micro-benchmarks, and on micro-kernels corresponding to database access over the Web. We implement these benchmarks with two commercial MA platforms: IBM's Aglets and Mitsubishi's

Concordia. Before presenting experimental results, we describe the basic characteristics of these two platforms. Further experimental results and comparisons with other MA platforms can be found in [12,29].

The **Aglets Software Development Kit (ASDK)** is an environment for programming mobile Internet agents (Aglets) in the Java programming language [16]. An Aglet is a Java object that has the ability to move (be dispatched) autonomously from one computer host to another. This transportation is possible between hosts with a pre-installed Tahiti server (a "place" in our terminology), which is an Aglet server program implemented in Java. Tahiti captures arriving Aglets and provides them with an Aglet context. In this context, Aglets can run their code, communicate with other Aglets, collect local information and move to other hosts. The protocol used for the transportation of the Aglet, is the Aglet Transfer Protocol (ATP) by IBM, [22]. ATP defines four standard request methods for agent services, which are `dispatch, retract, fetch` and `message`. The Tahiti server (we use v1.0.3 with size 2.25MB) can be installed at any platform that supports Java Virtual Machine (JVM). Fiji Applet is an abstract applet class of a Java package called "Fiji Kit", which allows Aglets to be fired from applets. For a Java-enabled Web browser (like Netscape Communicator) to host and fire Aglets, and thus become a "place," two more components are required and are provided by IBM: an Aglet (fiji) plug-in allowing the browser to host Aglets, and an Aglet router that must be installed at the Web server. The Aglet router's purpose is to capture incoming Aglets and forward them to their destination.

**Concordia** is a framework for the development, execution and management of mobile agent applications written in Java [19,40]. The Concordia system is made up of several integrated components. The Concordia server (a "place" in our terminology) is the major block, inside which the various Concordia Managers reside. One of these managers is the Agent Manager, which provides the communication infrastructure that enables agents to be transmitted from and received by nodes on the network, and the management of the life-cycle of the agent. Communication in Concordia relies on the Java RMI system. One of the central features of RMI is its ability to download the bytecode of an object's class if the class is not defined in the receiver's virtual machine. To ensure security of all its transmissions, Concordia uses the SSLv3 (Secure Socket Layer) protocol to transmit agent information from one system to another. Concordia provides an abstract class called `ConcordiaApplet` that extends Java's applet class. There is no need to install the Concordia System on a client machine because `ConcordiaApplet` implements a class provided by Concordia, namely `AgentTransporter` that acts as a lightweight Concordia Server. Concordia implements inter-agent messaging through the concept of events, which are Java objects posted at the Event Manager of a Concordia Server. Concordia supports *Service Bridges*, that is, gateways between Concordia and Java resources installed locally at the host machine. An agent can call methods of a Service Bridge and get back its results. Services offered by Service Bridges can be reg-

istered with a directory service running in one or more Concordia Servers. Experiments presented below were done with version 1.1.2 of Concordia.

### 5.2 Micro-Benchmarks

As described in Section 4, computing models that use MA technology employ the following basic components: a) *mobile agents* to materialize modules of the client-server model and its variations; b) *messenger agents* as an approach for flexible communication; c) *messages* as an efficient communication and synchronization mechanism. Therefore, micro-benchmarks devised to test basic performance properties of mobile-agent platforms must focus on measuring the performance of frequently executed components, i.e., messenger agents and messages. To this end, we propose the following benchmarks that measure:

- **[AC/L]**: The overhead of creating and launching messenger agents, which is represented by the time to create and dispatch mobile agents with minimal content.
- **[MSG]**: The overhead of messaging, that is, the time to create and post messages.
- **[ROAM]**: The overhead of agent traveling, which is represented by the time it takes an agent with minimal content to return to its host node after roaming along a given itinerary of hosts. The agent has minimal interaction with the resources of each host visited, e.g., it just queries the host's identification.
- **[SYNC]**: The synchronization overhead, which is represented by the time to exchange a message between two hosts (equivalent to the "ping-pong" benchmark [14]).

Benchmarks are parameterized by the number of iterations they execute. We measure the *total time to completion* of each benchmark for a chosen number of iterations. In our tests, we choose iteration numbers from 1 to 1000.

**Micro-Benchmark Experiments:** For the quantitative comparison we ran several tests implementing the micro-benchmarks presented above. In our tests, we examine two scenarios: the first scenario presumes the installation of the full agent-execution environment on the hosts of the system under scrutiny. The second scenario tests the case where the client has limited computing resources, as in the case of mobile-computing units, or connects from a machine with minimal configuration, i.e., Internet connectivity and a Java-enabled Web browser. In the second scenario, the client is communicating with the mobile-agent platform by downloading an applet enhanced with agent-handling capabilities (Fiji or Concordia applet).

**Test 1** corresponds to benchmark **[AC/L]**. For this test we launch and dispatch agents from a parked agent to a remote Agent Server. We measure the time it takes to create and launch the agents. For both platforms, we employ agents (Aglets and Concordia agents) of identical, minimal, functionality. The size of the Aglet is 1.64 KB whereas the size of the Concordia agent used is 693 bytes. The parked Aglet is 3.54 KB and the parked Concordia Agent is 1.65 KB.

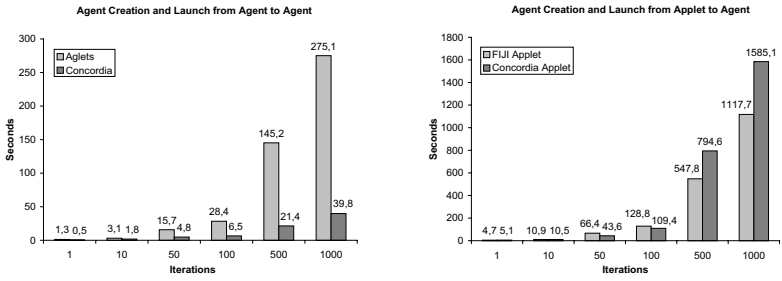



**Fig. 3.** [AC/L] Benchmark; results from Tests 1 and 2.

**Test 2** corresponds to benchmark [**AC/L**] under the second scenario, where we use an applet to launch agents. The Fiji Applet is 4.8 KB in size and the Concordia Applet is 3.61 KB.
**Test 3** corresponds to benchmark [**MSG**]. A parked Aglet/Concordia agent creates and sends/posts messages/events to a remote Aglet or Event Manager respectively. The message/event carries a simple piece of information - an integer expressing its ID. The size of the message is 5 KB whereas the size of the event is only 286 bytes.
**Test 4** corresponds to benchmark [**MSG**] following the second scenario, where messages are created and launched from an applet. The applet sizes are 5.19 KB for Fiji and 3.09 KB for Concordia.
**Test 5** corresponds to benchmark [**ROAM**] with one hop. An agent launches another agent to a remote Agent Server. At its arrival, the agent prints its id and returns back. Upon return to the sender, the agent is re-dispatched towards the same destination.
**Test 6** also corresponds to benchmark [**ROAM**] with one hop, where applets act as agent launchers.
**Test 7** corresponds to benchmark [**SYNC**], with message exchange taking place between two agents. It is a variation of Test 5 using messages: a message (or event) is sent to a remote receiver. Once this message is received, the receiver sends it back. The next message is sent after the return of the previous one.
**Test 8** corresponds to benchmark [**SYNC**] with message exchange occurring between an applet and an agent.

For the above tests we used a Pentium PC at 166MHz with 32MB of RAM running Microsoft Windows 95 as the PC from where we launched the agents and the messages and a Pentium Pro at 350 MHz with 64MB of RAM running Microsoft Windows 95. These computers were connected on a 10 Mbps Ethernet LAN.

**Discussion:** Our micro-benchmark tests provide useful insights into three important aspects of Aglets and Concordia performance: mobile agent dispatch from agent servers, mobile agent dispatch from applets, and messaging.

In particular, results from Tests 1 and 5 show that Concordia performs substantially better than Aglets in agent dispatching from agent servers (see the

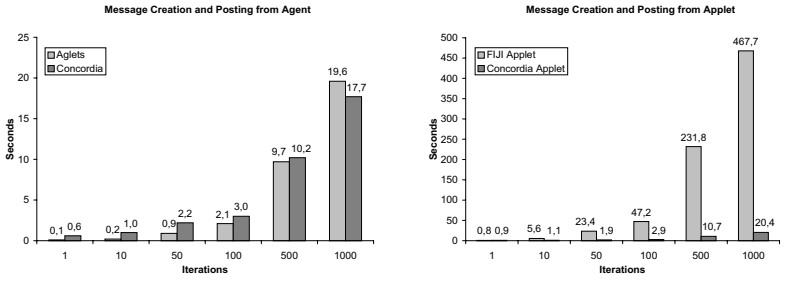



**Fig. 4.** [MSG] Benchmark; results from Tests 3 and 4.

diagrams on the left side of Figures 3 and 5). This is attributed to the fact that, when transporting an agent, Concordia dispatches an image of it. This image retracts its classes from the sender, on a need-to-use basis. In contrast, Aglets Workbench dispatches an Aglet together with all the objects reachable from this Aglet. Furthermore, it is plausible that additional overhead is incurred by Aglets due to the extra level of indirection introduced by their callback-based transport model.

Concordia performs worse than Aglets, however, when dispatching agents from an applet. This can be attested from Test 2 (Figure 3, right) and from a comparison between the two diagrams of Figure 5; note that in Test 6 the performance difference between Concordia and Aglets is smaller than in Test 5. This is probably due to the fact that the current handling of applets by the Aglets Workbench (through IBM's Fiji Applet and plug-in) is more optimized than the work-around we did to launch and receive Concordia agents from applets. This workaround requires manual installation of Concordia/application classes on the client-machine's local disk. It should be noted that, across both platforms, dispatching agents from applets performs substantially worse than dispatching agents from agent servers.

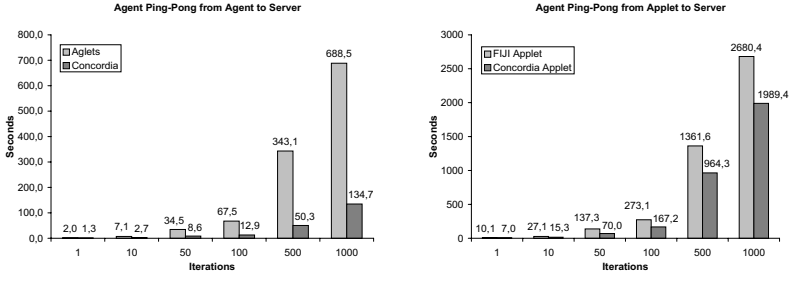



**Fig. 5.** [ROAM] Benchmark; results from Tests 5 and 6.

Results from Test 3 (Figure 4) show that Aglets outperform Concordia in message creation and posting. Concordia, however, performs better in the case of 1000 iterations. We believe this is an artifact of Tahiti-server performance behavior, which handles the transmission of incoming and outgoing messages. Further tests have shown that Tahiti saturates under heavy load. Concordia, on the other hand, separates the handling of messages (events) from the handling of agents and, apparently, Concordia's Event Manager is better optimized to sustain higher messaging loads than Tahiti. Turning to Test 7 (Figure 6), we observe a significant degradation of Aglets performance under the [SYNC] benchmark, with an increasing number of benchmark iterations. We believe this degradation is related to the implementation of message reply under Aglets, and with the erratic performance of the Tahiti server under heavier loads.

Interestingly, Concordia outperforms Aglets in message transmission from an applet, and in message exchange between an applet and an agent (see Tests 4 and 8, Figures 4 and 6). In our experiments, the use of applets under the Aglets platform has an additional overhead factor, which is due to security limitations: for a FijiApplet to communicate with an agent server (Tahiti), it has to make an extra hop and go through the Web server, where from the applet was downloaded to the client machine. This is not the case with the Concordia work-around we employed to dispatch and receive agents from an applet.

Another interesting observation from micro-benchmark measurements is that, in contrast to Aglets, Concordia performance scales impressively well as we increase the number of benchmark iterations. This is attributed to the way Concordia handles agent transportation by creating and maintaining a persistent image of an agent before dispatching it to another machine. Hence, Concordia avoids class loading on subsequent transfers, whereas Aglets must continuously load the needed classes on every Aglet transfer.

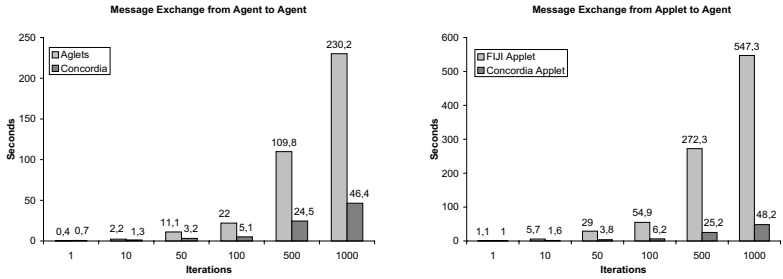



**Fig. 6.** [SYNC] Benchmark; results from Tests 7 and 8.

### 5.3 Web Database Micro-Kernels

In this chapter, we focus on micro-kernels that use mobile agents to provide database access over the Web and correspond to the computational models pre-

**Table 1.** Micro-kernels materialized with Mobile Agents.

| Computing Model | Kernel | Comments |
|---|---|---|
| C/S-MA | **Framework 1** | *Baseline* |
| C/A/S-MA | **Framework 2** | *Using messenger agents* |
| C/A/S | **Framework 3** | *Using messages* |
| C/A/S-MA-CSB | **Framework 4** | *Using Service Bridges and messenger agents* |
| C/A/S-CSB | **Framework 5** | *Using Service Bridges and messages* |

sented earlier. We propose a *micro-kernel* consisting of a short transaction (three queries) between a client and a remote database. The queries select all entries of a small student database. We measure the time required to launch an agent (or a message) from the client site, the time to carry this agent to the database server, the time to connect to the database and execute the query, and the time to bring the results back to the client. We measure the time to query the remote database for the first time and for any subsequent request. We expect these two measurements to be different, as the time of the first query includes the connection to the database and the downloading of the JDBC drivers from the client or its surrogates.

The kernel is implemented with mobile agents following the computing models **C/A/S** and **C/A/S-MA**. We also implement the kernel according to the "mobile" client-server model (**C/S-MA**). Concordia Service Bridges represent an alternative non-dynamic way to materialize the server-side of the C/A/S and C/A/S-MA models. Therefore, we implement our Web-database kernel with Service Bridges, adding another two frameworks in our benchmark suite. Table 1 summarizes the application kernels employed in our tests and the notation we use for them subsequently.

**Tests:** We ran tests to evaluate the five frameworks presented in Table 1. Details about the tests are given below. The measurements are presented in Figure 7.

**Framework 1** implements the C/S-MA model for Web-database connectivity. The client-side is implemented as an applet, which is downloaded by a client machine with a Java-enabled Web browser. Communication between the client and the server is done through the DBMS-agent launched by the applet. Upon arrival to the server, the DBMS-agent downloads the appropriate JDBC driver and connects to the database. Subsequently, it carries client queries and query results between the client and the remote database.

**Framework 2** implements the C/A/S-MA model. The client-side is again implemented as an applet, which is downloaded to a client machine with a Java-enabled Web browser. The applet launches two agents to the database server. One of these agents "parks" to the server and is responsible for downloading the necessary JDBC driver, connecting to the database and querying it. The other agent is the messenger that undertakes the responsibility of transferring the results to the client and the new client requests to the "parked" agent. The

"parked" agent is transported and connected to the database server only for the first query.

**Framework 3** implements the C/A/S model. Implementation is similar to Framework 2 except for the communication between the client and the database server, which employs messages instead of agents.

**Framework 4** implements the C/A/S-MA-CSB model: we created a Concordia Service Bridge that performs the Web access on behalf of an incoming agent. The incoming agent carries the SQL statement from the applet client to the Service Bridge, and returns the results back to the client.

**Framework 5**: This framework uses events for the communication between the applet and the Service Bridge. Both the applet and the Service Bridge are connected to the Event Manager of the Concordia Server at the database machine and they exchange Access Request events and Access Results events.

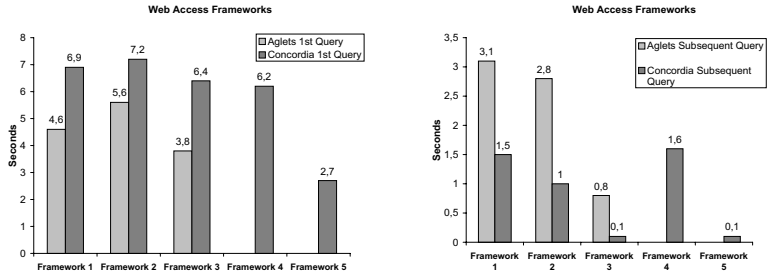



**Fig. 7.** Performance of Web-database kernel.

**Discussion:** Figure 7 presents our measurements from the frameworks presented above. A first remark is that the Aglets Workbench outperforms Concordia for the first query, as shown in the left diagram of Figure 7. This observation is consistent with the results of Test 2, and with our remarks on applet-performance under Concordia (Figure 3, right diagram). Concordia, however, outperforms Aglets in subsequent queries. For Frameworks 1 and 2, this observation is consistent with our results from Test 6 (see Figure 5, right diagram). In the case of Framework 3, the improvement of Concordia's performance over Aglets agrees with our [SYNC] micro-benchmark (see Figure 6).

The better performance displayed by Framework 1 with respect to Framework 2 for the first query, is due to the extra overhead incurred by the dispatch of a second agent under Framework 2. This agent parks at the server and results in the improved performance displayed by subsequent queries under Framework 2 over Framework 1.

It is also interesting to point out that the small performance difference between Frameworks 1 and 3 for the first query, is due to the more limited functionality of the agent sent to park at the server-side under Framework 3 (C/A/S model). This agent is "lighter" than the agent dispatched under Framework 1

(C/S-MA model), as it does not have to cope with dispatching itself back and forth to the client. Another interesting point is the performance benefit of using messages over messenger agents. This is expected, though, as messenger agents are "heavier" objects than messages, both in Aglets and Concordia. Messenger agents, however, offer greater flexibility as they can roam the network collecting more information before reaching their destination. Finally, Concordia's Service Bridges represent a very efficient approach for providing services to incoming agents at the server side. This approach, however, lacks the flexibility of dynamically "parking" a mobile agent at the server-side, at run-time, and having this agent negotiate with the server the services it will provide to incoming agents.

## 6 Conclusions

In this chapter we presented a short overview of various approaches for performance evaluation and analysis of mobile-agent applications and systems. Then, we described a quantitative performance analysis methodology that we developed to investigate performance properties of MA platforms and systems. We implemented this methodology with a hierarchy of benchmarks (micro-benchmarks and micro-kernels), which we ran on top of two popular Java-based mobile-agent platforms, IBM's Aglets and Mitsubishi's Concordia Results from micro-benchmark tests revealed interesting aspects of Aglets and Concordia performance, and enabled us to interpret the performance of micro-kernels. On the other hand, micro-kernel-performance measurements helped us assess the distributed computing models examined. Furthermore, these measurements confirm the validity and usefulness of the proposed micro-benchmarks. As expected, both platforms have their pros and cons, with Concordia providing better performance and robustness and Aglets offering improved flexibility. We are currently developing further benchmarks to extend our approach for assessing the performance of mobile-agent-based distributed applications.

## References

1. Internet Performance Modeling. Workshop on Internet Performance Modeling. Department of Computer Science, University of Dortmund, October 1999.
2. Y. Aridov and D. Lange. Agent Design Patterns: Elements of Agent Application Design. In *Proceedings of Autonomous Agents 1998*, pages 108–115. ACM, 1998.
3. N. Bhatti, A. Bouch, and A. Kuchinsky. Integrating user-perceived quality into Web server design. In *Proceedings of the 9th International World-Wide Web Conference*, pages 1–16. Elsevier, May 2000.
4. W. Brenner, R. Zarnekow, and H. Wittig. *Intelligent Software Agents. Foundations and Applications.* Springer, 1998.
5. M. Breugst, I. Busse, S. Covaci, and T. Magedanz. Grasshopper – A Mobile Agent Platform for IN Based Service Environments. In *Proceedings of IEEE IN Workshop 1998*, pages 279–290, Bordeaux, France, May 1998.
6. A. Carzaniga, G.P. Picco, and G. Vigna. Designing Distributed Applications with Mobile Code Paradigms. In *Proceedings of the 1997 International Conference on Software engineering*, pages 22–32, May 1997.

7. A. Castillo, M. Kawaguchi, N. Paciorek, and D. Wong. Concordia as Enabling Technology for Cooperative Information Gathering. In *Japanese Society for Artificial Intelligence Conference*, June 1998. http://www.meitca.com/HSL/Projects/Concordia/.
8. D. Lange and M. Oshima. *Programming and Deploying Java Mobile Agents with Aglets*. Addison Wesley, 1998.
9. M. Dikaiakos and D. Gunopoulos. FIGI: The Architecture of an Internet-based Financial Information Gathering Infrastructure. In *Proceedings of the International Workshop on Advanced Issues of E-Commerce and Web-based Information Systems*, pages 91–94. IEEE-Computer Society, April 1999.
10. M. Dikaiakos, A. Rogers, and K. Steiglitz. Performance Modeling through Functional Algorithm Simulation. In G. Zobrist, K. Bagchi, and K. Trivedi, editors, *Advanced Computer System Design*, chapter 3, pages 43–62. Gordon and Breach, 1998.
11. M. Dikaiakos and G. Samaras. A Performance Analysis Framework for Mobile-Agent Platforms. In *Proceedings of Workshop on Infrastructure for Scalable Mobile Agent Systems, Autonomous Agents 2000*, 2000.
12. M. Dikaiakos and G. Samaras. Quantitative Performance Analysis of Mobile-Agent Systems: A Hierarchical Approach. Technical Report TR-00-2, Department of Computer Science, University of Cyprus, June 2000.
13. M. Dikaiakos and J. Stadel. A Performance Study of Cosmological Simulation on Message-Passing and Shared-Memory Multiprocessors. In *Proceedings of the 10th ACM International Conference on Supercomputing*. ACM, May 1996.
14. J. Dongarra and W. Gentzsch, editors. *Computer Benchmarks*. North Holland, 1993.
15. G. Glass. Overview of Voyager: ObjectSpace's Product Family for State-of-the-Art Distributed Computing. Technical report, ObjectSpace, 1999.
16. J. Gosling and H. McGilton. *The Java Language Environment. A White Paper*. Sun Microsystems. http://java.sun.com/docs/white/index.html.
17. W. Meira Jr., T.J. LeBlanc, and V. Almeida. Using the Cause-Effect Analysis to Understand the Performance of Distributed Programs. In *Proceedings of Symposium on Parallel and Distributed Tools*, pages 101–111. ACM, 1998.
18. N.M. Karnik and A.R. Tripathi. Design Issues in Mobile-Agent Programming Systems. *IEEE Concurrency*, 6(3):52–61, July-September 1998.
19. R. Koblick. Concordia. *Communications of the ACM*, 42(3):96–99, March 1999.
20. David Kotz, Guofei Jiang, Robert Gray, George Cybenko, and Ronald A. Peterson. Performance Analysis of Mobile Agents for Filtering Data Streams on Wireless Networks. In *Proceedings of the Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM 2000)*, pages 85–94. ACM Press, August 2000.
21. D. B. Lange and M. Oshima. Seven Good Reasons for Mobile Agents. *Communications of the ACM*, 42(3):88–91, March 1999.
22. D.B. Lange and Y. Aridov. *Agent Transfer Protocol – ATP/0.1*. IBM Tokyo Research Laboratory, March 1997. http://www.trl.ibm.co.jp/aglets/.
23. T. Magedanz and A. Karmouch. Mobile Software Agents for Telecommunication Applications. *Computer Communications*, 23(8):705–707, 2000.
24. S. Papastavrou, G. Samaras, and E. Pitoura. Mobile Agents for WWW Distributed Database Access. In *Proceedings of the Fifteenth International Conference on Data Engineering*, pages 228–237. IEEE, March 1999.
25. E. Pitoura and G. Samaras. *Data Management for Mobile Computing*. Kluwer Academic Publishers, 1998.

26. A. Puliafito, S. Riccobene, and M. Scarpa. An Analytical Comparison of the Client-Server, Remote Evaluation and Mobile Agents Paradigms. In *Proceedings of the Joint Symposium ASA/MA '99. First International Symposium on Agent Systems and Applications (ASA '99). Third International Symposium on Mobile Agents (MA '99)*, pages 278–292. IEEE-Computer Society, October 1999.
27. O.F. Rana. Performance Management of Mobile Agent Systems. In *Proceedings of Autonomous Agents 2000*, pages 148–155. ACM, June 2000.
28. O.F. Rana and K. Stout. What is Scalability in Multi-Agent Systems? In *Proceedings of Autonomous Agents 2000*, pages 56–63. ACM, June 2000.
29. G. Samaras, M. Dikaiakos, C. Spyrou, and A. Liverdos. Mobile Agent Platforms for Web-Databases: A Qualitative and Quantitative Assessment. In *Proceedings of the Joint Symposium ASA/MA '99. First International Symposium on Agent Systems and Applications (ASA '99). Third International Symposium on Mobile Agents (MA '99)*, pages 50–64. IEEE-Computer Society, October 1999.
30. L.M. Silva, G. Soares, P. Martins, V. Batista, and L. Santos. Comparing the Performance of Mobile Agent Systems: a Study of Benchmarking . *Computer Communications*, 23(8):769–778, 2000.
31. T. Spalink, J. Hartman, and G. Gibson. The Effects of a Mobile agent on File Service. In *Proceedings of the Joint Symposium ASA/MA '99. First International Symposium on Agent Systems and Applications (ASA '99). Third International Symposium on Mobile Agents (MA '99)*, pages 42–49. IEEE-Computer Society, October 1999.
32. C. Spyrou, G. Samaras, E. Pitoura, and P. Evripidou. Wireless Computational Models: Mobile Agents to the Rescue. In *2nd International Workshop on Mobility in Databases & Distributed Systems. DEXA '99*, September 1999.
33. M. Strasser, J. Baumann, and F. Hohl. Mole - A Java Based Mobile Agent System. In J. Baumann, editor, *2nd ECOOP Workshop on Mobile Object Systems*, 1996.
34. M. Strasser and M. Schwehm. A Performance Model for Mobile Agent Systems. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 97)*, pages 1132–1140, June 1997.
35. D.L. Tennenhouse, J.M. Smith, W. D. Sincoskie, and G.J. Minden. Itinerant Agents for Mobile Computing. *Journal of IEEE Personal Communications*, 2(5), October 1995.
36. Transaction Processing Performance Council (TPC). *TPC Benchmark W (Web Commerce) - Draft Specification*, December 1999.
37. J. White. Telescript Technology: Mobile Agents. In J. Bradshaw, editor, *Software Agents*. MIT Press, 1997.
38. J.E. White. *General Magic White Paper*. http://www.genmagic.com/agents, 1996.
39. D. Wong, N. Paciorek, and D. Moore. Java-based Mobile Agents. *Communications of the ACM*, 42(3):92–95, March 1999.
40. D. Wong, N. Paciorek, T. Walsh, J. DiCelie, M. Young, and B. Peet. Concordia: An Infrastructure for Collaborating Mobile Agents. *Lecture Notes in Computer Science*, 1219, 1997. http://www.meitca.com/HSL/Projects/Concordia/.
41. M. Woodside. Software Performance Evaluation by Models. In C. Lindemann G. Haring and M. Reiser, editors, *Performance Evaluation: Origins and Directions*, pages 283–304. Springer, 1999.

# UML-Based Performance Modeling Framework for Component-Based Distributed Systems

Pekka Kähkipuro

SysOpen Plc, Pasilankatu 4 B, FIN-00240 Helsinki, Finland
pekka.kahkipuro@sysopen.fi

**Abstract.** We describe a performance modeling framework that can be used in the development and maintenance of component-based distributed systems, such as those based on the CORBA, EJB, and COM+ platforms. The purpose of the framework is to produce predictive performance models that can be used for obtaining performance related information on the target system at all stages of its life cycle. The framework defines a UML-based notation for describing performance models, and a set of special techniques for modeling component-based distributed systems. In addition, we present a transformation for converting the resulting models into a format that can be solved approximately for a number of relevant performance metrics.

## 1 Introduction

The primary domain for the Unified Modeling Language (UML) is functional modeling of software systems, but the language has enough expressive power for performance modeling as well [1, 2, 3]. The main advantage of using the UML for performance modeling is the possibility to create both functional and performance models in parallel with the same design tools and with the same diagrams, and thereby reduce the cognitive gap between these domains [4, 5, 6]. To represent performance related features that are not covered by the core UML, a number of standard UML extension mechanisms can be used.

UML models can be written in different ways, and not all of them are suitable for performance modeling. Therefore, we define a set of UML modeling techniques that support the construction of performance models for component-based distributed systems, such as those based on CORBA, EJB, and COM+. To cope with the complexity of distributed computing, the proposed techniques divide the overall model into a layered collection of separate but interrelated UML diagrams.

To provide practical support for software engineering, there must be a way to solve the resulting performance models for a number of relevant performance metrics. We follow a stepwise approach. As a first step, we normalize UML diagrams into a textual format and remove elements that are not relevant for performance modeling. For this purpose, we use our own textual notation, the Performance Modeling Language (PML). As a second step, PML based performance models are expanded into augmented queuing networks (AQN), where each resource instance of the system is represented explicitly. Finally, approximate techniques can be used for solving the resulting AQNs for the relevant performance metrics.

R. Dumke et al. (Eds.): Performance Engineering, LNCS 2047, pp. 167–184, 2001.

The rest of the work is structured as follows. In Section 2, we briefly describe the structure of the framework. Section 3 presents the details of the proposed modeling notation and techniques. Section 4 describes how component-based systems should be partitioned into layers, and Section 5 illustrates the framework with an example. Finally, section 6 briefly discuss the advantages and limitations of the framework, and draws a number of conclusions.

## 2 Framework Structure

The presented performance modeling framework can be divided into three elements:

- A UML-based performance modeling notation,
- A set of modeling techniques,
- Tools for solving, presenting, and analyzing the models.

The purpose of the performance modeling notation is to provide a common language for representing performance related information in software systems. We propose to use a subset of the UML notation with a few extensions for indicating performance related information in otherwise normal UML diagrams. The proposed extensions comply with the standard UML extension mechanisms.

The proposed set of performance modeling techniques provides means for creating precise performance models for complex distributed systems using various component-based distributed platforms. In particular, the techniques allow application level issues to be separated from the component infrastructure and the network through a layered structure. Our techniques are close to the normal UML modeling style as presented in the UML standard [7] and literature (e.g. [8, 9, 2, 10]). Therefore, they allow existing functional UML models to be extended into performance models.

Our set of analysis tools provides the necessary technical machinery for using the framework in a real context. As a minimal approach, we propose an algorithm for transforming UML based performance models into a solvable format. This way, it is possible to produce approximate solutions for the performance models, and to convert the solutions into a set of relevant performance metrics to be used in a performance modeling methodology. A full toolkit could have additional capabilities for graphical modeling and for visualizing the metrics obtained for the models.

The technical aspects of the framework can be described in terms of three performance model representations. The framework defines mappings between the representations, as shown in Fig. 1. The idea is to start from the UML representation and proceed downward using the mappings. Once the bottom has been reached, an approximate solution can be found for the model. The mappings also indicate how the obtained metrics can be propagated upwards.

The *UML representation* describes the system with UML diagrams. This representation may contain purely functional elements that are not needed for performance modeling.

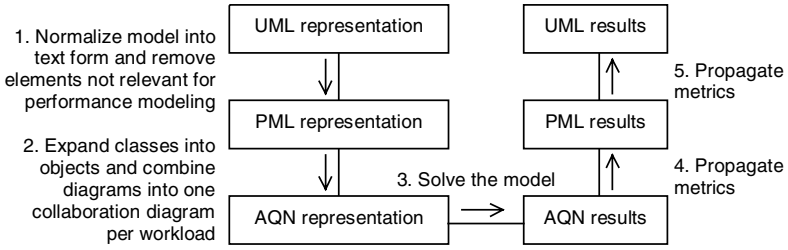



**Fig. 1.** Three performance model representations

The *PML representation* (Performance Modeling Language) provides an accurate textual notation for representing performance related elements in the UML diagrams. The PML representation has the same layered structure as the UML representation, and the mapping from UML to PML is straightforward. The purpose of this representation is to filter out those features that have no significance for performance modeling, such as graphical UML variations. Moreover, the PML representation has an important role in the development of the framework, as it is currently the input format for the prototype implementation of the modeling and analysis tool. However, the use of PML is not mandatory, and any other human-understandable UML representation could be used instead. For example, we might later opt for the human-usable textual UML notation that is currently being developed by the OMG.

The *AQN representation* describes the target system as an augmented queuing network. The AQN representation contains features that are not allowed in classical product-form queuing networks, such as simultaneous resource possessions, synchronous resource invocations, and recursive accesses to resources [11]. However, there are ways to find approximate solutions for the AQN representation. A special algorithm has been proposed and analyzed in [11]. In addition, simulation can be applied to the models. The AQN representation is fairly close to several layered queuing network models (e.g. [12, 13] ) and, therefore, it might also be possible to use related algorithms for solving the AQN representation. The AQN representation is obtained from the PML representation by expanding object classes into object instances that correspond to individual resources in the system. Moreover, the UML collaboration and sequence diagrams describing the behavior of the application and the infrastructure are combined into one or more workload specifications.

Similar tiered architectures are common in performance modeling, but the top representation is often a sophisticated performance modeling notation, such as a variant of stochastic Petri nets, to support advanced modeling techniques [14]. In our case, a non-technical top representation is used for hiding most of the underlying performance modeling issues and for making the models closer to the functional models used by software engineers. See [15] for a similar but less complete approach.

This framework is a generalization of the more technical framework presented in [11] and [16]. In this work, the focus is on general-purpose UML-based performance modeling while the earlier results concentrate on a single distribution technology (CORBA) and on a single algorithm for solving the models.

# 3 UML-Based Modeling Techniques

We now proceed to the modeling techniques that are specific to our framework. It is assumed that the reader is familiar with the basic UML notation. See [7, 8, 10] for discussions. For each UML diagram, we also indicate the corresponding PML representation. See [11] for a more complete presentation of the PML.

## 3.1 Resource Representation

Software and hardware resources are represented with UML classes. To distinguish these *resource classes* from other classes, we mark them with the *queue* or *delay* properties. The *queue* property denotes a queuing resource, such as a single-threaded software server or a physical device like a CPU or a hard disk. The *delay* property denotes a delay resource, such as think time, network delay, or a multi-threaded software server that does not impose queuing for its clients.

A request to an operation in a resource class indicates an access to that resource. In addition, a request to an operation in a non-resource class that is contained within some resource class is also considered as an access to that resource. Service demands for these operations can be defined explicitly with user-named properties. An operation may have a single property for the total service demand (e.g. $d = 10$), or it may have multiple properties to indicate the individual elements of the service demand. For example, there may be separate properties for the disk, CPU, and network adapter. Service demands are given in time units. For software resources, service demand indicates the measured or estimated time to execute the operation using whatever hardware resources necessary. For hardware resources, service demand indicates the actual time during which the corresponding device performs some work.

The model designer can freely select the symbols for service demands. If a symbol is not used in any triggering condition and if it is not explicitly bound to a lower-level resource, the service demand represented by the symbol is directly associated with the resource to which the property is attached. In our examples, we commonly use the following symbols: *d* denotes the measured or estimated service demand that is usually bound directly to the associated resource, *cpu* denotes service demand to be bound to the CPU resource, *disk* denotes service demand to be bound to the hard disk, and *adapter* denotes the service demand to be bound to the network adapter.

```
class CDatabase {
  property queue;
  Read() {cpu=10,disk=50};
  Status() {cpu=5};
  Write() {cpu=10,disk=80};
};
```

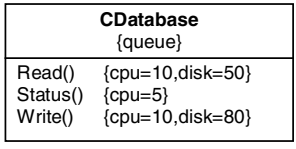



**Fig. 2.** A queuing resource with three operations

Accurate service demands are difficult to obtain without measuring real systems. Therefore, estimates are often used at early stages of development to produce approximate performance models. Service demands are not mandatory in class diagrams since they can be also defined in the workload descriptions. Fig. 2 illustrates a re-

source class *CDatabase* with three operations. The service demands of the operations are divided into CPU and disk components.

```
interface IDatabase {
  Read() {cpu=10,disk=50};
  Status() {cpu=5};
  Write() {cpu=10,disk=80};
};

class CDatabase {
  property queue;
  realizes IDatabase;
};
```

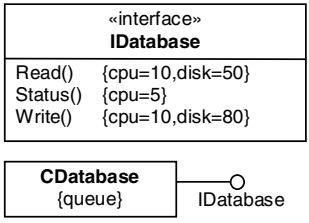



**Fig. 3.** An example interface

Service demands can also be specified for interfaces. Fig. 3 shows how the *CDatabase* resource can be specified with an explicit interface. Resource classes can be used like any other UML classes and mixed freely with non-resource classes. To simplify the instantiation of resources, we assume the existence of two predefined resource classes, *Queue* and *Delay*, with the properties *queue* and *delay* respectively.

### 3.2 Workload Representation

Performance workloads are modeled with collaboration diagrams (*workload diagrams*). For closed workloads, the number of jobs is indicated with the *population* property. For open workloads, the arrival rate of jobs is indicated with the *arrivalrate* property. Depending on the algorithm used for solving the performance model, different requirements can be set for the arrival processes. For example, if we want to comply with the requirements for classical BCMP networks and to ensure more accurate results with approximate algorithms, only Poisson arrival processes should be allowed for open workloads (see [11] for a detailed discussion). All non-Poisson arrival processes should have the *distribution* property to indicate the actual distribution. Fig. 4 illustrates a simple workload diagram where 100 users are accessing a database through a front-end application.

Service demands need not be specified explicitly in workload diagrams if they are given in class or interface specifications. Service demands in workload diagrams override those given elsewhere. It is possible to use anonymous invocations that only indicate the target resource but do not name the operation. They always require explicit service demand. Again, if the BCMP requirements are not met, the service time distribution should be indicated explicitly so that it can be taken into account when solving the model. Otherwise, we assume that the BCMP requirements are met. The example in Fig. 4 models the think time of the end user with an anonymous access to the *User* delay resource.

```
collaboration MainLoad {
  property population=100;
  1: CFront.Start() {cpu=10};
  1.1: CDatabase.Read();
  1.2: CDatabase.Write();
  2: User() {d=500};
};

actor User { property delay; };
class CFront { property queue; };
```

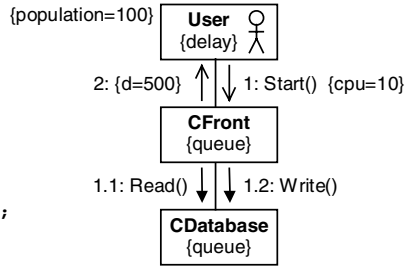



**Fig. 4.** An example workload diagram

The *thinktime* property can be used to represent think time or any similar delay when there is no need to explicitly present the corresponding delay resource (e.g. the end user). For instance, the example in Fig. 4 can be specified in a more concise form by omitting the *User* class together with the anonymous invocation to it, and by adding the *thinktime* property to the diagram.

Any class can be marked with the *singleuser* property. This Boolean property affects the class itself and all resources that it contains. An access to a *singleuser* resource is considered to have no contention. As a result, its service demand is simply added to the value of the *thinktime* property. Effectively, resources that are marked with the *singleuser* property are excluded from the underlying AQN model except for the increase in think time. This is typically done for user workstations and other similar resources that are dedicated to a single job and do not impose queuing. In some cases, however, it may be preferable to simplify the model by leaving out such resources and by increasing the *thinktime* property explicitly.

For conditional and iterative messages in collaboration diagrams, we specify a performance-oriented shorthand notation. Instead of giving a guard condition, we write down explicitly the execution probability for the message. This way, expressions like

```
[x > 0] 1: Operation() {probability = 0.7}
1.1: *[i = 1..10]: Operation()
```

can be shortened to

```
[0.7] 1: Operation()
[10] 1.1: Operation()
```

in performance models. This notation can be used in sequence and collaboration diagrams, and it is the only way to express conditional execution and iteration in PML. It may sometimes be difficult to estimate the correct execution probabilities and iteration counts since both can be data dependent (see [17] for a discussion and examples).

### 3.3 Triggering Properties

Triggering properties represent side effects of the application behavior in the infrastructure and in the network. These effects are normally excluded from application-level workload diagrams to keep them readable. Examples include network delays

and context switching delays during inter-process communication. Any UML class may specify up to nine triggering properties, as listed in Table 1.

**Table 1.** Triggering properties

| Property | Triggering condition |
|---|---|
| requestin | Request from an outside class |
| replyin | Reply sent by an outside class |
| msgin | Any incoming message (i.e. requestin or replyin) |
| requestout | Request to an outside class |
| replyout | Reply to an outside class |
| msgout | Any outgoing message (i.e. requestout or replyout) |
| requestpeer | Request sent between first-level contained classes |
| replypeer | Reply sent between first-level contained classes |
| msgpeer | Any internal message (i.e. requestpeer or replypeer) |

The value of a triggering property is a reference to a collaboration diagram (*triggering diagram*) describing the actions that take place when the triggering condition is satisfied. We use UML package names to indicate these references, but it is possible to use other referencing mechanisms, such as dependency stereotypes. Fig. 5 illustrates a process with a constant context switch delay when a request or a reply arrives from another process. Requests that are sent and received within the same process do not satisfy the *requestin* condition and do not entail any delay.

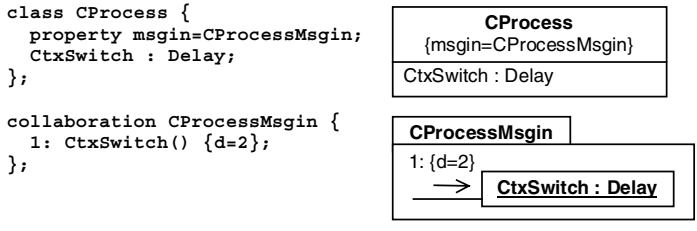



**Fig. 5.** A process with a constant context switch delay for incoming messages

A triggering diagram can use the service demand values of the operation that satisfies the triggering condition. Suppose, for example, that each operation contains an *adapter* property representing the use of the network adapter. Fig. 6 illustrates a node where the *adapter* property is mapped to the *Adapter* device. Service demands in triggering diagrams can be adjusted with arithmetic expressions. For instance, a fast network adapter might be modeled with the property $d = adapter * 0.9$. These and similar arithmetic expressions can refer to properties that have been specified for any modeling element. For example, if the *CNode* class has an *AdapterRate* property for indicating the relative rate of the adapter, the service demand of the network adapter could be defined with the expression $d = adapter * CNode.prop.AdapterRate$.

```
class CNode {
  property requestin=CNodeRequestin;
  Adapter : Queue;
};

collaboration CnodeRequestin {
  1: Adapter() {d=adapter};
};
```

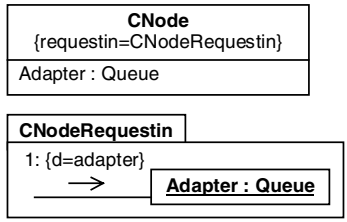



**Fig. 6.** The service demand of the network adapter depends on the *adapter* property

### 3.4 Service Demand Binding

If a service demand property is not used in any triggering condition, the service demand is associated directly with the object that receives the invocation. In some cases, however, the service demand should be bound to some other resource. For example, if there is a CPU resource in each node, the service demands of software server requests should be bound to the CPU resource in order to model CPU contention.

Service demand binding is implemented by declaring a *binding resource,* such as a CPU, with a *binding property*. The binding resource is often placed inside a container class, such as a node, to indicate the scope of the binding. The binding property indicates the name of the service demand property it binds. Fig. 7 illustrates how the *cpu* and *disk* properties can be bound to model CPU and disk contention. Arithmetic expressions can be used in binding properties. For example, the binding expression $d = cpu * 1.5$ could be used to model a slow CPU.

```
class CNode {
  Cpu : Queue {d=cpu};
  Disk : Queue {d=disk};
};
```

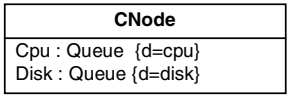



**Fig. 7.** Service demand binding

Nested service demand binding is allowed. For example, a disk driver resource could bind all application-level service demands for file access. The service demand of this resource might then be bound to the actual hard disk. It is also possible to bind the same service demand more than once. For example, if a node has two disks and data accesses are equally divided between them, this might be modeled by splitting the service demand of each disk access with the binding property $d = disk * 0.5$ for both disks.

### 3.5 Network Connections

In UML deployment diagrams, a connection between nodes may be refined by a stereotype for identifying the communication protocol or the network medium (e.g. «*TCP/IP*» or «*Ethernet*»). We propose an extension to the UML for specifying the

details of such association refinements. Essentially, we need the same triggering properties that are available for UML classes. Hence, we define a class stereotype «*connection*» for specifying association stereotypes for network connections. Usually, connection classes contain a delay resource for representing network delays and queuing resources for modeling physical devices, such as modems or routers. These resources are typically accessed only from triggering diagrams. In other words, network access is typically a side effect of some other behavior in the system.

Fig. 8 shows a connection with a constant delay for each message that passes between two nodes in a «*LAN*». Fig. 9 shows how this connection can be used in a deployment diagram.

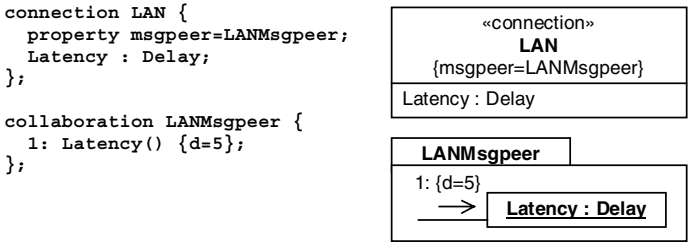



**Fig. 8.** An example definition for a network connection

### 3.6 Run-Time Configuration

A description of the run-time configuration is needed for obtaining performance metrics for the modeled system. This can be carried out with deployment diagrams containing elements for the relevant physical entities, such as nodes, networks, and processes. Application objects are instantiated to their appropriate locations in the configuration. Fig. 9 illustrates a deployment diagram of a simple system that contains two nodes attached to a LAN.

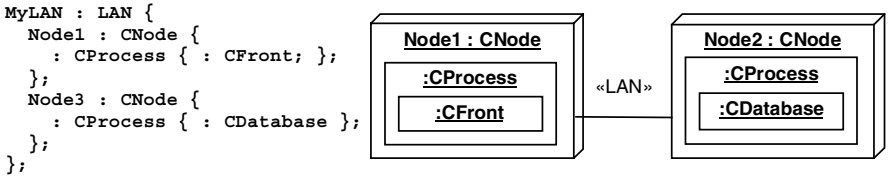



**Fig. 9.** An example deployment diagram

If the full run-time environment is not known, the configuration can be described with simple object diagrams. Still, it may be possible to obtain useful metrics for the model. The full configuration of complex systems may be specified with several deployment diagrams. For example, there may be separate diagrams for each node, and an additional diagram for the network connections between the nodes.

### 3.7 Creating the AQN Representation

So far, we have discussed how performance models can be represented in terms of UML modeling elements and diagrams. We now describe how such models can be transformed into augmented queuing networks. Essentially, an AQN is a queuing network where both synchronous calls and asynchronous message passing is allowed. See [11] for a definition for an AQN and for an example of the transformation.

**Classes.** The transformation of UML classes is straightforward. Each instance of a resource class is mapped to an AQN resource. All other classes disappear during the transformation. The actual number of instances for a particular resource class is determined from the object and deployment diagrams. An additional *thinktime* delay resource is added to the AQN.

**Workloads.** Each workload diagram is transformed into a class of jobs in the resulting AQN. The mapping is implemented in four steps. First, the *thinktime* property is mapped to an access to the *thinktime* resource. Second, the service demands for all invocations are determined from the workload, class, and interface specifications. The values in workload diagrams override those given elsewhere. If the target of an invocation is a non-resource class, the service demand is attributed to the closest surrounding resource class. Third, all accesses to *singleuser* resources are handled by increasing the service demand of the *thinktime* resource appropriately.

The fourth step during workload transformation is the expansion of class invocations into one or more accesses to resource objects. We define a special *class resolution algorithm.* The algorithm is based on the use of resolution contexts, i.e. packages and composite objects. For the first invocation in a workload diagram, the resolution context is the current package. For all subsequent invocations, resolution is first attempted in the chain of invocations that preceded the current one. If this fails, resolution is attempted in the smallest context containing the object that executed the previous invocation. If there is no match in that context, resolution is attempted in the next surrounding context, and so forth until the whole system is the context.

When one or more instances of the target class are found in a resolution context, the service demand is distributed evenly among them, and the subsequent resolution contexts are determined by these instances. Hence, a single invocation in a workload diagram may spawn any number of invocations in the AQN representation. The class resolution algorithm can be overridden by explicitly specifying a resolution context for the operation, or by using a named target object.

**Triggering properties.** If an invocation in a workload diagram satisfies a triggering condition, a sequence of additional resource accesses are copied from the corresponding triggering diagram into the workload diagram. A single message in a workload diagram may satisfy any number of triggering conditions for several objects. For example, an access to a remote object generates a message that leaves a node, passes through a network, and enters another node. As a result, three triggering diagrams might be involved in the transformation. If the message being expanded is a synchronous call, the calling object remains blocked during the additional accesses, and also the backward path is checked for triggering conditions. Otherwise, the sending resource is released immediately and asynchronous messages are generated.

**Service demand binding.** The transformation generates additional invocations for service demand bindings. After the original access, a new access is made to the binding resource. If the target resource of the original access is a queuing device, the

binding is transformed into a synchronous call. If the target resource of the original access is a delay device, an asynchronous messages is generated from the binding. Notice that the service demand of the original target resource may become zero after this transformation (i.e. all service demand properties get bound). This is typical for software resources when the performance model also includes a CPU resource.

**Conditional execution and iteration.** Conditional execution and iteration are both handled after expanding triggering properties and service demand bindings into additional resource accesses. For both cases, we require that the UML model contains a factor $f$ indicating the branching probability ($f < 1$) or the average repetition count ($f > 1$). The proposed shorthand notation gives a convenient way for expressing this factor. All service demands that are within the affected execution path are simply multiplied by $f$. A single service demand may be multiplied by several factors if there are nested conditional messages and iterations.

**Network connections.** Association stereotypes representing network connections are treated as if they were container classes for the nodes that are contained within the network. Accordingly, the transformation for network connections is the same as the mapping for classes that have triggering properties or service demand bindings.

## 4 Partitioning the Model into Layers

To cope with the complexity of component-based distributed systems, we proposed to partition performance models into the following six layers:

- Application layer,
- Interface layer,
- Behavior layer,
- Infrastructure layer,
- Network layer,
- Deployment layer.

Depending on the target environment, it may sometimes be appropriate to merge two adjacent layers or to divide a layer into multiple sublayers.

The *application layer* describes the application's static structure with UML class diagrams. Resources that are relevant for performance modeling are indicated with the *delay* or *queue* properties. In addition, if operations have been explicitly given in class diagrams, they may be equipped with service demand properties. Since the application layer requires only a few additional properties in otherwise normal class diagrams, it may be possible to maintain a single set of UML class diagrams for both functional and performance modeling.

The *interface layer* describes operations that are implemented by application layer objects. The use of an explicit interface layer reflects the principle of separating interfaces from implementations [18]. In many cases, interfaces also indicate service demands for each operation, unless these service demands have already been given at the application layer

The *behavior layer* describes the application's behavior in terms of interactions between application layer objects. The behavior layer is modeled with UML collaboration diagrams. All diagrams that represent essential workloads for the system must

comply with the framework. This means that a choice must be made between open and closed workloads. For the former, the arrival rate needs to be specified and, for the latter, the population is required. In addition, the probabilities and repetition counts for conditional execution paths and iterations must be estimated.

The *infrastructure layer* describes the infrastructure support. This may include, for example, the middleware, the operating system, and the hardware. This layer is represented with class diagrams equipped with triggering properties and corresponding collaboration diagrams. Additional links between applications and the infrastructure can be created with service demand binding. Elements at this layer, such as processes and nodes, are organized into nested classes according to their physical structure.

The *network layer* describes the performance characteristics of the underlying network. In particular, the «*connection*» stereotype allows network connections to use the same techniques that are available for the infrastructure layer

The *deployment layer* specifies the run-time configuration in terms of objects, processes, nodes, networks, etc. with deployment diagrams. This layer can be described at different levels of detail. For example, at an early stage of development, it may be useful to instantiate only the application objects. Later, when the effect of network traffic is investigated, network layer objects might be added, etc.

## 5 Example

We now illustrate the framework with an example. It shows how distributed applications can be modeled, and how a component infrastructure can be specified with the framework. It is not our goal to create a complete model for the target system.

The target system of our example is a reduced version of the TPC Web Commerce Benchmark revision D-5.0 (TPC-W) as described in [19]. We have slightly changed the original design that assumes HTTP communication between client and server nodes. In our approach, communication is implemented with the CORBA infrastructure, but the functionality is kept as close as possible to the original design. This way, we can retain the same primary metric of interest: system throughput in terms of end user interactions per second with a fixed number of clients.

To keep the example small, we omit the transaction processing part of the functionality and assume zero service demands for the operations that model the application functionality. As a result, the model concentrates on the communication part of the system and on the activities that take place within the infrastructure. It is possible to extend the model with transaction processing and database management by using traditional performance modeling techniques, such as those discussed in [17, 20].

We implemented a prototype system with the Java language using the JBuilder 3.0 development tool and the Visibroker 3.4 CORBA platform [21]. Tests were carried out in a dedicated 10 Mbs Ethernet LAN with 120 MHz and 400 MHz PCs.

The system functionality can be modeled with eight application-level classes. The *CClient* class represents the client application and it contains 14 operations for modeling the end user interactions. The *CShop* class provides a starting point for the client when it starts looking for application objects on the server. The *CProduct* class represents products to be sold. The *CCustomer* class represents end users that have chosen to purchase products. The *CCart* class represents shopping carts that collect products to be purchased. The *COrder* class represents orders that have been placed by cus-

tomers. The *CGate* class represents an interceptor that monitors user actions once he has identified himself and the *CLog* class represents a security log that registers every relevant operation request. Fig. 10. shows a class diagram of the system.

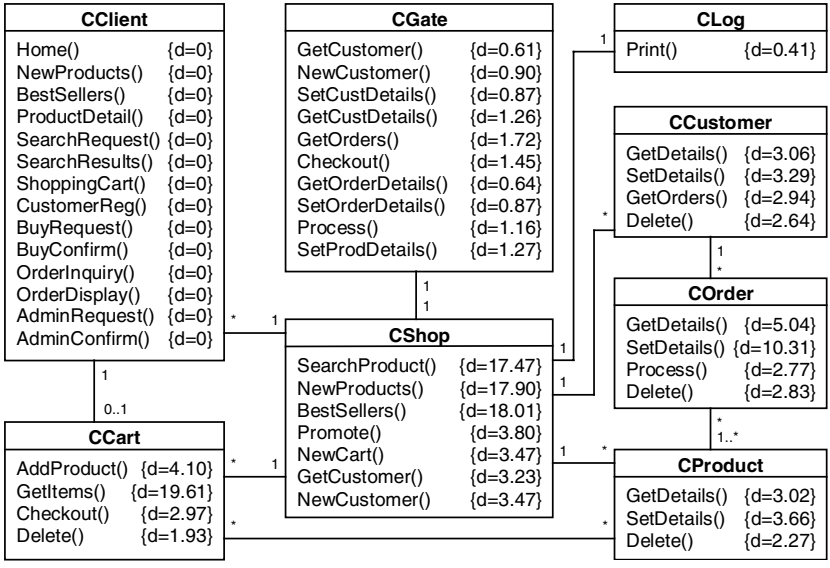



**Fig. 10.** Class diagram for the electronic commerce system

The application layer in Fig. 10 also includes the interface layer information. The service demand estimates (in milliseconds) in the class diagrams have been obtained from a baseline test with a single client. These results combine the network latency, the CPU time needed for parameter handling and other middleware activities at both ends, and the time spent in the network adapters.

The TPC-W specification defines the behavior of the client application with a transition probability matrix. It is used for selecting the next interaction after completing the previous one. The think time between interactions is specified in the TPC-W document to have negative exponential distribution with the mean of 7 seconds. Since our implementation does not execute any application code, we executed our tests with the mean of 2 seconds to impose a higher workload for the server node.

Since each end user interaction contains only a few remote invocations, the behavior of the electronic commerce system can be modeled with one workload diagram. The first nine interactions are illustrated in Fig. 11 and the last five are shown in Fig. 12. The execution probability for each interaction is directly the share indicated by the TPC-W probability matrix.

The network and infrastructure layers are represented with a single set of diagrams due to the simplicity of the dedicated LAN environment. We have a *MemBus* queuing resource in each node for modeling additional queuing that is related to context switching. Also, we include a *Cpu* queuing resource in each node. To cope with dif ferent CPU speeds, we specify two separate classes, *CFastNode* and *CSlowNode*, for modeling the nodes. In our example, the *CFastNode* class needs triggering properties

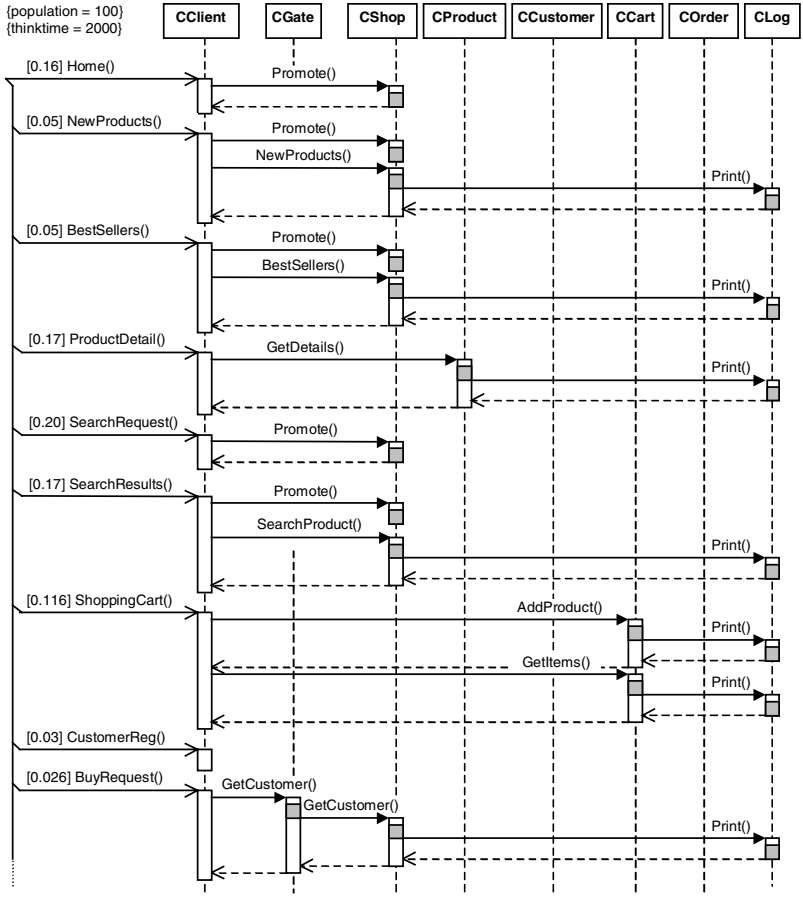



**Fig. 11.** The first nine interactions of the behavior layer

for outgoing invocations and incoming replies, while the *CSlowNode* class requires triggering properties for incoming invocations, outgoing replies, and messages within the same node. The service demands that we obtained from the baseline test are normalized into machine cycles and divided evenly between the four triggering properties. Property *d* is the measured service demand from the baseline test, and property *dd* is the amount of service demand to be attributed to the *Cpu* resource during an outgoing request. To calibrate the scaling factors in the service demand expressions, we carried out experiments with a lightly loaded server (20 concurrent clients) using the baseline application. Fig. 13 illustrates the resulting infrastructure layer.

The deployment layer for the electronic commerce benchmark is illustrated in Fig. 14. The system has two nodes. The client node contains a multi-threaded client application that simulates up to 100 simultaneous end users accessing the system. The server node contains five multi-threaded and two single-threaded CORBA object implementations that are together responsible for the server side functionality.

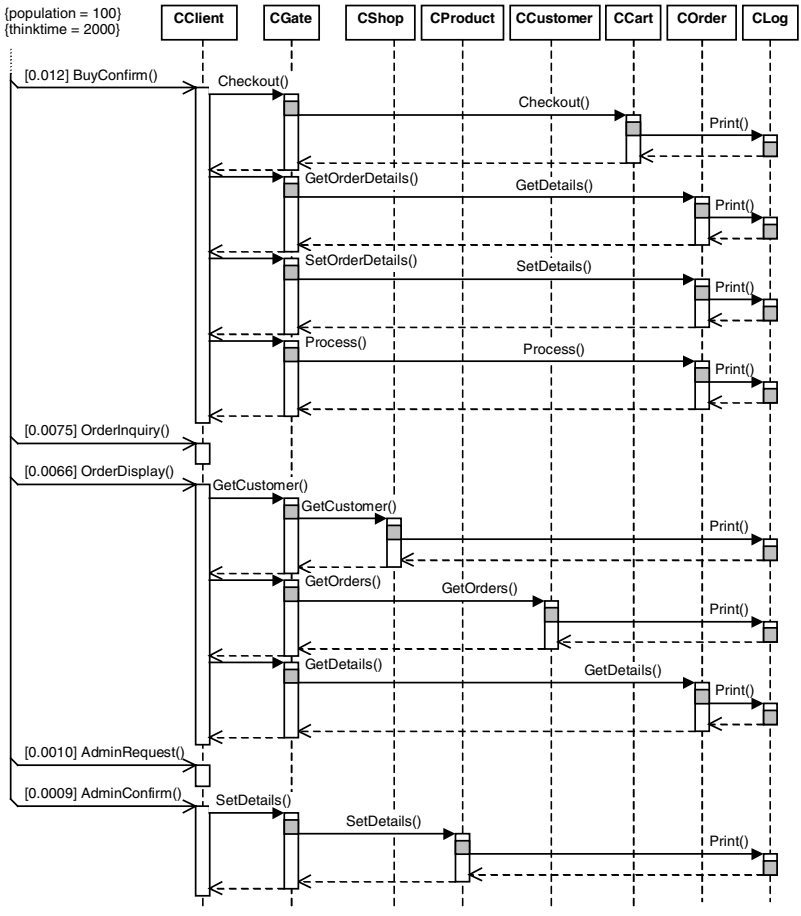



**Fig. 12.** The last five interactions of the behavior layer

To validate the presented performance model, we conducted measurements with our benchmark implementation that directly realizes the model described in the above figures. To find the maximal throughput, we let the number of concurrent clients vary between 10 and 100 with an increment of 10 clients. Fig. 15 indicates that the model is able to predict the throughput of the system reasonably well – the greatest difference between the measurement and the prediction is only 18%. A more detailed analysis of this example and additional examples can be found from [11].

## 6 Conclusions

We have presented a UML-based performance modeling framework for modeling component-based distributed systems. In the framework, UML class diagrams are used for modeling resources in the performance models, and UML collaboration dia-

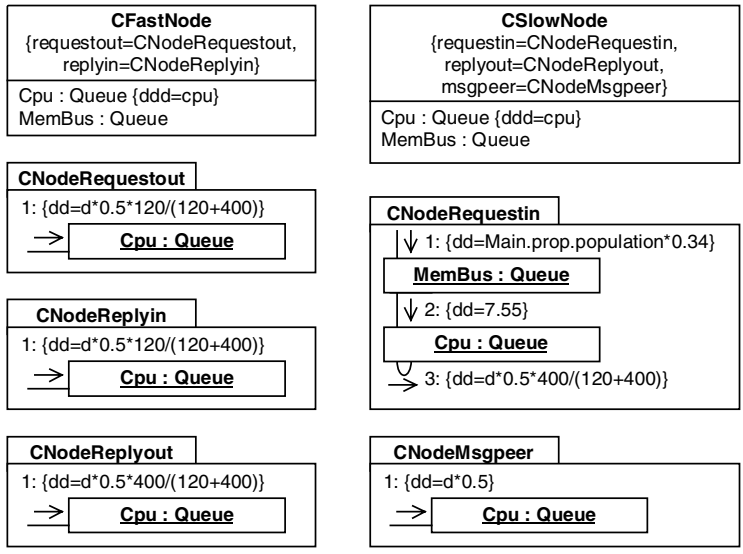



**Fig. 13.** The infrastructure layer for the electronic commerce system

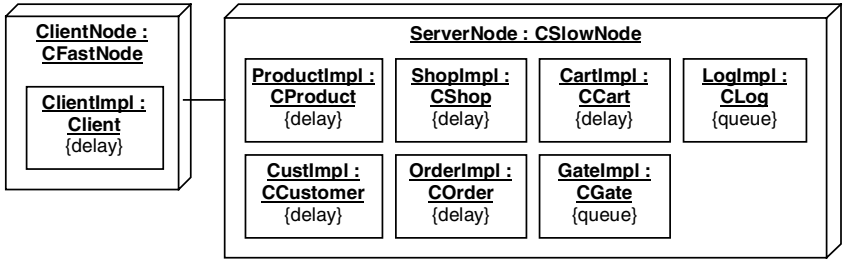



**Fig. 14.** The deployment layer for the electronic commerce system

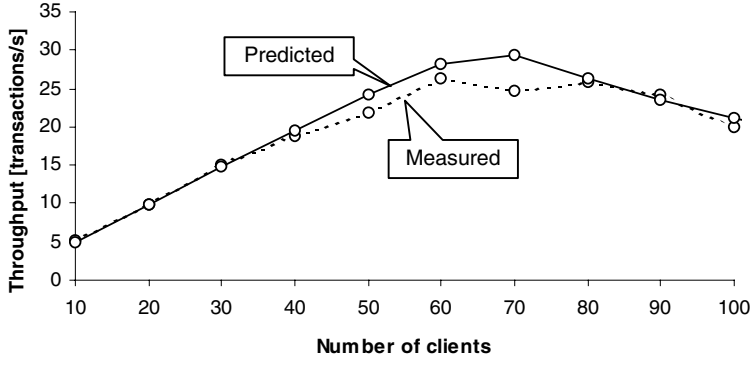



**Fig. 15.** Throughputs for the electronic commerce system

grams are used for representing the workloads. To cope with complexity of component-based distributed systems, we divide the functionality of the system into six layers: application, interface, behavior, infrastructure, network, and deployment layers.

All proposed techniques are compatible with the core UML and follow the guidelines for UML extensions. Also, the techniques allow the coexistence of performance related modeling elements with other UML elements.

We have also described how the resulting UML diagrams can be transformed into a solvable format. This way, relevant performance metrics can be obtained for the models, and the modeling framework can be used in performance engineering.

We briefly discuss some observations. First, the example in section 5 is fairly simple but it gives a reasonably good picture of the target system. This suggests that performance modeling is feasible for component-based distributed systems in spite of the additional complexities caused by the component infrastructure.

Second, the UML diagrams that we used for presenting the performance model are relatively easy to read. Performance related additions are given with simple properties attached to classes, operations, and diagrams. This suggests that the expressive power of the UML is sufficient for performance modeling, and the use of special performance modeling notations is not necessary in normal software engineering.

Third, the usability of the infrastructure layer is worth noting. It provides a flexible tool for modeling the performance aspects of the middleware, the operating system, and the hardware in a structured manner with very little links to the application level. The complexity and accuracy of the infrastructure layer may range from a straightforward representation of communication delays to a full model for the middleware, operating system, and hardware. The infrastructure layer is also a good target for model calibration, since changes at this layer do not distort the application level.

An important limitation of the framework is the partial support for UML diagram types. UML state and activity diagrams might be used to express performance related information, and future extensions of the framework should take them into consideration. For example, activity diagrams might be used to combine several workload diagrams into a single class of jobs. This way, complex workload specifications could be split into smaller pieces. Another important limitation is the lack of support for all UML features in collaboration diagrams. In particular, it is not possible to spawn, kill, or synchronize threads, although multi-threading is supported. The addition of full thread support is a topic for further research.

Currently, the UML is rapidly progressing towards new directions. There is an ongoing initiative to specify next version of UML, and this initiative may change the way performance-related information is represented [22]. The emerging UML profile for scheduling, performance, and time may also influence our framework [3]. An interesting feature in the framework could also be the support for stating performance requirements in UML diagrams. Such requirements can be specified in our framework but they are currently treated only as informal comments.

## References

1. Pooley, R., King, P.: The unified modeling language and performance engineering. IEE Proceedings Software. February,146(1999)1, 2-10
2. Douglass, B.P.: Real-Time UML. Second Edition. Addison-Wesley, Reading, MA (1999)
3. Object Management Group: UML Profile for Scheduling, Performance, and Time, Request for Proposal. OMG TC Document ad/99-03-13. Framingham, MA (1999)

4. Utton, P., Hill, B.: Performance Prediction: An Industry Perspective. In: Marie, R., Plateau, B., Calzarossa, M., Rubino, G. (eds.), Computer Performance Evaluation – Modeling Techniques and Tools. LNCS 1245. Springer-Verlag, Berlin, (1997) 1-5
5. Waters, G., Linington, P., Akehurst, D., Symes, A.: Communications software performance prediction. In: Kouvatsos, D. (ed.): 13th UK Workshop on Performance Engineering of Computer and Telecommunication Systems, Ilkley, West Yorkshire. BCS Performance Engineering Specialist Group (1997) 38/1-38/9
6. Shousha, C., Petriu, D., Jalnapurkar, A., Ngo, K.: Applying Performance Modeling to a Telecommunication System. In: Proceedings of the First International Workshop on Software and Performance WOSP 98. ACM, New York, NY (1998) 1-6
7. Rational Software Corporation: Unified Modeling Language Documentation, version 1.1. Cupertino, CA (1997)
8. Eriksson, H-E., Penker, M.: UML Toolkit. John Wiley & Sons, New York (1998)
9. Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process. Addison-Wesley, Reading, MA, (1998)
10. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual. Addison-Wesley, Reading, MA (1999)
11. Kähkipuro, P.: Performance Modeling Framework for CORBA Based Distributed Systems. PhD Thesis, Report A-2000-3. Department of Computer Science, University of Helsinki (2000)
12. Rolia, J.A., Sevcik, K.C.: The Method of Layers. IEEE Transactions on Software Engineering. Vol. 21, No. 8, August (1995) 689-699
13. Ramesh, S., Perros, H.G.: A Multi-Layer Client-Server Queueing Network Model with Synchronous and Asynchronous Messages. In: Proceedings of the First International Workshop on Software and Performance WOSP 98. ACM, New York, NY (1998) 107-119
14. Haverkort, B.R.: Performance of computer communication systems: a model-based approach. John Wiley & Sons, New York, NY (1998)
15. Petriu, D.C., Wang, X.: From UML description of high-level software architecture to LQN performance models. In: International Workshop on Applications of Graph Transformation with Industrial Relevance, AGTIVE'99, Monastery Rolduc, Kerkrade, The Netherlands, September 1-3, 1999. LNCS, Springer-Verlag, Berlin (2000)
16. Kähkipuro, P.: UML Based Performance Modeling Framework for Object-Oriented Distributed Systems. In: France, R., Rumpe, B. (eds.): «UML»'99 – The Unified Modeling Language, Beyond the Standard. LNCS 1723. Springer-Verlag, Berlin (1999) 356-371
17. Smith, C.U: Performance Engineering of Software Systems. Addison-Wesley, Reading, MA, (1990)
18. Kähkipuro, P.: Object-Oriented Middleware for Distributed Systems. Licentiate Thesis, Report C-1998-43, Department of Computer Science, University of Helsinki (1998)
19. Transaction Processing Performance Council (TPC): TPC Benchmark W (Web Commerce), Revision D-5.0. TPC, San Jose, CA (1999)
20. Menascé, D.A., Almeida, V.A.F., Dowdy, L.W.: Capacity Planning and Performance Modeling. Prentice Hall, Englewood Cliffs, NJ (1994)
21. Inprise Corporation: JBuilder 3.0 Documentation. Scotts Valley, CA (1999)
22. Object Management Group: UML 2.0 Request for Information, Version 1.0. OMG TC Document ad/99-08-08. Framingham, MA (1999)

# Scenario-Based Performance Evaluation of SDL/MSC-Specified Systems

Lennard Kerber

Universität Erlangen-Nürnberg, Institut für Informatik 7, Martensstr. 3,
D-91058 Erlangen, Germany
kerber@informatik.uni-erlangen.de

**Abstract.** The standardised languages SDL (Specification and Description Language) and MSC (Message Sequence Chart) are very popular in the field of telecommunications, since they support all essential steps in the protocol life cycle. With SDL the protocol behaviour is completely specified by communicating extended finite-state machines. The formal basis of SDL enables the use of code generation tool chains, which allows an automated implementation of the specification. With MSC the communication between processes is described by example.
Since telecommunication systems are real time systems, functional and performance aspects must be closely integrated into the development process. Our approach for an early performance prediction is based on scenarios given in MSCs extended with quantitative data. The scenarios are automatically transformed to an SDL specification which yields a prototype implementation via a code generation tool chain. The resulting implementation is executed on the target machines. By monitoring this system, various performance characteristics can be gained, such as processor load, channel utilisation or response time. This allows an early performance prediction for the given scenarios in a system environment which integrates the target hardware and system software in a realistic way.

## 1 Introduction

Today's huge and complex mobile communication networks must be developed and maintained under the continuous pressure of releasing new featured products to the market in time. One approach to attack this problem is to use high-level description and implementation techniques. The standardised languages SDL (Specification and Description Language, [6]) and MSC (Message Sequence Chart, [19]) are today's choice to fit these needs.

Methodologies developed with SDL and MSC are very popular in the field of telecommunications, as they integrate all essential steps in the protocol life cycle. The use of SDL and MSC ranges from requirement specification over the system architecture and behaviour specification to testing. In order to avoid the risk of implementation errors during the manual implementation from the specification, and to speed up the development process, two decades ago protocol engineers

R. Dumke et al. (Eds.): Performance Engineering, LNCS 2047, pp. 185–201, 2001.

started to derive their implementation directly from the high-level specification in SDL instead of coding all parts in low-level programming languages. Today, code generation is an integral part of commercial tool vendors [30] and also a research area where new, efficient implementation strategies are developed to synthesise hardware and software [4,9,26].

Telecommunication systems are real time systems, thus performance issues play an important role through the whole development process. To deal only with functional aspects in early design phases until first executable components are available, is not advisable, since it remains unknown, how design decisions influence the performance the the final system. It is well known that late corrections of design errors lead to much higher costs than timely corrections. Thus, continuous performance analysis is necessary to accompany the functional design process. During the phases where no executable components are available the analysis procedure must be model-based.

Formal description techniques have been extended with quantitative measures recently to form a basis for performance modelling. This has the following advantages:

- a separate performance model need not be constructed and
- performance model and functional model are consistent.

In order to support the protocol development process, one solution is to extend the formal description languages SDL and/or MSC with quantitative measures. Several approaches extend SDL, and an overview of these approaches can be found in [24]. Since many design decisions are already made in an SDL-specification, it is important to analyse different design possibilities according to their performance impacts before they are finalised within the specification. Since MSC-specifications are available very early in the protocol life cycle, MSC is a prime candidate to enable these analyses. Thus we have extended the MSC language with performance aspects to the language PMSC (Performance Message Sequence Chart). The two main aspects of these extensions are annotations to generate a system model for performance evaluation and annotations to specify performance requirements for the system. The system model for performance evaluation is scenario-based, since it uses a partial description of the system behaviour, which is given in MSC. Annotations in the context of PMSC are language extensions that are embedded in comments of the original language, such that standard tools can still process these MSC specifications. Extensions to specify performance requirements in MSC have been proposed by different research groups [27,14,28] and have been integrated in the latest version of MSC, namely MSC2000 [20].

## 2 SDL and MSC

SDL and MSC are languages standardised by the International Telecommunication Union (ITU). They are widely used languages in the telecommunication sector for standardisation, development and implementation of protocols. Both

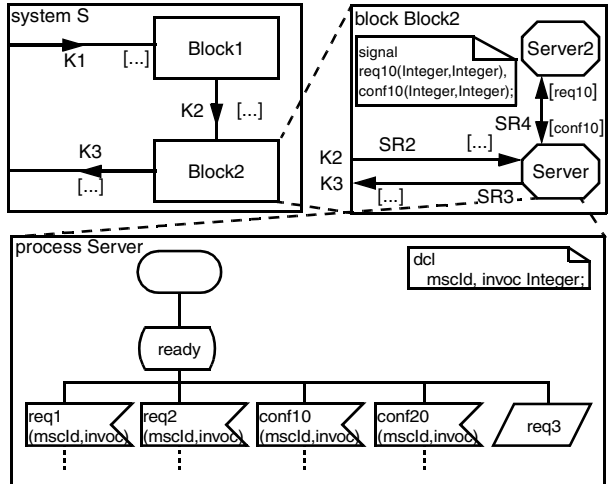



**Fig. 1.** Hierarchy of a SDL specification on system , block and process level

languages have a graphical representation for user interaction and a textual representation for tool processing. They both have a formal semantics.

MSC and SDL generate different views on the system behaviour. With MSC, the system behaviour is described by example, with a global view on the system, where all relevant instances are shown in one diagram. With SDL, the behaviour is described completely, with the view on local components (SDL-processes).

### 2.1 Specification and Description Language

SDL is the main language to describe services and protocols in the telecommunication sector. A system is described hierarchically (figure 1). On the system level, only a static block structure is visible with its channel structure and the signals the blocks exchange. The blocks can be decomposed by other static block structures. On the lowest level of blocks, processes are specified. Within these processes, the dynamic behaviour is specified with extended finite state machines. These finite state machines are extended with variables, one (infinite) input queue, and timer. These processes run conceptually in parallel. They exchange messages, so-called signals. The transfer form the sending process to the receiving process may take time or be immediate (depending on the channels connecting the processes). When a signal arrives at its destination process, it is stored in the appropriate input queue. The SDL process's current state of its finite state machine and the contents of the input queue determine which transition is fired next. If the first message of the input queue does not trigger any transition in the current state, then it is implicitly consumed by the process, i.e. thrown away. To prevent a signal from implicit consumption it is possible to save a signal in the input queue. This means that all saved signals are hidden in the queue and the remaining visible signals in the queue are processed in a

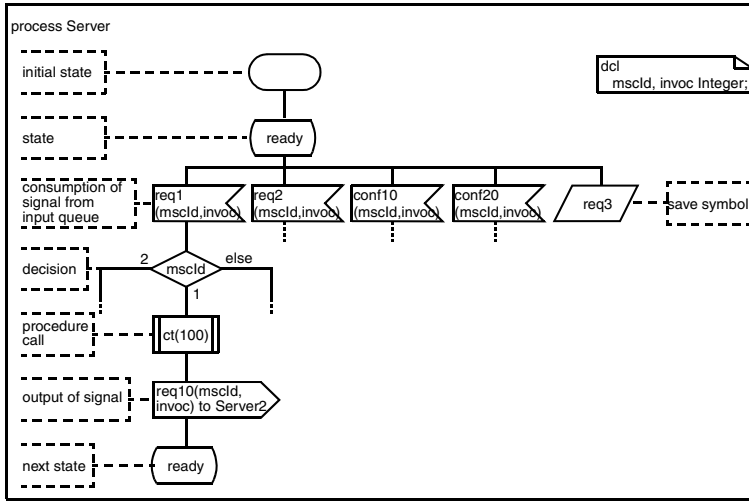



**Fig. 2.** Example of a SDL process

first-come-first-served manner. The positions of the hidden signals in the queue are not changed.

A graphical SDL example is shown in figure 2. The dashed open frames attached to SDL symbols are comments, while the dotted ends of the lines are not SDL syntax, but indicate that the transitions must be continued. The SDL process *Server* starts in its initial state and reaches the state *ready* immediately. Assuming that the signal *req4, req3* and *req1* are waiting in the queue in that order, then

1. the signal *req4* will be implicitly consumed, i.e. erased from the queue, because no transition can be triggered with *req4* in state *ready*,
2. the signal *req3* will be hidden in the queue, because it is saved, and this signal will remain in the queue,
3. the signal *req1* triggers its transition and the parameters of the signal are assigned to the local variables *mscId* and *invoc*.
4. Depending on the value of *mscId* a decision is made,
5. if the value of *mscId* equals *1*, then the procedure *ct* is called with parameter *100* and a signal *req10* with the parameters *mscId* and *invoc* is send to process *Server2* and the next state will be *ready*.

The semantics of SDL defines the state space of the specification. This state space can be used for various analyses and transformation techniques, e.g.:

- state space exploration, simulation,
- checking the SDL-specification for deadlocks, lifelocks or implicit consumption of signals at a process,
- deriving test cases automatically and
- code generation for an executable prototype or end system.

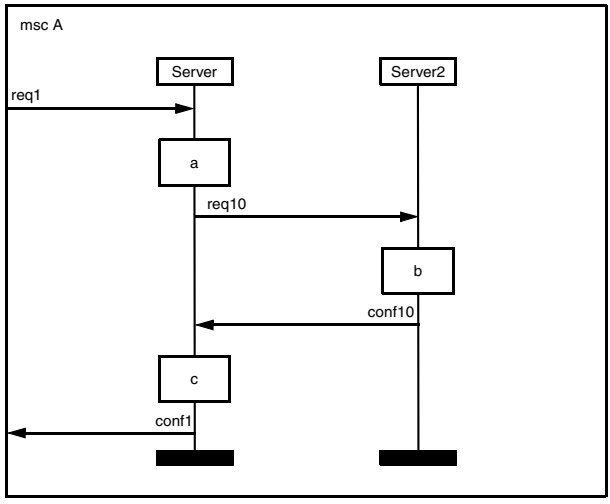



**Fig. 3.** Simple MSC

### 2.2 Message Sequence Chart

In telecommunication industry, MSCs are the first choice to describe example traces of the system under development. MSCs are used throughout the whole protocol life cycle from requirements analysis to testing. In a simple MSC, the asynchronous message exchange between parallel working instances is specified.

The simple MSC *A* shown in figure 3 defines a partial-order on the events occuring on the instances *Server* and *Server2*. The events are ordered along the instance lines and according to the communication. In MSC *A* a *req1* message is received at the instance *Server* before a local action *a* and the sending of a message *req10* occurs. The instance *Sever2* acknowledges the *req10* message after a computation *b* with a *conf10* message, which triggers the *Server* instance to compute *c* and send an *conf1* message back to the environment.

In addition to communication and local action events, other events can be specified, i.e. timer, instance create and instance stop events. The ordering of events on one instance axis can be explicitly annihilated by a so-called coregion. On the opposite, events on different instances or within coregions can be ordered by a general ordering relation.

To define longer traces hierarchically, simple MSCs can be composed by operators in high-level MSC. The MSCs are referenced with so-called MSC references. These MSC references can also reference other high-level MSC, but recursive references to itself are forbidden. The main operators allowed in high-level MSC are:

**Weak Sequence (seq):** When two MSC are composed in sequence, events of an instance in the second MSC can occur after all events of the same instance have occured in the first MSC. This means that it need not delay its events

till all events in the first MSC have happened. Thus events of the second MSC may take place before events of the first MSC.

**Delayed Choice (alt):** This operator is used to specify an alternative between two MSCs. Since the semantics is based on deterministic transition systems the delayed choice operator is introduced. This operator delays the choice behind the common preambles of the different branches the execution may take. Thus, the syntactic location of the choice within the specification is irrelevant, because the choice is resolved at that point where the executions of the different branches differ.

**Parallel Operator (par):** The parallel operator in MSC is an unsynchronised free merge of the events from the different MSCs.

**Loop (loop):** The loop operator defines a repeated weak sequential composition of its argument. The number of iterations through the loop can be fixed to

- a closed interval of finite numbers,
- an open interval where only the minimal number of iterations is specified and
- infinity.

Loops that are not fixed to a closed interval will be denoted in this paper as unbounded loops.

Additional operators are option and exception. These operators are only syntactic abbreviations which can be rewritten by delayed choice constructs.

Beside the textual notation high-level MSC allow a graphical composition. The main idea is that the nodes are MSC references or MSC operator expressions and the edges allows the definition of (weak) sequences, (delayed) choices, and unbounded loops.

In the latest version, MSC is extended with time [20]. The semantics assumes that the events are timeless and that time passes between events. The distance on the time axis between events or the absolute timing of events can be restricted. Moreover, time values may be assigned to time variables and used in such restrictions. This allows very flexible specifications of timed traces.

When MSC and SDL are used together the following analysis and transformation are often used:

- checking the consistency between the MSC and SDL specification and
- transforming the MSC specification to an SDL specification.

The major drawback of the transformation is that its correctness[1] is undecidable when unbounded loops are allowed in the MSC specification[2]. This means of course that a complete consistency check cannot be algorithmically established.

[1] We assume that the transformation for an MSC specification to an SDL specification is correct, iff the traces of both specifications, restricted to the communication events, are equal.

[2] This is a direct consequence of the undecidability result on checking MSCs for races by Muscholl and Peled [25].

## 3 Performance Message Sequence Chart (PMSC)

The idea of PMSC [14] is to enable an early performance evaluation of the system under development. Moreover, time constraints should be specified on the system's execution. These time constraints can be directly used as an evaluation objective for the performance model.

### 3.1 Model for Performance Evaluation

Several modern models for performance evaluation have a modular structure. This structure follows the so-called 3-phase-methodology [18], in which the model of the system is composed by two sub-models, a load model and a machine model. The machine model describes the processing units, their interconnections and how they execute tasks. The load model describes both the tasks with their internal structure in subtasks (application model) and the intensity, i.e. how often the tasks are requested (traffic model). By a mapping of subtasks to processing units the model of the whole system is obtained and can be analysed. The advantage of the modular structure is its high flexibility to combine different load and machine models.

The system model of PMSC also follows the 3-phase-methodology. The application model is given in MSC extended with quantitative data. The machine model is represented with queueing stations.

The application model is based on a set of scenarios given in MSC. Each scenario defines a pieces of load that can be requested by the environment. This piece of load is denoted as a service in PMSC. The PMSC services must be loop-free and can be composed of different MSCs. Each service can be requested by a traffic source which determines the behaviour of the environment. Different services can be requested independently. Thus, different services are composed in parallel in the MSC specification.

An overview of an example system model is given in figure 4. The application model consists of two services *A* and *B*. They are requested by the traffic sources *traffic source A* and *traffic source B*. Each service is given by one simple MSC. The instances *Server*, *Server2*, and *Server3* are mapped on the queueing stations *CPU1* and *CPU2*, while the communication between the instances is mapped on the queueing station *Channel*.

The semantic model for PMSC differs from standard MSC in some ways. In MSC2000, events are instantaneous and time passes only between events, while in PMSC the MSC events are time consuming and therefore called actions in PMSC. Moreover, PMSC assumes an SDL-like implementation model for the MSC instances. Instances have a buffer for incoming messages and the consumption of a message triggers an atomic transition[3]. These assumptions change the semantics of parallel composition from the free merge between MSC-events to the free merge of transitions. A transition is characterised by the following:

[3] "Atomic transition" means that no other event from other transitions of the *same* instance may occur, while a transition is active. A transition is active, when its first event has occured, but its last event has not.

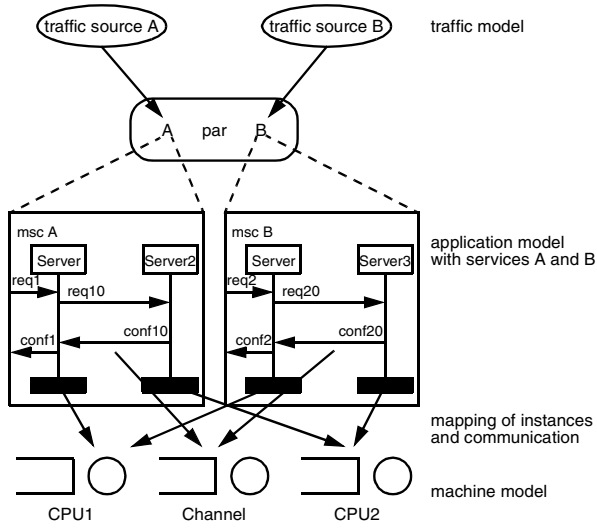



**Fig. 4.** System Model

*A transition is a set of actions from one instance. There is only one input action in a transition that is the very first action of the transition. The transition ends before the next input specified for this instance.*

The transitions of the MSC example from figure 3 are shown in figure 5 with vertical rectangles[4]. Note that transitions can contain the sending of zero or more messages and the performing of zero or more local actions.

The time-consuming actions are annotated with resource requirements. The load imposed on the machine through a single call of a service is determined by the resource requirements of its individual actions. The local actions are annotated with abstract machine operations. These operations must be supplied by the machine on which the action is mapped. The processing time needed for such an action is determined in the machine model (table 1). To define the load imposed on the channels the message lengths can be specified. From these message length the communication times are also derived with the machine model.

[4] This notation should not be confused with an activation region in MSC2000 which is denoted by a rectangle (with a white filling).

**Table 1.** Machine Model

| | Comparison | Assignment | byte |
|---|---|---|---|
| CPU1 | 1ms | 1ms | — |
| CPU2 | 2ms | 2ms | — |
| Channel | — | — | 0.8$\mu$s |

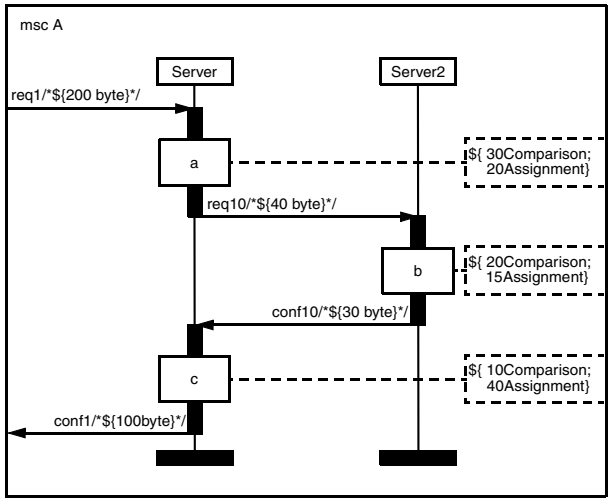



**Fig. 5.** MSCs with transitions and resource requirements

An example is given in figure 5. The action *a* from the instance *Server* is annotated with the abstract machine units of *30 Comparison*s and *20 Assignment*s. When this instance is mapped on *CPU2* (table 1) then the transition will take $30 \cdot 2ms + 20 \cdot 2ms = 100ms$ CPU time.

### 3.2 Definition of Time Constraints

MSC96 specifies functional requirements on the system. These requirements had been extended by temporal behaviour [27,14,28] and led to the time extension in MSC2000 [20]. The notation of MSC2000 is not sufficient for PMSC, since in PMSC actions are time consuming [21] and not instantaneous as in MSC2000. Thus, each PMSC action, i.e. MSC event, consist of a start and an end event. In PMSC, the start and end event of an action can be marked. If two marked events are causally ordered, then a time constraint on the elapsed time can be specified. In figure 6 the ending of the action *receive message req1* is marked with the label *start*. The marker *finish* denotes the end of the action *send message conf1*. According to the time requirement (*required span*{*start,finish*}<=*500ms*) the second message must be sent within 500ms after the first message has been received.

## 4 Evaluation Technique

Our approach generates an early prototype from the PMSC specification for performance evaluation. This is done in two steps. First, the scenarios in PMSC are automatically transformed to an SDL specification. Second, the SDL specification is implemented via a code generation tool chain, resulting in a prototype

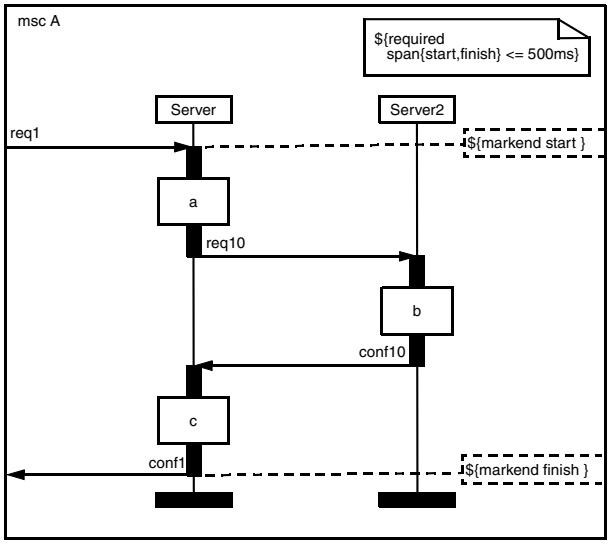



**Fig. 6.** Time constraints in PMSC

implementation. This resulting implementation is executed on the target machines with the SDL runtime environment. By monitoring this system, various performance characteristics can be gained, such as processor load, channel utilisation, or the response time. This allows an early performance prediction for the given scenarios in a system environment which integrates target hardware and system software in a realistic way.

The technique uses the standard tools for the code generation from SDL and standard tools for the monitoring of the performance characteristics. For this paper, the most interesting part is how the SDL specification is generated from the MSC specification. This is discussed in the remaining part of the section with the special focus on our first prototype tool LISA [16].

Each service defines its resulting messages and transitions. For the transformation one of the the following assumptions has to be made:

1. only a single instance of a given service can be active at a given time, or
2. more than one instance of a service may be active at the same time

The two approaches differ in the use of the PMSC language and the possibilities to represent service-related information in the processes.

In the first case, the idea is to give a precise definition of the application behaviour, with its maximal parallelism, e.g. a service *data transfer* can only be active once, but be activated in parallel with another service *connection establishment*. It is possible that in the resulting SDL specification parallel active services have pairwise disjoint message names. This enables the use of SDL states to represent the whole state information of a service and access the messages as specified in the MSC by their names and save all other messages in its input queue.

In the second case, the services are seen as classes of different jobs that the machines must perform and which are triggered from the environment, e.g. a service *data transfer* can be active in parallel when modelling an IP switch where the first IP packet has not left the system before the second arrives, and the message names of the first and second service are the same. In this case the SDL process cannot distinguish messages from different invocations of the same service by the message names in its input queue. Thus, the SDL process must consume the message to analyse its parameters.

The second assumption has been taken by the tool LISA. For simplicity, LISA interprets the instances in the MSC specification as a collection of transitions without any causality between them[5], in order to avoid the storage of service-related information.

The instance *Server* (figure 2), for example, has two transitions resulting from MSC *A* (figure 3) which are triggered through the messages *req1* and *conf10*. The first transition results in the leftmost SDL transition in figure 2[6]. Note that message *req1* can also be used in other MSCs. To distinguish the different *req1* messages from different MSCs at the SDL process *Server*, the parameter *mscId* has been automatically generated, where its value equals *1* for MSC *A*. The parameter *invoc* is passed with all messages without changes. It is used in the environment to distinguish the different parallel invocations of the service. The procedure *ct* is responsible for the time consumption of the transition. Its parametrisation assumes that process *Server* is mapped on *CPU2* (see figure 1). The parameter *100* means that it consumes *100 ms* CPU time. This is exactly the amount of time that has to be consumed on *CPU2* before the message *req10* is sent to *Server2*. In the transformation performed by LISA additional parameters are added to the message to achieve the same message lengths as specified in the MSCs. With an integer occupying *4* bytes, this padding parameter is for the *req1* message *200 - 2·4 = 196* bytes long and for the *conf1* message *40 - 2·4 = 32* bytes. The padding parameters are not shown in figure 2.

## 5 Case Study

The tool LISA was jointly developed with Lucent Technologies[7] to analyse one network element of a GSM (Global System for Mobile communication) network. The presentation of this section mainly follows [12,17].

[5] This simplification has no influence on the utilisation prediction but effects the precision of the response time prediction, since specified causalities are ignored. To remedy this problem a tool LISA-2 is currently under implementation.

[6] The *save*-symbol in figure 2 is not part of the generated SDL process. Because of the simple transformation the process *Server* can receive all possible signals in its state *ready*.

[7] The tool LISA and the case study described in here is part of the cooperation project MONA (MObile Network Analysis) between Lucent Technologies, Network Systems GmbH, Global Wireless Systems Research, and the University of Erlangen-Nürnberg, Institute of Computer Science 7.

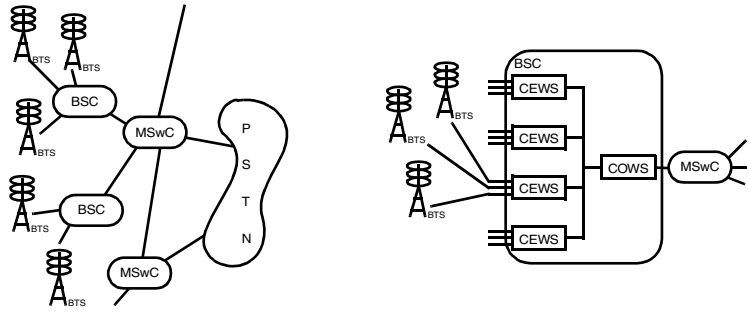



**Fig. 7.** Overview GSM scenario (left) and Structure of Base Station Controller (BSC) (right)

GSM is one of the most popular systems for mobile telecommunications especially in Europe and Asia. Figure 7 (left) gives a simplified view of the network architecture with the three main network elements BTS, BSC and MSwC. The BTS (Base Transceiver Station) is controlling the radio transmission and the interface to the mobile phones. Several BTS are connected to a BSC station (Base Station Controller), while the BSC are connected to the MSwC (Mobile Switching Centre[8]). These MSwC provide access to the PSTN (Public Switched Telephone Network) which is typically ISDN-based.

Within the GSM network, the BSC is a critical element in the network from the performance point of view, because it manages all the signalling traffic and protocol handling from and to the connected radio cells for the real-time radio resource management (establishment and release of radio connections, handovers, etc.).

A realistic BSC software architecture as implemented in a commercial product was examined at Lucent. The unit is implemented as a cluster of several standard UNIX workstations, connected by a 10 MBit Ethernet-LAN running TCP/IP (see figure 7 (right)). One workstation, the common workstation (COWS), mainly handles the tasks of a master controller, while the rest, the cell workstations (CEWS), are more concerned with the actual communication services towards the BTS. To allow an adaption to the variable number of connected BTS, the CEWSs run *server* SDL processes and other SDL processes for each radio transceiver. For the performance analysis we assume that the BSC runs in a well planned scenario, where each radio transceiver serves an area of equal mean traffic load. Therefore, equal signalling load is assumed for each of the SDL processes. Note that a different amount of BTS is typically served by one BSC cell workstation which implies that the overall load for the workstations varies.

The BSC application, i.e. the protocol handling for GSM signalling traffic, is specified in SDL and automatically implemented using proprietary code genera-

[8] Mobile Switching Center is usually abbreviated with MSC which is misleading in the context of Message Sequence Charts.

**Table 2.** Typical percentage of processor load for common and cell workstations within a BSC cluster at the same external traffic demand [12]

| | Radio Side (CEWS) | Net Side (COWS) |
|---|---|---|
| SDL Application (Call Handling) | 12% | 51% |
| Inter-Process Message Transfer Mechanism | 43% | 26% |
| Lower Layer System Calls | (LAPD) 45% | (SS#7) 23% |

tion tools. The BSC internal communication is built with a proprietary middleware that offers transparent and absolute addressing of SDL process instances. The run-time environment bridges the gap between the high-level description of a message communication in SDL and the low-level communication mechanisms in operating systems. On the high level, the message communication between processes or between processes and the environment are visible, while on the low level the communication mechanisms of operating systems and machine interfaces are necessary to achieve the communication between processes and the environment. Measurements at the current BSC show that major parts of the performance are consumed for the run-time environment of the SDL system, i.e. transparent message transfer for SDL-processes and system calls (see table 2). This underlines the need for performance evaluation techniques that integrate the aspects of the run-time environment. Thus, it is not sufficient to describe only transition delays from the application point of view. In LISA these aspects are integrated implicitly through the use of the target operating system, middleware and machine.

We have specified the GSM protocol standards for the Base Station Controller by approximately 100 use cases as Message Sequence Charts for performance evaluation with LISA. These use cases reflect the behaviour of the important SDL-processes and (after the implementation) operating system processes. After an automated generation of SDL processes from the use cases, these processes were executed on an existing BSC UNIX workstation cluster as shown in figure 7 (right). We have added one workstation as a dedicated representative for the environment to the cluster. It generates the traffic for the BSC and receives all its responses. In contrast to the target system these stimuli are communicated via the internal communication medium, i.e. the Ethernet. Despite of this difference the software and hardware structure is the same for the prototype as for the target system.

Each use case represents a service of the protocol, e.g. connection establishment or handover, and is triggered according to its traffic characteristics. The time requirements are checked for each service. The structure of the MSCs specified are very much like the MSC in figure 3.

The technique does not need any assumptions on the relationship between the growth of the utilisation of the workstations and Ethernet in the BSC and the external traffic demands, since we measure the performance of an early prototype. Most techniques for an early performance evaluation assume a linear

relation which is not obvious for a UNIX workstation environment [12]. But indeed, LISA has predicted a linear relationship between the external traffic demands and the utilisation of the BSC components which is confirmed with measurements at the real BSC implementation.

## 6 Related Work

Most performance evaluation approaches for SDL/MSC-specified systems focus on SDL rather than on MSC. An overview of these approaches can be found in [24]. Prominent examples are [23,8,5,29]. [23] focuses on a methodology for integrating SDL and performance evaluation without investigating the automatization steps needed from the SDL specification to the performance model. QSDL (Queueing SDL,[8]) and SPECS (SDL Performance Evaluation of Concurrent Systems,[5]) extend SDL with processors and time consuming transitions and evaluate their system by discrete event simulation. The SPEET (SDL Performance Evaluation Tool,[29]) approach transforms the specification to C code which can be linked to an emulated target environment.

In contrast to these techniques based on functionally complete descriptions, our approach can be classified as a scenario-based approach in the context of SDL/MSC, since our application model is based on a partial system description, specified in MSC.

[13] also uses a scenario-based approach to evaluate SDL specified systems. For a set of scenarios, some MSC traces are derived from the specification and transformed to a so-called Layered Queueing Network (LQN) model, a special queueing model for performance analysis. With the LQN model the system response time or the system utilisation can be derived.

Another scenario-based analytical technique to evaluate SDL/MSC specified systems is to calculate the utilisation of processors and channels, and the throughput of the system on the basis of MSC as proposed by [11]. Its underlying model estimates the utilisation as linear with the offered load.

A queueing model is derived from MSC in [2]. The MSCs are extended with synchronous communication and state information on the instance axises. The MSC instances represent either software processes of the system or the environment. The software processes are mapped on a queueing network model. In contrast to PMSC, the quantitative values are added to the queueing network model and not to the MSC. Moreover, the performance model generated in this approach represents the application and not the machines.

In [7], UML use case diagrams, sequence diagrams, and deployment diagrams are used to generate a queueing network model. All diagrams are extended with quantitative data to automatically parameterise the queueing model. The UML sequence diagrams, which are similar to MSCs, represent the application model in this approach. The communication in these sequence diagrams is only allowed to be synchronous and must follow a strict request-reply pattern and the use cases are loop-free. In PMSC, we do not have restrictions on the communication patterns and consider only asynchronous communication.

A simulation environment for an MSC dialect TID (Timed Interaction Diagrams) is presented in [3]. This approach is similar to PMSC. The main difference is the sequential composition of MSCs which TID defines as a strong sequence where all actions in the first TID must be accomplished before any action in the second TID may take place, while PMSC has adopted the weak sequence operator from standard MSCs. Thus, MSC can be used in PMSC as it is, while in TID synchronisation barriers are introduced.

Transformations from MSC to SDL are investigated in [1,22]. Both approaches focus on a functional analysis of the given use cases to find inconsistencies. The fundamental problem with this transformation is the undecidability result for checking the trace-equivalence between the two specifications when the MSCs contain unbounded loops. This is a direct consequence of the proof for the undecidability to detect so-called race conditions in HMSC [25, Theorem 4.3]. In order to avoid this undecidability, PMSC service specifications have been restricted to loop-free MSC specifications.

## 7 Conclusion

Our approach integrates performance analysis in early phases of the SDL/MSC protocol engineering process. The performance model has a modular structure with a traffic, application and machine model. The application model is built by MSCs with annotations for resource requirements of actions and messages. These requirements are specified in abstract units. The actual time consumption depends on the mapping of actions/messages to machines and is calculated from these abstract units and information in the machine model. In addition, the external stimuli for the application are specified in a traffic model. Time constraints on causally dependent events can be specified which must be checked by the evaluation. To extend MSC has the advantage that only minimal specification effort is needed in order to gain the required performance model, since the MSCs from the functional specification can be reused and "only" quantitative values have to be added there.

For the evaluation the scenarios specified in extended MSCs are transformed to a prototype in our current approach. This is achieved in two steps. First, the MSCs are transformed to SDL in such a way that the SDL system can execute all transitions specified in the MSCs. The transition's run time and message lengths in the SDL system are generated according to the annotated MSC specification. Second, the derived SDL system is automatically implemented with a code generation tool chain. This generated prototype runs on the real hardware, and response time, processor or net utilisation are measured. Thus, the influences of the target system software, middleware, and hardware that implement the transparent communication for the SDL processes, is automatically integrated into the performance measurement.

**Acknowledgements.** I would like to thank Dr. Markus Siegle and Ralf Münzenberger for many valuable discussions on my work and proofreading of earlier versions of this paper.

## References

1. M. Abdalla, F. Khendek, and G. Butler. New results on deriving SDL specifications from MSCs. In Dssouli et al. [10], pages 51–66.
2. F. Andolfi, F. Aquilani, S. Balsamo, and P. Inverardi. Deriving Performance Models of Software Architectures from Message Sequence Cahrts. In Woodside et al. [31], pages 47–57.
3. L. Braga, R. Manione, and P. Renditore. A Formal Description Language for the Modelling and Simulation of Timed Interaction Diagrams. In Gotzhein and Bredereke [15], pages 245–260.
4. O. Bringmann, W. Rosenstiel, A. Muth, G. Färber, F. Slomka, and R. Hofmann. Mixed Abstraction Level Hardware Synthesis from SDL for Rapid Prototyping. In *10th IEEE International Workshop on Rapid System Prototyping*, Clearwater, USA, June 1999.
5. M. Bütow, M. Mestern, C. Schapiro, and P.S. Kritzinger. Performance Modelling with the Formal Specification Language SDL. In Gotzhein and Bredereke [15], pages 213–228.
6. CCITT. *Recommendation Z.100: Specification and Description Language SDL, Blue Book.* ITU, Geneva, 1992.
7. V. Cortelessa and R. Mirandola. Deriving a Queueing Network based Performance Model from UML Diagrams. In Woodside et al. [31], pages 58–70.
8. M. Diefenbruch, J. Hintelmann, and B. Müller-Clostermann. The QUEST-Approach for the Performance Evaluation of SDL-Systems. In Gotzhein and Bredereke [15], pages 229–244.
9. M. Dörfel, F. Slomka, and R. Hofmann. A Scalable Hardware Library for the Rapid Prototyping of SDL Specifications. In *10th IEEE International Workshop on Rapid System Prototyping*, Clearwater, USA, June 1999.
10. R. Dssouli, G. v. Bochmann, and Y. Lahav, editors. *SDL'99: The Next Millennium*, Montreal, Canada, June 1999. Ninth SDL Forum, Elsevier Science B.V.
11. W. Dulz. A Framework for the Performance Evaluation of SDL/MSC–specified Systems. In A. Javor, A. Lehmann, and I. Molnar, editors, *ESM96 European Simulation Multiconference*, pages 889 – 893, Budapest, Hungary, June 1996. Society for Computer Simulation International.
12. W. Dulz, S. Gruhl, L. Kerber, and M. Söllner. Early Performance Prediction of SDL/MSC-specified Systems by Automated Synthetic Code Generation. In Dssouli et al. [10], pages 457–471.
13. H. El-Sayed, D. Cameron, and M. Woodside. Automated performance modeling from scenarios and sdl designs of distributed systems. In *Proc. of Int. Symp. on Software Engineering for Parallel and Distributed Systems*, pages 127–135, Kyoto, Japan, April 1998. IEEE Press.
14. N. Faltin, L. Lambert, A. Mitschele-Thiel, and F. Slomka. An Annotational Extension of Message Sequence Charts to Support Performance Engineering. In A. Cavalli and A. Sarma, editors, *SDL'97: Time for Testing - SDL, MSC and Trends*, pages 307–322, Evry, France, September 1997. Eighth SDL Forum, Elsevier.

15. R. Gotzhein and J. Bredereke, editors. *Formal Description Techniques IX*. IFIP, Chapman & Hall, Oktober 1996.
16. S. Gruhl. Automatic Generation of Synthetic Load from Formal Use Cases for Performance Evaluation. Diplomarbeit, IMMD7, Universität Erlangen-Nürnberg, November 1998.
17. Stefan Gruhl, Micheal Söllner, and Lennard Kerber. Automated Performance Prototyping for SDL/MSC-specified Systems. In *International Conference on Computer Communication (ICCC)*, Tokyo, Japan, September 1999.
18. U. Herzog. Performance Evaluation and Formal Description. In V.A. Monaco and R. Negrini, editors, *Advanced Computer Technology, Reliable Systems and Applications, Proceedings*, pages 750–756, Bologna, Italy, May 1991. IEEE CompEuro, IEEE Computer Society Press. Invited paper.
19. ITU-T. *ITU-T Recommendation Z.120: Message Sequence Charts (MSC)*. ITU, Geneva, 1996.
20. ITU-T. *ITU-T Recommendation Z.120: Message Sequence Charts (MSC)*. ITU, Geneva, 2001. To be published.
21. L. Lambert. PMSC for Performance Evaluation. In *1. Workshop on Performance and Time in SDL/MSC*, Erlangen, Germany, 1998.
22. N. Mansurov and D. Zhukov. Automatic synthesis of SDL models in use case methodology. In Dssouli et al. [10], pages 225–240.
23. J. Martins and J. Hubaux. A New Methodology for Performance Evaluation Based on the Formal Technique SDL. In *Proc. Design and Analysis of Real-Time Systems (DARTS '95)*, Bruessels, November 1995.
24. A. Mitschele-Thiel and B. Müller-Clostermann. Performance Engineering of SDL/MSC Systems. *Journal on Computer Networks and ISDN Systems*, 31(17):1801–1815, June 1999.
25. A. Muscholl and D. Peled. Analyzing Message Sequence Charts. In S. Graf, C. Jard, and Y. Lahav, editors, *2nd Workshop on SDL and MSC*, pages 3–17, Col de Porte, Grenoble, France, June 2000. SDL Forum Society.
26. Peter Langendörfer and Hartmut König. COCOS - A Configurable SDL Compiler for Generating Efficient Protocol Implementations. In *Proc. of 9th SDL Forum*, Montreal, Canada, June 1999. Elsevier.
27. C. Schaffer. MSC/RT: A Real-Time Extension to Message Sequence Charts (MSCs). Technical Report TR140-96, Johannes Kepler Universität Linz, Institut für Systemwissenschaften, 1996.
28. I. Schieferdecker, A. Rennoch, and O. Mertens. Timed MSCs - an Extension to MSC'96. In A. Wolisz, I. Schieferdecker, and A. Rennoch, editors, *Formale Beschreibungstechniken für verteilte Systeme*, GMD-Studie Nr. 315, pages 165–174, Berlin, Germany, June 1997. GI/ITG, GMD-Forschungszentrum.
29. M. Steppler. Performance analysis of communication systems formally specified in sdl. In *Proceedings of The First International Workshop on Simulation and Performance '98 (WOSP '98)*, pages 49–62, Santa Fe, New Mexico, USA, 12th–16th October 1998.
30. Telelogic, Malmö, Sweden. *Telelogic Tau 4.1, SDL Suite Getting Started*, September 2000.
31. M. Woodside, D. Menasce, and H. Gomaa, editors. *Second International Workshop on Software and Performance WOSP2000*, Ottawa, Canada, September 2000.

# Characterization and Analysis of Software and Computer Systems with Uncertainties and Variabilities

Shikharesh Majumdar[1], Johannes Lüthi[2], Günter Haring[3], and Revathy Ramadoss[4]

[1] Systems and Computer Engineering Department
Carleton University, Ottawa, K1S 5B6, Canada
majumdar@sce.carleton.ca
[2] Institut für Technische Informatik
Universität der Bundeswehr München, D-85577 Neubiberg, Germany
luethi@informatik.unibw-muenchen.de
[3] Institut für Informatik und Wirtschaftsinformatik
Universität Wien, Lenaugasse 2, A-1080 Wien, Austria
rodo@ani.univie.ac.at
[4] Nortel Networks, Richardson, USA

**Abstract.** Conventional solution techniques for analytic performance models of computer and telecommunication systems use single values as inputs. Uncertainties or variabilities in model parameters may exist in many types of systems. Using models with a single aggregated mean value for each parameter for such systems can produce inappropriate and misleading results. This chapter presents intervals and extended histograms for characterizing system parameters that are associated with uncertainty and variability. Adaptation of existing analytic performance evaluation methods to this interval-based parameter characterization is described. The application of this approach is illustrated with two examples: a hierarchical model of a multicomputer system and a queueing network model of an EJB server implementation.

## 1 Introduction

Analytic techniques are often used for performance analysis of computing systems because of their relatively low cost in comparison to simulations and benchmarks. An analytic model accepts a set of single valued parameters (such as mean resource demands) and produces a single point measure for each performance index of interest (such as the mean response time and mean processor utilization). However such a single point characterization of parameters is inadequate when *uncertainties* or *variabilities* are associated with system parameters. Uncertainties may be associated with parameters for various reasons. As an example, consider software performance engineering that is concerned with the integration of performance modeling with the various phases of software design as well as

R. Dumke et al. (Eds.): Performance Engineering, LNCS 2047, pp. 202–221, 2001.

implementation and is receiving a great deal of attention [39]. Although uncertainties may be associated with one or more system parameters in early stages of system design, the expert designer may have a good idea about the range of values associated with these parameters from experience with similar systems. Intervals are also useful in expressing confidence intervals associated with parameters measured on systems.

We propose to use intervals for capturing such parameter uncertainties. The probability of occurrence of any value within an interval can follow any given arbitrary distribution. In hierarchical performance modeling, bounding techniques used in one layer of the modeling hierarchy may produce input intervals in the next layer. For example, Hartleb and Mertsiotakis propose bounding techniques for the runtime of parallel programs [11], which are integrated in the modeling tool PEPP [8]. Bounds for the mean runtime of parallel programs are also provided by the *serialization analysis* technique used in the PAMELA approach [9]. Additionally, input intervals can also be derived for systems represented using fuzzy modeling techniques [14] via so-called $\alpha$-cuts of parameters characterized by fuzzy numbers [19]. Performance analysis algorithms that can handle intervals are also suitable for a low cost computation of performance bound intervals, best/worst case analysis, and sensitivity analysis studies.

Parameter variabilities can occur in systems which are characterized by different phases of operation. As an example consider a client-server system, where different mean demands at a given device may occur during various time periods. Such a variability may be exhibited by a point-of-sale system where different amount of work per transaction occurs during different periods of the day. Variabilities in service demands can also occur implicitly in systems. For example, different service demands have been observed in a database system described in [4]: during periods of time when less memory was available for transaction processing a larger number of I/O operations was observed. Even though the system is in steady state during each phase of operation, neither a conventional single class nor a multiclass queueing network is adequate for the computation of system performance. Variabilities in system parameters may also arise in other situations as well. As described by Buzen [4] variabilities in parameters can lead to large errors if a single mean value is used for every model parameter of interest. We propose histograms that are expressed by a series of intervals with associated probabilities to capture such variabilities.

This chapter focuses on the characterization of uncertainties and variabilities in model parameters through intervals and histograms and adaptation of existing analytic techniques to handle such non-single-valued parameters. A number of existing popular analytic models are considered in the paper. This includes queueing network models [15] that have been used extensively in various studies of computer system performance. Conventional solution techniques for queueing networks are inadequate for handling parameters characterized by uncertainties or variabilities as specified through histograms. Moreover the existing techniques compute only a single mean value for a performance measure of interest. The

techniques discussed in this paper produce an interval or a histogram for performance measures of interest. Standard queueing theory is not applicable for modeling most concurrent systems and more complex techniques such as the two-level model presented by Thomasian and Bay [42] are required. The incorporation of the techniques for handling parameter uncertainties into such methods is discussed.

To handle the proposed parameter characterization method, existing algorithms must be adapted to expect intervals as input parameters. Depending on how the model output is related to the different interval parameters, computing systems can be divided into two classes. The simpler of the two is a class in which the performance measure of interest is a monotonic function of the input parameters. A short discussion of this simpler case is presented in Section 2. More details including the application of this technique to the well-known mean value analysis (MVA) algorithm can be found in [22]. The second class of systems is one in which such a monotonic relationship between model inputs and output is not known to exist. An interval arithmetic-based technique is proposed for analyzing such systems. Non-monotonic behavior has been observed in several different situations. The classical studies of system thrashing (see [37] for example) have shown that the throughput of a computer system increases initially with an increase in the level of multiprogramming but it soon reaches a plateau and starts decreasing with further increase in the multiprogramming level. At very high levels of multiprogramming an application receives only a few frames of memory that results in a heavy page faulting activity known as thrashing. A recent study of client-server distributed systems shows a non-monotonic relationship between system throughput and the duration of the timeout [36] associated with the receiving of a response for Remote Procedure Calls (RPC) that are used as the basic communication mechanism between client and server nodes.

This chapter concentrates primarily on the analysis of mean values for performance measures of interest. Performance bounds for single class queueing networks characterized by uncertainties and variabilities are presented in [18]. Extension of conventional bottleneck analysis techniques to such models is considered in [20]. The association of fuzzy numbers to parameters of models with uncertainties is proposed in [19]. None of these works deal with the computation of performance measures for models with interval-based parameters that is addressed by the research discussed in this chapter. A part of this research is based on the use of interval arithmetic. Various programming languages such as Fortran, Pascal, C, and Prolog support interval arithmetic-based computation [35]. Interval arithmetic has been used in many different areas such as Critical Path (PERT) analysis, X-Ray Diffraction Crystallography and a variety of global optimization problems [35]. A more elaborate survey of applications of interval arithmetic can be found in [28]. Previous application of interval arithmetic to performance analysis include the computation of robust throughput bounds for general multiclass queueing networks [26], [29]. The first presents a technique for the computation of upper and lower bounds on throughput for

different customer classes in a queueing network with servers employing a FIFO queueing scheme. Bounds for various other methods of scheduling at the servers are discussed in the second paper. Interval arithmetic is also used for the computation of bounds on the performance of concurrent software using rendezvous style communication [25].

## 2 Interval Parameters

Typically, performance measures in analytical models can be expressed as mathematical functions of a number of input parameters. Thus, in the sequel we consider real functions:

$$f : \mathbb{R}^n \to \mathbb{R}, \quad x = (x_1, \ldots, x_n) \to f(x). \tag{1}$$

As discussed in Section 1, we are interested in using intervals as input parameters for performance measure functions. In the following, we give some basic definitions of intervals and related terms. A detailed introduction can be found in books like e.g. [31], [32].

### 2.1 Definitions and Introduction

A real interval is a set of the form

$$X = [\underline{x}, \overline{x}] = \{x \in \mathbb{R} \mid \underline{x} \leq x \leq \overline{x}\}, \tag{2}$$

where $\underline{x}, \overline{x} \in \mathbb{R}$ and $\underline{x} \leq \overline{x}$. $\underline{x}$ and $\overline{x}$ are called *endpoints* of the interval. In particular, $\underline{x}$ is called *lower bound*, and $\overline{x}$ is called *upper bound* of the interval $X = [\underline{x}, \overline{x}]$. If $S$ is a nonempty bounded subset of $\mathbb{R}$, we denote the *hull* of $S$ by $\square S = [\inf(S), \sup(S)]$. The hull is the tightest interval enclosing $S$. An interval vector $X = (X_1, \ldots, X_n)$ is also referred to as a *box*. For a real function $f$, continuous on every closed box on which it is defined, the *range* of a box $X$ is defined as:

$$f^*(X) = \square\{f(x) \mid x \in X\} = \{f(x) \mid x \in X\}. \tag{3}$$

Given a performance model with interval parameters we are usually interested to find the range of associated performance measures. Because of the continuity of $f$, the range is itself an interval: $f^*(X) = [\underline{f}, \overline{f}]$. In general, the computation of the range is a global optimization problem with box constraints. I.e., the global minimum $\underline{f}(x) = \min_{x \in X} f(x)$ and the global maximum $\overline{f}(x) = \max_{x \in X} f(x)$, subject to $x \in X$ have to be found.

In the special case of so-called $N$-monotonic functions, the range can be computed using only single value evaluations of $f$ with appropriate combinations of parameter interval endpoints as input parameters. To be more specific, let $f(x_1, \ldots, x_n)$ be monotonically increasing w.r.t. all parameters $x_i$, $i \in I$ and monotonically decreasing w.r.t. all parameters $x_i$, $i \in D$, where

$I \cup D = \{1, \ldots, n\}$. Then the range of $f$ with interval parameters $X_1 = [\underline{x}_1, \overline{x}_1], \ldots, X_n = [\underline{x}_n, \overline{x}_n]$ can be computed as follows:

$$f^*(X_1, \ldots, X_n) = [f(y_1, \ldots, y_n),\ f(z_1, \ldots, z_n)]\,, \tag{4}$$

where $y_i = \underline{x}_i$, $z_i = \overline{x}_i$ if $i \in I$, and $y_i = \overline{x}_i$, $z_i = \underline{x}_i$ if $i \in D$. In [22], such a situation is discussed in detail for the example of the *Mean Value Analysis* (MVA) algorithm for closed single class queueing networks.

### 2.2 Interval Arithmetic

For many performance measures, monotonicity properties do not hold and general optimization methods are often difficult to apply and of high computational complexity. However, an enclosure $F(X) \supseteq \{f(x) | x \in X\}$ for the range can be obtained using so-called *interval arithmetic* (for a detailed introduction see for example the book [32]): on the set of intervals, the *elementary operations* $\circ \in \{+, -, \cdot, /, \hat{}\} =: \Omega$ are defined by setting:

$$X \circ Y = \square\{x \circ y \,|\, x \in X, y \in Y\} = \{x \circ y \,|\, x \in X, y \in Y\}, \quad \forall \circ \in \Omega. \tag{5}$$

Furthermore, the elements $\varphi$ of a predefined set $\Phi$ of elementary continuous real functions are extended to interval arguments by defining:

$$\varphi(X) = \square\{\varphi(x) \,|\, x \in X\} = \{\varphi(x) \,|\, x \in X\}, \tag{6}$$

for all intervals $X$ such that $\varphi(x)$ is defined for all $x \in X$.

From monotonicity properties it follows that the elementary operations $\circ \in \{+, -, \cdot, /\}$ can be computed in terms of the endpoints of the intervals $X = [\underline{x}, \overline{x}], Y = [\underline{y}, \overline{y}]$: $X + Y = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$, $X - Y = [\underline{x} - \overline{y}, \overline{x} - \underline{y}]$, $X \cdot Y = [\min(\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}), \max(\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y})]$, and $X/Y = X \cdot [1/\overline{y}, 1/\underline{y}]$, if $0 \notin Y$. Analogously, (piecewise) monotonicity of the elementary functions can be exploited to define their evaluations along the lines of computations with the interval endpoints of the argument. E.g., because of the monotonicity of the exponentiation function we know that for any interval $X = [\underline{x}, \overline{x}]$, $\exp(X) = [\exp(\underline{x}), \exp(\overline{x})]$. Using the interval extensions of elementary operations and functions, an arithmetic expression can be evaluated with intervals by substituting the variables by the corresponding intervals and step by step application of interval arithmetic. E.g., given the intervals $X_1 = [1, 2]$ and $X_2 = [4, 5]$, the arithmetic expression $f(x_1, x_2) = (2x_1 + x_2)x_1$ is evaluated as follows: $f(X_1, X_2) = (2 \cdot [1, 2] + [4, 5]) \cdot [1, 2] = ([2, 4] + [4, 5]) \cdot [1, 2] = [6, 9] \cdot [1, 2] = [6, 18]$.

Interval arithmetic can serve as a tool to obtain interval extensions of real functions. However, due to an effect known as *dependency problem*, in general, interval arithmetic does not provide the exact range of a function. This effect is also known as *overestimation*. The root of the dependency problem is the memoryless nature of interval arithmetic if a parameter occurs multiple times in an

arithmetic expression since each occurrence of an interval variable in an expression is treated independently [32]. For example, the expression $X-X$ is evaluated to $\{x_1 - x_2 \mid x_1, x_2 \in X\} = [\underline{x} - \overline{x}, \overline{x} - \underline{x}]$, instead of $\{x - x \mid x \in X\} = [0, 0]$. Sometimes an expression can be re-formulated to avoid multiple occurrence of parameters or at least to reduce the number of occurrences of an interval parameter. For example, the expression $f(X, Y) = \frac{X-Y}{X+Y}$ may be rewritten as $1 - \frac{2}{1+X/Y}$ if $0 \notin Y$. However, in general multiple occurrence of interval parameters cannot always be avoided. Therefore the dependency problem often causes crucial overestimation of the actual range of an evaluated function.

### 2.3 Interval Splitting

A way to overcome overestimation due to the dependency problem is to split the original input parameter intervals into subintervals and evaluate the arithmetic expression using these subintervals as input parameters. The principal idea for interval splitting is to subdivide the input parameter intervals into a number of subintervals, compute interval evaluations of the arithmetic expression with the subintervals as input parameters, and find the overall result by computing the minimum of all lower bounds and the maximum of all upper bounds of the intermediate results. Analogously, an interval parameter vector (box) is split into subboxes. In [38] it is shown that the results obtained from interval splitting converge to the actual range if the width of the subintervals approaches zero. This means that it is guaranteed that interval splitting is indeed a technique to obtain arbitrarily tight interval results.

In the *brute force splitting* (BFS) algorithm, in every iteration the input parameter intervals are split into two subintervals of equal length. The parameter (sub)intervals considered in iteration $s$ (i.e. splitting degree $s$) are collected in $P^s$, the set of potential input parameter intervals. In every iteration, the splitting degree $s$ is incremented and a new set $P^s$ of input parameter intervals to be considered is initialized. Subsequently, $P^s$ is filled with subintervals of all intervals $X \in P^{s-1}$. The desired performance evaluation algorithm is applied to each subinterval combination and the minimum of all lower bounds and maximum of all upper bounds are computed to obtain the result of the current iteration. These steps are iterated until the difference between successive iterations becomes smaller than a predefined stopping criterion $\epsilon$.

Note that the number of subintervals at splitting degree $s$ is $2^s$. More general, if $n$ parameters are characterized by intervals (i.e. we have an $n$-dimensional input parameter box), it holds that $|P^s| = 2^{sn}$. The application of the BFS algorithm for the solution of interval-based computer performance models is presented in [27].

The high complexity of BFS can be significantly reduced if not every subinterval is considered for further splitting. Given a subinterval $X$ it may eventually be concluded from the obtained interval results that neither the lower nor the upper bound of the range is produced by that subinterval. In that case, $X$ need

not be considered any further in the search for the lower and upper bound of the range; i.e., $X$ need not be split into additional subintervals. This idea of selective interval splitting was introduced by Skelboe in the context of general purpose optimization of rational interval functions [38]. The application of this techniques to performance models is reported in [34]. In the example modeling an Enterprise JavaBeans (EJB) implementation, presented in Subsection 3.2 we use a new selective splitting algorithm described in [24]. This algorithm combines interval and conventional evaluations of a performance model to reduce the number of subintervals that have to be considered.

## 3 Application of Interval-Based Techniques

We have adapted a number of performance models to handle interval parameters. These models can be divided into two classes. Models with monotonic behavior and models with non-monotonic behavior. For models in the first class the performance measures are monotonic functions of input parameters. The single class closed queueing network discussed in [22] is an example of such a model. In [22], a generalization of monotonicity results proved by Suri [41] to separable closed QNMs consisting of queueing as well as delay devices is presented. These monotonicity properties are exploited to obtain an efficient interval solution. Multiclass open queueing networks also demonstrate monotonic behavior and are discussed in detail in [30]. In such a model with monotonic behavior the conventional open queueing network methods are applied twice with different combinations of lower and upper bounds of each parameter value (see Eq. (4)). The upper and lower bounds that characterize the performance measure of interest are directly obtained from these evaluations. Models in which the performance measures are known to be non-monotonic functions of input parameters or for which monotonicity properties can not be proved to hold for all possible parameter values belong to the second class. The application examples presented in this section belong to this second class of systems. As we will see we have used interval arithmetic-based techniques in their solution. More examples are discussed in [30].

### 3.1 Modeling Concurrent Applications on a Multicomputer System

Standard queueing networks cannot handle concurrency in execution and more sophisticated models are required. A number of advanced techniques for the estimation of performance for such systems have been proposed (see for example [42] and [33]). The interval based performance analysis as provided by the interval splitting technique has been applied to a number of such other analytic models where the existence of an explicit monotonic relationship between model inputs and outputs is not known. Examples include models of client-server systems communicating through synchronous message passing and the model of a concurrent application running on a multicomputer [34]. For conservation of space

we report on the latter case study that uses a hierarchical modeling technique based on both Markov chains and queueing networks as proposed by Thomasisan and Bay. A brief explanation of this technique is presented in the context of an example concurrent application running continuously on a multicomputer system consisting of two computing nodes (see Fig. 1). A more detailed description of the general technique can be found in [42].

The application structure is described by the precedence graph [6] shown in Fig. 1 (a). Each node in the precedence graph represents a process with its own independent thread of control. Directed edges represent precedence relationships among processes. A process can start execution only after all its predecessors have completed. The application runs continuously: as soon as Process 5 is completed another statistically identical application is started immediately on the system. The multicomputer system on which the concurrent application runs is composed of two computing nodes inter-connected by a full duplex communication link (see Fig. 1 (b)). Processes 1 and 4 execute on computing node 1 while the other processes execute on computing node 2. Processes 1c and 4c are special in the sense that they model message transmission through the communication link and message processing at the CPU of the receiver system. Each of these two processes makes 9 visits to the communication server. The mean total CPU demand for Process 1c is a variable factor in this example whereas Process 4c exhibits a mean total CPU demand of 100 msec. The service demands of the other processes in the application are presented in Table 1.

A two-level hierarchical decomposition technique is used for the solution of the model. At the higher level the system is specified by a Markov chain (see Fig. 1 (c)) in which each state is labeled with the labels for processes that execute simultaneously on the system while a closed multiclass queueing network is used at the lower level for modeling resource contention. A multiclass closed queueing network model is to be solved for each of these states. The requirement for the application of a product form queueing network are assumed to be valid and an exact MVA algorithm is used for computing the throughput of a given class that corresponds to a given process appearing in the state label. These throughputs provide the transition rate for the Markov chain model. For example, the solution

**Table 1.** Service demands for processes in the concurrent application

| Process | CPU demand (msec) | No. of visits to Disk A | No. of visits to Disk B | Disk service time per visits (msec) |
|---|---|---|---|---|
| 1 | Variable | 5 | 5 | 50 |
| 2 | 800 | 15 | 15 | 50 |
| 3 | 800 | 15 | 15 | 50 |
| 4 | 1000 | 10 | 10 | 50 |
| 5 | 500 | 5 | 5 | 50 |

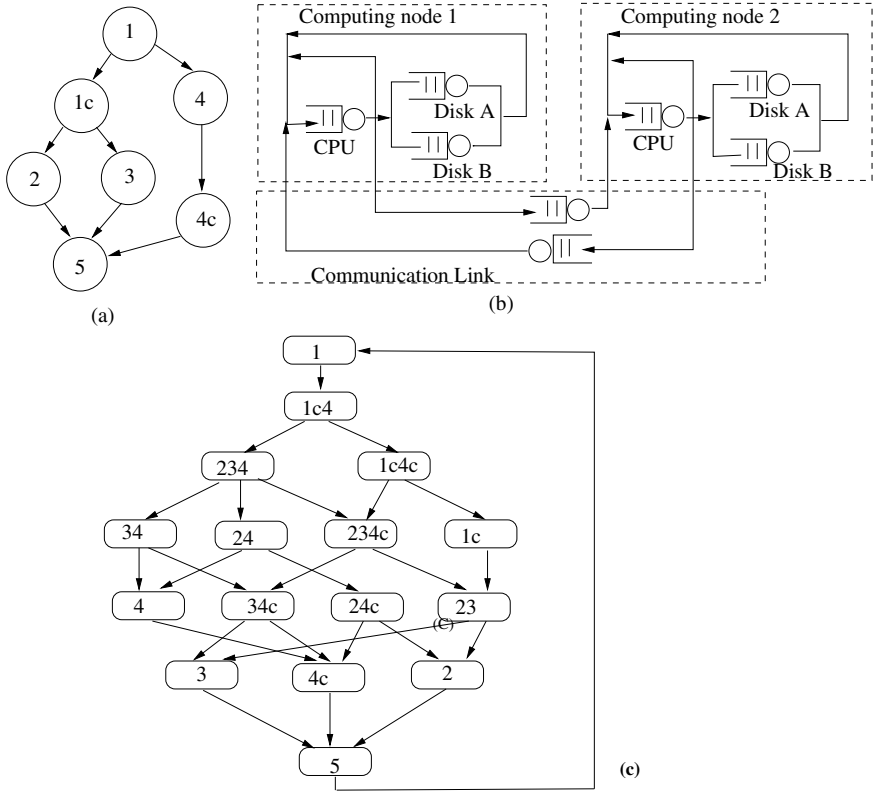



**Fig. 1.** Multicomputer system running concurrent applications. (a) Application task graph, (b) Queueing network model of the multicomputer system, (c) Markov model.

of the queueing network for state 1c4c provides the throughput for two classes of customers: a class that corresponds to Process 1c and a class that corresponds to Process 4c. The first throughput is the transition rate from state 1c4c to state 234c while the second is the transition rate from state 1c4c to state 1c. The transition rates from other states that do not give rise to any resource contention can be easily computed from the total service demand at the resources used by the process. Thus if $p$ is the vector of steady state probabilities and $T$ is the transition rate matrix then the steady state probabilities can be computed by solving the set of linear equations: $pT = 0$.

A CLP-BNR [1] program was written for solving the two-level model for the concurrent system of Fig. 1. Note that the system throughput is obtained as $p_1\mu_1$ where $p_1$ is the steady state probability of the system being in state 1 and $\mu_1$ is the rate of leaving state 1. Analysis of sensitivity of performance to characteristics of different system components leads to the identification of a number of key components. These key components have a strong effect on system performance and need special attention during system upgrading or redesign. As an example the sensitivity of the system throughput to two parameters that correspond to the execution times of two tasks were computed. The results are presented in Table 2. The variation in each of these parameters is captured in an interval shown in the fourth column of the table whereas the corresponding change in throughput is recorded in the fifth column. The system performance seems to be fairly sensitive to the CPU demand of Process 1: a change of approximately 29.5% in system throughput can be expected if the execution time for Process 1 is changed from 250 to 500 msec. The message processing time corresponding to inter-process communication as captured in the CPU demand of Process 1c seems to have a relatively smaller impact on performance: only a change of 4.7% in system throughput is observed when this parameter is changed by a factor of 5.

**Table 2.** Effect of changes in parameters for concurrent application

| Case | Process | Parameter varied | Interval for parameter (msec) | Interval for throughput |
|---|---|---|---|---|
| I | 1c | CPU demand | [100, 500] | [0.12634, 0.14623] |
| II | 1 | CPU demand | [250, 500] | [0.10426, 0.18011] |

### 3.2 Model of an EJB Server Implementation

As another example consider the model of an EJB (*Enterprise JavaBeans*) server implementation. The server works as the central scheduler in a distributed, three-tier, client-server architecture. The real system under study is the Kensington

Enterprise Data Mining system [5] whose application server (or scheduler) implements the EJB-1.1 specification [40]. The behavior of a method execution is modeled since it is the most common operation in the system. A detailed description of this execution and the derivation of the model and its approximate solution can be found in [16].

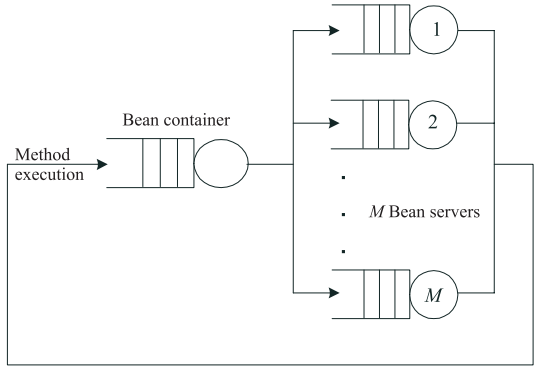


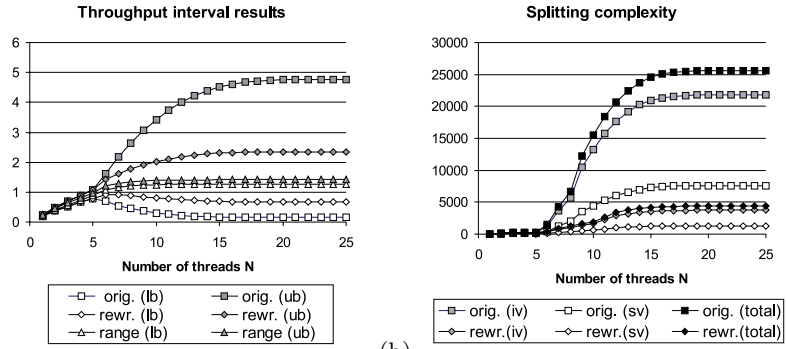

**Fig. 2.** Short-circuited FES sub-network

The model that is used to illustrate the application of interval techniques corresponds to a sub-model of the system described above which is also derived in [16]. This sub-model (shown in Fig. 2) is obtained via the application of the Flow Equivalent Server method (FES) (see [10], for example) in order to reduce the complexity of the original model. Blocking is the critical non-standard characteristic in this network. A client that has completed service in the container node may be blocked under certain circumstances (see [16] for details). In this case blocking time is the time required for the first of the $M$ parallel bean servers to clear its queue in a blocking-after-service discipline. By aggregation of the $M$ parallel servers into a single node, the sub-model is reduced to a two-node queueing network with more complex service times. The blocking behavior is analyzed using an M-state Markov chain that produces the probabilities $\xi_{kn}$ that $n$ out of the $M$ bean servers are busy, given that there are $k$ customers at the parallel servers altogether. Based on these probabilities, load dependent service rates for the two nodes of the reduced queueing network and subsequently the throughput of the sub-model are computed.

Due to restricted access to the real system, accurate measurements to obtain the service rate parameters for various components of the system cannot be performed. In order to capture this type of parameter uncertainty the timing parameters of the model are replaced by intervals laid around parameter estimates obtained via expert guess. In the following experiments we use parameter values taken from evaluations described in [16]: $M = 6$ bean servers per container,

the estimates for the service rate of the bean servers $\mu$ and the mean service time of the outer server $m_1$ are subject to uncertainty. Thus, these parameters are characterized by the intervals $\mu^{(iv)} = 1/4.1 \pm 5\% = [0.2317, 0.2561]$ and $m_1^{(iv)} = 0.4 \pm 5\% = [0.38, 0.42]$.

The computational steps for the solution (implemented in the programming language C) of the sub-model throughput, given that the service demand parameters $\mu$ and $m_1$ are intervals have been adapted to interval arithmetic. During this adaptation several equations have been re-formulated to decrease overestimation due to the dependency problem. Moreover, monotonicity properties of intermediate computation steps have been exploited to further reduce the amount of overestimation. The details of this adaptation process can be found in [23]. Fig. 3(a) depicts throughput intervals for thread populations $N = 1, \ldots, 25$. Three interval plots are shown: the actual throughput range obtained via interval splitting ('range'), throughput intervals obtained by a direct interval adaptation of the original solution algorithm without modifications ('orig.'), and results obtained by an interval adaptation that uses re-written formulae to reduce the effect of the dependency problem. The plots denoted by 'orig.' and 'rewr.' are computed without interval splitting. It can be seen in this figure that using the original expressions, the throughput interval is much more overestimated than the throughput interval obtained using the rewritten expressions.




(a) (b)

**Fig. 3.** Comparison of original and rewritten expressions: (a) throughput interval results and (b) computational complexity: number of necessary interval evaluations to obtain interval results with desired accuracy $\epsilon = 10^{-2}$ for varying population $N$.

Unfortunately, due to the dependency problem, even the throughput intervals obtained by using the rewritten expressions are more than 10 times as wide as the actual range of the throughput (the innermost intervals in Fig. 3(a)). Thus, in both cases, interval splitting has to be applied to obtain reasonable tight enclosures of the throughput range. However, even though interval splitting may

be necessary for both, original as well as optimized (w.r.t. interval computation) expressions, the computational effort is significantly reduced when using the rewritten formulae. Fig. 3(b) depicts the computational complexity required to obtain the range for the throughput with an accuracy of $\epsilon < 10^{-2}$. To obtain the range of the throughput, a new splitting approach is used, which performs both, evaluations with interval parameters as well as evaluations with single value parameters (see [24] for details). For each version of the expressions (original and rewritten), three different plots are shown: the number of necessary interval evaluations (iv), the number of necessary single value evaluations (sv), and the weighted sum $iv + sv/2$ (total) — the computational complexity for an interval evaluation is approximately twice as high as for a single value evaluation. Note that using the rewritten expressions decreases the number of necessary evaluations in the interval splitting algorithm by a factor of more than 5.

## 4 Histogram Parameters

### 4.1 General Approach: VU-Lists

The interval parameter approach discussed in the previous sections can be extended to characterize variabilities and uncertainties in model parameters. Suppose we have $K$ parameters $d_1, \ldots, d_K$, describing a model. Variabilities as well as uncertainties may be captured by the $K$-dimensional histogram (VU-list) consisting of $I$ parameter vectors of dimension $K$:

$$(D_1^{(1)}, \ldots, D_K^{(1)}) : p^{(1)}, \ \ldots \ , \ (D_1^{(I)}, \ldots, D_K^{(I)}) : p^{(I)}, \tag{7}$$

where $\sum_{i=1}^{I} p^{(i)} = 1$. The $i^{\text{th}}$ entry in that list consists of a parameter vector $(D_1^{(i)}, \ldots, D_K^{(i)})$ and an associated probability of occurrence $p^{(i)}$. Every parameter $D_k^{(i)}$ is an interval $D_k^{(i)} = [\underline{d}_k^{(i)}, \overline{d}_k^{(i)}]$. Parameters without uncertainty are represented by intervals of zero width. In the following subsection we discuss how such VU-lists may be obtained.

### 4.2 Sources for VU-Lists

*Combination of 1-Dimensional Histograms:* if several model parameters are specified as stochastically independent one-dimensional histograms, these can be combined to obtain a VU-list. Suppose we have $K$ parameter histograms, and the number of histogram bins of the $k$-th parameter histogram is denoted by $m_k$. The histograms can be combined to a $K$-dimensional VU-list of the length $I = \prod_{k=1}^{K} m_k$.

*Multi-Dimensional Histograms:* if the parameter values of different parameters are not independent, they may be specified as a $K$-dimensional histogram which directly corresponds to a VU-list.

*Parameter Clusters:* often, using a multi-dimensional histogram based on measurements counted in a predefined grid is inefficient because there may be many histogram bins with low probabilities of occurrence. If similar measurement results appear in groups of measured points, the analyst may try to construct parameter clusters using clustering techniques as described for example in [13]. Each cluster can be characterized by a $K$-dimensional box, i.e. a combination of parameter intervals that corresponds directly to a VU-list representation.

*Distribution Approximation:* if there is reason to characterize model parameters as certain probability distributions, VU-lists can be used to represent an approximation of these distributions. Stochastically independent parameter distributions may be approximated by one-dimensional histograms using equidistant supportive points. The supportive points may also be chosen in a way that the obtained probabilities $p_{k,i}$ are all equal. One-dimensional parameter histograms obtained for different resources via approximation of parameter distributions may be combined to VU-lists as described above. If independence of distributions for multiple parameters cannot be guaranteed, a multi-dimensional probability distribution has to be used which may directly be approximated by a corresponding VU-list.

### 4.3 Adaptation of Analysis Techniques

Existing analytic techniques have to be adapted to handle the proposed parameter characterization. The structure of the proposed VU-lists (lists of vectors with interval parameters and associated probabilities of occurrence) calls for the following general algorithm adaptation strategy:

1. Adapt the existing technique to handle interval parameters.
2. Solve the model parameterized by a VU-list for every component of that list. This produces an intermediate interval result for every VU-list component. The intermediate results have got the associated probabilities of occurrence corresponding to the probabilities of their respective input parameter combinations.
3. The list of intermediate results corresponds to separate analyses for every variability phase (i.e. VU-list component) of the model. Using the associated probabilities of occurrence, these intermediate results can be combined to aggregated results that provide a better view of the overall behavior.

Techniques for Step 1 are discussed in Section 2. Step 2 is straight forward and should not require any further discussion. The next subsection summarizes some techniques for Step 3. A more detailed description can be found in [21].

### 4.4 Combination of Intermediate Results

As discussed in Section 4.3, using an analytic performance evaluation algorithm with the capability to handle intervals as input parameters, a set of performance

measure intervals for every VU-list component is produced. Suppose that for performance measure $X$ we have $I$ intermediate interval results with associated probabilities of occurrence. I.e., $X$ is characterized by the list:

$$X^{(1)} = [\underline{x}^{(1)}, \overline{x}^{(1)}] : p^{(1)}, \ \ldots \ , \ X^{(I)} = [\underline{x}^{(I)}, \overline{x}^{(I)}] : p^{(I)}. \tag{8}$$

Without additional assumptions on the distribution of the performance measure within the intermediate intervals $X^{(i)} = [\underline{x}^{(i)}, \overline{x}^{(i)}]$, $i = 1, \ldots, I$, we can only rely on the information that $x \in X^{(i)}$ with probability $p^{(i)}$, $i = 1, \ldots, I$. Based on this information we can construct bounds for the mean values for performance measures by weighted summation of the intermediate intervals using the probabilities of occurrence as weights:

$$M(X) = [\underline{\mu}(X), \, \overline{\mu}(X)] = \sum_{i=1}^{I} p^{(i)} X^{(i)} = \left[ \sum_i p^{(i)} \underline{x}^{(i)}, \ \sum_i p^{(i)} \overline{x}^{(i)} \right]. \tag{9}$$

Denoting the actual mean value of the performance measure $X$ by $\mu(X)$, it holds that $\mu(X) \in M(X)$.

The list of intermediate results can also be aggregated to reveal information about the distribution of the performance measure of interest to the analyst. Without assumptions for the distribution of the performance measure within the intermediate interval results it is impossible to construct a probability density function (pdf), because within an interval $X^{(i)} = [\underline{x}^{(i)}, \overline{x}^{(i)}]$ infinitely many pdfs $f$ could produce the associated probability of occurrence $p^{(i)} = \int_{\underline{x}^{(i)}}^{\overline{x}^{(i)}} f(x)\, dx$. However, as described by Berleant [2], the intermediate intervals can be used to construct bounds for the cumulative distribution function (cdf). For each intermediate interval $X^{(i)} = [\underline{x}^{(i)}, \overline{x}^{(i)}]$ consider the two extreme situations where (a) the whole weight $p^{(i)}$ is concentrated at the lower bound $\underline{x}^{(i)}$ or (b) the whole weight $p^{(i)}$ is concentrated at the upper bound $\overline{x}^{(i)}$. Using all lower endpoints of the intermediate intervals, an upper bound for the cdf can be constructed. The upper endpoints yield a lower bound for the cdf. We denote the cdf for the performance measure $X$ by $F_X$, and we denote the characteristic function of a set $A \subset \mathbb{R}$ by $\chi_A(x)$. The lower (upper) bound for $F_X$ constructed as described above is:

$$\underline{F}_X \ = \ \sum_{i=1}^{I} p^{(i)} \cdot \chi_{[\overline{x}^{(i)}, \infty)} \quad \text{and} \quad \overline{F}_X \ = \ \sum_{i=1}^{I} p^{(i)} \cdot \chi_{[\underline{x}^{(i)}, \infty)}. \tag{10}$$

The cdf bounds can be used to obtain probability intervals $P_X^{(\mathrm{iv})} = [\underline{p}_X^{(\mathrm{iv})}, \, \overline{p}_X^{(\mathrm{iv})}]$ which contain the corresponding probabilities $P_X$ associated with the actual performance measure distribution $F_X$. Given an interval $A = [a, b]$, we have:

$$P_X(X \in A) \in P_X^{(\mathrm{iv})}(X \in A) = [\max\{0, \, \underline{F}_X(b) - \overline{F}_X(a)\}, \ \overline{F}_X(b) - \underline{F}_X(a)]. \tag{11}$$

Fig. 4 shows the construction of $P_X(X \in A)$ and of $\overline{p}_X^{(\mathrm{iv})}(X \in A)$. In this example, we have $P_X^{(\mathrm{iv})}(X \in A) = [0, \overline{F}_X(b) - \underline{F}_X(a)]$.

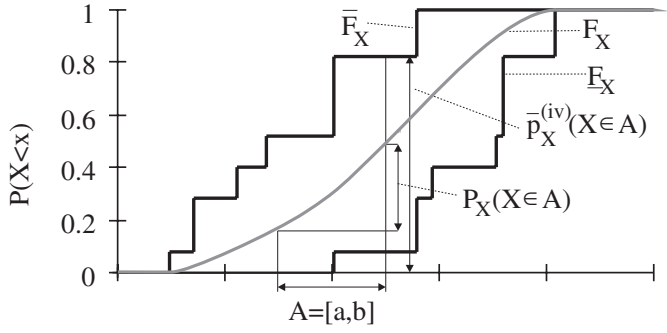



**Fig. 4.** Sketch of probability interval construction.

In general, we have no information about the distribution of the performance measures within the intermediate interval results. However, to produce approximations for the expected performance measure mean values as well as for the distribution of the performance measures we can make the simplifying assumption that the results are uniformly distributed within the intermediate intervals.

If we assume that within each interval $X^{(i)} = [\underline{x}^{(i)}, \overline{x}^{(i)}]$ the results are uniformly distributed, the overall mean value is obtained as the weighted sum of the interval midpoints:

$$\mu(X) \;=\; \sum_{i=1}^{I} p^{(i)} \frac{\underline{x}^{(i)} + \overline{x}^{(i)}}{2}. \tag{12}$$

If none of the intermediate intervals is of zero width, the uniformity assumption can be used to construct an approximation for the probability density function (pdf) describing the respective performance measure distribution. Assuming uniform distribution of values within the performance measure intervals, an approximation of the pdf $f_X^{(app)}$ of the performance measure of interest may be obtained via weighted summation:

$$f_X^{(app)}(x) = \sum_{i=1}^{I} \frac{p^{(i)}}{\overline{x}^{(i)} - \underline{x}^{(i)}} \cdot \chi_{X^{(i)}}(x). \tag{13}$$

A number of examples using this approach of histogram-based parameter characterization can be found in [17] and in [30].

## 5 Conclusions and Future Work

Computer systems are becoming more and more complex putting a greater demand on performance models for understanding their behavior as well as for

prediction of their performance. Because of its relatively lower cost in comparison to simulation and benchmarking analytic modeling is a popular performance evaluation method. Existing modeling tools do not consider variabilities and uncertainties in system parameters. In early stages of system design exact values may not be known for each parameter of interest, but a range of values may be estimated by the designer. System operations are sometimes characterized by phases with device demands varying from one phase to another. Moreover, using bounding techniques in one layer of a hierarchical performance model may produce uncertainty on another layer. We have proposed methods for the characterization and performance analysis for systems that exhibit such parameter uncertainties and variabilities.

Uncertainty without any variability is characterized by associating intervals with parameters. Variability with uncertainty in a parameter value is proposed to be characterized by a histogram. Based on such a characterization of input parameters this research concerns the adaptation of existing analytic performance evaluation algorithms to handle interval and histogram parameters. An interval arithmetic-based technique is proposed for the adaptation of existing algorithms to interval parameters. Depending on the widths of the parameter intervals some inaccuracy may result. A technique called interval splitting is described for improving the accuracy of the results. These techniques are observed to demonstrate a reasonable degree of scalability as long as the number of parameters associated with uncertainty or variability is not very high.

Two examples illustrate the application of the proposed methods: a model of a concurrent system that requires a two level modeling technique employing both queueing networks and Markov chains is presented for handling parameter uncertainties and for studying the sensitivity of system performance to variation in key parameters. As another example that illustrates the effectiveness of the proposed parameter characterization techniques we present an interval version of a queueing network model with blocking modeling an Enterprise JavaBeans server implementation.

We believe that an interval-based analytic performance evaluation technique is useful in the context of software performance engineering. The low cost associated with the model makes it attractive at earlier stages of system design and implementation. Uncertainties in parameter values are expected to exist in these earlier stages. As more and more components are implemented more accurate estimates (narrower intervals) of parameters can be used to provide tighter intervals for performance metrics of interest.

Future research can span a number of different directions. These include the adaptation of other analytic models such as Stochastic Petri Nets to interval parameters, and application of the techniques to large industry scale problems. Moreover, interval-based parameters could be especially useful for software performance models derived from UML (unified modeling language) diagrams [3], like for example discussed in [7] or [12]. Parallelizing the algorithms for a network of workstations is also worthy of future research.

**Acknowledgements.** This research was supported by the Telecommunications Research Institute of Ontario, a center of excellence funded by the province of Ontario and by the Natural Sciences and Engineering Research Council of Canada. We would also like to thank Catalina M. Ladó for her participation in preparing the EJB example presented in Subsection 3.2.

## References

1. Bell Northern Research Ltd., Computing Research Laboratory, P.O. Box 3511, Station A, Ottawa, Canada. *CLP-BNR Prolog User Guide*, 1993.
2. D. Berleant, "Automatically Verified Arithmetic on Probability Distributions and Intervals." *Applications of Interval Computations*, pp. 227–244. Kluwer Academic Publishers, Dordrecht, 1996.
3. G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*. Addison-Wesley, $3^{rd}$ Ed., 1999.
4. J.P. Buzen, "A Modeler's View of Workload Characterization," *Workload Characterization of Computer Systems and Computer Networks*, pp. 67–72, North-Holland, 1986.
5. J. Chattratichat, J. Darlington, Y. Guo, S. Hedvall, M. Kohler, and J. Syed, "An Architecture for Distributed Enterprise Data Mining." *Proc. $7^{th}$ Int. Conf. on High Performance Computing and Networking Europe (HPCN Europe'99, April 12–14, Amsterdam, The Netherlands)*, April 1999.
6. E.G. Coffman, Jr. and P.J. Denning, *Operating System Theory*. Prentice-Hall, 1973.
7. V. Cortellessa and R. Mirandola, "Deriving a Queueing Network based Performance Model from UML Diagrams," *Proc. $2^{nd}$ Int. Workshop on Software and Performance (WOSP 2000, September 17–20, Ottawa, Canada)*, Sept. 2000.
8. P. Dauphin, F. Hartleb, M. Kienow, V. Mertsiotakis, and A. Quick, "PEPP: Performance Evaluation of Parallel Programs, User's Guide – Version 3.3," Technical Report 17/93, IMMD VII, Univ. of Erlangen-Nürnberg, Germany, Sept. 1993.
9. A.J.C. van Gemund, "Compile-time Performance Prediction with PAMELA," *Proc. $4^{th}$ Int. Workshop on Compilers for Parallel Computers*, Delft, the Netherlands, pp. 428–435, Dec. 1993.
10. P.G. Harrison and N.M. Patel, *Performance Modelling of Communication Networks and Computer Architectures*. Addison-Wesley, 1993.
11. F. Hartleb and V. Mertsiotakis, "Bounds for the Mean Runtime of Parallel Programs," *Proc. Sixth Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, pp. 197–210, 1992.
12. F. Hoeben, "Using UML Models for Performance Calculation," *Proc. $2^{nd}$ Int. Workshop on Software and Performance (WOSP 2000, September 17–20, Ottawa, Canada)*, Sept. 2000.
13. R. Jain, *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, Inc., New York, 1991.
14. A. Kaufmann and M.M. Gupta, *Fuzzy Mathematical Models in Engineering and Management Science*, North Holland, Amsterdam, 1988.
15. E.D. Lazowska, J. Zahorjan, G.S. Graham, and K.C. Sevcik, *Quantitative System Performance – Computer System Analysis Using Queueing Network Models*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1984.

16. C.M. Lladó and P.G. Harrison, "Performance Evaluation of an Enterprise JavaBean Server Implementation," *Proc. $2^{nd}$ Int. Workshop on Software and Performance (WOSP 2000, September 17–20, Ottawa, Canada)*, pages 180–188, Sept. 2000.
17. J. Lüthi, S. Majumdar, and G. Haring, "Mean Value Analysis for Computer Systems with Variabilities in Workload," *Proc. IEEE Int. Computer Performance & Dependability Symposium (IPDS'96), Urbana-Champaign, IL, USA, September 4–6, 1996*, pp. 32–41, IEEE Computer Society Press, Los Alamitos, Sept. 1996.
18. J. Lüthi, S. Majumdar, G. Kotsis, and G. Haring, "Performance Bounds for Distributed Systems with Workload Variabilities & Uncertainties," *Parallel Computing*, vol. 22, no. 13, pp. 1789–1806, Feb. 1997.
19. J. Lüthi and G. Haring, "Fuzzy Queueing Network Models of Computing Systems," *Proc. $13^{th}$ UK Workshop on Performance Engineering of Computer and Telecommunication Systems*, Ilkley, UK, July 1997.
20. J. Lüthi, "Interval Matrices for the Bottleneck Analysis of Queueing Network Models with Histogram-Based Parameters," *Proc. IEEE Int. Computer Performance & Dependability Symposium (IPDS'98, September 7–9, 1998, Durham, NC, USA)*, pp. 142–151, IEEE Computer Society Press, Los Alamitos, Sept. 1998.
21. J. Lüthi. "Histogram-Based Characterization of Workload Parameters and its Consequences on Model Analysis," *Proc. MASCOTS'98 Workshop on Workload Characterization in High-Performance Computing Environments, July 19–24, 1998, Montreal, Canada*, pp. 1/52 – 1/64, July 1998.
22. J. Lüthi and G. Haring, "Mean Value Analysis for Queueing Network Models with Intervals as Input Parameters," *Performance Evaluation*, vol. 32, no. 3, pp. 185–215, April 1998.
23. J. Lüthi and C.M. Lladó, "Interval Parameters for Capturing Uncertainties in an EJB Performance Model," submitted for publication, preprint available as Technical Report No. 2000-08, Fakultät für Informatik, Universität der Bundeswehr München, D-85577 Neubiberg, Germany, Dec. 2000.
24. J. Lüthi and C.M. Lladó, "Splitting Techniques for Interval Parameters in Performance Models," submitted for publication, preprint available as Technical Report No. 2000-07, Fakultät für Informatik, Universität der Bundeswehr München, D-85577 Neubiberg, Germany, Dec. 2000.
25. S. Majumdar, "Interval Arithmetic for Performance Analysis of Distributed Computing Systems," *Proc. Canadian Conf. on Electrical and Computer Engineering*, Quebec, Canada, Sep. 1991.
26. S. Majumdar, J.E. Neilson C.M. Woodside, and D.C. Petriu, "Robust Box Bounds: Network Performance Guarantees for Closed Multiclass Queueing Networks with Minimal Stochastic Assumptions," *Proc. Infocom'92*, pp. 2006–2016, May 1992.
27. S. Majumdar and R. Ramadoss, "Interval-Based Performance Analysis of Computing Systems," *Proc. MASCOTS'95*, pp. 345–351, IEEE Computer Society Press, Jan. 1995.
28. S. Majumdar, "Application of Relational Interval Arithmetic to Computer Performance Analysis," *Constraints (special issue "Interval Constraints")*, Vol. 22, No. 3, March 1997.
29. S. Majumdar and C.M. Woodside, "Robust Bounds and Throughput Guarantees for Closed Multiclass Queueing Networks," *Performance Evaluation,* vol. 32, no. 2, pp. 101–136, March 1998.
30. S. Majumdar, J. Lüthi, R. Ramadoss, and G. Haring, "Performance Analysis of Computing Systems with Interval Parameters," Technical Report, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada K1S 5B6, 2001.

31. R.E. Moore, *Interval Analysis*. Prentice-Hall, 1966.
32. A. Neumaier, *Interval methods for systems of equations*. Cambridge University Press, Cambridge, 1990.
33. D.C. Petriu, S. Majumdar, J. Lin, and C.E. Hrischuk, "Analytic Performance Estimation of Client-Server Systems with Multithread Clients," *Proc. MASCOTS'94*, Durham, NC, USA, Jan. 1994.
34. R. Ramadoss, *Interval-Based Performance Analysis of Computing Systems*. Master's thesis, Ottawa-Carleton Institute for Electrical Engineering, Faculty of Engineering, Department of Systems and Computer Engineering, Carleton University, Ottawa, Ontario, Canada, Oct. 1994.
35. H. Ratschek and J. Rokne, *New Computer Methods for Global Optimization*. Ellis Horwood, 1988.
36. J.A. Rolia, M. Starkey, and G. Boersma, "Modeling RPC Performance," *Proc. CASCON'93*, pp. 677–690, Oct. 1993.
37. A. Silberschatz, J.L. Peterson, and P. Galvin, *Operating Systems Concepts*. Addison-Wesley, 3rd edition, 1992.
38. S. Skelobe, "Computation of Rational Interval Functions," *BIT*, vol. 14, pp. 87–95, 1974.
39. C.U. Smith, *Performance Engineering of Software Systems*. Addison-Wesley, Reading, MA, 1990.
40. Sun Microsystems. Enterprise JavaBeans 1.1 Architecture. http://java.sun.com/products/ejb, 1999.
41. R. Suri, "A Concept of Monotonicity and Its Characterization for Closed Queueing Networks," *Operations Research*, vol. 33, pp. 606–624, 1984.
42. A. Thomasian and P.F. Bay, "Analytic Queueing Network Models for Parallel Processing of Task Systems," *IEEE Trans. on Computers*, vol. 35, no. 12, pp. 1045–1054, Dec. 1986.

# The Simalytic Modeling Technique

Tim R. Norton

Simalytic Solutions, LLC
711 Beacon Ridge Drive
Colorado Springs, CO 80906
tim.norton@simalytic.com

**Abstract.** While the advent of desktop and departmental servers started the popularity of multiple-tier application designs (2-tier, 3-tier, n-tier, etc.), the e-commerce explosion has forced many more applications into a multiple-tier implementation using the Internet. However, the popularity of providing web access to new and existing applications has created significant problems for the performance analyst and the capacity planner. This chapter introduces "Simalytic" (**Sim**ulation/An**alytic**) Modeling,[1] a new approach to modeling the performance of transaction based client/server applications specifically addressing the problems of modeling client/server applications for capacity planning. This technique uses a general purpose simulation tool as an underlying framework, combined with analytic results to represent individual nodes, to predict the capacity requirements in an enterprise model. It combines modeling tools already in place for the individual systems (platform-centric and general purpose) to address heterogeneous environments with a modeling framework that connects the parts of an application.

## 1 Objectives

Upon completion of this chapter, the reader will:

- Understand the concepts of a hybrid model, using both simulation and analytic queuing theory models, applied to performance analysis and capacity planning of client/server applications.
- Understand how to create a Simalytic Model to analyze the performance and capacity requirements of client/server transactions.
- Understand the advantages to the Simalytic approach (reduced model complexity, improved model execution times, application of appropriate node-level modeling tools, and application of the Spiral Modeling Methodology™).

[1] Simalytic™, Simalytic Modeling™, Simalytic Modeling Technique™, Simalytic Enterprise Modeling™, Simalytic Function™, and Spiral Modeling Methodology™ are trademarked by Tim R. Norton. All other trademarked names and terms are the property of their respective owners.

R. Dumke et al. (Eds.): Performance Engineering, LNCS 2047, pp. 222–238, 2001.

## 2 Introduction

Complex application designs utilize the features and services of different types of computers running different operating systems connected by a variety of communication network techniques [1, 2]. Performance and capacity planning considerations for client/server applications require that we understand how to select the right systems at each level and, once selected, that we can insure those systems are the right size. If any system is too small then the whole application will fail. If they are too big, the cost of running the application may exceed the revenue it generates. Neither is a very attractive situation.

Capacity planning has traditionally been focused on determining if a given system has "enough capacity" to service an application. Today, planning the capacity of large computer installations with multiple systems requires an understanding of not only the operating systems, the platforms, the clients, the servers, the networks, the transaction systems, etc., but also the relationships between them. Once those relationships are defined and understood, the application's performance can be assessed against the business objectives and goals. Projected business volumes are then modeled to predict the capacity required to meet those goals at future volumes. Instead of planning the capacity of individual systems, the application responsiveness needs to be predicted across the entire enterprise. Only then can the true capacity requirements be identified.

There are many modeling tools and techniques that address both performance and capacity for each of the systems in today's multi-platform environment [3, 4]. The Simalytic™ Modeling Technique provides a bridge across these existing tools to allow the construction of an enterprise level application model that takes advantage of models and tools already in place for planning the capacity of each system. The advantages of using the Simalytic Model Technique are:

**Rapid Analysis:** A Simalytic model can be constructed with minimal effort for each node model. Analysis of the application using this high level model allows additional effort to be applied only to the nodes in the critical-path areas, saving time and effort by avoiding areas with minimal impact on application responsiveness.

**Spiral Methodology:** Simalytic Modeling provides a mechanism to promote the exchange of information between the users, the developers and the modelers. As more information becomes available, the application model is refined. Early assumptions can be replaced with the results from more sophisticated tools as more details become available. This spiral approach allows modeling earlier in the design phase as well as after implementation.

**Reuse:** Simalytic Modeling provides for the direct incorporation of existing tools and techniques into the high level model. The investment in time, effort and money expended for training on the tools used for existing system models is preserved. It is easier to get someone who has built a number of models of a system, using a tool they are well trained on, to do another model, than it is to start over in different tool.

**Distributed Model Development:** Simalytic Modeling provides easy distribution of modeling activities to multiple people or organizations. Node models can easily be "sub-contracted" to the modelers most familiar with those tools and systems.

**Applicable Tools:** Using the most applicable tool for each node of the Simalytic Model increases both the speed (construction and execution) and the accuracy of the model. For example, the network component can be assumed constant or modeled

with greater detail if the modeler determines that the network is a major component of response time. If the simulation tool does not provide the level of complexity needed, then a specialized network modeling tool can be used to create network nodes for the whole Simalytic Model or just for a critical part of the network between two servers.

# 3 Background Topics

Before discussing Simalytic Modeling, it is important to quickly review the background topics that support Simalytic Modeling. The review will provide for a common basis in the discussion of the implementation. Additional information on the following topics is available in [5-7] and in the references cited in those works.

## 3.1 Capacity Planning

The capacity of a system can be measured many different ways, depending on the business the system supports. Generally, the way a system is measured centers around the performance of one or more of the applications it supports. The system "has enough capacity" if everything is getting done when it is needed. Capacity planning is making decisions about the resource requirements of a computer system based on the forecasting of application performance using the goals of the business. What do we have to buy and when do we have to buy it to make sure that the applications that run the business perform at the level required to insure that the business succeeds?

## 3.2 Transaction Based Applications

Transaction processing allows the end-user to enter a relatively small independent unit of work and receive some information as a response in near real-time. Transactions include business (entering an order at a terminal), database (SQL commands), or interactive (keystrokes followed by a carriage-return). Transactions can be counted to establish load (e.g. arrival rate) and measured to establish performance (e.g. response time). The responsiveness of the transactions determine if that application meets the needs of the business. Projected business volumes are then modeled to predict the capacity required to meet the business goals at those volumes.

## 3.3 Client/Server Environments

The term "client/server" means that some application component is requesting service from another application component. These components may, or may not, be on different physical computing hardware or under different operating systems. Each application component is associated with a system (hardware and operating system) with some form of communication link to the other application components.

Modeling in this environment is a challenge because each of the systems requires a different knowledge base and expertise [2, 8]. None of the systems can be modeled independently because the transaction arrival rate for one system may be dependent

on the response times of the others. The responsiveness of one part of the model will have an impact on the other two. The use of different data collection utilities and different modeling tools on each of the different platforms greatly increases the complexity of modeling the application.

### 3.4 Modeling Capacity Projections

Capacity planning has always relied on modeling because of the need to predict future requirements. A capacity planner can analyze the workloads and make predictions based on experience or simple trending. Models can be constructed to understand how an application functions without predicting future performance. Capacity Modeling is the use of models to predict capacity requirements based on performance expectations.

### 3.5 Response Time Modeling

As applications move towards being transaction based, the definition of application performance becomes centered around transaction response time. Modeling the response time then becomes crucial to the ability to predict the future performance of that application and plan the required capacity.

### 3.6 Modeling Tools

In addition to the choice between analytic and simulation tools, the capacity planner or performance analyst has the choice between platform-centric and general purpose tools. The basic difference between these two groups is the problem set the tools were designed to address.

### 3.7 Platform-Centric

Platform-centric means the tools contain detailed information about the platform, but do not allow more than one platform to be modeled at a time. For example, they include information about the processors for each vendor system, task scheduling and I/O prioritization by operating system release, and seek and rotational times for various disks. Platform-centric models are generally easier to build because they are made of large component "building blocks," and the relationships between them, predefined in the tools. However, these tools cannot be used to model an environment not built into the tool. Platform-centric tools are *generally* implemented using analytic modeling techniques and process performance data collected from running systems.

### 3.8 General Purpose

General purpose means the tool contains the features to allow the user to model almost anything, but with little or no "built-in" understanding of any given computer

platform. These tools are used to model more than just the hardware, including application design, traffic flow and communications protocols. System components are modeled using either a sub-model to implement the underlying architecture or a pre-determined delay value. Although many general purpose tools provide libraries of sub-models for a variety of systems and devices, they generally do not provide the required level of granularity. Building the relationships between the submodels requires an in-depth understanding of all of the submodels used. General purpose tools are *generally* implemented using simulation modeling techniques.

## 4 The Simalytic Modeling Methodology

What is Simalytic Modeling? Simalytic™ Modeling (from **Sim**ulation and An**alytic**) is a hybrid modeling technique that uses a general purpose simulation modeling tool as a underlying framework with the results of an analytic modeling tool to represent the individual nodes or systems. A Simalytic Model creates an enterprise level model to predict the capacity requirements of an application executing on heterogeneous computer systems. A discussion of the industrial changes that provided the impetus for Simalytic Modeling are available in [5]. Details of the mathematical foundation and validation of the technique are available in [6] and [7].

There are two key differences between the existing modeling tools and the Simalytic Modeling Technique. The first is the ability to use the results from not only different tools, but different modeling techniques altogether, as submodels within an enterprise model. The second is the ability to use the results from tools currently being used to model individual nodes in the system. These differences reduce the time and effort to build an enterprise level application model by using the results from commercially available platform-centric tools or existing detailed application models.

Simalytic Modeling brings together existing performance models (usually platform-centric analytic models) and application information (best expressed as simulations). The analytic models rely on averages, such as average response time, average service time and average arrival rate. These models are generally more efficient to execute than simulation models, but, because of the use of averages, variability generally causes the accuracy to decrease as the data collection interval increases. Simalytic Modeling allows the application to be modeled over longer periods of time to understand the application dynamics without increasing this error. The simulation framework allows the use of trace-driven models, which are common in capacity planning. As an additional benefit, the trace data can provide the transaction arrival distributions, which is often a major issue in model construction.

Simalytic Modeling is based on a hybrid technique that allows the models to use the best features of each tool. Submodels allow some part of the model to be replaced with a different model, using a different technique, as long as it provides appropriate functionality; similar to the FESC (flow-equivalent service center) decomposition technique discussed in [9]. A valid model must exist for each system or node to be included in the application enterprise model. The application details must be understood, and consistently defined, at the enterprise level.

An enterprise level model is constructed by starting with a high level simulation model of the application, where each system is a single server. Then, instead of using a pre-defined service time, each server uses a transform function, the Simalytic

Function™, that maps the transaction arrival rates to service times. As the simulation model is run, the service time dynamically adjusts at each node depending on the combination of application transaction arrival rate and other work at the node. The Simalytic Function provides the workload characterization that allows each system to be represented as a single load-dependent server in the simulation framework.

## 5 Steps to Build a Simalytic Model

As with any modeling effort, creating a Simalytic Model requires more than just putting the pieces together in some modeling tool. A substantial amount of information is required about the applications and systems involved. This section presents all of the steps to build a Simalytic Model. The most critical step, Workload Analysis, is a complex and involved process, and only some of the issues involved, those that relate directly to the construction of a Simalytic Model, are discussed here. Other areas, such as calibration techniques for queuing theory tools and features of simulations tools are assumed to be covered in the documentation for the specific tools. Keep in mind that a Simalytic Model requires the same level of analysis once the model has been completed as any other capacity planning model.

The major phases to creating a Simalytic Model are: Workload Analysis, Node Models, Simulation Model, Simalytic Model, and Model Analysis. Each of these phases is discussed in detail in the following sections.

### 5.1 Workload Analysis Phase

In the Workload Analysis Phase the modeler collects information about the application to be modeled. This includes identifying, defining, documenting and measuring the application. This phase includes the same type workload analysis done for system level modeling efforts, but it must be done consistently for all of the systems and includes collecting additional information about the application at the enterprise level. The steps in the Workload Analysis Phase are:

**Identify:** Identify the workload. Because Simalytic Modeling takes an enterprise view of the application, the identification process is at a global level. How a workload is identified will differ between platforms and depends on what database and middleware the application uses. Although workload identification is done on each platform, it cannot be done independently. How the workloads are correlated across the platforms must be considered during identification.

Start by identifying end-user's business transactions. This is often a series of prompts and replies that, taken together, make an single activity such as entering an order. Regardless of the modeling technique used, workload analysis is complex and requires substantial effort. The objective of this step is to define business activities, such as orders entered, in terms of measurable work elements. The workload projections and response time measurements are at the business level. The models are built at the transaction level. There must be a valid mapping between these two levels.

This step must be done in conjunction with developers and end-users. It is a compromise between what end-users would like and what is realistic, considering

how the application works. For example, if new orders and status inquiries use the same transaction, it may not be possible to separate them into different workloads.

**Document:** Document the application topology, such as transaction routings, serial or parallel execution, and client/server architecture (e.g. 2-tier, 3-tier, etc.). The documentation technique used should be whatever best supports the application and has the support of the users and developers, who must maintain the documentation. Therefore, the challenge of this step is to develop a mapping technique from the developer supported documentation to what is required for model construction.

The objective of this step is to produce a topology description of the application that can be easily and accurately translated into a simulation model. When this step is completed, the modeler should be able to track a business transaction through the entire environment (including all splits, protocol translations, routing decisions, etc.).

**Measure:** Measure the workload. Application measurement, at both the business and the system levels, is a key enabler for Simalytic Modeling. The overall Simalytic Model can be calibrated only if the responsiveness of the business transactions can be measured. The node level model can be calibrated only if the IT transactions can be measured. The measurement of the IT transactions is generally already implemented for the node level models currently being done. However, in today's client/server environments, the measurement of the business transaction is very difficult.

In this step, the modeler must determine the ability to measure the application at each system and from the end-user's point-of-view (end-to-end response time). The node level measurement data are usually readily available. The modeler must have additional information about the number, frequency and response times of the business transactions. If the application or the system does not collect the data, the modeler may need to observe the application users and collect the data manually.

The objective of this step is to determine the feasibility of the modeling effort. If adequate measurement data cannot be collected then the modeler must determine if the effort generate enough interest to increase the quality of the measurement data.

**Correlate:** Correlate the workload across systems. The final step of Workload Analysis is to determine the correlation between the workloads at each system. The definition for a workload at one system must mean the same thing at other systems. There cannot be any additional or missing transactions. For example, if a workload is defined as three transactions, then the measurement of it at one system must include all of those transactions that are routed to that system. In addition, it cannot include any other transactions that run on that system not included in the overall definition of the workload. This appears to be a straight-forward requirement, but it becomes complex as the client/server environment grows and applications attempt to reuse functions. A legacy application transaction might be invoked by more than one client/server business transaction to look up customer information. This transaction would need to be included in multiple workloads, which would then cause errors in the model. This situation cannot be resolved without some application modification to enable collecting data about which workload includes each transaction.

The objective of this step is to insure the consistency of the workloads across the entire enterprise model. Such problems need to be identified and resolved with data collection, application changes, model changes or simplifying assumptions.

### 5.2 Node Models Phase

In the Node Models Phase, the modeler models all of the systems supporting the application. This phase includes the same type of modeling done for system level modeling efforts, but coordinates the node level models to integrate with the additional information about the application from the enterprise point-of-view. The steps in the Node Models Phase are:

**Build:** Build a model of each node. Building a model of each node used by the application is not significantly different from any existing system level modeling efforts. Whatever tool is currently used to model each system should be used for that node in the overall model. The major difference is that the workload definitions used in the node models are those developed in the prior Workload Analysis phase. Only the application of interest should be modeled as an identifiable workload with response time predictions. All other activity at each node should be included only to understand resource usage and correctly influence the workload of interest.

The objectives of this step are to build a model of each system and to take advantage of any existing models. Although the workload definition may change, the processes already in place to collect measurement data and calibrate the models, and possibly some of the actual models, for any of the nodes can be effectively reused.

**Calibrate:** Calibrate the model of each node. The node level models must be valid before continuing. There must be a high degree of confidence in the predictive nature of each of the node models. Because the Simalytic Model will connect the nodes together using the workload definitions, an error or poor results from any one node model can impact the accuracy of the Simalytic Model for the entire application. The calibration techniques used are dependent on the modeling tools. Also, care must be taken to insure that steps taken to calibrate one node do not contradict assumptions made in a different node model.

The objective of this step is to have a solid predictive model for each node that presents a consistent view of the application across all nodes.

**Run:** Run the models. Develop a profile of the application by running each of the node models for a series of arrival rates from very low to very high (either the model saturates or the 'knee' of the response time curve is well established). The actual arrival rates used will depend on the application and should make sense to the users. The arrival rate increment should be as fine as is practical, considering the time and resources required for each execution of the model. The increment does not have to be uniform across the range; use a larger increment when there is little change in the response times and use a smaller increment when there is a large change.

The objective of this step is to establish a response time curve that can be used to extrapolate the response time when presented an arrival rate not modeled.

**Create:** Create a model results table. Create a table of response times and arrival rates for each system for the workload of interest. Other workloads on the system will not be modeled but they are still accounted for in the node level system models. A key assumption is that the other workloads provide a consistent load on the system and thus a consistent level of interference. If this is not true, then the table must be extended to include some external parameters, such as time of day. The response time values must then be based on the combination of those parameters and the arrival rate.

The objective of this step is to characterize the application performance and responsiveness. The information in this table will be used to create the Simalytic Function when the Simalytic Model is constructed.

### 5.3 Simulation Model Phase

In the Simulation Model Phase, the modeler builds an overall model of the application with each of the systems supporting it represented as a node. Information from the Workload Analysis Phase is used to connect each of the systems together in an enterprise view of the application. The steps in the Simulation Model Phase are:

**Build:** Build an overall model. Using the simulation tool of choice, build a model of the application with a single server for each node. This model is defined by the application topology documented in the Workload Analysis stage. It identifies what transactions are routed to which nodes under what circumstances. This model can be built before any of the node level performance data has been collected by making assumptions as to the expected performance at each node. Use the model to identify how sensitive the application is to changes in the performance of any given node. If large variations in service time have only minimal response time impacts, then that node model may be deferred. This approach, an implementation of The Spiral Modeling Methodology™ [10], allows the initial model to be developed quickly and then successively refined as additional node level information becomes available. Each iteration of the spiral increases the accuracy of the application model with minimal effort. Iterations are continued only until the desired level of overall accuracy has been achieved.

The objective of this step is to build a model of the application that represents the overall application behavior across the enterprise.

**Set:** Set the overall model parameters. Set the service time of each node to the best estimate of the application response time or, if available, the lowest response time in the table created in the Node Models Phase. Set each node to have enough servers so that there is ***no*** queuing at any node. How this is done will differ with each of the simulation tools. Generally, it is some type of replication factor within the node. The value must be very high so that there is never any queuing to get a transaction through the node. It is generally not a good idea to create multiple nodes because of the problems that creates with transaction routing. In the enterprise model, the service time and the response time for each server will be the same because the queue time is accounted for in the response time data for the server. (The service time of each node in the simulation model is the response time from the analytic model of that node.)

The objective of this step is to set the simulation model such that the response time at any node can be controlled by the Simalytic Function when it replaces the static service time in a later phase. The simulation model at this stage can be used to verify the application topology and conduct sensitivity analyses of user expectations.

**Calibrate:** Calibrate the overall model. Calibrate the simulation model against the end-user response time for the very low arrival rate and verify that there is no queue time at any of the nodes. Because the response time from the queuing theory tool includes the queue time in the node, any queue time in the simulation model will, in effect, double count the queue time. The simulation tool is being used to control the

flow and routing of transactions, not calculate the queue time. This step should insure that the topology and routing information is correct before continuing.

The objective of this step is to verify that the simulation model accurately reflects both the application topology and the user's response time at very low arrival rates.

### 5.4 Simalytic Model Phase

In the Simalytic Model Phase, the results of the system models are incorporated into the overall model of the application. This phase uses the information from the Workload Analysis and the Node Models Phases to provide the predictive capabilities to the enterprise view model. The steps in the Simalytic Model Phase are:

**Create:** Create the Simalytic Function. Using the table of response times and arrival rates created from the node models, create a Simalytic Function for each node. This can either be a look-up table or a formula established by fitting a curve to the response time data. The details of the function and how it is implemented will depend on the simulation modeling tool used for the overall model framework.

The initial function is implemented by converting the interarrival times over a small number of transactions to an arrival rate and thereby to the associated response time [5]. It is most likely that the initial function will not provide accurate enough results due to issues such as arrival distributions and will need to be enhanced to include additional information. This function is referred to as the Simalytic Function because it not only includes the results of the node models, but also the additional features to select the most appropriate result for each transaction visit. To do this, the arrival rate used will more than likely need to be modified by some technique. One approach is to maintain a rolling average over some number of transactions (small enough to maintain the responsiveness of the model to workload changes but large enough to minimize the influence of isolated instances of very small interarrival times). The size of the averaging window will depend on the variability of the application. Some experimentation may be required to achieve the best results. Another approach would be to examine the node to determine the current number of transactions being serviced or the node utilization, then select an arrival rate more consistent with that node state. The technique chosen is a trade-off between rapid development and model accuracy. Implement the simplest Simalytic Function possible and enhance it as required, and only when necessary, to achieve the desired accuracy.

The objective of this step is to create a Simalytic Function for each node that accurately reflects the application's behavior.

**Replace:** Replace the static service times. Replace the service time for each node with the function created in the prior step. Again, how this is done will differ with each simulation tool. For example, some simulation tools support load dependent servers and the response time values can be entered into the server. However, this technique may not be viable if a more complex Simalytic Function is required and the load dependent server cannot implement a complex function.

The objective of this step is to implement the Simalytic Function in each node of the overall simulation model. The service time used for each transaction visit is the value returned by the Simalytic Function.

**Calibrate:** Calibrate the Simalytic Model. First calibrate it against the prior simulation model for the very low arrival rate to insure the overall model structure is still correct. Next, calibrate it against known end-user response times for known arrival rates. Enhance the Simalytic Function as required to get the required accuracy based on the objectives of the modeling effort.

The objective of this step is to insure that the Simalytic Model provides valid application prediction within the required level of accuracy.

### 5.5 Model Analysis

The next phase uses the Simalytic Model to analyze the application. At this point, the Simalytic Model can be used the same as any model which has been calibrated. How a model is used to answer "what-if" questions is very dependent on the questions themselves. Therefore, the details of the phase will not be discussed here other than to note that all of the phases of constructing a Simalytic Model should be considered as a spiral development process. The completion of each phase may identify additional information or requirements for one of the prior phases. This provides the added benefit of allowing the modeler to implement a quick, simple Simalytic Model and then continue to refine it based on the business requirements and objectives.

## 6 Simalytic Model Implementation

An actual Simalytic Modeling effort is complex and involves creating and calibrating multiple models. In order to focus on the model building process, a simplistic example is used to illustrate the steps presented above.

### 6.1 Implementation Example

This implementation of a Simalytic Model uses a hypothetical client/server environment to illustrate the process. Assume the workload of interest is an Order Entry application on one server. Also, a Shipping application on another server is used by the Order Entry application. The Order Entry user types in the name of an existing customer and gets not only their address, but also shipping information about any recent orders. This may provide better service, but it also causes some number of the Order Entry transactions to be sent to the Shipping server. If the Shipping workload outgrows its system, it can impact the responsiveness of the Order Entry transactions. In addition, growth in the Order Entry workload will impact the Shipping system, but only if the orders are from existing customers. The systems cannot be modeled independently because the service time for one system is dependent on the response time of the other. When the Order Entry transaction rate increases, more transactions are sent to the Shipping server. The increased response time at Shipping will cause the overall response time for those transactions to increase, which will be seen as either longer average response time or reduced through-put for the application.

Figure 1 shows a diagram of this system. The response time is measured from Arrivals to Departures, either through the Shipping node or around it. This example shows how the Simalytic Model connects what is happening in the application on the different servers. If the Order Entry system is modeled by itself, the workload representing the long transactions (those also sent to Shipping) would not reflect the increased response time due to the load at Shipping. Because of the additional application information in the Simalytic Model, the Simalytic Model can adjust the response time in the Shipping server based on the current load, which will then be reflected in the Order Entry transactions that visit the Shipping server.

As with any modeling effort, there must be business objectives to analyze using the model. Assume that the manager of the Order Entry department has requested a model to determine when the Order Entry system will need to be upgraded in order to maintain the required response time of less than 1.7 seconds. The arrival rate is assumed to have a constant increase over the next 18 months and the percent of the Order Entry transactions that also query the Shipping system is 30%. The response time goal for the Shipping system is less than 10 seconds (because these transactions generally do not involve a waiting customer). The objectives of the analysis are to answer two questions: "When does the Order Entry system fail to meet the business response time goal?" and "What must be upgraded to again meet the goal?"

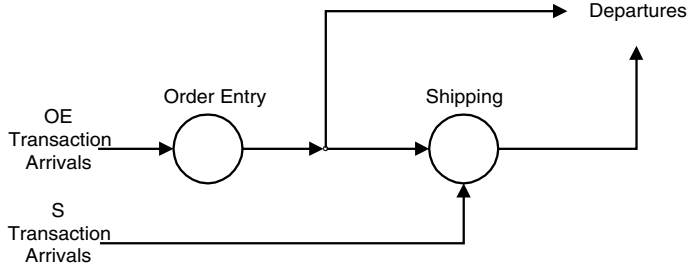



**Fig. 1.** A Simple Enterprise Model. This diagram shows the topology of the example environment. Only those Order Entry transactions for existing customers are routed to the Shipping server

## 6.2 Implementation Process

Although the implementation process for the example follows the steps presented above, many assumptions have been made about the information collection process to simplify the example. The actual implementation uses some inexpensive and readily available tools. The framework is implemented using Simul8 [11], a general purpose simulation modeling tool developed primarily to model manufacturing situations. The performance characteristics of the nodes are developed using the results of the OpenQN analytic tool [9]. OpenQN is a simple Pascal program that reads an input file of workload parameters and produces a report. OpenQN was selected because it is

easy to use, fast and included with Dr. Menascé's book [9]. Finally, the Simalytic Function™ was implemented with Microsoft's Visual Basic because of the OLE (Object Linking and Embedding) interface to Simul8. Simul8 allows OLE to be used for service times and transaction distributions as well as feedback of current state information and model control. Although this interface was one of the main reasons Simul8 was selected for the research, the most current version of Simul8 includes an internal scripting language, Visual Logic, with the same functionality and much faster execution (see [12] for details).

### 6.3 Workload Analysis Example

**Identify:** For this example, there is a single Order Entry transaction, OE, and a single Shipping transaction, S. The OE transactions are the workload of interest. The S transactions need to be included only to the extent they impact the OE transactions. However, some additional information about the S transaction response times is included to illustrate the pit-fall of modeling the systems independently. The S transaction arrival rate is kept constant at 0.1 arrivals per second. Only the OE transaction arrival rate is changed to represent growth in that workload.
**Document**: Refer Figure 1. Assume measurement data shows 30% of the OE transactions are routed to the Shipping server. Also assume that all of the transactions that execute on either server use the same resources. This means that there is no difference on the Order Enter server between the OE transactions that route to Shipping and those that don't. It also means there is no difference between the transactions that execute on the Shipping server (i.e. an OE transaction routed to Shipping consumes the same resources on the Shipping server as an S transaction).
**Measure**: Because this is a hypothetical client/server environment, there are no actual measurements. Therefore, the results of a pure simulation model of the environment will be used to represent these measurements.
**Correlate**: The workload correlation is assumed.

### 6.4 Node Models Example

**Build:** The service times of 0.10 for the Order Entry server and 2.00 for the Shipping server are used in the OpenQN model for each node. (Details of the device level service times and calculated response times are available in [12].)
**Calibrate:** The model of each node is assumed to be calibrated for this example.
**Run:** An OpenQN model was run for each node.
**Create:** The results of the OpenQN model of each node are used to create a table of arrival rate and response time pairs. The models were not run for a large number of arrival rates because there was no significant change in response times. When response times started to change significantly, the arrival rate step was reduced (from 2.00 to 0.25 for the Order Entry server and from 0.05 to 0.02 for the Shipping server) to better define the knee of the response time curve (2.46 seconds at 16.25 arrivals/second to 6.06 seconds at 16.50 arrivals/second for the Order Entry server and 10.65 seconds at 1.30 arrivals/second to 119.96 seconds at 1.42 arrivals/second).

**Simulation Model Example. Build**: The overall simulation model was built using Simul8 as shown in Figure 2.
**Set:** The service times of both the Order Entry and Shipping servers are set to the lowest response times from the Create step above. The replication factor for each server is set to 100 to avoid queuing. The average response times for that model run (ten trials) are 0.56 for OE and 2.01 for S. These response times are very close to the expected values of 0.7 and 2.00, respectively (the expected OE response time is 0.7 because of the routing to Shipping: 0.1+0.3*2.0=0.7) The OE response time is slightly lower because the routing was slightly lower than 30% in most of the trial runs due to the small number of transactions at the low arrival rate.
**Calibrate:** The results of the above model are compared to a pure simulation model, also built in Simul8, to calibrate the model. The service times are the same for both models. The simulation model is also shown in Figure 2.

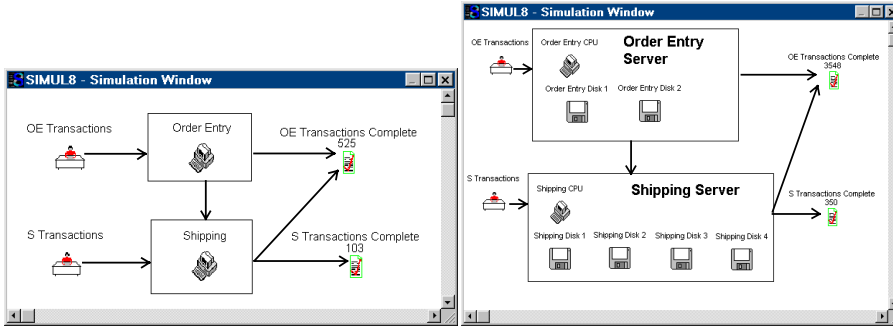



**Fig. 2.** Order Entry Example Models. The Simalytic Model (left) and the equivalent simulation model (right) for the Order Entry example are implemented with the Simul8 simulation tool

**Simalytic Model Example. Create:** Create the Simalytic Function. The Simalytic Function was created using Microsoft's Visual Basic. For this example, it is a very simple function that calculates the rolling average of the interarrival times for each workload and looks up the corresponding response time in a table.
**Replace:** Replace the static service times with the Simalytic Function.
**Calibrate:** The results of the above model are compared to a pure simulation model to calibrate the model as shown in Figure 3. The Simalytic results track the simulation results. The slight under predicting is consistent with the simple implementation of the Simalytic Function and is fully explained in the author's ongoing research.

## 6.5 Model Results

How do the these results relate to the questions asked in the *Implementation Example* section? Figure 3 shows the answer. When the business volume grows to 3.33 OE transactions per second the response time will exceed the goal of 1.7 seconds.

Furthermore, the way to keep OE transactions under the goal is to upgrade the Shipping server rather than the Order Entry server because the OE response time is directly related to the longer Shipping transactions.

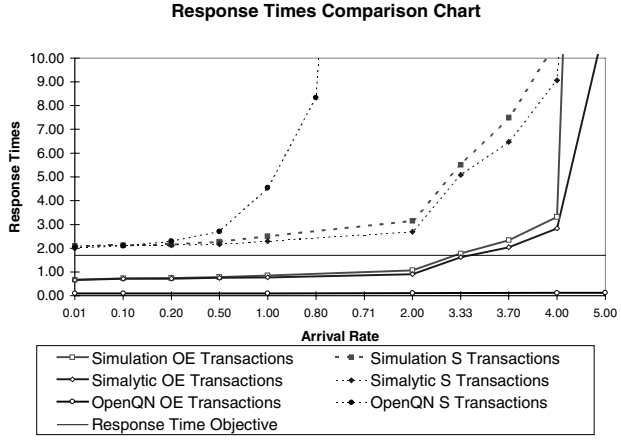



**Fig. 3.** Response Time Comparison. This comparison chart shows the improved prediction accuracy of a Simalytic Model over an analytic single server (OpenQN) model

Figure 3 graphs the response times for the different models. The response times are shown in pairs; one for the OE transactions and one for the S transactions. The OpenQN lines represent only the workload at the respective servers. All of the others include OE transactions sent to Shipping in the OE workload. The Simulation lines represent the pure simulation model and the Simalytic lines represent the Simalytic Model results. Because the S transaction workload arrival rate was kept constant, the Arrival Rate axis is the OE arrival rate except for the OpenQN S Transactions line, which shows what happens at the Shipping server. In all other cases, the increase in S transaction arrival rate is due to the transactions sent to Shipping from Order Entry.

From Figure 3 we can make the following observations: OpenQN shows the Order Entry System response time to be flat (response time of 1.7 not predicted until almost 16 transactions per second). The simulation and Simalytic Model results are very close and show the OE workload exceeding the goal at 3.33 transactions per second .

Which is the best approach to use? The queuing theory tool greatly underestimates the OE workload response time because it does not account for the impact from the Shipping system. The results of a simple single-server simulation model (not shown) greatly overestimate the OE workload response time because of the rapid queue buildup at a single server. Only the full simulation model and the Simalytic Model represent the actual workload behavior. Which is the best is determined by the complexity of the modeling effort. The Simalytic Model is more attractive when the nodes are too complex to be easily modeled with a general simulation tool.

## 7 Summary

The traditional view of planning the capacity of a system is evolving because of the desire to predict the performance of the application. Applications designed to exploit a client/server architecture greatly increase the complexity of both the computer system configurations and the applications themselves. Predicting the responsiveness of those more complex applications requires a more complex modeling methodology. But adding complexity to a modeling effort also adds time, effort and cost. There are many techniques and tools that are beginning to address this evolution, but none of them can provide the desired level of detail for every situation and every application.

By following these steps for implementing a Simalytic Model, the modeler can rapidly produce an application model at the level of detail needed to make business decisions. Combining different modeling techniques (simulation and analytic queuing theory) and different modeling tools (platform-centric and general purpose) reduces the time, effort and cost of developing an application model. Using the Spiral Modeling Methodology enables rapid model development and, as more detailed results are required, more sophisticated tools can then be used to increase the understanding of critical sections of the model. Simalytic Modeling supports this spiral approach by allowing an initial simple Simalytic Function for any node in the application model to be replaced with more detailed results as needed.

This level of analysis provides insight into the application's future performance that would not otherwise be available. Using the Simalytic Modeling Technique both improves the understanding of the application as well as identifies which systems require more detailed analysis. It protects the investment an organization has made in the acquisition of existing tools and the training in their use. It allows the most appropriate tools to be used for each modeling effort.

Capacity planning is still fundamental to business success. But just as application designs are moving away from single system solutions, modeling for capacity planning must move away from single system analysis and begin predicting the application across the enterprise.

## References

1. Wilson, G.L.: Capacity planning in a high-growth organization. presented at Computer Measurement Group, Orlando, FL (1994)
2. Hatheson, A.: Two Unix Client/Server Capacity Planning Case Studies. presented at Computer Measurement Group (1995)
3. Pooley, R.: Performance Analysis Tools in Europe. Informationstechnik und Technische Informatik 37 (1995) 10-16
4. Smith, C.U.: The Evolution of Performance Analysis Tools. Informationstechnik und Technische Informatik 37 (1995) 17-20
5. Norton, T.R.: Simalytic Enterprise Modeling: The Best of Both Worlds. presented at Computer Measurement Group, San Diego, CA (1996)
6. Norton, T.R.: Simalytic Hybrid Modeling: Planning the Capacity of Client/Server Applications. presented at 15th IMACS World Congress, Berlin, Germany (1997)
7. Norton, T.R.: Simalytic Modeling: A Hybrid Technique for Client/Server Capacity Planning. presented at Summer Computer Simulation Conference, Arlington, Virginia (1997)

8. Gunther, N.J.: Performance Analysis and Capacity Planning for Datacenter Parallelism. presented at Computer Measurement Group (1995)
9. Menascé, D., Almeida, V., Dowdy, L.: Capacity Planning and Performance Modeling: from mainframes to client-server systems. Englewood Cliffs, New Jersey: Prentice Hall (1994)
10. Norton, T.R.: A Practical Approach to Capacity Modeling. In: Anaheim, C.A.: Computer Measurement Group. vol. Workshop Proceedings, CMG, Inc. (1998) 1-123
11. Visual: Simul8. 141 St James Rd., Glasgow, UK G4 0LT: Visual Thinking International Limited
12. Norton, T.R: Don't Plan Capacity When You Should Predict Applications. presented at Computer Measurement Group, Orlando, FL (1997)

# Resource Function Capture for Performance Aspects of Software Components and Sub-systems

M. Woodside[(1)], V. Vetland[(2)], M. Courtois[(3)], S. Bayarov[(4)]

[(1)] Dept. of Systems and Computer Engineering, Carleton University

[(2)] Telenor ASA, Oslo, Norway

[(3)] MPC Data Ltd, Bradford on Avon, Wiltshire, UK

[(4)] Rational Software Canada

email: cmw@sce.carleton.ca, vidar.vetland@telenor.com, mcourtois@mpc-data.co.uk

**Abstract.** The performance of a software system is determined by its resource demands, and the degree of competition for such resources during execution. The demands are in part determined by pre-existing software components including libraries, operating systems, middleware, and increasingly, also by application level components. A suitable description of the resource demands of a component can be used for rapid performance and capacity analysis of a planned system. Resource demands may be found by theoretical analysis (as in big-O complexity analysis), or by measurement, as considered here. This paper describes the general notion of a workbench and repository for the gathering and maintenance of resource demand data, in the form of resource functions, and two research prototypes. The key elements are a test harness for each software component, automation based on a stored plan for running the test, function fitting for parametric dependencies, and a repository for retrieval of demand values. These tools simplify the process of getting the essential data for performance analysis, so that it can be economical and timely.

## 1 Introduction

Performance engineering is used to achieve quality-of-service goals for software products, and to avoid performance disasters. The resource demands made by the software components and operations determine the performance of the design, but are often not well understood. Systematic capture of resource function data can make performance engineering easier, faster and cheaper; component-based software techniques only increase the opportunities to capture and use the data.

This work assumes that performance analysis is done by calculation on some kind of model, which uses resource functions as data. Direct performance measurements and

R. Dumke et al. (Eds.): Performance Engineering, LNCS 2047, pp. 239–256, 2001.

tuning (either on a full-scale deployed system, or some kind of execution-driven simulation of it), can be done later.

Resource demands are quantified for one invocation of an operation provided by a software component, in terms of CPU cycles or seconds, the number of I/O operations, and the number of operations invoked on other devices and servers. They describe the workload generated by the operation, and they are essential input data to any analysis to predict the performance of a design. A resource function describes how the demands depend on some of the parameters of the operation. Message size is a typical argument, in resource functions for communications overhead costs. Figure 1(a) shows the CPU cost of a component from the ACE communications-software framework [4], and Figure 1(b) shows the CPU cost of sending an Internet message with a certain implementation of TCP/IP [16]. Other studies of communications costs include [5] and [25].

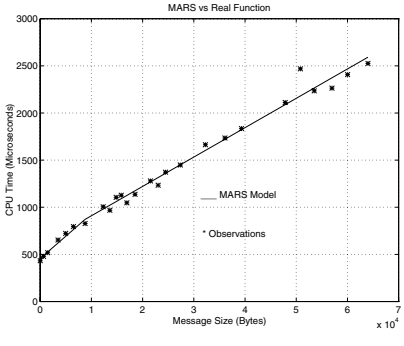



**Figure 1** (a) CPU demand resource function for the ACE supplier service handler used in the event server

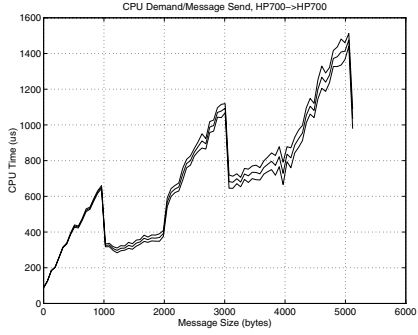



**Figure 1** (b) CPU demand resource function for TCP/IP messaging (mean and 95% confidence intervals)

Resource demand data and functions are needed for models used for performance predictions, capacity planning, and scalability analysis. Smith [22] described building a queuing model from an execution graph (a kind of flow-chart) which identifies the operations in a given scenario and a table of the resource demands of each operation. Cortelessa described essentially the same steps, starting from a UML sequence diagram for the scenario and leading to an extended queuing model, in [3]. Sheikh and Woodside carried out a similar analysis for transactions in a distributed database system, from which they derived a layered queuing model [19]. Menasce and Gomaa attached resource functions to high-level pseudo-code for planned software, and derived a queuing model from the pseudo-code [13]. A commercial performance engineering tool for client-server systems, Strategizer, provides resource functions for a commercial database system [9], and uses them to build a simulation model.

Resource functions are also needed for performance assertions. In real-time systems with deadlines, worst case CPU demands and other resource dependencies are used in schedulability analysis, to check the feasibility of the deadlines [12]. In the RE-

SOLVE system for component-based software engineering, it has been proposed to make and verify assertions about the asymptotic form of resource functions based on algorithmic complexity analysis [21].

Systematic performance engineering requires systematic data gathering for resource functions. This work describes an approach based on

- a "Workbench" which is a test environment for measuring resource demands, with a degree of test automation,
- resource function characterization by fitted functions, also automated,
- a Repository for resource functions, including the measurements and records of the test conditions

When building systems with components, performance modelling can even be automated (or partly automated), as described in Figure 2.

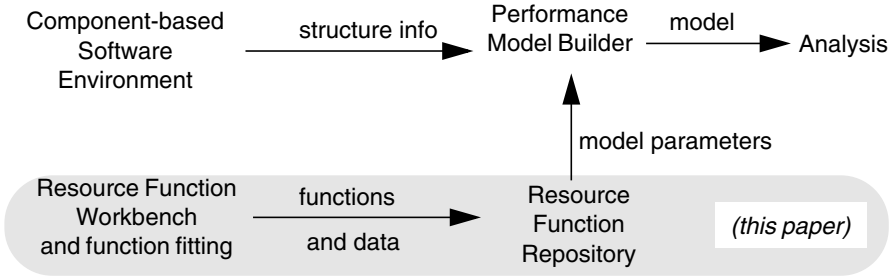



**Figure 2** Components of a complete performance engineering environment

This paper is mainly concerned with resource functions determined by measurement, but the Repository is equally useful for functions determined by theoretical analysis of the code and algorithms [21], compiler analysis [12], or expert judgement [22]. Measurement techniques and instrumentation used in this work are well documented elsewhere. For example, Bentley, Kernighan and van Wyk give a detailed prescription for measuring the demands of individual C language instructions in [2].

This paper explores the issues of data gathering and preparation, which is akin to data mining. It describes a prototype of the Workbench for running the experiments, and experience with different function fitting techniques including a particularly robust automated fitting approach based on Regression Splines. A second prototype which includes all the parts shown in Figure 2, including a limited and focused version of the Workbench and Repository, is also described.

## 2 Resource Functions

Resource demands of a component or operation *A* describe what *A* requires from the rest of the system. There must be a boundary around *A* which delimits what is to be captured. Demands that are generated within the boundary for CPU execution, and for

operations outside the boundary, are captured as illustrated in Figure 3. Thus the resource functions represent:

- the CPU execution cost of *A*.

  The boundary of *A* defines whether the cost of operating system calls are included, or not; commonly this question is actually decided by the instrumentation and what it captures when the component executes.
- the memory use by *A* (not considered here, however).
- invocations from A to operations on other components (hardware and software) which are outside the boundary of A. This represents a delegation of processing, and the resource demands must be quantified elsewhere.

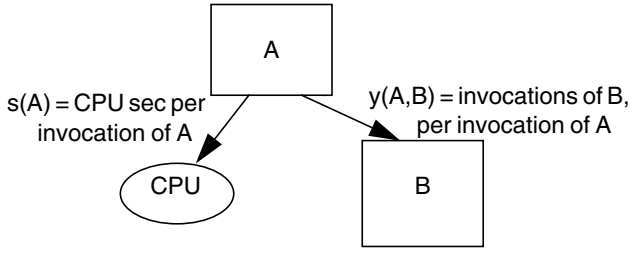



**Figure 3** Resource demands of a component *A*

The resource function *arguments* are parameters which affect the demands, such as:

- message sizes for communications operations, as shown by the examples, and also other data sizes, such as the size of arrays to be sorted or searched, of graphs to be traversed, or of images to be processed,
- the number of sub-operations to be executed, such as mean loop counts or the depth of a complex transaction,
- alternative execution platforms, including the hardware, the version of the operating system or real-time kernel, and the compiler,

It is a core problem of resource demand description, to limit the number of parameters examined. An unexamined parameter is a source of possible error, if its value is different in the final deployed product and the way it is used, and no analysis can ever, in practice, be complete.

Some examples of resource function analysis will illustrate the concept and its difficulties. The study by Pozzetti et al. [16] is a good example. They set up an instrumented version of the code in an isolated system with no other workload, and did experiments across a wide range of the chosen parameter, message size. It required a substantial research project to determine the form of the function shown in Figure 1(b), and to have confidence in its rather bizarre shape. Special drivers and instrumentation were required. The ability to repeat the experiment and get the same results was essential to their confidence in the correctness of the results. This work shows the importance of the ability to repeat any experiment, to keep the drivers and instrumentation together with the data, and to explore the data interactively, as is common in data mining.

Xu and Hwang investigated the delay (for CPU processing and the interconnection network) for three communications operations, and for two alternative message-passing libraries, on the IBM SP2 [29]. They found that the delay depends on the number of nodes in the computer, and the number of bytes in a message, with known functional forms suggested by the algorithms. They fitted functions to the measured delays; the fitted functions conveyed more information, more compactly, than just a table of values. Although this work confounded the CPU and network resources, it shows the use of fitted functions clearly. The function arguments were both quantitative (the message size and number of nodes) and qualitative (which message-passing library to use).

The research group developing the RESOLVE component-based approach to software engineering [20] have considered resource functions for CPU demand for any component (e.g. [21]), based on the algorithms used by the designer. Their analysis is asymptotic and only predicts the functional form as the arguments become large; it gives insight into scalability problems.

Smith in [22] considers capacity planning in systems that have not yet been built, and advocates the use of expert judgement for resource functions. Demands for other operations have a prominent role, and are successively reduced using the resource functions for the other operations, to obtain total demands for CPU, disk I/O, etc. Measured resource functions can be included; [23] describes a typical study with database resource functions measured on a prototype.

These are a few examples. In [26], Vetland et. al. considered the general problem of systematic parameter capture for modelling. From all this experience the following five requirements on resource functions, and the process and tools for measuring and handling them, can be extracted.

1. A test configuration is required, which should be reproducible and representative. It may not exactly represent the conditions under which the component will run, but it should be close enough to give useful data, which can perhaps be adjusted when it is used. The present work used both standard operating system instrumentation, and special instrumentation packages, for capturing measurements.
2. It should be easy to create a test for a new component, since software technology is constantly changing. If the process of instrumenting a component and creating tests takes too long, it will not be used in urgent situations. This is a challenge; we may turn it around and say that we must accept the use of "obvious" and not very sophisticated experiments. The present work uses a custom-written test harness to invoke the component, which can imitate the typical usage in practice.
3. It must be possible to run the measurement experiment automatically, over different processing environments, so that the information can be maintained as models and versions change. If the process of capturing the data is arduous (as it often is, without automation), then it will not be practical to use it in urgent projects. The workbench described here provides all the automation features, based on capabilities which must be provided by the harness.
4. It must be possible to apply the parameter changes in the experiments and run them many times over ranges of parameter values, with the entire parameter con-

trol plan being stored and maintained for use in later repetitions. The workbench design also provides this feature.

5. A compact representation is needed for the function which captures the variation of the demand, as the parameters change. From this representation, values for the analysis of a given case can be found by interpolation. If the demand value is roughly constant, or is a linear function of the parameters, this is staightforward. However, there are no guarantees, as shown by the zigzag shape of the results in Figure 1(b).
6. it should be possible to combine resource functions from different sources, including theoretical analysis and expert judgement, in a repository that supplies parameter values for modelling.

The inevitable errors in demand values should be kept in mind in using the results of the performance analysis, and predictions must be checked against trials as a product emerges. Deviations (if any) can be traced back to the resource functions and used to compensate the predictions, or to correct the measurement experiments.

Requirements 1 - 4 for the Workbench and Repository are discussed next in Sections 3 and 4, Requirement 5 for function representation in Section 5, and Requirement 6, for support for performance analysis, in Section 6.

## 3 Resource Demand Workbench and Repository

The first four of the requirements just stated motivate the construction of some special resource function tools, which are the bottom two components in Figure 2. Resource demand data is often gathered in isolated spasms of activity which are difficult to mount, and whose impact is limited. The information is put to immediate use, but its value quickly degrades. The users forget under just what conditions the data was gathered, the software components that were measured evolve a little, a new version of the operating system arrives (or a new model of computer) and soon the work is all to be done again. Each time is like a new research project, which is costly and discouraging. The Workbench is intended to lower the effort and increase the retention of information. Figure 4 shows the elements of the Workbench.

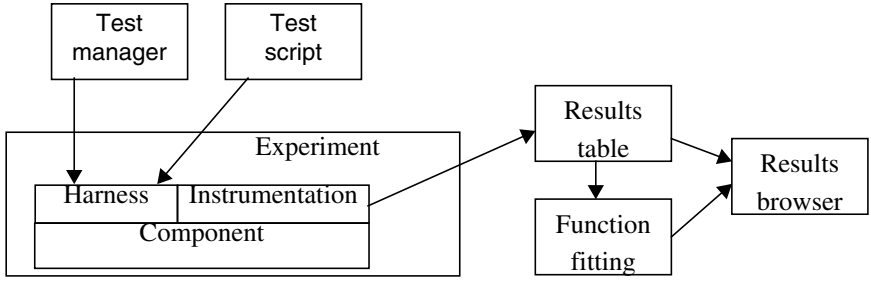



**Figure 4** Elements of a Resource Function Workbench

The last three requirements in the list above describe the need for storage and access to the results, provided by the Repository in Figure 2. The storage can be as simple as a file with a table of values, if the resource data is just a set of point values, or it could be a data base with an appropriate format for each function. For generality, it should be possible to store resource demand values in three formats:

- point values for demands, including the name of the operation or component, the names and units of the demands and their units (CPU time in seconds, disk pages, etc.). This could be a list of (name, description, value) elements;
- tables covering parameter variations; it would include all the above data and also the names of the parameters, and their values for each record.
- functions for the demands. The process of fitting empirical functions tends to lead to a standard function form, which may simplify the presentation. One example is multivariate linear functions expressed as an array of coefficients; another, described below, is linear regression splines.

Access to the resource values will be either automated, to fill parameter slots in predefined analysis models, or via a human interface for browsing and informal analysis. Next we examine how these requirements were met in prototypes.

# 4 WorkBench Experience

This section describes two prototype environments which illustrate what must be done to meet the requirements listed above. The first one is a Workbench for hands-on measuring and analysis of resource functions, described in [28] and based partly on the IMSE Experimenter [15]. The Workbench was designed to run on a UNIX network, and to analyse a wide range of software components that might run on many different UNIX platforms. . The second is an automated facility called RefCAM that handles a limited, well-defined set of components that exist in a large number of variations for different real-time platforms.

The main features of the Workbench will be described while considering a software component which marshals and unmarshals data for Remote Procedure Calls (RPCs), using the External Data Representation (XDR) format. Marshaling overhead can be an important factor in the performance of distributed applications. The marshaling function code is generated automatically by a stub generator, based on data type declarations for the RPC arguments. It is then compiled and linked with a main program, which declares the arguments and calls the marshalling function. (It is not necessary to actually execute the RPC.) The main program is the harness, and is controlled by the workbench. Three cases are described. The first case describes the Workbench and the harness, for experiments in which the parameters variations do not lead to recompilation. The second case has parameter changes that require the component to be recompiled, and in the third case the harness code itself is modified between runs.

### Case 1: Parameter variation by simple repeated execution.

We consider the marshalling of an array (of variable length) of bitmapped images which are structures, consisting of an integer giving its type, and a variable length array

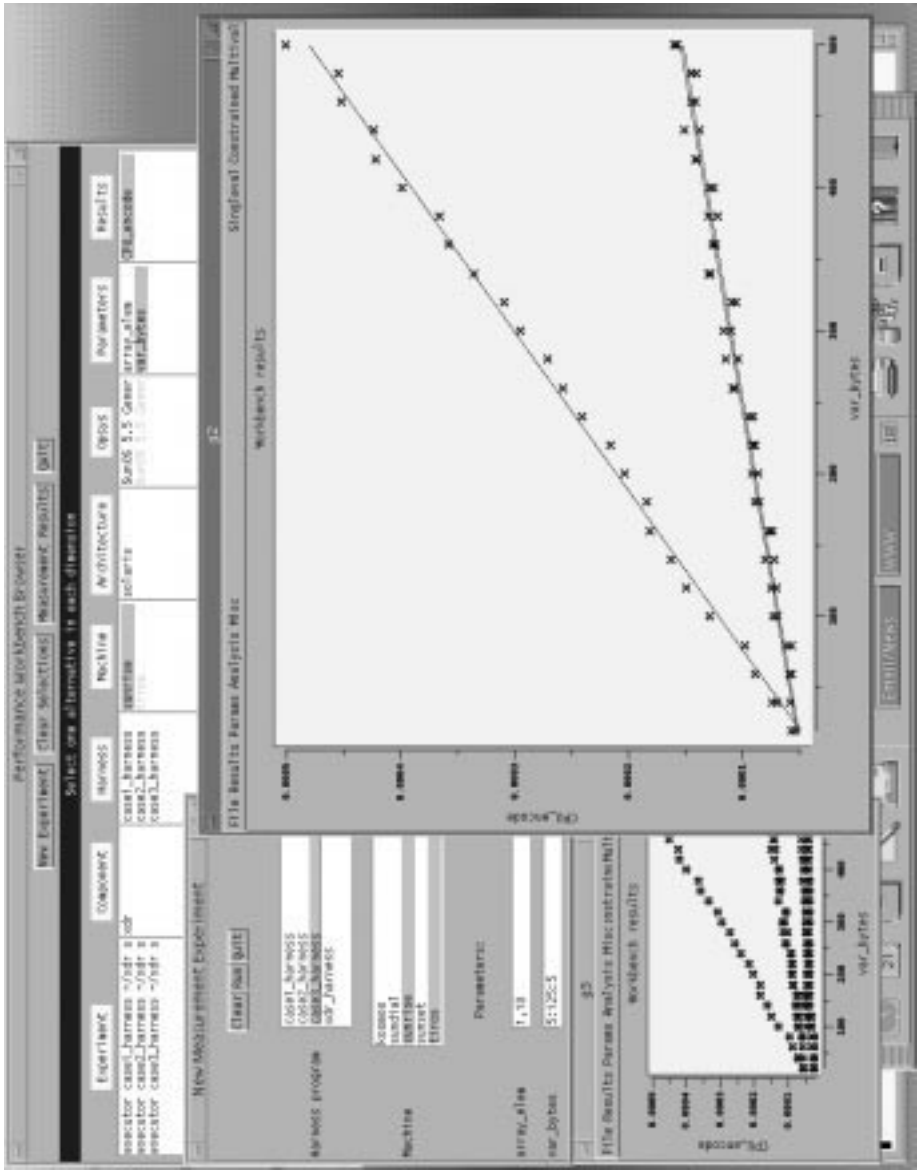


**Figure 5** The control windows of the Performance Workbench

of untyped binary data. The lengths of the arrays can be set at runtime, and we wish to find the CPU demand for different values of two parameters *m* and *n*:

*m* = bitmap size in bytes,

*n* = number of images

The harness code for this component is written in the appropriate programming language (C in this case) and has the sequence:

- command line invocation with arguments *n* and *m*
- declarations
- initialize the data structures to be marshalled, filling the image bitmap arrays with arbitrary values
- start timing the CPU usage, using *getrusage*
- call the marshalling function *N* times (optional, to provide resolution if the timing facilities have poor resolution)
- stop timing and retrieve the CPU usage over the interval; divide it by *N*
- print the result to standard output, in the workbench harness output format described below

The Workbench has a configuration record for the harness, giving its name, the parameter names, and the names of the resource demand measures that it will output. All harnesses are UNIX executables that accept a space-delimited sequence of parameter values on the command line, and produce results in a standard format defined by the BNF:

*experiment-output:: harness-name NL*

*component-under-study-name NL*

*parameter-settings NL*

*results NL*

*parameter-settings :: parameter-setting + parameter-settings | parameter-setting*

*parameter-setting :: parameter-name = value NL*

*results :: result + results | result*

*result :: result-name = value NL*

In the BNF, the atom NL stands for a newline character.

Figure 4 shows the workbench facilities for accessing the harness and defining and running experiments. At the top is the browser which lists the components, harnesses, computers, operating systems and architectures, that are known to the workbench. When certain items are selected, it affects what is displayed in other columns. For instance the component called *xdr* which we have been describing, is selected by default since it is the only component. Its two parameters and one demand value result are listed in the two columns at the right. The machine *sunrise*, which is selected, has the architecture and operating system shown.

The experiments listed in the left-most column have been previously defined using the "New Measurement Experiment" window at the left. To define an experiment, one selects the harness and a set of machine names, and one enters values for the parameters either as a list (shown in the Figure for the variable *array_size,* representing *n*) or as a sequence (shown for the variable *var_bytes*, which represents *m*, the size of the images).

The experiment is run from this window, and invokes the harness program remotely on the machines. The harness is invoked once for each combination of parameter settings, so in this case there will be 50 runs on the machine called *sunrise*, and 50 on *tiros*.

The results of the experiment on *sunrise* are shown in Figure 6. The harness for this

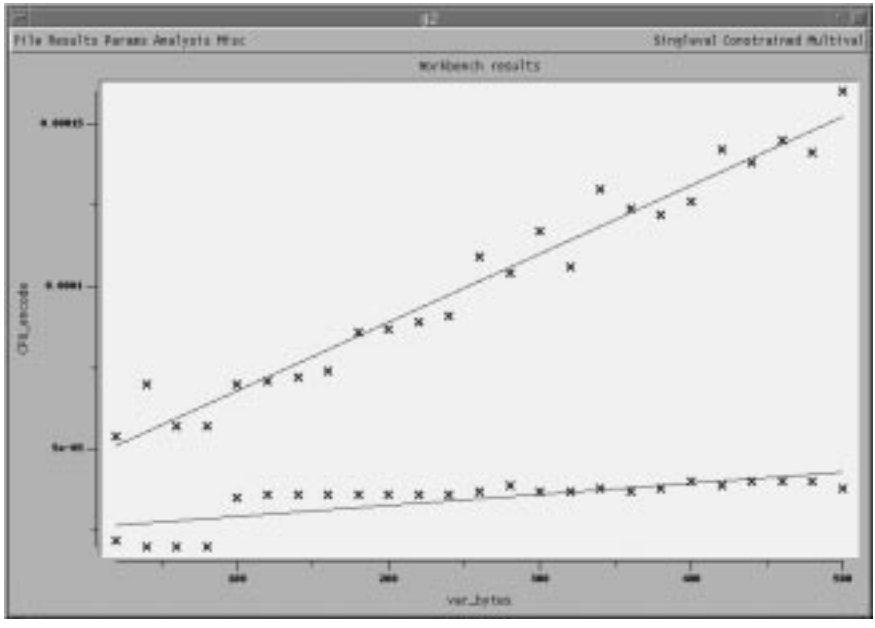



**Figure 6** Resource functions for CPU demand for marshalling image data in Case 1

experiment passes "opaque" data, which transfers the bytes with no conversions, so the overhead is small. It appears that a straight line is a reasonable fit for the upper group of data for $n = 10$, but is not so good for the lower group of data for $n = 1$, which show a distinct step at around 100 bytes. Nonetheless the error in both cases are not large, so a straight line could be used. The resource function found here for $n = 10$ is:

CPUtime = $[46.9 + 0.21m]$ μsec

## Case 2: Parameter variation requiring recompilation

A parameter change may require recompilation. For example, to evaluate the use of fixed size arrays $m$ and $n$, instead of variable sizes, the sizes could be defined constants with values set by a compiler directive. Now the harness could be a shell script which invokes the compiler and linker, runs the program and outputs the results. Recompilation is required for parameters which change the platform, or compiler options such as optimization, linker options such as stack size, or heap size, and application options such as buffer sizes or number of threads.

In this case the resource measurements were almost the same as in Case 1.

### Case 3: Parameter variation requiring changing the source code

The most complex kind of parameter change actually modifies the source code of the harness or the component. As an illustration, the example was changed to pass a structure containing $n$ structures, each with $m$ floating variables. For each run the harness modified the code and recompiled it, regenerated the stubs and then ran the measured part of the experiment. The result for $n = 10$ on *sunrise* was, this time:

CPUtime = [31.5 + 0.89 $m$] μsec.

This is a considerably heavier demand than before. The results graph included at the bottom right in Figure 4 shows the measured values and fitted functions for the three cases just described, all for $n = 10$.

### Harness configurations

The harnesses used with the workbench are executables, either programs linked with the component being measured, or scripts that cause it to be run. Figure 7 shows a variety of configurations for the harness and the component or components.

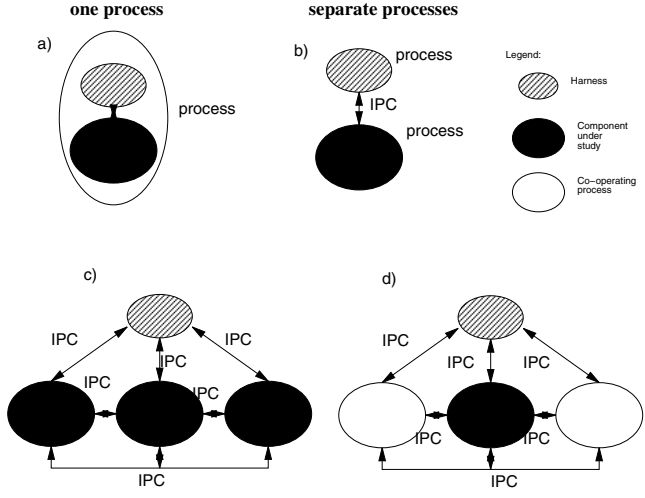



**Figure 7** Harness configurations for different kinds of components (components in black)

The use of measurements entails some errors introduced by the instrumentation, due to measurement overhead and timing granularity, which are discussed in [2]. The repeatability of data obtained with the Workbench, just using the UNIX *getrusage* function, was encouraging. The degree to which the values measured in test conditions match those in the field is always a question, which must be checked with experiments on the integrated product. Resource functions are valuable prior information, but they may have to be compensated or adjusted if experience shows there are systematic errors.

# 5 Resource Function Representation

The advantages of fitted functions were discussed above. Many practitioners try to choose a functional form in advance, based on prior knowledge or simplicity. For example Menasce [13] used a linear function of message size to estimate the overhead costs of Internet messaging, in a distributed client-server system. Xu and Hwang [29] used complexity analysis of the algorithms in two message-passing libraries to suggest the functional forms shown in Table 1, and then fitted parameters by measurement. They measured total delay for the CPU and interconnection network together, for MPI (the public-domain Message-Passing Interface library) and MPL (the IBM Message-Passing Library) running on the SP2 computer, for messages of $m$ bytes in a network of $n$ nodes.

**Table 1**: Resource Functions for Message Passing on the SP2

| Operation | MPI Library | MPL Library |
|---|---|---|
| Broadcast | 40log $n$ +0.37 $m$ log $n$ | 16 log $n$ + 0.025 $m$log $n$ |
| Gather | 84 + 24 $n$ + 0.045 $mn$ | 15 + 17 log$n$ - 0 02$m$ + 0.025$mn$ |
| Scatter | 105 + 24$n$ + 0.03$m$ + 0.026$mn$ | 15 + 17 log$n$ - 0.02$m$ + 0.025$mn$ |

Linear regression may be useful. When the study of marshalling overhead described in Section 3 above was extended to more complex messages, it was studied for an array of $n$ structures, each with m1 integers, *m2* floats (32-bit real values) and *m3* doubles (64-bit reals). Stepwise linear regression was performed on the CPU demand, with the four parameters just listed as independent variables, and also the total bytes to be converted, and a linear function was found [27]. All the variables are significant at the 95% level:

CPU demand = 25.2 + 0.09**bytes* + 2.3 *m1* + 0.21 *m2* + 0.52 n + 0.24 *m3*.

Another more complex experiment with the Workbench considered the cost of calculating a minimum spanning tree on graphs of different sizes and shapes [27]. A sample of graphs was selected from the Stanford "GraphBase" [11], randomly selecting graphs from those with certain pre-selected values of four topological parameters called P, W, E, N. The CPU demand was measured for the spanning-tree calculation, and the regression function formed. As expected there was substantial variation in the demands, even between graphs with the same parameter values, due to the random construction of the graphs. Stepwise regression was applied, which automatically retains only those terms with statistical significance, and which can be set up to include product and power terms. Stepwise regression chose the function form automatically, and can be forced to consider only linear terms, or a set of given nonlinear functions of the data.

## Regression Splines

However, regression analysis is not a complete solution to the question of representation. In general it still requires a little research project on each function, to find the

best form, and there may not be time for this. Some functions are quite non-smooth, like the resource function for TCP/IP execution shown by the points recorded in Figure 8. In this case there are jumps due to a decision made by the implementation, at certain message lengths, to change the size of buffers used to hold the message data. Not all cases are so difficult, but when simple functional forms are inadequate, a fall-back is needed.

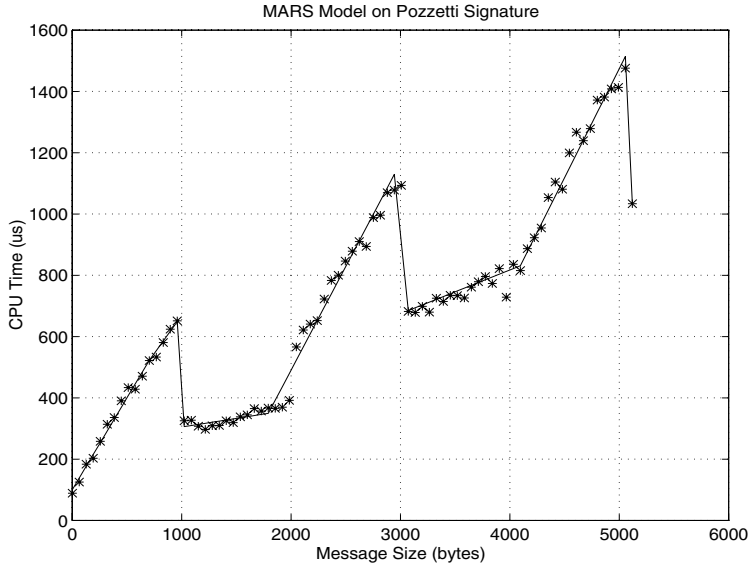



**Figure 8** CPU demand data and fitted regression spline for TCP/IP messaging (same data as shown in Figure 1(b))

For these cases Courtois and Woodside in [4] adapted a robust function-fitting algorithm which does not require any assumed functional form, called Multivariate Adaptive Regression Splines (MARS) [8]. MARS fits functions of any number of variables, made up of rectangular patches; each patch is a product of linear functions (one in each dimension). A fitted MARS function is expressed as a sum of terms, each of which is a product of a coefficient and a set of factors of the form:

$[x\text{-}a]^+ = \max(0, x\text{-}a)$, or: $[a\text{-}x]+ = \max(0, a\text{-}x)$.

For resource functions with a single parameter, MARS fits a piecewise linear function, as illustrated in Figure 8. The standard version of MARS in [8] adaptively chooses the location of the "knots", as the corner points in the piecewise linear function are called. In [4] another adaptive feature was added, to control the number of points that are measured, and to measure additional points in the parameter space until the accuracy reaches a desired threshold. The location of the additional points is determined by a heuristic algorithm that attempts to fill in gaps in spacing of the points and to add points where the local approximation error is relatively large.

Accuracy in MARS is indicated by a measure called Generalized Cross Validation (GCV), and [4] also examined the relationship between GCV and the usual concept of confidence intervals. A heuristic relationship was found which allows approximate confidence bands to be computed from GCV, to estimate the approximation accuracy and control the stopping point for data gathering.

A good illustration of the problem of characterising real resource functions is an event server, described in [4]. The event server was built from components in the ACE communications software framework [18]. The resource functions of two of its components depend strongly on the message size and the number of consumers of events, while the other components have roughly constant resource functions. The Supplier Service Handler component (which takes messages from senders) depends on the message size only, as shown in Figure 1(a). The points are data values measured with the Workbench, and the smooth line is the MARS fit. The Consumer Router component sends the message to all subscribers, one at a time, with demands which depend on the message size and also on the number of subscribers, or consumers, as indicated in Figure 9. The points represent the measured values and the two-dimensional grid shows the MARS function. The rectangular patches can be seen in the Figure. The fit in both functions is excellent

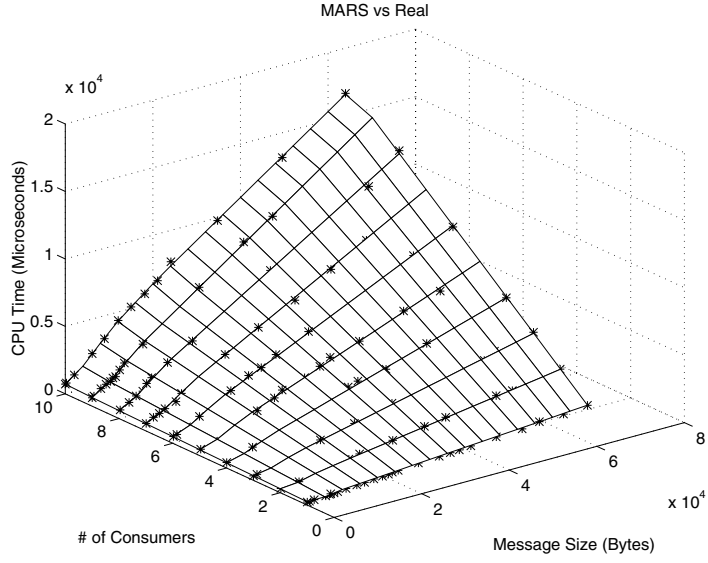



**Figure 9** CPU demand resource functions for the consumer router component in the event server

MARS-based representation appears to be ideal for nonlinear resource functions fitted to empirical data, as it does not require a hypothesis about the functional form. This was a valuable property in the two cases shown in Figure 8 and Figure 9, where there was no theory to guide the choice. However in other cases, linear representation may be sufficient. This was the case in the system described in the next section.

# 6 Performance analysis interface

There are many different kinds of performance analysis using models based on resource functions. Typically the analyst defines the important scenarios in terms of what operations are executed, and how many times. A table is built up that includes the values of the demands per operation (from the resource functions), and the total demands per execution of each scenario. The demands can be used to build various kinds of model:

1. a load table, which identifies the system bottleneck [3][22],
2. a schedulability analysis, for the feasibility of a set of deadlines [12],
3. a queuing model, which can also predict response times and capacity values [22],
4. an extended or layered queuing model [3],[7],[17],[19],[30] which can also identify logical bottlenecks, as well as predicting mean response and throughput values,
5. a simulation model, which can be instrumented to predict all aspects.

To go closer to the design, a model may be based directly on the specification of the software in a software tool (instead of on scenario analysis). A number of authors have considered modelling within SDL, using extended modelling primitives within an SDL tool to define the resources and their demands (see for example [24], which contains further references).

More powerful models can be created by using a separate model builder outside the software environment, which can describe a variety of system deployments without complicating the software design description. It can scale up the system in various ways and add a model environment describing the user, file systems, web servers, etc. Design information is extracted from the design tool into the modelling tool, and resource data is taken from the repository. This approach has been described for designs described in SDL [6] and also in the ObjecTime design tool.

## CASE tool integration

In the prototype environment built to work with Objectime [1], [31] , a specialized Resource Function system was included which illustrates how the entire concept in Figure 2 may be implemented in practice. The goal was to analyze the performance of prototype C++ code which runs on a custom layer called RTS, over a wide range of real-time kernels [14]. Most of the execution demand is CPU time for RTS operations (actor state changes, messaging, etc.) which were captured in resource functions.

The measurements of RTS were driven by standard test scripts, written in the tool itself and code-generated for the desired kernel. Instrumentation was kernel-specific, but the CPU time per operation, for about a hundred RTS operations, was stored in a file of standard format. For message passing operations a range of message sizes was used, and linear functions were fitted. The function forms were first scrutinized manually to be sure they were appropriate, and then generated automatically when needed. The entire process of capturing resource functions for one kernel/processor combination was then a single script that ran in a few minutes.

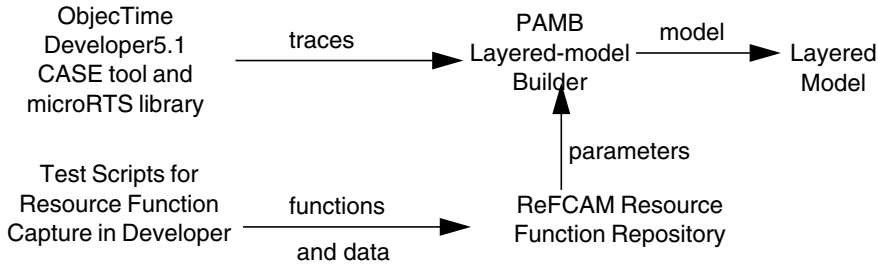



**Figure 10** A prototype performance engineering environment based on a CASE tool with a component library [1][31]

The model-builder (called PAMB, and more fully described in [1] and [31]), created a layered queuing model structure from traces made by executing the design. The traces contained tags for the occurrence of RTS operations, including message sizes. The model-builder accessed the demand values for each RTS function from the repository "RefCAM" to calibrate the model parameters. The calibrated layered model can then be studied in combination with other components, on large and small platforms and for a range of workloads and environments.

# 7 Conclusions

The systematic capture of resource data and resource functions has been explored, and its feasibility has been shown by prototype demonstrations. The essential elements are: repeatable experiments to capture data, a test environment to help with maintenance of the data over time, efficient representation by fitted functions, and automated access to the data by performance analysis tools.

The Workbench/Repository approach is adaptable to any performance engineering analysis method that depends on calculation and requires data for operations. The repository can also provide functions derived without measurement, for instance by complexity analysis, or by analysis of compiler output.

**Acknowledgements**

This research was supported by the Natural Sciences and Engineering Research Council of Canada through its program of Industrial Research Chairs, by the Ottawa Center for Research and Innovation, and by ObjecTime Ltd. (now Rational Software Canada).

# 8 References


[1] S. Bayarov, *Resource Functions for Model Based Performance Analysis of Distributed Software Systems*, Master's Thesis, Dept. of Systems and Computer Engineering, Carleton University, Ottawa, Canada, 1999.

[2] J. Bentley, B. W. Kernighan, and C. Van Wyk, "An Elementary C Cost Model", *Unix Review*, vol. 9, no. 2, pp. 38-48, 1991.

[3] V. Cortelessa and R. Mirandola, "Deriving a Queueing Network based Performance Model from UML Diagrams", in *Proc. Second Int.l Workshop on Software and Performance (WOSP2000),* Ottawa, Canada, September 17-20, 2000, pp. 58-70.[1]

[4] M. Courtois and M. Woodside, "Using Regression Splines for Software Performance Analysis and Software Characterization", in *Proc. Second Int. Workshop on Software and Performance (WOSP2000),* Ottawa, Canada, September 17-20, 2000, pp. 105-114.

[5] J. J. Dongarra and T. Dunigan, "Message-Passing Performance of Various Computers" University of Tennessee and Oak Ridge National Library, Tech. Report, August 1995.

[6] H.M. El-Sayed, D. Cameron, and C.M. Woodside, "Automated performance modeling from scenarios and SDL designs of distributed systems", *Proc. Int. Symposium on Software Engineering for Parallel and Distributed Systems (PDSE'98)*, Kyoto, Apr. 1998.

[7] G. Franks, S. Majumdar, J. Neilson, D. Petriu, J. Rolia, M. Woodside, "Performance Analysis of Distributed Server Systems", *Sixth Int. Conf. on Software Quality (6ICSQ)*, Ottawa, Canada, Oct. 1996, pp 15-26.

[8] J. H. Friedman, "Multivariate Adaptive Regression Splines", *Annals of Statistics*, vol. 19, no. 1, pp. 1-141, 1991.

[9] Hyperformix Strategizer Features, http://www.hyperformix.com/low-res/products/strategizer.htm.

[10] P. Jogalekar and M. Woodside, "Evaluating the Scalability of Distributed Systems", *IEEE Transactions on Parallel and Distributed Systems*, v. 11, n. 6, pp. 589-603, 2000.

[11] D. E. Knuth, *The Stanford Graph Base*. New York: ACM Press, 1993.

[12] Jane W. S. Liu, *Real-Time Systems*, Prentice-Hall, 2000.

[13] D. Menascé and H. Gomaa, "A Method for Design and Performance Modeling of Client/Server Systems", *IEEE Trans. on Software Engineering*, v. 26, n. 11, pp. 1066-1085, 2000.

[14] ObjecTime Ltd, *Developer 5.1 Reference manual*, ObjecTime Ltd (now Rational Software Canada), Ottawa, Canada, 1998

[15] R. J. Pooley. "The Integrated Modelling Support Environment - a new generation of performance modelling tools", in G. Balbo and G. Serazzi (editors), *Proc. Fifth Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, Torino, Italy, Feb. 1991, pp 1-15. Elsevier, 1992.

[16] E. Pozzetti, V. Vetland, J. Rolia, and G. Serazzi, "Characterizing the Resource Demands of TCP/IP", in *Proc. 1995 Conf. on High Performance Computing and Networking (HPCN95),* May, 1995.

[17] J. Rolia, K. Sevcik, "The Method of Layers", *IEEE Trans. on Software Engineering*, v 21, n 8, Aug. 1995, pp 689-700.

[18] D. C. Schmidt, "The ADAPTIVE Communication Environment", *11th Sun User Group Conf.,* San Jose, CA, Dec., 1993.

[19] F. Sheikh and C. M. Woodside, “Layered Analytic Performance Modelling of Distributed Database Systems”, in *Proc. Int. Conf. on Distributed Computer Systers,* Baltimore, U.S.A., May 1997, pp. 482-490.

[20] M. Sitaraman and B. W. Weide, “Component-based software using RESOLVE”, *ACM Software Eng. Notes*, vol. 19, no. 4, pp. 21-67, 1994.

[21] M. Sitaraman, J. Krone, G. Kulczycki, W.F. Ogden, A.L.N. Reddy, “Performance Specification of Reusable Software Components”, Research report, Univ. of West Virginia, 2000.

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

[23] C.U. Smith, B. Wong, “SPE Evaluation of a Client/Server Application”, *Proc. Computer Measurement Group*, Orlando, Dec. 1994.

[24] M. Steppler, “Performance Analysis of Communication Systems Formally Specified in SDL”, *Proc. 1st Int. Workshop on Software and Performance*, Santa Fe, NM, Oct. 1998, pp 49-62.

[25] J. D. Touch, “Performance Analysis of MD5”, in *Proc SIGCOMM '95,* 1995, pp. 77-86.

[26] V. Vetland, P. Hughes, A. Solvberg, “Improved Parameter Capture for Simulation Based on Composite Work Models of Software”, *Proc. of the 1993 Summer Computer Simulation Conf.,* July 1993.

[27] V. Vetland, M. Woodside, “Systematic Performance Characterization of Software Components”, Report SCE-97-06, Dept. of Systems and computer Engineering, Carleton Univ., Ottawa, March 1997.

[28] V. Vetland and C. M. Woodside, “A Workbench for Automation of Systematic Measurement of Resource Demands of Software Components”, *Transactions of the Computer Measurement Group*, Issue 92, pp. 42-48, June, 1997.

[29] Z. Xu and K. Hwang, “Modeling Communication Overhead: MPI and MPL Performance on the IBM SP2”, *IEEE Parallel and Distributed Technology*, pp. 9-23, Spring 1996.

[30] C.M. Woodside, “Throughput Calculation for Basic Stochastic Rendezvous Networks”, Performance Evaluation, vol.9, no. 2, pp 143-160, Apr. 1989.

[31] M. Woodside, C. Hrischuk, B. Selic, and S. Bayarov, “Automated Performance Modeling of Software Generated by a Design Environment”, *Performance Evaluation.* To appear, 2001.

# Shared Memory Contention and Its Impact on Multi-processor Call Control Throughput

T. Drwiega

Nortel Networks, PO Box 3511, Station C, Ottawa, ON, Canada, K1Y 4H7
Drwiega@nortelnetworks.com

**Abstract.** A Shared-Memory Multi-Processor Call Control (SMMPCC) is described when the executing transactions contend for access to shared memory addresses. First, memory contention is characterized, then the system throughput is modelled analytically. Based on the model, bounds for the throughput are found. Also shown are how the throughput depends on the number of processors, contention level and on how far into transaction execution a contention on memory access happens.

## 1 Introduction

A Shared-Memory Multi-Processor Call Control (SMMPCC) executes several transactions in parallel. Those transactions are parts of the call control and they will be hereafter referred to as subcalls. Executing subcalls may need to access the same address(es) in the shared memory. Accesses by two or more concurrently executing subcalls to the same shared memory address may result in an inconsistency of the state of the accessed data. For example, one subcall can modify data, another subcall can read it and act upon the read value even though it has been modified in the meantime again by the first subcall.

There are a number of mechanisms for assuring consistency of data accessed in shared memory. We will consider locking. A subcall accessing a shared memory address locks it to block other subcalls from accessing it. All addresses locked by a subcall are released when the subcall is completed (this is called committing).

An access request to a locked address is blocked, unless it is the requesting subcall who locked the address. A subcall whose request is blocked is immediately rolled back to the state before the execution and restarted later. All addresses locked by the subcall prior to its rollback are released and their contents is restored.

The processor time used by rolled back subcalls does not result in any useful work done and therefore decreases its processing power. The more subcalls are rolled back the less processing time is available for doing useful work. The total real time used by rolled back subcalls, and consequently the processing power of a SMMPCC depends on factors such as: number of processors, level of the load offered to the core, average real time used by rollbacks, and level of potential memory contention in the offered load of subcalls.

The subject of this paper is the impact of blocking on shared memory access on the processing power of SMMPCC. In Section 2 some characteristics of shared memory

R. Dumke et al. (Eds.): Performance Engineering, LNCS 2047, pp. 257–266, 2001.

contention observed in call processing software are discussed. In Section 3 an analytical model for memory contention is described. A validation of the model against simulation results is discussed in the Appendix. The model has been used to study SMMPCC throughput under different numbers of processors, fraction into subcall when contention occurs and the potential memory contention in the offered load of subcalls. The results of that study are discussed in Section 4. Section 5 contains key conclusions drawn from the analysis.

## 2 Blocking on Memory Access

Processing a call is performed in stages, called subcalls, which correspond to the phases of the call, such as off-hook, digits, supervision, release, etc. Subcall execution may require accessing some data from the shared memory. Subcalls, which have to access the shared memory, can block, or be blocked on memory access when they are executing concurrently in different processors.

In order to be able to model memory contention, accesses of shared memory by subcalls have been studied. The following key characteristics of memory contention were found:

1. There is a fraction of subcalls which do not access the shared memory.
2. Shared memory addresses accessed by two different subcalls are accessed in the same order by both subcalls. For example, if one subcall accesses addresses A, **C**, **E**, K, **U**, V, then the other subcall accesses B, **C**, D, **E**, O, **U**, Y, Z.
3. Subcalls which access shared memory addresses do not necessarily have common addresses with all other subcalls accessing shared memory. Many pairs of subcalls access disjoint sets of addresses. Moreover, depending on the call mix, subcalls form groups, which we call *c*ontention groups, such that members of a group access common shared memory addresses, and subcalls from different groups do not contend, or very little contend, for shared memory addresses.

The first observation means that a part of the load of subcalls offered to the SMMPCC will never block on memory access. The second observation means that if two subcalls concurrently try to access the same addresses in the shared memory, the subcall that accesses the first common address cannot be blocked by the other call.

Based on the observations the important parameters characterizing memory contention between subcalls are:

1) the fraction of subcalls which do not access shared memory, and therefore never block,
2) the number of contention groups and their sizes,
3) for each pair of subcalls, the time into the subcalls execution when the first common address is accessed.

The first two parameters are indicators of potential for memory contention in the call processing software, for a given mix of subcalls, which we call potential blocking. Potential blocking is independent of the number of processors and of the intensity of the traffic offered to SMMPCC.

The above parameters are an input to the model for studying the impact of memory contention on SMMPCC throughput.

## 3 Model for Memory Contention in SMMPCC

Impact of memory contention on the performance of multi-processor systems has been modelled and studied by many researchers, see e.g. [1] and [2] and the references therein. Most of them assumed that shared memory addresses form a pool, not necessarily homogeneous as in [3], from which processes draw them randomly. Consequently, processes access shared memory addresses in random orders, independent from one another. Another case, see e.g. [4], assumed that each process accesses all required shared memory addresses at the beginning of their execution.

The models described in the literature are inapplicable to memory contention with characteristics 2 and 3. We have developed an analytical model incorporating these findings for the purpose of studying SMMPCC performance under memory contention.

SMMPCC is modeled as a multi-server system with a shared queue, where an arriving subcall is executed immediately if there is an idle server, or joins the queue otherwise. When a server becomes idle it starts processing the head of the queue, if the queue is not empty.

Assume that an SMMPCC of k processors is offered subcalls, at average rate of $\lambda$ such that it works at 100% occupancy, i.e. all processors are always busy, but all subcalls are, eventually, successfully executed. $\lambda$ is therefore the SMMPCC throughput. For convenience, assume that the average time of a successful execution is 1.

Subcalls which contend for shared memory addresses form n groups, such that only processes from the same group can be in contention. Denote the arrival rate of first offered subcalls from an i-th contention group by $f_i\lambda$ and the average execution time by $w_i$ (i=1,2,...,n). From the Little's law the average number of successfully executing subcalls (not including rollbacks) from the i-th group is $f_i\lambda w_i$.

We make a simplifying assumption about contention within each group. Since subcalls from any group contend with each other for shared memory addresses, we assume that they all, within the same group, have to access the same location of the shared memory. A subcall, which accesses that location, executes successfully and locks that location for the rest of its execution. All requests for access to a memory location which is locked are blocked and the requesting subcalls are rolled back. We assume that the period for which a memory location is locked by a subcall from an i-th contention group is a negative exponential random variable with mean $q_i$ ($<w_i$).

Based on those assumptions memory contention in a contention group can be modelled as a single server system with quasi-random input and blocked arrivals cleared. Executing subcalls from that group are sources of requests for service, and the portion of the shared memory which they request to access is the server. Subcalls which have not requested the shared memory access yet are considered idle sources; they will request service. The subcall which has locked the shared memory portion is a busy source. It will not issue any requests, and it will release the server when it commits.

Subcalls which are blocked from accessing the shared memory, are rolled back and restarted later. Denote the rate of restarts from the i-th group by $\zeta_i$. The rate of rollbacks is also $\zeta_i$. A rolled back subcall from i-th group spends, on average, $(w_i-q_i)$ time in the system. From the Little's law the mean number of executing subcalls which will be rolled back is $\zeta_i(w_i-q_i)$ (i=1,2....,n).

The average number of executing subcalls from the i-th group is $f_i\lambda w_i + \zeta_i(w_i-q_i)$. Since all k processors execute subcalls from all groups, the probability that a processor is serving a subcall from this group is $[f_i\lambda w_i+\zeta_i(w_i-q_i)]/k$. An idle source from the i-th group, sends requests for the shared memory address at the rate of $1/(w_i-q_i)$. Thus a processor, if it has not locked the shared memory of the i-th group, sends requests for it with an average rate of $[1/(w_i- q_i)]*[f_i\lambda w_i+\zeta_i(w_i-q_i)]/k$. If they are granted the access, they keep it, on average, for the $q_i$ period of time. An average request load offered by a processor to the server modeling the i-th group shared memory is therefore

$$a_i = [q_i/(w_i-q_i)]*[f_i\lambda w_i+\zeta_i(w_i-q_i)]/k.$$

From the Engset formula (see e.g. [5]) the probability that a request by a processor executing a subcall from the i-th group for the shared memory access will find the address locked by another subcall is

$$P_i = (k-1)a_i / [1+ (k-1)a_i]. \quad (1)$$

On the other hand the probability of rollback for a subcall from the i-th contention group is a ratio of the rollback rate to the total arrival rate of subcalls from that group:

$$P_i = \zeta_i / (\zeta_i + f_i\lambda). \quad (2)$$

From (1) and (2) we obtain a formula for $\zeta_i$:

$$\zeta_i = d_i\lambda^2/(1 - e_i\lambda) \quad (3)$$

where $e_i = (k-1)f_iq_i/k$, and $d_i = f_ie_iw_i/(w_i-q_i)$.

For any offered load, i.e. when $\lambda>0$, the rollback rates are positive if and only if

$$1 - e_i\lambda > 0 \qquad \text{for } i=1,2,...,n. \quad (4)$$

The traffic carried by a k processor system working at 100% occupancy is k. This traffic is a sum of the through traffic and the rollbacks:

$$k = \lambda 1 + \sum_{i=1}^{n} \zeta_i (w_i - q_i) = \lambda 1 + \lambda^2 \sum_{i=1}^{n} \frac{d_i(w_i - q_i)}{1 - e_i\lambda} = \lambda + \lambda^2 \sum_{i=1}^{n} \frac{h_i}{1 - e_i\lambda} \quad (5)$$

where $h_i = d_i(w_i-q_i)$ for i=1,2,...,n.

(5) is equivalent to a polynomial equation of $\lambda$ of a degree between 2 and n+1. For degrees 2 and 3 it can be solved explicitly and for higher degrees - numerically. For the general case, we will find the interval where the sought solution is located and we will show some properties of the solution.

We assume that there are some contention groups among the processes, i.e. $f_i>0$ for at least some i. Otherwise, if there is no contention, there are no rollbacks and $\lambda=k$.

**Theorem 1**. Equation (5) has exactly one solution which satisfies condition (4).

**Proof**: From (4) we have that

$$\lambda < \lambda_{sup}=1/e_{max}, \quad (6)$$

where $e_{max}$ is the largest among all $e_i$ values. It can be checked that the function of $\lambda$ $T(\lambda) = k - \lambda 1 - \lambda^2 \Sigma h_i/(1-e_i\lambda)$ is monotonically decreasing in the interval $(0,\lambda_{sup})$, is positive at 0 and tends to minus infinity when $\lambda$ approaches $\lambda_{sup}$ from the left. This means that there is exactly one zero of $T(\lambda)$ in that interval, and that is the solution of the equation (5) that satisfies the condition (4). •

If there is no memory contention, the throughput increases with k and goes to infinity if k does. When even a small fraction of subcalls arriving at SMMPCC request access to the same portion of the shared memory the throughput is limited, even if the number processors is not. We will formulate this asymptotic property of the throughput as the following theorem.

**Theorem 2**. The solution of the equation (5) which satisfies (6) tends to a finite value $1/(f_jq_j)$, as k tends to infinity, where j is the contention group with the largest value of $f_jq_j$ among all the groups.

**Proof**: When k tends to infinity then so does the rightmost side of the equation (5). $\lambda$ is bounded from above by $\lambda_{sup}$ which tends to $1/(f_jq_j)$. The components of the summation tend to finite limits, with the exception of those corresponding to the groups with products $f_iq_i$ equal $f_jq_j$. They grow to infinity with k. Therefore the right side of (5) tends to infinity iff $\lambda$ tends to $1/(f_jq_j)$ when k goes to infinity. •

Theorem 2 provides the upper bound on the SMMPCC throughput for a given potential memory contention level. It shows that a target throughput may not be achievable by just adding more processors to the system. If the asymptotic throughput is lower than the target, reduction of potential memory contention is required. The asymptotic throughput also shows the maximum throughput gain that can be achieved by adding more processors to an SMMPCC.

The highest level of potential blocking is when all subcalls are in contention for shared memory access with each other. In our model this corresponds to n=1 and $f_1$=1. In this case we have an explicit formula for the throughput

$$\lambda = k / [1 + (k-1)q_1]. \quad (7)$$

Because $q_1<1$ therefore $\lambda$ is always greater than 1 and it tends to $1/q_1$ when k grows to infinity. If under the strongest potential blocking the throughput is greater than 1 we can conclude that the throughput is always greater than 1. In other words the processing power equivalent to at least one processor is doing useful work. Formula (7) indicates a dependence between the throughput and the average period of time for which subcalls lock contentious memory locations. This dependence is discussed in more detail later.

The model was validated by means of simulation. A comparison of throughput obtained from the model and simulations is discussed in the Appendix.

# 4 Impact of Shared Memory Contention on SMMPCC Throughput

In this section SMMPCC throughput is discussed as a function of shared memory contention. In order to show that function more clearly we did not take into account other factors that impact the throughput, such as memory access delays, scheduling overhead, etc. For the same reason we reduced the number of the input parameters by assuming that all contention groups are of the same size, the same average execution time and the average period for which they lock their memory portions, i.e. $f_i = f$, $w_i = 1$ and $q_i = q$ for i = 1,2,...,n.

If there was no memory contention then all processors' time could be used to do useful work. However, in the presence of memory contention the rolled back subcalls use a fraction of the total real time. This fraction grows with the SMMPCC occupancy and achieves its maximum when the system is working at its full capacity. The number obtained by multiplying this fraction at capacity by the number of processors in the system, can be interpreted as the number of processors, possibly not integer, that work only on rollbacks. The difference between that number and the total number of processors in the SMMPCC, which we call available processing power, can be thought of as the expected number of processors doing all useful work.

The SMMPCC throughput at capacity is obtained from the processing power and the subcall real time requirements. The available processing power can also be interpreted as the SMMPCC throughput at capacity when the average subcall real time requirement is one unit of time, i.e. when single processor core throughput at capacity is 1. We assume that subcall real time requirement in this paper, since we are interested in throughput behavior and not its values for a particular subcall mix.

SMMPCC throughput depends on potential blocking, number of processors in the core and the real time used by a rollback. We discuss the dependence on the number of processors first.

## 4.1 SMMPCC Throughput for Different Numbers of Processors

We eliminated the impact of the point in subcall execution where contention occurs by setting q at a fixed level of 0.8.

Figure 1 shows SMMPCC throughput for three different cases of potential blocking. In all cases, 80% of first offered subcalls did not access shared memory. In the first cas the remaining 20% of the subcalls were in the same contention group, i.e. n = 1, $f_1 = 0.2$. In the second and third cases, the 20% of subcalls accessing shared memory were divided into 2 and 4 contention groups, respectively. The model parameters in those cases were: n = 2, $f_1$, $f_2 = 0.1$, and n = 4, $f_i = 0.05$ for i=1,2,3,4. Throughput when no subcall accesses shared memory, so there is no memory contention, is shown as a reference.

SMMPCCs with more processors have more processing power to execute subcalls. On the other hand, they process more subcalls concurrently, thus the probability of memory contention is higher. Consequently more and more real time is spent on rollbacks and additional processors increase throughput by smaller amounts. The difference in throughput between the no- memory contention case and any scenario with memory contention grows with the number of processors.

The increments of SMMPCC throughput decrease with the number of processors until additional processors practically do not increase the throughput as it approaches the asymptotic level given by Theorem 2. Those levels are respectively: 6.25, 12.5 and 25 for one, two and four contention groups in Figure 1.

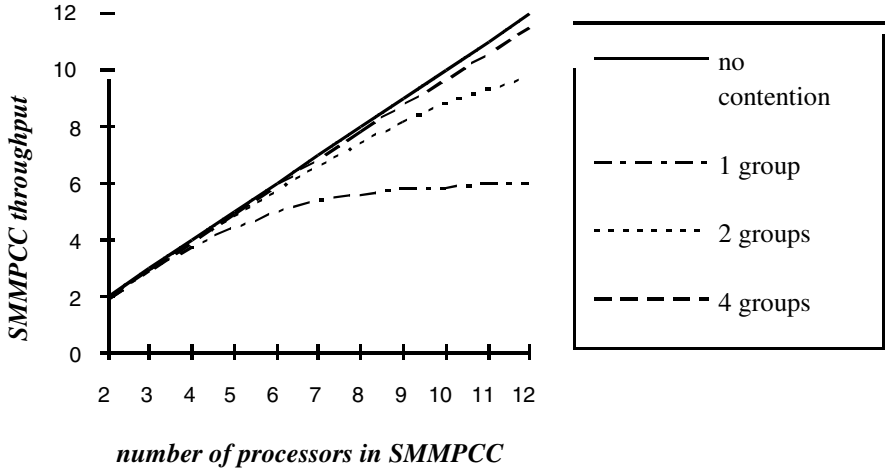



**Fig. 1.** SMMPCC throughput as a function of the number of processors

As the chart on Figure 1 shows, the SMMPCC throughput strongly depends on the potential blocking of the offered load. The load where the potentially blocking subcalls are split into four groups results in the highest throughput. In this case the probability of concurrent execution of subcalls from the same group is smaller than when there are two or one blocking group.

The difference in throughput between the presented case of potential blocking is very small for 2 to 5 processors. The probability that two subcalls will compete for an access to a memory address is quite small when 80% of first offered subcalls does not require such access. In general, small numbers of processors are robust to changes in potential blocking as long as that blocking is not too high.

### 4.2 SMMPCC Throughput Dependence on the Fractions into Subcall when Contention Occurs

A fraction of subcall execution before contention occurs impacts SMMPCC throughput in two ways. Firstly, because the total real time used by rolled back subcalls is a total of those real times. Secondly, because the probability of rollback depends on how long subcalls own the addresses which they have locked.

These two ways counter each other. The sooner a subcall accesses and locks a contention spot the longer it owns it and the probability that another subcall will try to access the locked address is higher. There will be more blocked subcalls but they will waste less real time. If, however, shared memory addresses are accessed, on average, late in subcall executions, time intervals when contention may occur are shorter, but

when it occurs its real time cost is higher. The question is, which factor is more important: probability of blocking or real time waste per rollback.

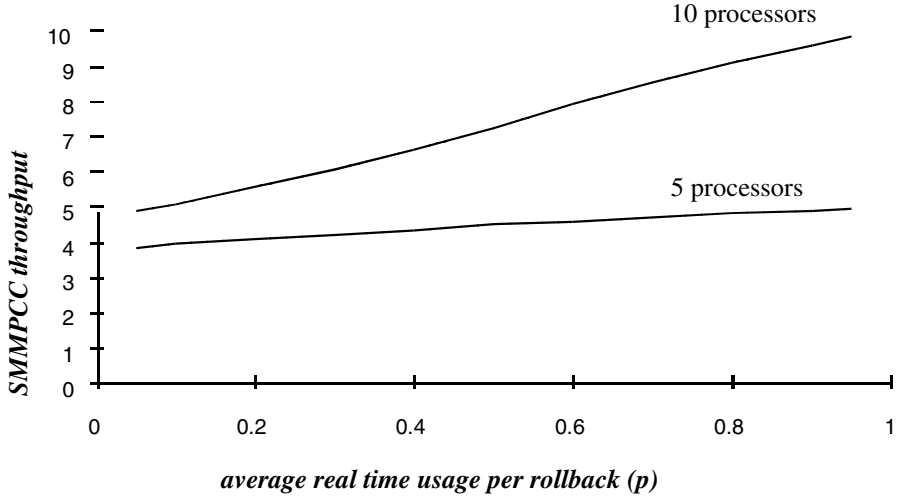



**Fig. 2.** SMMPCC throughput as a function of subcall fraction before contention

The chart on Figure 2 shows SMMPCC throughput when the fraction of subcall which is executed before contention occurs (1-q) ranges from 0.05 to 0.95. Since average subcall work time is one unit of time, these real time usages can also be interpreted as fractions of subcall work times. 20% of first offered subcall belong to a single contention group contention group, i.e. $n = 1$, $f_1 = 0.2$.

Figure 2 shows that the SMMPCC throughput increases when contention happens later into subcall execution. The shorter locking times, and subsequently the smaller probabilities of rollback have stronger effect on throughput than the increase in average real time wasted per rollback. In the 10 processor case the increase is much greater
but there is more room for it. When a rollback happens, on average, at 0.05 into the execution, the throughput is slightly above five processors, which is half of the maximum throughput. Five processors start at the same point with throughput about 4.5 which is 90% of the maximum.

## 5 Conclusions

Throughput of an SMMPCC increases with the number of processors.

In the presence of memory contention the throughput tends to a finite limit, which depends on potential blocking, and not on the number of processors.

Throughput is higher when subcalls access contentious memory addresses later in their executions.

For higher contention levels the SMMPCC throughput depends very little on the number of processors.

Processing power is never less than equivalent to one processor.

## References

1. Thomasian, A.: Two-Phase Locking Performance and Its Thrashing Behavior. ACM Transactions on Database Systems, December, 18(1993)4, 579-625
2. Tay, Y.C., Suri, R., Goodman, N.: A Mean Value Performance Model for Locking in Databases: The No-Waiting Case. Journal of the Association for Computing Machinery, July, 32(1985)3, 618-651
3. Thomasian, A.: On a More Realistic Lock Contention Model and Its Analysis (1994) 2-9
4. Thomasian, A.: Performance Evaluation of Centralized Databases with Static Locking. IEEE Trans. Software Engineering, April, 11(1985)4, 346-355
5. Cooper, R.: Introduction to Queueing Theory. Macmillan, New York (1972)

## APPENDIX

The model has been validated against simulations of shared memory contention. The simulator generated subcalls and queued them at a single queue where they waited for execution. For each subcall its execution length was generated from a negative exponential distribution. The subcalls were generated at a rate such that the utilization of the simulated SMMPCC was almost 100%. The generated subcalls were assigned with probabilities $f_1$, $f_2$, ... , $f_n$ to the contention groups. Subcalls which were not selected to any contention group were considered contention free.

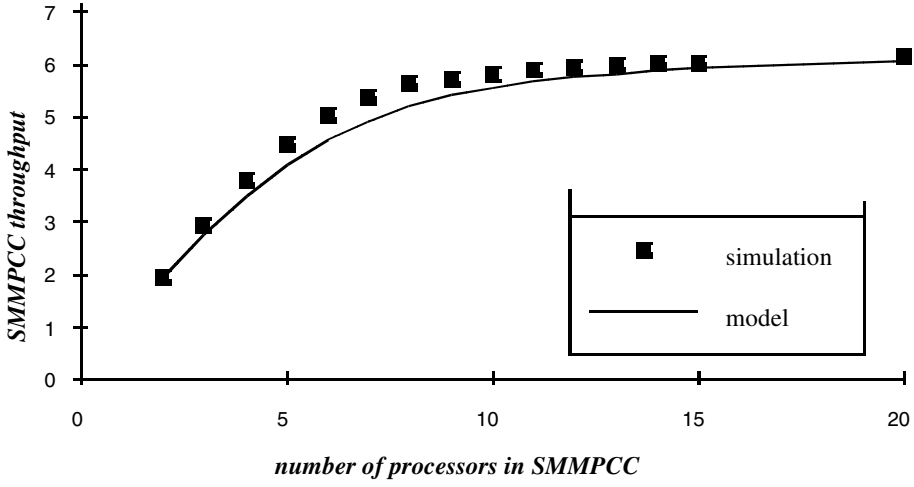



**Fig. 3.** SMMPCC throughput from the model versus simulations

When a processor ended an execution, it immediately started executing the head of the subcall queue. If the executed subcall belonged to a contention group, at the fraction p into the execution the processor requested the portion of the shared memory contentious for this group. If the memory was not locked by another processor, the access was granted and the memory was locked. Otherwise, the subcall was rolled

back, i.e. its execution was aborted and the executing processor began executing the head of the subcall queue. The rolled back subcall was kept in a buffer until the subcall, which locked the contentious memory ended its execution, and then it was sent back to the subcall queue.

Figure 3 shows a comparison between model and simulation results. A single contention group generated 20% of the first offered subcalls, and the rollback real time usage p = 0.2. The comparison was made for the number of processors from 2 to 20.

The model accuracy, as compared against the simulation results, is within a few percent (max 10%), and always on the conservative side. It is very high for low and high contention levels in the system. In Figure 3 this corresponds to less than 5, or more than 10 processors. Under the assumed potential contention level, the blocking in less than five processor systems is low. It becomes medium when the number of processors is between 5 and 10, and is high for larger systems.

# Performance and Scalability Models for a Hypergrowth e-Commerce Web Site

Neil J. Gunther

Performance Dynamics Company, 4061 East Castro Valley Boulevard,
Suite 110, Castro Valley, CA 94552, U. S. A.
njgunther@perfdynamics.com
http://www.perfdynamics.com/

**Abstract.** The performance of successful Web-based e-commerce services has all the allure of a roller-coaster ride: accelerated fiscal growth combined with the ever-present danger of running out of server capacity. This chapter presents a case study based on the author's own capacity planning engagement with one of the hottest e-commerce Web sites in the world. Several spreadsheet techniques are presented for forecasting both short-term and long-term trends in the consumption of server capacity. Two new performance metrics are introduced for site planning and procurement: the *effective demand*, and the *doubling period.*

## 1 Introduction

Most Web sites are configured as a highly networked collection of UNIX and NT servers. The culture that belongs to these mid-range environments typically has not embraced the concepts of performance engineering in applications development. Moreover, many of the larger commercial web sites are large because they are success disasters. In other words, a suggested dot com business idea was implemented in a somewhat ad hoc fashion and later found to be far more attractive than site architects had originally anticipated. Now the site is faced with explosive levels of online traffic and this has brought capacity planning concerns to the forefront. Highly publicized Web sites such as Amazon.com, AOL.com, eBay.com, eToys.com, and Yahoo.com are just a few examples of success stories that have experienced variants of this effect.

Here, *success* is measured in terms of a site's ability to attract internet traffic. This is not necessarily equivalent to financial success. As many e-commerce Web sites have discovered (perhaps to the chagrin of Wall Street watchers), connections per second and dollars per connection are not always positively correlated. Nonetheless, it has become abundantly clear that server performance and site scalability are critical issues that can seriously affect the commercial viability of a Web site. The piece of the puzzle that is still missing is the concept of *planning* for site capacity. In fact, given the chaotic environments and undisciplined culture that are so typical of hypergrowth Web sites, one could be forgiven for thinking that *capacity planning* is a web-oxymoron [1]. There are, however, some encouraging signs that this situation is changing. A quick review of recent trade literature [2], [3], [4], [5] indicates that

R. Dumke et al. (Eds.): Performance Engineering, LNCS 2047, pp. 267–282, 2001.

hyper-growth Web sites are beginning to appreciate the importance of application performance engineering and server scalability planning [6]. Even though additional server capacity is usually foreseen to be necessary, it must be procured well in advance of when it is actually needed. Otherwise, by the time the new servers arrive, traffic will have grown explosively and the additional capacity will be immediately absorbed with no advantage gained from the significant fiscal outlay.

In this chapter, we present a case study in capacity planning techniques developed by the author for one of the world's hottest web sites (which shall remain nameless for reasons of confidentiality). A notable attribute of the work described here is that the performance models used for planning must be lightweight and flexible to match the fast-paced growth of these environments [1]. The chapter falls logically into two parts. The first part (section 6) deals with determining short-term effective demand. The second part (section 7) uses peak effective demand statistics to project long-term server growth. Long-term growth projections, expressed in terms of the capacity *doubling time,* set the pace for the hardware procurement schedule.

## 2 Analysis of Daily Traffic

One of the most striking features of e-commerce traffic is the unusual variance in its intensity during the day. Mainframe capacity planners are already familiar with the bimodal "10-2" behavior of conventional commercial workloads due to online user activity [7]. The 10-2 designation refers to the twin peaks in CPU utilization occurring around 10am in the morning and 2pm in the afternoon (local time) within each nine-to-five day shift. There are two peaks because of the lunchtime slump in system activity around midday.

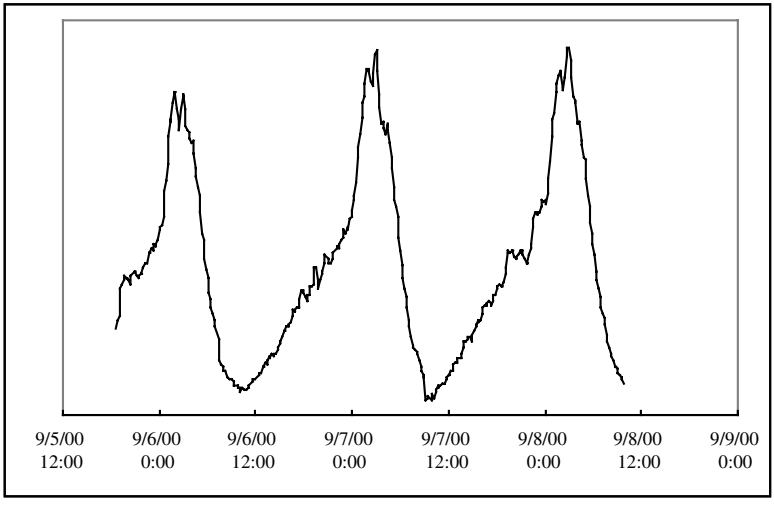



**Fig. 1.** Characteristic daily peaks in traffic and resource utilization seen in many successful North American Web sites. Time is expressed as Coordinated Universal Time (UTC)

On the World Wide Web, however, the "day shift" can last for twenty-four hours and together with randomized traffic arrivals it is not clear whether any characteristic peaks exist at all; bimodal or otherwise. Worldwide access twenty-four hours a day, seven days a week, across all time zones suggests the traffic intensity might be very broad without any discernable peaks.

On the contrary! Not only did the measured traffic intensity peak, there was always a dominant peak around 2:00 hours UTC (Coordinated Universal Time). The single daily peaks in Fig. 1 also show up consistently in the measured CPU utilization of in-situ Web site servers as well as the measured outbound LAN (Local Area Network) packets inside the Web site. Moreover, this traffic characteristic is quite universal across all hyper-growth Web sites. The reason for its existence can be understood as follows.

### 2.1 Bicoastal but Unimodal

North American Web site traffic can be thought of as being comprised of two major contributions: one from the east coast and the other from the west coast. This naive partitioning corresponds to the two major population regions in the USA. If we further assume the traffic intensities (expressed as a percentage) are identical but otherwise phase-shifted by the three hours separating the respective time zones, Figure 2 shows how these two traffic profiles combine to produce the dominant peak. As the west coast contribution peaks at 100% around 4:00 hours UTC, the east coast contribution is already in rapid decline because of the later local time (7:00 hours actual UTC). Therefore, the aggregate traffic peak occurs at about 2:00 hours UTC.

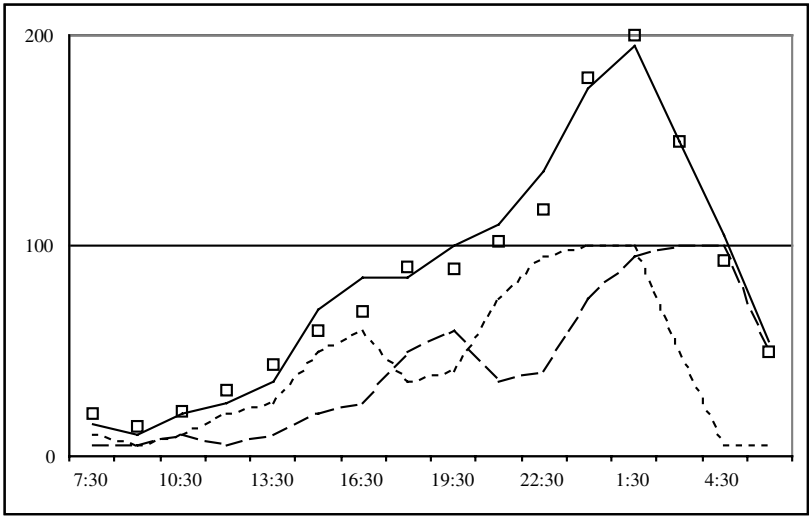



**Fig. 2.** Measured daily traffic intensity at a USA site (boxes). The single dominant peak (solid line) at 2am UTC is the sum of contributions from the east coast (dotted line) and the west coast (dashed line) phase-shifted by 3 hours for the respective time zones. As the western contribution is peaking, the eastern contribution is declining rapidly because of the later local time

A benign bimodal character can be seen in the aggregate data of Fig. 2 with a small hump occurring around 16:30 hours UTC. This corresponds to a much more distinct bi-modality in both component traffic profiles and is reminiscent of the „10-2“ mainframe traffic characteristic mentioned earllier but now reversed and shifted in time.

### 2.2 All American!

Another important conclusion can be drawn from this simple traffic analysis. There is essentially no significant traffic coming from time zones in Europe, Asia, Australia or the Pacific region. If these regions were contributing (albeit at less intensity) they would broaden the dominant peak significantly. Hence, the 10-2 bimodal profile seen in mainframe traffic can be understood as belonging to very localized usage where most users are connected to the mainframe by terminals or workstations and they are active in the same time zone. Although hyper-growth Web site traffic is delocalized into multiple time zones, it is still confined mostly to North America and within that continent, it is dominated by the activity of the coastal populations.

Several corollaries follow from these observations about the hyper-growth traffic profile:

- When it appears daily, we know we are looking primarily at North American users.
- This activity will be reflected in the consumption of other site resources e.g., server utilization, and network bandwidth.
- We can use it as a quick validation of any modeling predictions (e.g., section 6 and Fig. 3).

In the subsequent sections, we shall focus entirely on back-end server capacity rather than network capacity. The site in this study had plenty of OC12 bandwidth to handle outbound packets so, network capacity never caused any (non-pathological) performance problems.

Since the measured server utilization is bounded above by 100% at CPU saturation, a peak in server utilization caused by the characteristic traffic profile discussed above would remain obscured. One of the objectives in this study was to reproduce a server utilization profile that included such peaks. To this end we now introduce the concept of effective server demand.

## 3 Effective Server Demand

Effective demand is a measure of the work that is being serviced as well as the work that could be serviced if more capacity was available. It is similar to the mainframe notion of "latent demand" [7], [8]. The performance metric is dimensionless in the same way that CPU-busy is dimensionless but unlike CPU-busy, which is bounded above by 100%, effective demand can be expressed as an unbounded percentage. For example, an effective demand of 167% means that the application workload could have been serviced by one and two-thirds servers, even though only one completely saturated server was physically available to accommodate that workload.

Once a server becomes saturated, the run-queue necessarily begins to grow and this can have an adverse impact on user-perceived response time. Although there was some discussion about making response time measurements to gauge the expected impact of capacity upgrades, no commitment was undertaken by the site management during the engagement period.

### 3.1 Modeling Assumptions

The effective demand was calculated using the spreadsheet regression tool. The predictor is the effective demand (expressed as a percentage). Appropriate regressor variables can be determined from a factorial design analysis [9], [10], [11]. Typical regressors can include CPU clock frequency, the number of user submitting work to the system, the run-queue length, I/O rates, etc. One performance metric that cannot be used is the measured CPU utilization itself, since that is the variable we are trying to predict [8].

As with any statistical trend analysis of CPU capacity, we assumed that the CPU-intensive workload would remain so as more CPU capacity was added. It could happen that removing the CPU bottleneck by adding just a small amount of CPU capacity simply brings a disk or memory bottleneck to the surface and this would become the new scalability inhibitor.

### 3.2 Statistical Approach

There is no way to see such shifting bottlenecks a priori without the aid of more abstract performance tools; such as multi-class queueing models [12]. But a queueing model require that the underlying abstraction of the system be accurate. Without more sophisticated measurements than were available and more time (a precious commodity in hyper-growth Web sites) to validate such abstractions, the fallback to statistical data modeling was declared appropriate [13].

We analyzed ten weeks of performance data comprising two-minute samples of approximately two hundred metrics. During that period, many upgrades had been performed on the back-end servers of interest. These included upgrading CPUs and cache sizes, upgrading the version of ORACLE, and modifying the proprietary application software across the site. As a consequence, only a subset of that data was stable enough to employ for forecasting server capacity consumption.

## 4 Choosing Statistical Tools

As mentioned in section 3.1, it was decided to undertake a purely statistical analysis of the raw performance measurements of hyper-growth traffic at this Web site. A statistical analysis of this type typically uses analysis of variance (ANOVA [11]), regression [9], and time-series analysis (ARMIA [14]). However, because of the relatively short time over which performance metrics had been collected, no seasonal or other long-term periodic trends were present and time-series analysis was not needed.

Having elected to apply statistical analysis, the next issue was the selection of statistical tools to use for the analysis and, in particular, whether to purchase statistical software or build a custom application integrated into the current environment? The methods presented in this chapter had not been used elsewhere by the author and this meant that some prototyping was inevitable.

It was also not clear at the outset which statistical functions would be required. For example, implementing (which really means debugging) a set of statistical functions based on Numerical Recipes in C [15] or Perl [16] would be more time consuming than using an environment where a reasonably complete set of statistical functions already existed.

### 4.1 Spreadsheet Programming

The choice became obvious once it was realized that almost every desktop at the Web site had a PC running Microsoft Office and thereby had EXCEL readily available. What is not generally appreciated is that EXCEL is not just a spreadsheet [17]; it is a programming environment with Visual Basic for Applications (VBA) [18] as the programming language.

VBA is a reasonable prototyping language because it is object-oriented, comes with an integrated debugger, and has a macro-capture facility. As well as being ubiquitous, EXCEL is also platform-independent. The author could develop the tool off-site on an iMac and email it to the work place as an attachment to be downloaded on a resident PC for testing.

### 4.2 The Internet Is Your Friend

Moreover, there are several Internet news groups [19] devoted to spreadsheets and VBA programming so that expert help is readily available for any obscure VBA programming problems. As we shall see later, data filtering was also an important requirement for selecting measured performance data according to some specified criteria. Plotting (or "charting" as it is called in EXCEL) is integrated into spreadsheets with a VBA programmable interface.

A broad set of standard statistical analysis functions (e.g., Regression, ANOVA, moving average) is available by default. This proved very useful because it was not known exactly which functions would be needed to analyze the performance data. EXCEL, as packaged, does not include the more sophisticated analysis tools (such as ARIMA) but these can often be found on the internet or as commercial add-in packages. Finally, there is an option to import data over the Web and also to publish spreadsheets as HTML pages.

## 5 Planning for Data Collection

Ideally, data center operations and capacity planning should be distinct groups. But for understaffed Web sites, economics often dictates the need for a leaner infrastructure than might otherwise be available traditionally.

Most data center operations, including hyper-growth Web sites, understand the need to collect performance data. Many data center managers are also quite prepared to spend several hundred thousand dollars on performance measurement tools. However, what is not yet well recognized in most operations centers is that the purpose for which such data is intended should determine how it is collected and stored. This means that operations center management has to think beyond immediate monitoring and schedule planning into a longer-term picture.

### 5.1 Use It or Lose It

The Web site in this study had purchased a commercial data collection product and had been using it for about six months prior to this capacity planning study. In order to conserve disk storage consumption these products have default settings whereby they aggregate collected performance metrics. For example, data might be sampled every few minutes but after eight hours all those sampled statistics (e.g., CPU utilization) will be averaged over the entire eight-hour period and thereby reduced to a single number. This is good for storage but bad for modeling and analysis. Such averaged performance data is likely far too coarse for meaningful statistical analysis. Some commercial products offer separate databases for storing monitoring and modeling data.

Either the default aggregation boundaries should be reset, or operations management needs to be prescient enough to see that whatever modeling data is needed gets used prior to aggregation or is siphoned off and stored for later use. In either case, data collection and data modeling may need to be scheduled differently and well in advance of any data aggregation. This idea is new to many Web site data centers.

### 5.2 Saved by the SymbEl

And so it was with the site described here. Fortunately, alternative data was available because the platform vendor had installed their own non-commercial data collection tools written in the SymbEl or SE scripting language [20]. Since the user interface to this monitoring tool employed a java-enabled browser, and the job of the tool was simply to collect performance data in the background, it was called „SE Percolator“. Thankfully, SE Percolator was unaware of sophisticated concepts such as aggregation and kept two-minute samples of more than two hundred performance metrics. It was this data that was used in our subsequent statistical modeling.

## 6 Short-Term Capacity

As discussed in section 3.1, the effective CPU demand is calculated using regression techniques. We new present that technique in more detail.

### 6.1 Multivariate Regression

Because of a prior succession of changes that were made to the system configuration, only five weeks of data were stable enough to show consistent trending information. An EXCEL macro was used to analyze the raw metric samples and predict the effective demand using a multivariate linear regression model of the form

$$U^* = a_1 X_1 + a_2 X_2 + \ldots + a_0 , \qquad (\mathbf{1})$$

where each of the X's is a regressor variable, and the α's are the coefficients determined by an ANOVA analysis of the raw performance data. Here, $U^*$ refers to the estimated effective utilization of the server and it can exceed 100%.

### 6.2 Spreadsheet Macros

A Perl script [16] extracts the raw SymbEl Percolator data (section 5.2) averaging over15-minute samples of the relevant time-stamped performance metrics for each day of interest. The extracted data is then read directly into a spreadsheet using the EXCEL Web Query facility. This produces about 100 rows of data in the spreadsheet. Two macros are then applied to this data.

The first of these filters out any row if it contains a measured CPU utilization of 95% or greater. Such rows are eliminated from the ANOVA calculations as being too biased for use by equation (1). The regressor variables (labeled $X_1$ through $X_6$ in Table 1) are then used by the macro to calculate the α coefficients of equation (1). Once these coefficients are known, the $U^*$ value (third column in Table 3) is computed for each row in the spreadsheet. In general, the estimated and measured utilization will be fairly close until it gets near to saturation (e.g., 95% or greater), in which case significantly larger values of $U^*$ are estimated as shown in Figure 3. The complete VBA code for these modeling macros can be found in [1].

**Table 1.** Example of EXCEL spreadsheet layout for the multivariate regression model

| DateTime | $U$ | $U^*$ | $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | $X_6$ |
|---|---|---|---|---|---|---|---|---|
| 9/29/99 0:00 | 25.25 | 32.90 | 32 | 19 | 16.45 | 18.96 | 15.04 | 131.56 |
| 9/29/99 0:16 | 27.25 | 36.85 | 45 | 11 | 17.01 | 22.49 | 14.18 | 136.08 |
| 9/29/99 0:32 | 47.12 | 54.01 | 50 | 42 | 29.52 | 33.32 | 27.07 | 236.13 |
| 9/29/99 0:48 | 45.88 | 51.19 | 53 | 38 | 27.29 | 32.09 | 24.62 | 218.29 |

## 7 Long-Term Capacity

Summary statistics from the short-term multivariate model described in Section 6 were then taken over into a weekly spreadsheet. About eight weeks worth of these summary statistics were needed to make reasonable long-term growth predictions.

### 7.1 Nonlinear Regression

For this phase of the exercise, a nonlinear regression model was also built in using spreadsheet macros. The maxima of the weekly effective demands, $U^*$, calculated from the multivariate model were fitted to a single parameter exponential model of the form,

$$U(w) = U_0 e^{bW}, \tag{2}$$

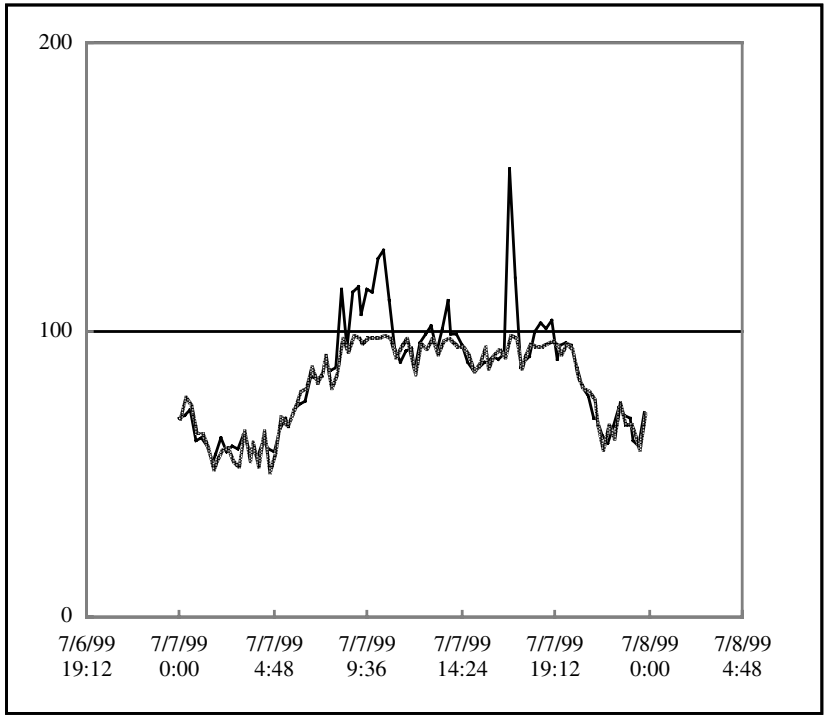



**Fig. 3.** Effective demand (solid) sometimes exceeding 100% compared with raw CPU utilization data (dashed line). Note the spike in utilization near 18:00 hours as discussed in section 2

where the long-term demand U(w) is expressed in terms of the numbers of weeks since the data analysis began. The constant $U_0$ is the CPU utilization at the zeroth week and the parameter 'b' is fitted by the Trending capabilities of EXCEL. That parameter specifies the growth rate of the exponential curve. For the data in Fig. 4, $U_0$ = 80.62 and b = 0.0309. We chose an exponential nonlinear server growth-model because it is expected that server capacity will need to scale beyond simple linear growth. Similar expectations exist for business growth models and it can be seen in the measured growth of network traffic.

### 7.2 Doubling Period

A more intuitive grasp of the growth parameter comes from considering the time it takes to *double* the required server capacity. Thus, the natural logarithm of two divided by the rate parameter b in Fig. 4 gives a doubling time corresponding to 5.6 months. In other words, every 6 months, twice as much server capacity will be consumed as is being consumed now!

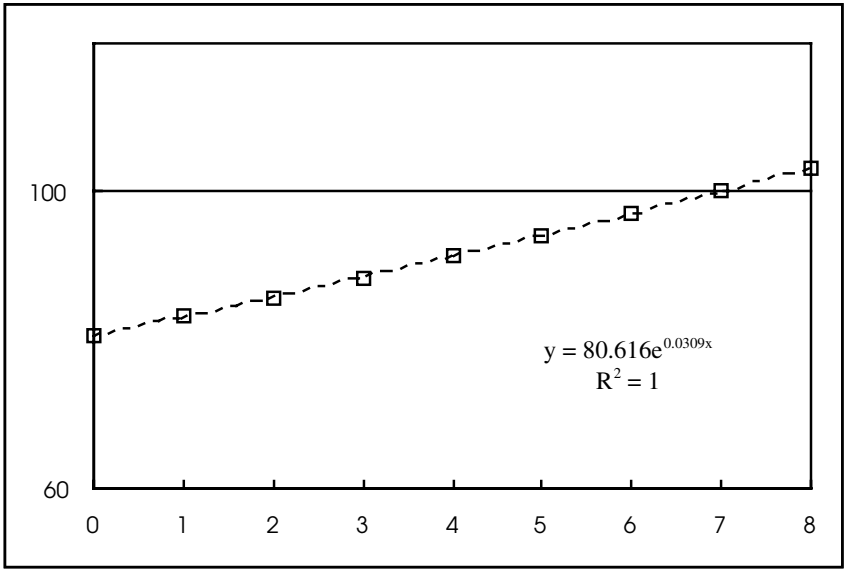



**Fig. 4.** Exponential regression (dashed line) on eight weeks of peak demand (boxes) from the short-term multivariate model in section 6. The curvature parameter b = 0.0309

This is at least an order of magnitude faster than the growth of typical mainframe data processing workloads and four times faster than Moore's Law [12] which predicts the number of transistors that can be packed into VLSI circuitry every two years. Nonetheless, this startling rate of growth agrees with growth of network traffic measured at this Web site

Fully loaded back-end database servers cost tens of millions of dollars. Considering that even the most lucrative hyper-growth Web sites only have revenue streams in the range of tens of millions of dollars per annum, a doubling time of six months amounts to nothing less than a Web definition of bankruptcy!

## 8 Upgrade Paths

The final task was to translate the results from these effective demand models into procurement requirements. This provides a set of upgrade curves corresponding to

different possible CPU configurations, cache sizes, and clock speeds. Since the above regression models only pertain to the measurements on the current system configuration, we need a way to extrapolate to other possible CPU configurations.

### 8.1 Super-Serial Scaling

The next task was to translate this explosive growth rate into a set of sizing requirements for server procurement. For this purpose, we used the super-serial scaling model [12] which relates the effective capacity C(p) of a server to the number of physical processors (p) as the function defined by,

$$C(p) = \frac{p}{1 + \sigma\{(p-1) + \lambda p(p-1)\}} \quad . \tag{3}$$

This function can be fitted to arbitrary measured CPU configurations. The throughput, X(p), can then be predicted by multiplying an arbitrary measured throughput by C(p) in equation (3) [22]. Fitting the super-serial model to server scaling data provides estimates for two parameters corresponding to such confounded hardware and software factors as:

- serial delays ($\sigma$) due to contention for mutex [20] and database locks [21].
- coherency delays ($\lambda$) due to stale cache refetching [21].

Because it would take us too far afield to present it here, the interested reader is encouraged to see references [12], [21] for a more extensive discussion of the super-serial model and its derivation.

The back-end servers at this web site were running the ORACLE database management system and could have up to 64-way CPUs. In the author's experience, the contention factor for ORACLE is typically around 3% or $\sigma = 0.030$ and the coherency term is typically $\lambda = 0.002$. This allowed us to make a realistic estimate of capacity factors for server configurations of interest.

Referring now to Table 2 which contains vendor scaling estimates, we see that adding 12 more 333 MHz CPUs with 4MB secondary cache to the existing 52-way, shows a 15% gain in headroom. But keeping the CPU configuration fixed at 52-way and increasing the CPU model from 333 MHz/4MB to 400 MHz/8MB, shows a 32% headroom gain. Performing both of these upgrades together shows a 52% gain in headroom. This seems rather optimistic for an ORACLE application.

Referring to Table 3 which contains more conservative estimates, we see that adding 12 more CPUs to go from a 52-way to 64-way at the same clock speed 333 MHz and secondary cache size of 4MB, indicated a 6% gain in headroom. Keeping the CPU configuration fixed at 52-way but increasing the CPU clock speed from 333 MHz/4MB to 400 MHz/8MB, shows a 32% gain in headroom. Performing both of these upgrades together reveals a 39% headroom gain.

## 8.2 Server Scalability Calculation

Going from a 333 MHz to a 400 MHz clock while maintaining a 52-way configuration on the backplane and using the vendor scaling data in Table 2, we first

**Table 2.** Vendor scaling data in transactions per minute. Reading left to right corresponds to changing CPU clock frequency (ΔCLK) with the same CPU configuration, while reading downward corresponds to increasing the number of physical CPUs (ΔCPU) at the same clock speed. These changes are also expressed as percentages in the outside column and row

| System | 333/4 | 400/8 | ΔCLK | Percentage |
|---|---|---|---|---|
| 52-way | 115,755 | 152,432 | 36,677 | 0.32 |
| 64-way | 133,629 | 175,969 | 42,340 | 0.32 |
| ΔCPU | 17,874 | 23,537 | 60,214 | N/A |
| Percentage | 0.15 | 0.15 | N/A | 0.52 |

**Table 3.** Super-serial scaling model in transactions per minute. Reading left to right corresponds to changing CPU clock frequency (ΔCLK) with the same CPU configuration, while reading downward corresponds to increasing the number of physical CPUs (ΔCPU) at the same clock speed. These changes are also expressed as percentages in the outside column and row

| System | 333/4 | 400/8 | ΔCLK | Percentage |
|---|---|---|---|---|
| 52-way | 57,605 | 75,859 | 18,254 | 0.32 |
| 64-way | 60,875 | 80,165 | 19,290 | 0.32 |
| ΔCPU | 3,270 | 4,306 | 22,560 | N/A |
| Percentage | 0.06 | 0.06 | N/A | 0.39 |

calculate the increase ($\delta$) in CPU capacity from the tabulated throughput values:

$$\delta = \frac{X(400) - X(333)}{X(333)}$$
$$= \frac{36,677}{115,755}$$
$$= 31.7\%$$

Scalability envelopes corresponding to estimates in both Tables 2 and 3 and shown in Figure 5.

Next, the increase in CPU capacity is used to adjust the weekly growth curve, C(w), downward because a lower curve takes longer to reach saturation. At week 20, $U_{333}(20)$ for the baseline server is 149.56 percent. By upgrading the CPU clock speed to 400 MHz, we know $\delta = 0.317$, and therefore the corresponding decrease in the effective demand can be determined as:

$$\begin{aligned} U_{400}(20) &= U_{333}(20) \times (1 - 0.317) \\ &= 149.56 \times 0.683 \\ &= 102.15 \end{aligned}$$

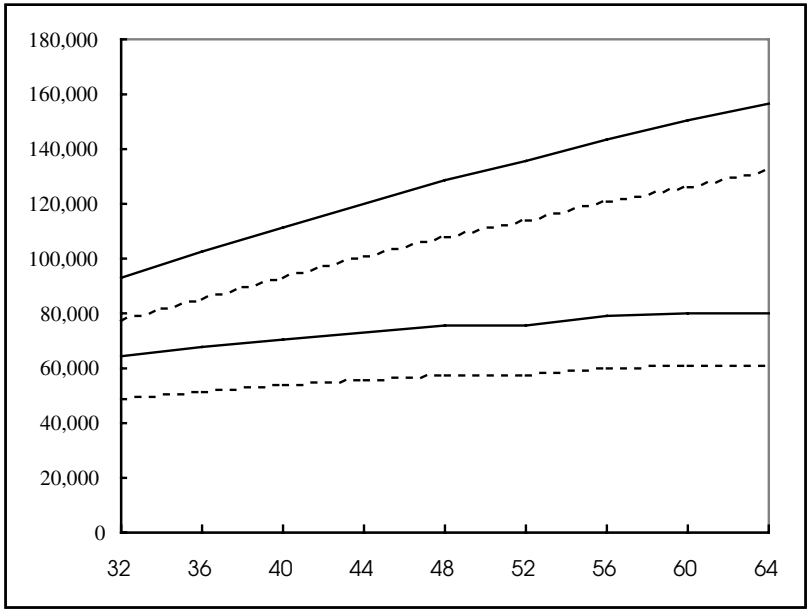



**Fig. 5.** Scalability envelopes corresponding to the values calculated in Tables 2 and 3. The solid lines represent the vendor (upper) and realistic (lower) CPU throughput (in transcactions per minute) for the 400 MHz/8 MB cache processor. Similarly, the dashed lines represent the vendor (upper) and realistic (lower) CPU throughput (in transcactions per minute) for the 333 MHz/4 MB cache processor

The corresponding long-term growth curves are shown in Figure 6 for the more conservative super-serial model.

Each of these scenarios could have been made more accurate if actual workload measurements had been available. Unfortunately, this Web site, like so many, was only in the early stages of using load test and software QA platforms prior to deployment. If the load testing had been further advanced, it might have been possible to use those workload measurements to improve some of our capacity predictions. In the meantime the site needed to make some fiscally responsible decisions regarding expensive server procurement and they needed to make those decisions quickly.

Under those circumstance (not unusual, as we noted in the Introduction to this chapter) any sense of direction is more important that the actual compass bearing [13].

## 9 Summary

In this case study, we have shown how multivariate regression can used to analyze short-term effective CPU demand in highly variable data. The effective demand is a measure of the work that is being serviced as well as the work that could be serviced if more capacity was available. Daily data was collected and the *effective demand* calculated using spreadsheet macros. A weekly summarization of the peak effective demand was then used to generate a *nonlinear regression* model for long-term capacity growth and, in particular, extraction of *doubling* period.

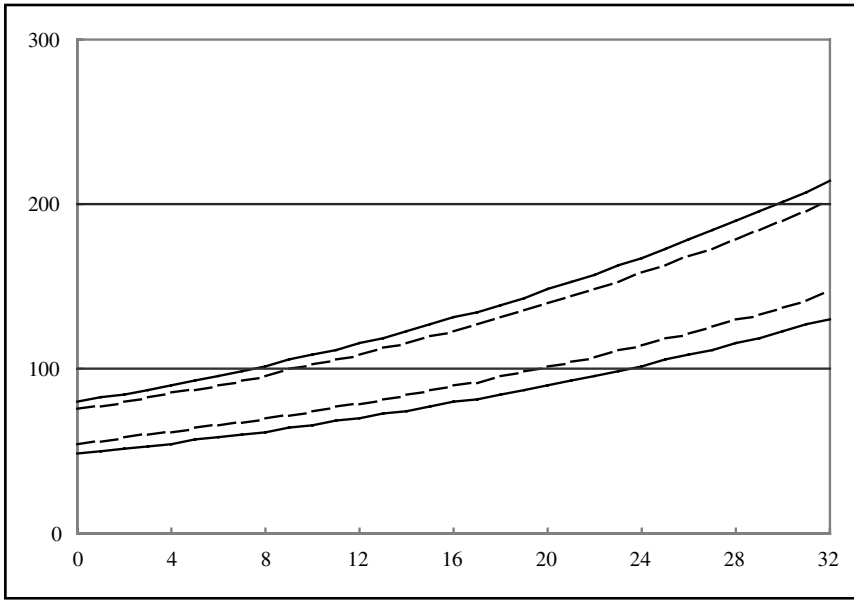



**Fig. 6.** Super-serial CPU capacity projections corresponding to the values calculated in Table 2. The x-axis represents the number of weeks since the capacity analysis data was begun. Week 20 corresponds to the Y2K boundary. The horizontal lines at 100 and 200 percent represent respectively a single server and two servers. The upper and lower solid curves represent respectively the worst case (maintain the current CPU configuration) and best case (fully loaded back-plane with fastest clock speed) projections

The study presented in this chapter provided the first quantitative capacity model of the growth rate of this very successful Web site, A doubling period of about six months, was quickly recognized as having potentially devastating implications for the fiscal longevity of the Web site. Several performance engineering actions (that were not predicted by our capacity model) were immediately undertaken.

A purchase order for a clustered back-end server was written without further delay. In addition, the chief software developer immediately undertook to implement some thirty software changes that had been planned but never implemented. These software changes alone recovered thirty percent headroom on the existing back-end server and corresponded to increasing the doubling time to approximately 15 months. As a side effect, the model presented in this chapter was immediately invalidated!

Other important conclusions concerning hyper-growth Web sites also stemmed from our study. Almost all the significant traffic comes from just two North American time zones with very little coming time zones in Europe, Asia, Australia or the Pacific region. This relative localization of potential Web traffic is responsible for the large single peak discussed in section 2. The long-term growth of this peak is of prime concern for server procurement.

As Web connectedness continues to extend outside North America, this dominant peak can be expected to broaden and sustained capacity will become the new issue.

Most hyper-growth Web sites understand the need to collect performance data. Data center managers are prepared to spend hundreds of thousand of dollars on performance monitoring tools. What is not yet well recognized is that the purpose for which such data is intended should determine how it is collected and stored. This means that operations management has to think beyond immediate performance monitoring and schedule capacity planning collection and storage into a longer-term operations plan.

## References


1. Gunther, N.J.: e-Ticket Capacity Planning: How to Ride the e-Commerce Growth Curve. Proceedings of the Computer Measurement Group (CMG) Conference, Orlando, Florida (2000)
2. McCune, J.C.: Keeping Tabs on Your Web site. Beyond Computing, March (2000)
3. Radosevich, L.: Designing a Growing Back End. InfoWorld, Aug. 23, online article at http://www.infoworld.com/cgi-bin/displayArchive.pl?/99/34/t25-34.34.htm (1999)
4. Nash, L.S.: Hyper Growing Pains. ComputerWorld, Sept. 11, online article at http://www.computerworld.com/cwi/story/0,1199,NAV47_STO49841,00.html (2000)
5. Power, D.: Sound Planning Gets Lost in e-Speed Pace. Executive Technology February 2(2000)2, P. 21
6. Shea, B.: Avoiding Scalability Shock: Five Steps to Managing Performance of e-Business Applications. Software Testing and Quality Magazine, May/June issue (2000)
7. Hall, G., August, L.: MVS Performance Management Reporting. Proceedings of the Computer Measurement Group (CMG) Conference, Reno, Nevada (1992) 443-451
8. Forst, F.: Latent Demand; The Hidden Consumer. Proceedings of the Computer Measurement Group (CMG) Conference, Orlando, Florida (1997) 1011-1017
9. Allen, A.O.: Probability, Statistics, and Queueing Theory with Computer Science Applications. Academic Press, New York (1990)
10. Box, G.E.P., Hunter, W.G., Hunter, J.S.: Statistics for Experimenters – An Introduction to Design, Data Analysis, and Model Building. Wiley, New York (1978)
11. Jain, R.: The Art of Computer Systems Performance Analysis. Wiley, New York (1990)
12. Gunther, N.J.: The Practical Performance Analyst. 2nd printing. iUniverse.com Inc., Lincoln, Nebraska (2000)
13. Gunther, N.J.: Guerilla Capacity Planning. Class notes for the Performance in the Valley Series, Castro Valley, California http://www.perfdynamics.com/PitV/guerilla.html (2000)

14. Sheshkin, D.J.: Handbook of Parametric and Nonparametric Statistical Procedures. CRC Presss, Boca Raton, Florida (1997)
15. Press, W.H., Flannery, B.P., Teukolsky, S.A., Vetterling, W.T.: Numerical Recipes in C. Cambridge Univ. Press, Cambridge, U.K. (1988)
16. Wall, L., Christiansen, T., Schwartz, R.L.: Programming Perl. 2nd edition. O'Reilly & Assoc. Inc., Sebastopol, California (1996)
17. Levine, D.M., Berenson, L.M., Stephan, D.: Statistics for Managers Using Microsoft EXCEL. Prentice-Hall, New Jersey (1999)
18. Boonin, E.: Using EXCEL Visual Basic for Applications. Que Corporation, Indianapolis, Indiana (1996)
19. EXCEL Newsgroups: microsoft.public.excel.programming and comp.apps.spreadsheets
20. Cockcroft, A., Pettit, R.: Sun Performance and Tuning. 2nd edition. SunSoft Press, Mountain View, California (1998)
21. Gunther, N.J.: A Simple Capacity Model for Parallel Transaction Systems. Proceedings of the Computer Measurement Group (CMG) Conference, San Diego, California (1993) 1035-1044
22 Gunther, N.J.: Scalable Server Performance and Capacity Planning. UCLA, Winter Extension Course http://uclaextension.org/shortcourses/winter2001/ scalable_server_ winter01.htm (2001)

# Performance Testing for IP Services and Systems

Frank Huebner[1], Kathleen Meier-Hellstern[2], and Paul Reeser[2]

[1]Concert Technologies, 200 Laurel Avenue
Middletown, N.J. 07748, U.S.A.
frank.huebner@concert.com
[2]AT&T Labs, 200 Laurel Avenue
Middletown, N.J. 07748, U.S.A.
{meierhellstern, preeser}@att.com

**Abstract.** The dynamic world of IP services and systems is focused on rapid time-to-market in the face of high demand uncertainty. Performance testing is frequently either omitted or performed in a cursory manner. Successful IP services experience phenomenal growth, and are often unprepared for the performance problems that they encounter while scrambling to scale to meet rapid demand increases. This growth spurt poses particular challenges for performance analysts. In this paper we describe methods for assessing performance through empirical performance testing combined with "gray-box" modeling. The paper defines the performance testing process and shows through a case study how the behavior observed in the stress tests can provide developers with opportunity areas for performance improvement. Based on the success of this approach, we recommend that software stress testing be incorporated formally into the software development process, especially when the development consists largely of integrating external vendor components.

## 1 Introduction

IP applications often use C++, Java or third-party software to enable rapid development and functionality changes. The price for the convenience of flexibility is often performance and scalability [1,3]. Performance assessment of these applications is aimed at identifying capacity, performance, and scalability-limiting bottlenecks prior to field deployment. Traditionally, detailed performance modeling has been possible because development has taken place in-house. However, several factors have radically changed our ability to perform traditional modeling. First, accelerated delivery cycles (fast time-to-market) require the involvement of performance analysts at a much earlier stage in the product lifecycle. Second, development consists of the integration of off-the-shelf vendor systems as more and more software is acquired from the outside, rather than developed in-house. Both of these factors strain the performance analyst's ability to apply traditional methods to assess system performance.

One particularly problematic effect of this "buy rather than build" approach is that we cannot "see" into the system software to the same extent that we could with in-house development. As a result, the vendor's software product is (at best) a "gray box" to the integration developer and to the performance analyst. Rather than being

R. Dumke et al. (Eds.): Performance Engineering, LNCS 2047, pp. 283–299, 2001.

able to analyze the internal flows within the code to identify capacity-limiting bottlenecks, we can only analyze what comes out of the box as a function of what goes in. Therefore, new tools and techniques are required to effectively analyze the performance of software acquired from a vendor for integration. In this paper we describe methods for assessing performance through empirical performance testing methods [2,4] combined with "gray-box" modeling. Section 2 gives an overview of the performance testing process. Section 3 gives a detailed case study that illustrates how performance testing and gray-box modeling can be used to improve performance.

## 2 Performance Testing Process

Since the details of software internals for IP services and systems are often hidden from performance engineers, performance testing is an integral aspect of assessing their performance. In this section, we present discussions of key components of a successful performance testing strategy for Internet applications. We outline the process flow of a performance testing effort, and discuss important factors that affect the outcome and results of a performance testing project.

### 2.1 Types and Sequence of Performance Tests

Depending on the properties of the system/service under study, various types of performance tests can be carried out:

- *Atomic Transactions vs. Operational Profiles*: The transactions used to generate load to test the system can either be in the form of atomic transactions or transaction workload mixes that reflect the operational profile. Atomic transactions are individual transactions selected on the basis of their frequency of occurrence, importance or resource consumption. Tests using atomic transactions are done in order to assess the impact of a particular type of transaction on the system, as well as to isolate performance issues associated with a particular transaction type. In operational profile performance testing, the performance of the system is evaluated using transaction mixes that to some degree resemble the anticipated operational profile of the production environment. Generally, the transaction mixes will be composed of an appropriate combination of the atomic transactions. The advantage of operational profile testing is that it gives a realistic view of the system performance under field conditions. However, its drawback is that the operational profile may change rapidly (e.g., time-of-day fluctuations) or over time (trends, e.g., by introducing new services that use different system resources or change user behavior).
- *Load Testing vs. Stress Testing:* Load testing evaluates and models the performance of various system components under different controlled levels of load. The number of concurrent users may be fixed at a predefined number (with each user generating load at a prescribed rate), or may vary according to the load (i.e., a new "user" may be spawned at each arrival epoch). In stress testing, load is generated by a number of concurrent users. As soon as a user receives a response,

it immediately submits the next request (with negligible client delay). Each of the simulated users does so independently and in parallel. Tests are run with increasing numbers of users until the system is saturated. Either load testing or stress testing can be done using atomic transactions or with an operational profile.
One caution: stress testing tries to determine the "maximum load" that the system can realistically handle. Stress testing using operational profiles may be deceptive, since it assumes linear growth of load without changing the transaction mix. In most cases, however, a new "killer application" (that changes the transaction mix considerably) is responsible for driving a system to its maximum load level.

In addition to the types of tests, the *sequence* of tests is also important. Ideally, a systematic performance testing effort would integrate all possible combinations of tests in order to develop a comprehensive signature profile. In our experience, we have found it helpful to begin with stress tests using atomic transactions, since they give insight into the maximum load that the system can handle for specific transaction types. These kinds of tests also allow you to determine which transactions are the "heavy hitters" in terms of system resource consumption. Next, we perform testing with operational profiles in both load and stress test mode. These tests provide valuable insight into the impact of transaction interactions on system performance and capacity. For example, a system may be able to handle $x$ transactions per second (TPS) of type A and $y$ TPS of type B, while the maximum throughput with a mix of types A and B transactions is limited to considerably less than $(x+y)/2$ TPS. Finally, we perform a "soak" run where load tests are performed with the expected operational profile and transaction volume for a long period of time (days) in order to discover latent performance issues.

## 2.2 Performance Testing Process Flow

We summarize here the generic process flow of performance testing efforts, consisting of two main phases: test planning and test execution/analysis. Figure 1 gives a flow diagram of the performance testing process.

**Performance Test Planning Activities.** The performance test planning activities shown in Figure 1 are described below.

- *Lab Configuration Verification.* In order to achieve realistic results, the lab configuration should be identical to the field, although this is not always possible due to cost considerations. If the lab and field are not identical, then there should be a clear understanding of how field performance will be estimated from lab results. Differences between the lab and field can usually be ignored for non-bottleneck resources. Extrapolation of performance for bottleneck resources can be more difficult, since resource consumption generally does not grow linearly with increased load. Extrapolation to additional processors (assuming multiprocessor scalability has been demonstrated) usually works well, but extrapolation to different I/O subsystems has proven difficult.

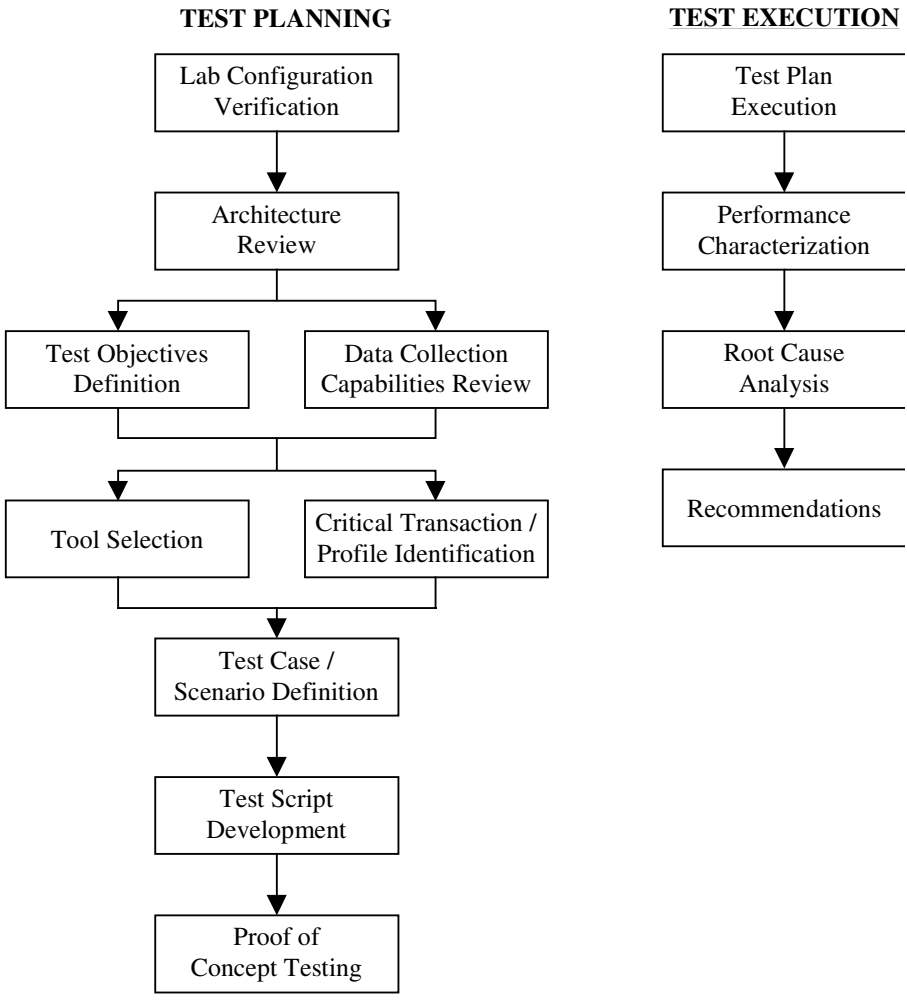



**Fig. 1.** Performance testing process flow diagram

- *Architecture Review.* Whatever information is available about the system internals and expected system operation should be understood before testing begins. Some bottlenecks can be identified from qualitative descriptions alone (sometimes referred to as a *qualitative* performance assessment). Often, knowledge gained in this phase leads to a list of suspected bottlenecks. An understanding of these candidate chokepoints can be used to design more effective test cases that identify if the suspected bottlenecks are present or not.
- *Test Objectives Definition.* "Success criteria" for performance testing should be defined before testing begins. If requirements are not available, then testing should focus on scalability to identify system breakpoints. Also, in the absence of performance requirements, if several (subsequent) software releases are to be

tested, then baselining the earliest release available and comparing against subsequent releases can be used to assess performance/capacity trends.

- *Data Collection Capabilities Review.* Identify required measurements that capture all aspects of system performance and yield information on potential performance problems. Bottleneck diagnosis must be based on an understanding of what is happening inside the system. Therefore, key diagnostic data must be obtained from log files and system accounting utilities. Some system accounting utilities used in Unix-based system performance testing are *sar*, *ps*, *iostat*, *mpstat* and *vmstat*. For NT systems, we use utilities such as *perfmon*.
- *Tool Selection.* Many sophisticated vendor tools exist to automate testing tasks, including generating user load and collecting performance statistics. With script-based benchmarking tools, one executes a typical sequence of user transactions, and captures the request/reply strings and keystrokes into a script. The client driver (load generator) then replays a specified number of simultaneous scripts in parallel to emulate concurrent users. Examples of such tools include LoadRunner® by Mercury Interactive and Silk Performer® by Segue. Both tools can be used for either load or stress testing. When cost is an issue, there are free tools that can be used, or custom scripts can be developed. Typically, these have a less sophisticated user interface, are more cumbersome to use when the load test parameters need to be continually changed, and may be limited to stress testing. However, they are an excellent way to get started when time or money is an issue. Examples of free tools are http-load (http://www.acme.com/software/http_load/) or Microsoft's WCAT (http://msdn.microsoft.com/workshop/server/toolbox/wcat.asp). A very good and up-to-date listing of tools (free and commercial) can be found at http://www.softwareqatest.com/qatweb1.html.
- *Critical Transaction/Profile Identification.* The goal in this phase is to identify "heavy hitter" transactions (either from a volume, resource consumption, or importance perspective). The heavy hitter transactions form the set of atomic transactions. Operational profiles are critical in establishing credible transaction mixes, and may also include different load and/or operational scenarios (e.g., a service "ramp-up" mode that includes many user registrations, a "steady state" mode that reflects a mature service, or various failure scenarios). When bottlenecks are identified and must be fixed (usually at considerable cost to the organization), there must be convincing evidence that the loads used in the testing phase were "realistic".
- *Test Case/Scenario Definition.* When designing test cases, it is important to "start simple". Performance testing efforts can easily bog down in complicated tests that shed little insight into the causes of anomalous system behavior. As noted in Section 2.1, we usually start with atomic transaction stress tests, followed by load and stress tests using one or more operational profiles, ending with long load tests at the expected system load and operational profile to uncover any latent problems. Tests should be conducted under different loads and operational profiles.
- *Test Script Development.* Instrumentation can vary from writing simple shell scripts, to comprehensive recording of operational sequences using load test tools (e.g., LoadRunner®). The proper setting of test objectives (see above) helps to limit the potential waste of time and resources spent on incorrect/unnecessary scripting.

- *Proof of Concept Testing.* This critical step should not be overlooked. It is used to identify if the lab is configured correctly, if the load test scripts are coded correctly, and if the system instrumentation is functioning as expected. This step also identifies flaws in the basic testing approach. We have learned through painful experience not to neglect this step, when days of load test results have been rendered useless due to a simple change that could have been made at the outset of testing.

**Performance Test Execution and Analysis Activities**

- *Test Plan Execution.* Execute all load test scenarios and capture resource consumptions, task flows, response times, traffic loads, and failures during all scenarios. Testing should ideally be performed iteratively, executing and analyzing the results of a few tests, then modifying test scenarios as appropriate, before proceeding. This strategy requires the capability to analyze test results quickly, but has proven very helpful in eliminating the execution of unnecessary test cases (from which no relevant additional information would have been obtained).
- *Performance Characterization.* Produce load-service performance characterizations and models. The test data should be used to produce load-service curves. In some cases, gray-box modeling is useful for identifying possible bottlenecks, and can be helpful in the diagnosis process. (A detailed case study in gray-box modeling is presented in Section 3.)
- *Root Cause Analysis.* Perform a root cause analysis for all observed anomalies and failures. Root cause analysis is often time-consuming and requires consultation with developers or vendors to identify the causes of observed behaviors. Re-testing with different test parameters may also be required to pinpoint suspected problems.
- *Recommendations.* Produce performance requirements compliance statements, engineering rules, guidelines for system monitoring and future problem detection, and recommendations for bottleneck removal and performance improvement.

## 2.3 Ensuring a Successful Performance Testing Effort

The guidelines in the previous section discuss various technical aspects of performance testing. However, there are numerous non-technical challenges in making a performance validation effort succeed. Often performance does not have the same priority as feature delivery (unless there has been a crisis), and performance is often the first thing to be cut when schedules inevitably slip. Assuming that the service or product is actually deployed and used, then ignoring performance inevitably leads to disaster, even though this harsh reality is not in the minds of engineers urgently trying to deliver a product. The following is a selected set of effective strategies that we have used in a variety of projects to achieve successful performance results.

1. Obtain organizational alignment and resource commitment to the testing process: Organizational and resource issues are likely to delay or stall a performance testing effort. A systematic effort to verify objectives, identify expectations, obtain resource commitments, and determine processes that will be followed throughout

the testing process is a critical prerequisite for a successful performance testing project.
2. Schedule time for performance testing: Performance testing must be planned into the project schedule as part of the delivery process. Failure to do so may result in deploying a new release with unidentified failures and performance bottlenecks.
3. Form a dedicated performance test team that works across applications: Due to release cycle constraints, it is usually infeasible to expect system testers to conduct performance testing. Performance testing is a specialized expertise within the testing discipline, and as such is best performed by people with experience in this area.
4. Allow for iterative performance testing that continues beyond the end of the system test cycle: Each performance testing cycle identifies performance issues and limitations that must be verified or further investigated through additional testing of the current or future releases.
5. Proactively manage third-party software vendors: Since the system dynamics and performance issues of third-party software are often unknown to the performance test team, it is important to share results of "gray/black-box" analysis techniques with the vendor to obtain more information about the internal software bottlenecks.

## 3 Using Stress Test Results to Drive Performance Modeling: A Case Study in "Gray-Box" Vendor Analysis

In this section we present a case study that illustrates how the testing methods defined in the previous section can be combined with gray-box modeling to diagnose and improve performance. The application is a Web-based e-commerce service. The development effort is centered around the integration of an external vendor's Java-based product that performs dynamic Web page construction and retrieval in a distributed object-oriented environment (see Figure 2). Specifically, the distributed execution environment consists of a front-end (FE) sub-system running a standard Web server and a Java virtual machine (JVM) servlet engine (SE), plus a back-end (BE) sub-system containing business logic and data (e.g., SMTP mail gateways, POP3/IMAP4 mail servers, LDAP/X.500 directory servers, etc.).

The vendor product consists of a number of Web page templates together with a number of Java servlets. Collectively, these servlets:

- parse the requested template for scripting language (dynamic) content,
- issue the appropriate BE requests to process the necessary business logic and/or retrieve the required data,
- process the returned data into the necessary (XML) data structures,
- perform protocol conversion (as necessary), and
- construct the resulting Web page for return to the user.

The protocol/language between the end-user and Web server is HTTP/HTML, and that between the Java servlets and BE is determined by the particular BE application (e.g., IMAP, LDAP, etc.). Within the JVM environment, data is passed between the various servlets as XML data structures. Hence, the scripts must perform a variety of

protocol and language conversions during the course of request execution. The Web page templates are written by the integration development team based on an API specification provided by the SE vendor, and the SE supports the standard BE protocol command sets, but the SE execution environment and scripts are vendor-proprietary, and the internals are not exposed to the team (i.e., we get binary code only). As a result, the vendor's SE application is a "gray box" from the standpoint of capacity planning and performance analysis. Therefore, traditional approaches to characterizing performance (e.g., analysis of internal flows, queueing, contention, etc.) are not available to us, and we must rely on other techniques to identify performance bottlenecks and determine system capacity. One such technique (discussed in Section 2) is performance testing.

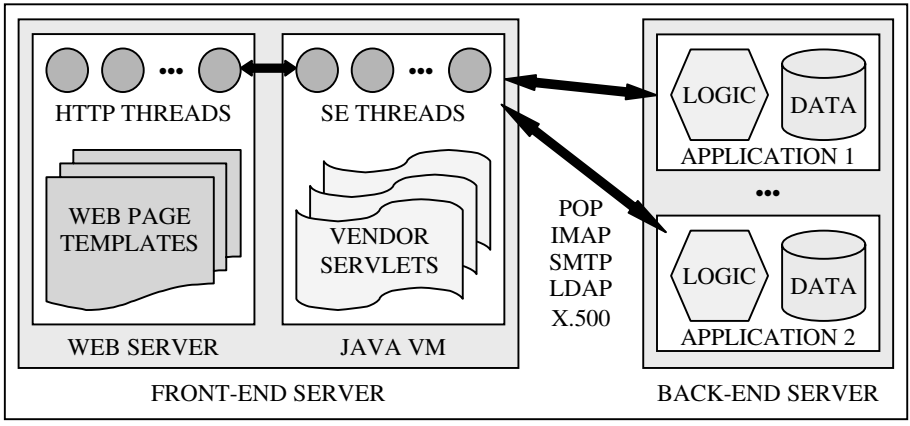



**Fig. 2.** System architecture

In this section, we present a methodology for "reverse-engineering" stress test results to build performance models of the system internals, and use those models to "see" inside the box. This approach provided significant insights into the performance of the vendor's proprietary code, and afforded us better leverage in the vendor management process. In particular, the approach identified significant software bottlenecks in the vendor's Java environment and servlet code that prevented the system from fully utilizing the hardware (CPU) resources. The approach also exposed the system's behavior under overload, and revealed the need for overload controls. With this type of quantitative evidence in hand, we were better positioned to negotiate an effective contract.

### 3.1 Stress Testing Environment

We conducted a series of load tests using Silk Performer to generate repeated requests at various concurrency levels (simulated users). The test configuration (see Figure 3) consisted of a Windows NT server running the load generation scripts (driver), a Solaris server running the FE software (Netscape WS, Jrun JVM, and vendor SE), and a Solaris server running the BE application (for this test, a POP3 message store

server). The FE and BE servers are collectively referred to as the system-under-test (SUT).

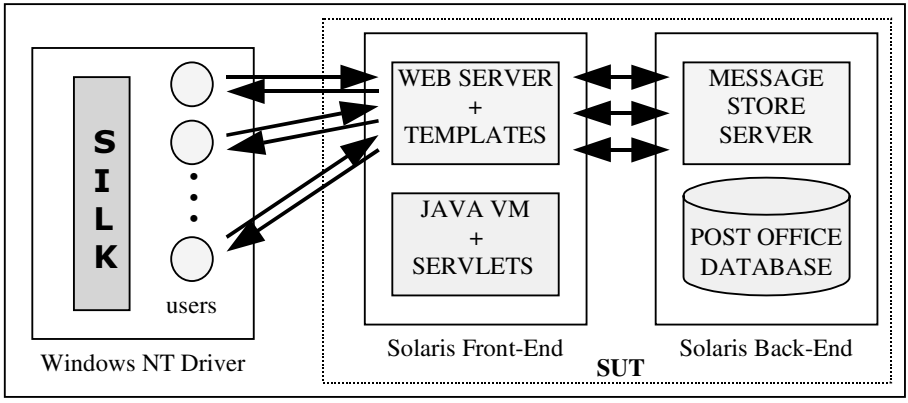



**Fig. 3.** Stress test configuration

The Silk scripts emulated a prescribed number N of concurrent users repeatedly generating the same request (e.g., fetch folder list, read message, send message, etc.). The number of concurrent simulated users was varied from N=1 to 20. The number of repeated requests per user at each concurrency level (typically 2000) was sufficient to achieve statistical stability. The tests were run in "stress" mode; that is, as soon as a user receives a response, it immediately submits the next request (i.e., with negligible delay). Each of the N simulated users does so independently and in parallel. As a result, the concurrency level (i.e., the number of requests in the system) equals N at all times (other than ramp-up and ramp-down).

The hardware (CPU, memory, disk, I/O) and software (process/thread) resource consumptions were measured on all three machines. Although the FE resource consumption was most relevant to this study, the driver and BE utilizations were measured to ensure that they did not introduce any capacity-limiting bottlenecks during the tests. In addition, response time was measured from the user perspective (FE+BE) as well as from the FE perspective (BE only). Again, although the FE performance was most relevant, the BE delay was measured to ensure that it did not limit system throughput as the concurrency level increased. The requested data was cached in the BE to further ensure consistent, non-bottlenecking BE performance.

### 3.2 Stress Testing Results

The stress test results for a particular request type are shown in Figure 4. In particular, Figure 4 plots the end-to-end (FE+BE) response time on the left-hand axis, and the FE CPU utilization on the right-hand axis, as a function of the concurrency level N. Throughout the tests, both the BE and driver CPU utilizations were low. BE delay was consistent at around 900ms. Driver delay between receipt of a response and submission of a request was negligible. All FE resource consumption levels were well within normal ranges, and did not signal any obvious bottlenecks.

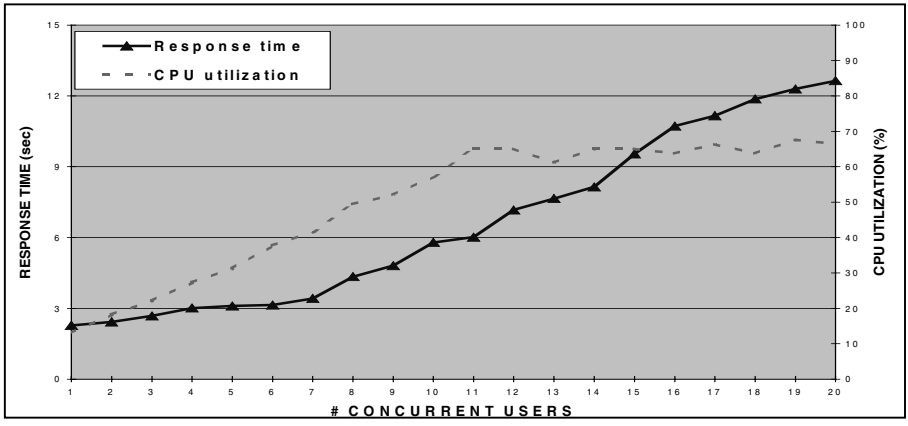



**Fig. 4.** Stress test results

These test data and curves are the sole source of information available to us to assess the performance of the vendor's software. The first question to be answered is, "What performance implications can we reverse-engineer from these stress test results?" At first glance, the response time behavior seems as expected: as more and more "load" (in the form of concurrent users) is offered to the system, the delay increases. Furthermore, the delay is not yet exhibiting the classic "hockey stick" behavior that is observed when an open system saturates, so the system seems to be behaving well. Yet, the CPU utilization curve is disturbing. CPU consumption levels off at ~65-70%, well below expectation if the CPU were the bottleneck. As we will see in the next section, care must be exercised in interpreting these stress test plots!

### 3.3 Stress Testing Theory

To better understand the stress testing results, we turn to elementary queueing theory.

Let $N$ denote the number of concurrent users simulated in the tests,
$D$ denote the end-to-end (FE+BE) delay between successive requests,
$\lambda$ denote the system arrival rate (offered load, in requests/second),
$T_B$ denote the holding time (in sec) of the (yet TBD) system bottleneck,
$\rho_B$ denote the (TBD) system bottleneck resource utilization, and
$\lambda_{\max}$ denote the maximum sustainable throughput (in requests/sec).

$N$ and $D$ are the x- and y-axis variables in the stress test plot in Figure 4, respectively. The offered load $\lambda$ can be determined as follows: since these are *stress* tests, each simulated user submits requests as quickly as possible. Since the driver delay is negligible, the time between successive requests by each user is $D$; that is, each user submits 1 request every $D$ seconds, at a rate of $D^{-1}$ requests/sec.

Therefore, $N$ users generate offered load at the rate $\lambda = ND^{-1}$. Furthermore, the stress testing environment is a *closed* system; that is, there are a fixed number of jobs in the system, corresponding to the number of concurrent users.

Next, the (to-be-determined) bottleneck resource utilization is given by

$$\rho_B = \lambda T_B = (NT_B)D^{-1}. \quad \textbf{(1)}$$

Thus, as the bottleneck resource becomes saturated,

$$\rho_B \to 1, \text{ so } (NT_B)D^{-1} \to 1, \text{ so } D \to NT_B. \quad \textbf{(2)}$$

Therefore, the relationship between $N$ (stress test x-axis) and $D$ (stress test y-axis) approaches a *linear asymptote* (with slope $T_B$) when the capacity-limiting system bottleneck becomes saturated (not a vertical "hockey stick", as expected in the case of open systems). Finally, at saturation,

$$\rho_B = \lambda T_B \to 1, \text{ so } \lambda_{\max} \to T_B^{-1}. \quad \textbf{(3)}$$

That is, the maximum throughput is given by 1 / {bottleneck resource consumption}.

Observation 1: Stress testing open systems creates a closed system environment.

Observation 2: The x-axis (concurrency level) in stress test plots is very deceptive. These are *not* traditional "load v. service" plots; rather, these are "concurrency v. service" plots. And contrary to expectation for open systems, increasing the concurrency level does *not* necessarily translate into increasing the offered load level.

### 3.4 Stress Test Results, Revisited

With this new awareness in hand, we now revisit the stress test plots (see Figure 5). As can be seen, we have *already* hit the asymptote at a concurrency level of ~7 simulated users. In other words, the (TBD) system bottleneck has already become saturated ( $\rho_B \approx 1$ ), the request arrival rate $\lambda$ has stopped increasing and has leveled off at a value $\lambda_{\max} \approx T_B^{-1}$, and the "concurrency v. delay" curve is riding along its asymptote. In fact, it appears that the curve actually starts to *diverge* from the asymptote at higher levels of concurrency.

We can translate these *stress* test results into equivalent *load* test results. Figure 6 plots the end-to-end (FE+BE) delay $D$ as a function of the load $\lambda = ND^{-1}$. As can be seen, the maximum system throughput $\lambda_{\max}$ momentarily peaks at ~2 requests/sec, then settles to a sustainable rate of ~$1^2/_3$ requests/sec. Thus, the (TBD) bottleneck resource holding time $T_B \approx (1^2/_3)^{-1}$ ~600ms. Furthermore, the divergence behavior is explained: as $N$ increases beyond ~7 simulated users, there is actually a

drop in capacity of ~25% (a "concurrency penalty"), likely due to context switching, object/thread management, garbage collection, etc.

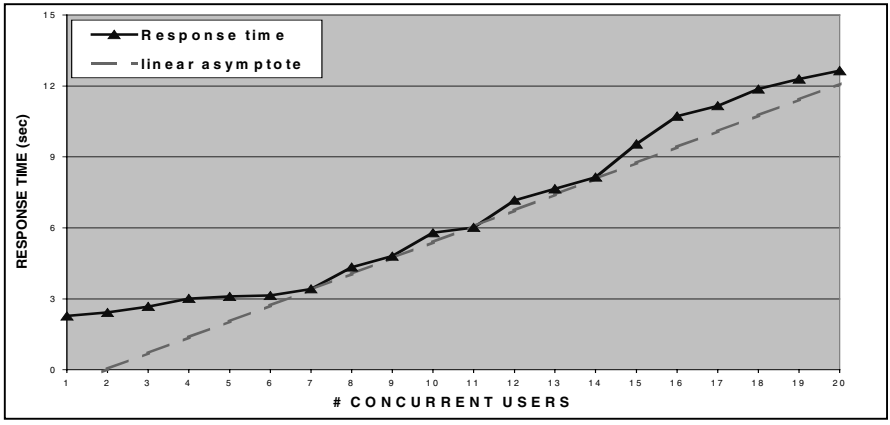



**Fig. 5.** Linear asymptote

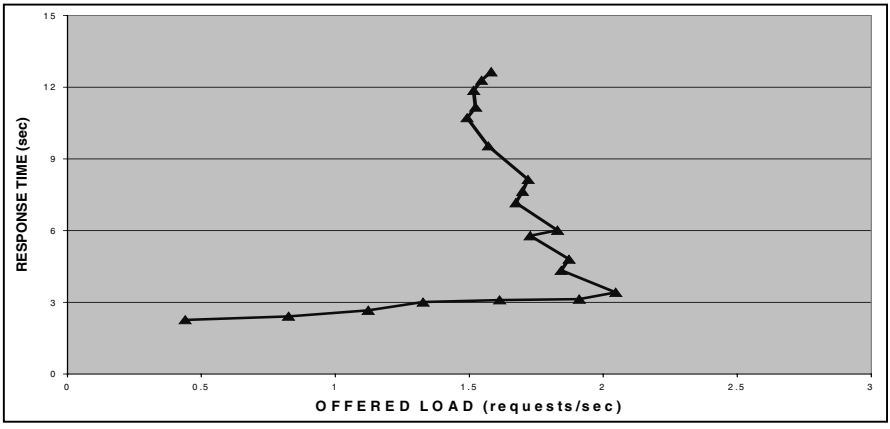



**Fig. 6.** Equivalent load test results

Finally we can shed some light on our fundamental question, "What performance implications can we reverse-engineer from these stress test results?" In particular, we now know the maximum system throughput, the holding time of the bottleneck resource, and the degree of performance degradation under overload. Furthermore, we know that the system performance is not easily explained by a simple "hard" resource bottleneck (e.g., CPU, memory, disk, I/O).

Some unknown "soft" resource bottleneck exists in the vendor's software that causes the delay to rise abnormally sharply, and prevents full utilization of the CPU resources. Candidate "soft" resources include OS/application threads, file descriptors, TCP transaction control blocks, I/O buffers, virtual memory, object/code locks, kernel-level resource contention, etc. So, the next question to be answered is, "What software bottleneck is limiting our observed performance?"

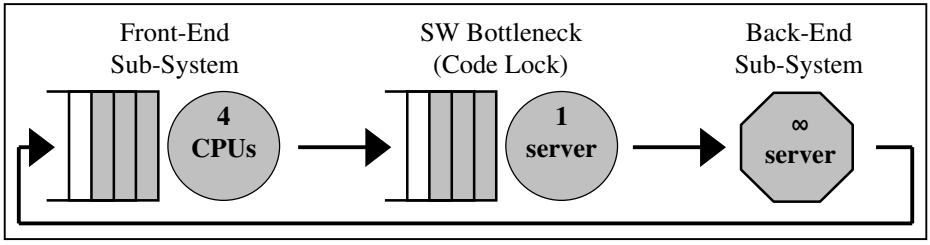



**Fig. 7.** Queueing model

We theorized that requests are being serialized through a lock on one or more significant *synchronized* code regions. It could be one sizable synchronized region in the application code, or numerous smaller synchronized code segments (e.g., some Java-related kernel system call). To test this theory, we developed a simple queueing model, as outlined in the next section.

### 3.5 Stress Testing Model

A simple analytic queueing model for our system is shown in Figure 7. The model consists of three serial nodes: the FE sub-system, the software (SW) bottleneck, and the BE sub-system. (The driver is not explicitly modeled, since the Silk client delay is negligible.) The FE sub-system is modeled as a 4-server queue to represent the 4 CPUs in our actual test machine; the service time at this node is the total CPU execution time in the FE sub-system (TBD). The SW bottleneck is modeled as a single-server queue to represent the theorized code lock; the service time at this node is the CPU execution time within the synchronized code regions (TBD). The BE sub-system is modeled as an infinite-server queue to represent the fact that BE delay is insensitive to load; the service time at this node is the measured BE response time (~900 ms).

First, consider the FE node:

Let $T_{FE}$ denote the FE sub-system CPU execution time per request (in sec),
$C = 4$ denote the number of CPUs in the FE sub-system,
$\alpha$ denote the CPU "concurrency penalty" (explained in detail below),
$\rho_{FE}$ denote the FE sub-system CPU utilization, and
$D_{FE}$ denote the FE node delay (queueing + service time, in sec).

The concurrency penalty is modeled as an "inflation factor" applied to execution time. That is, the actual execution time per request with $N$ concurrent users is given by

$$T_{FE}\,(1+\alpha\,N)\,. \tag{4}$$

Applying basic Markovian queueing results, the FE node utilization is given by

$$\rho_{FE} = \lambda\,T_{FE}\,(1+\alpha\,N)\,/\,C\,, \tag{5}$$

and the FE node delay is given by

$$D_{FE} = T_{FE}\,(1+\alpha\,N)\,/(1-\rho_{FE})\,. \tag{6}$$

Next, consider the SW node:

Let $T_{SW}$ denote the execution time of the synchronized code segment (in sec),

$\rho_{SW}$ denote the SW bottleneck code lock utilization, and

$D_{SW}$ denote the SW bottleneck queueing delay (in sec).

Applying basic Markovian queueing results, the SW bottleneck utilization is given by

$$\rho_{SW} = \lambda\,T_{SW}\,/\,(1-\rho_{FE})\,, \tag{7}$$

and the SW node delay is given by

$$D_{SW} = T_{SW}\;\rho_{SW}\,/\,(1-\rho_{SW})\,. \tag{8}$$

Note that the code lock utilization is a function of the FE CPU utilization, since the locked code actually executes on the FE sub-system. Note also that the SW bottleneck delay only includes the queueing delay, since the service time is already accounted for in the FE sub-system delay.

Finally, let $D_{BE}$ denote the BE sub-system response time (in seconds). Then the end-to-end (FE+SW+BE) delay is given by

$$D = D_{FE} + D_{SW} + D_{BE}\,. \tag{9}$$

This relationship reduces to a cubic equation in $D$ that can be easily solved iteratively to find the unique fixed-point solution.

### 3.6 Model Parameterization

Now that we have a hypothetical model for the system, we can use our stress test results to parameterize the model. In particular, we need to estimate the measured values of $\alpha$, $T_{FE}$, $T_{SW}$ and $D_{BE}$ to see if the model accurately predicts the measured end-to-end delay $D(N)$.

As mentioned earlier, the BE delay $D_{BE}$ was consistently measured at ~900ms. Next, recall from Section 3.4 that the bottleneck resource *holding* time was ~600ms and the CPU utilization was ~67%. In our model, the SW bottleneck *holding* time is given by

$$T_{SW}\,/(1-\rho_{FE})\,. \tag{10}$$

Therefore, $T_{SW}$ ~ 600(1–$^2/_3$) ~200ms. Also, recall from Figure 6 that we observed a concurrency penalty of ~25% as the concurrency level went from 7 to 20 users. Thus, the concurrency penalty $\alpha$ ~2%. Finally, the FE CPU execution time is given by

$$T_{FE} \approx C\hat{\rho} / \hat{\lambda}, \tag{11}$$

where $\hat{\rho}$ = net CPU utilization (not including the no-load CPU) and $\hat{\lambda}$ = maximum observed throughput. Fitting the linear part of the CPU consumption curve in Figure 4 suggests a no-load (y-intercept) value of ~8%. Thus, $T_{FE}$ ~ 4(0.67-0.08)/2 ~ 1200ms.

The resulting fits to the end-to-end (FE+BE) response time curve and FE CPU utilization curve with $\alpha$ =0.02, $T_{FE}$ =1200ms, $T_{SW}$ =200ms, and $D_{BE}$ =900ms are shown in Figures 8 and 9, respectively. As can be seen, this simple model provides a reasonable fit to the data. By applying elementary queueing theory to develop a simple "back-of-the-envelop" model, we have augmented our understanding of the performance of our gray box. In particular, in addition to the maximum system throughput, bottleneck resource holding time, and degradation under overload, we now also know the total CPU consumption of the vendor's code (~1200ms) and the CPU consumption of the serialized code segments (~200ms), and we believe that we have identified the software bottleneck that is limiting our performance.

### 3.7 Vendor Follow-Up

Our approach provided significant insights into the performance of the vendor's proprietary code, and afforded us better leverage in the vendor management process. In particular, the approach characterized the current system performance limitations, identified a significant software bottleneck in the vendor's servlet code that prevented the system from fully utilizing the CPU resources, and exposed the system's behavior under overload. With this type of quantitative evidence in hand, we were better positioned to negotiate effective contracts.

Vendors sometimes will attempt to deflect responsibility for poor performance back to the customer team, and an adversarial relationship can develop. In this case, however, we could present the vendor with undeniable evidence that a software bottleneck existed in the vendor's code. As a result, the vendor and customer could move directly to a constructive, cooperative effort to identify the root cause of the bottleneck.

The joint development-integration team ran a number of profiling tools to uncover the source of the code serialization. Two culprits became evident: first (at the application level), writes to the log file were synchronized, and second (at the kernel level), object creation method calls are single-threaded. In order to address the first source, the logging function was rewritten to allow concurrent writes to the transaction log. To reduce the impact of the (unavoidable) second source, many objects were moved from transaction (request) scope to application scope to avoid much of the object creation, de-referencing, and garbage collection activity.

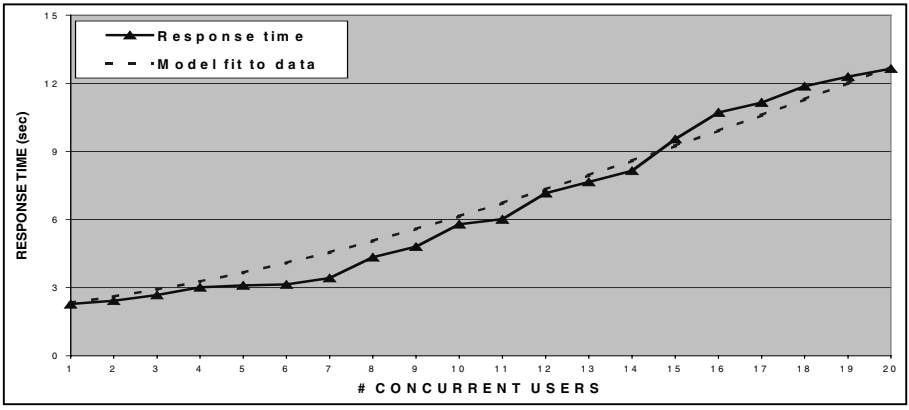



**Fig. 8.** Test results vs. model, I

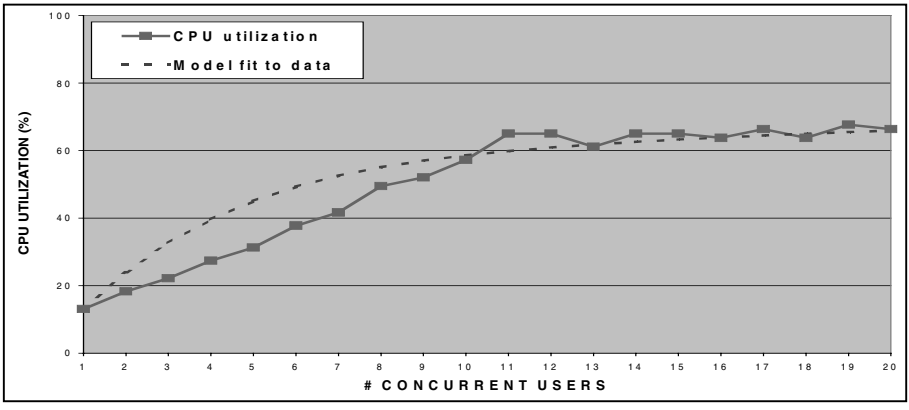



**Fig. 9.** Test results vs. model, II

Collectively, these changes resulted in a dramatic improvement in the vendor's software performance with the next release. In particular, the software realized a 4-fold increase in throughput, as well as improved behavior under overload (i.e., virtually no concurrency penalty).

## 4 Conclusions

The case study presented in the previous section illustrates a number of valuable lessons for Internet performance assessment. First, stress testing can provide significant "hidden fruit" for performance assessment, provided that the test results are interpreted properly. Second, gray-box modeling is possible without knowing the internal details of the software architecture, and can provide significant additional performance insights into the code. And finally, sophisticated analytic models are not

always required to "see" inside the box — this model was fairly simple. (However, the appropriate use of simple models is vital.)

At a broader level, this paper defines the performance testing process and shows by example how the behavior observed in load and stress tests can provide developers with opportunity areas for performance improvement. Based on the success of this approach, we recommend that software stress testing be incorporated formally into the software development process, especially when the development consists largely of integrating external vendor components. Such an approach has a number of significant benefits, including up-front capacity/scalability planning, early identification of performance limitations/bottlenecks, early engagement of the Quality Assurance/test team, and quantitatively driven vendor management. Furthermore, in the case of vendor-provided software, our approach offers perhaps the only way to "see" inside the binary code, and in the case of new programming languages such as Java, this approach offers an opportunity to expose anomalies that could not have been foreseen or predicted from past experience.

## References

1. Bulka, D.: Java Performance and Scalability. Vol. 1, Addison-Wesley, New York (2000)
2. Cockroft, A., Petit, R.: Sun Performance and Tuning – Java and the Internet. Sun Microsystems Press, California, 2nd Edition (1998)
3. Neffenger, J.: The Volano Report: Which Java platform is fastest, most scalable? Java World, http://www.javaworld.com/javaworld/jw-03-1999/jw-03-volanomark.html (1999)
4. Taylor, S.: A Developer's Crystal Ball. ADT– Software Engineering (1997)

# Performance Modelling of Interaction Protocols in Soft Real-Time Design Architectures

Carlos Juiz[1], Ramon Puigjaner[1], Ken Jackson[2]

[1]Universitat de les Illes Balears, Dpt. de Ciències Matemàtiques i Informàtica,
Escola Politècnica Superior
Ctra. de Valldemossa, km. 7.5, 07071 Palma (Spain)
Phone: +34-971-172975, 173288 Fax: +34-971-173003
{dmicjg4,dmirpt0}@uib.es
[2]Quality Systems and Software,
Oxford Science Park
Oxford, OX 4GA (UK)
Phone: +44-1865-784286
JacksonKen@CompuServe.com

**Abstract.** Architectures for designing soft real-time systems do not typically provide any performance tool that will enable the designer to analyse the performance of the system that is being designed. Elementary intercommunication data areas known as interaction protocols are the building blocks of every soft real-time system. The interaction protocols are grouped into families depending on their writing/reading functionality. In this chapter, we extent the interaction protocol families, that are commonly used in every soft real-time architecture, in order to describe a complete performance taxonomy. The components of this taxonomy transfer data among processes. Their performance models can be used in a software package to complement the automatic design and generation of soft real-time systems. Despite the performance of the whole system can be solved through simulation in this chapter we propose new semaphore queues as analytical approximations for most of these soft real-time components.

## 1 Introduction

Real-time systems differ from traditional software systems in that they have a dual notion of logical correctness. Logical correctness of a real-time system is based on both the correctness of the output and timeliness. That is, in addition to producing the correct output, real-time systems must produce it in a timely manner. A *hard real-time* system is a system that satisfies explicit or bounded *response-time constraints* whereas a *soft real-time system* is not so critical to produce an output within a *deadline*. In consequence, significant tolerance can be permitted but the software system tends to be large and complex [18].

A *Real-time network architecture* models a system in terms of a set of connected components. The connections contain data, which is shared among the connected processes included in the components, but which is not owned by them [22]. One of the most usual approaches is based on the concept of a system as a set of interacting

R. Dumke et al. (Eds.): Performance Engineering, LNCS 2047, pp. 300–316, 2001.

components. These components are basically processes, tasks and paths that make subsystems. Thus, the processing system is composed by connected subsystems [10]. In order to characterise the interaction between the components is necessary to express a protocol representing the timing constraints on the interaction and to its support by an explicit interconnection in the implemented system.

On the other hand, a *software design* defines how a software system is structured into components and also defines the interfaces among them. The nature of each component depends on the concepts and strategies employed by a method. A software *design method* is a systematic approach for creating a system design. During a given design step, the method may provide a set of structuring criteria to help the designer in decomposing the system into its components [7].

The *Data Interaction Architecture* (*DIA*) is an example of a layered architecture [21]. A set of well-known notations exists for each of these layers and they are basically modelled with a software design method called *MASCOT* [20]. *MASCOT* is a design and implementation method for real-time software development and brings together a co-ordinated set of techniques for dealing with the design, construction (*system building*), operation (*run-time execution*) and testing software [11].

At the heart of *MASCOT* there is a particular form of software structure supported by complementary diagrammatic and textual notations [1]. These notations give visibility to the design as it emerges during development, and implementation takes place via special construction tools that operate directly on the design data. Design visibility greatly eases the task of development management, and the constructional approach ensures conformity of implementation to design, and allows traceability of *real-time* performance back to design.

*DIA* uses protocol symbols to show the nature of each dynamic interaction between system elements on each layer. The functional model allows a system to be described in terms of independently functional units interacting through defined transfers of information. Here is introduced the notion of protocol component which is used to annotate the interaction between processes with a *MASCOT* representation.

In this chapter, we will concentrate in proposing a new *taxonomy of interaction protocols* and also in showing the validity of some exact and approximate analytical models of some of these *soft real-time* components. Thus, we will propose queueing-based models [3] for analysing these components, known as *interaction protocols* in the *DIA* design terminology, without losing the descriptive advantages of other performance techniques, e.g. stochastic Petri nets [2].

In the following section, we review the definition and basic classification of interaction protocols in *DIA* into four different *protocol families* [23]. Section 3 proposes new protocol extensions that are not included in the *protocol families* but with frequent appearance in the design of *soft real-time system*. In section 4, new protocol members are proposed with these extensions and are also composed in order to create new ones. Once the proposed *protocol taxonomy* is presented, we also state the difficulties to find exact analytical solutions for the different *soft real-time* components. Therefore, in section 5 we remind the *semaphore* queue paradigm as a valid approximation to some of these components and finally the conclusions are given in Section 6.

## 2 Basic Interaction Protocols

The interaction protocols derive from the explicit representation of intermediate data and from the realisation that such data can only be destructively or non-destructively written or read. These temporal effects are the dynamics of the interaction form; in fact, they show the relation between temporal effects and shared data concepts, and that is the interest for their performance modelling and analysis in a soft real-time system. The basic interaction protocols are four: *pool*, signal, *channel* and *constant*. A *pool* allows reference data to be passed from one process to another. A *signal* allows event data to be passed from one process to another. A *channel* allows message data to be passed from one process to another. And finally, a *constant* can be regarded as configuration data. It essentially provides a write capability.

### 2.1 Channel

The *channel* consists on a simple *producer-consumer*. The behaviour of the most basic *channel* is depicted as follows. The *producer* process sends a data item to a buffer every certain production time. Once the data item arrives to the buffer, the *producer* process takes also a certain time to put the data into the store. So, the storage time corresponds to the service time of putting the data into the first available position in the buffer. The buffer has a finite capacity and then it is necessary to implement a *semaphore* for controlling the *channel* full condition. The order in which the data are stored and consumed is FIFO. On the other hand, the inter-arrival time of consumptions is zero. That occurs because the *consumer* process is always waiting for data to be retrieved. As soon as data are stored into the buffer the *consumer* get them. This retrieval time corresponds to the service time of getting the data out of the first occupied position in the buffer. Then, it is necessary at least to implement a *semaphore* controlling the *channel* empty condition. Both processes run independently each other but orderly, a data item is always first stored and then retrieved. Therefore, writing is non-destructive but reading is a destructive operation.

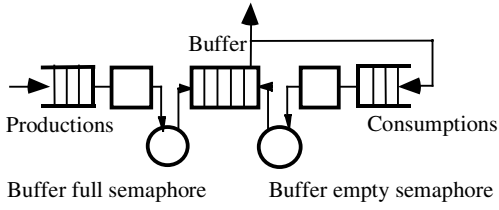



**Fig. 1.** The queueing network of a basic *channel* where the consumer process is permanent

### 2.2 Pool

The *pool* is a data storage that can be written to and/or read from by any task, and at any time. It is an effective means of sharing data among a number of tasks, not merely a select pair. Senders write information into the store and receivers read such stored

information as a unit. Thus, writer and reader tasks need have no knowledge of each other. Reading has no effect on the stored data, however writing is a destructive operation. Moreover, *pool* usage is a random affair; it is impossible to predict in advance when tasks will access the data [5].

There are several variants of the *readers-writers* problem, but the basic structure is the same. Tasks are of two types: reader tasks and writer tasks. All tasks share a common variable or data object. Reader tasks never modify the object, while writer tasks do modify it. Thus writer tasks must *mutually exclude* all other reader and writer tasks, but multiple reader tasks can access the *shared data simultaneously*.

In consequence, there are two different classes of servers for two different customer classes. In the basic *pool* model either only a writer task can store information into the *pool* or up to *C* reader tasks can be reading it. Due to the limited simultaneous service for the readers at *pool*, their service rate is proportional to their number. Thus, there is only one server for writer tasks but there are up to *C* servers for readers tasks available in the *pool*.

In figure 2, we remind the classical untimed *Petri* net is shown representing the *readers-writers* problem [19]. Therefore, writing is destructive but reading is a non-destructive operation.

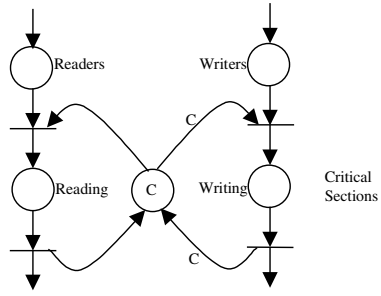



**Fig. 2.** The *readers-writers* synchronisation problem modelled with an untimed *Petri* Net

## 2.3 Signal

The basic *signal* allows event data to be passed from one process to another. This data can be overwritten at any time by the writer, but can only be actioned once by the reader, i.e. the action or inaction of the reader has no direct effect on the timing of the writer. Therefore, reading is also a destructive operation.

## 2.4 Constant

The basic *constant* can be regarded as configuration data. It essentially provides a *write once* capability. Generally the value of a constant is established at build time, and it would not expect to find any *soft real-time system*, which show a process writing to a *constant*. The definition of a *constant* is only to complete the basic

interaction protocols; hence the writing and the reading are non-destructive operations. Thus, there is no interest for performance study purposes.

## 3 Protocol Extensions

In *DIA* there are only two ways in which the four basic protocols are extended: the number of intermediate items (*capacity*) and the void values (*null*). The *capacity* constraint means varying the buffering. Whatever the degree of buffering, items are always read in the order they are written, so items are *queued*. This is an important feature from the performance analyst viewpoint. However, if the value of the item to be passed can be explicitly identified as being void (*stimulus* function) or not, has no interest for the performance analysis. That is, although the void value protocols have particular functional significance, they need not be given any special treatment, their timing and synchronisation properties can be taken to be identical to their non-void counterparts. Thus, we will only mention the void value protocols when the non-void were going to be explained.

It is convenient at this point to introduce the notion of a *protocol family*. In *DIA* every basic protocol defines a *protocol family*. Thus the *channel family* is regarded as all the protocols derived from the *channel* (including the void extensions): *rendezvous*, *bounded buffer*, *directional handshake*, *dataless channel* and *bounded stimulus buffer*. Likewise the *signal family* comprises all the protocols derived from the *signal*: *flash data*, *overwriting buffer*, *prod*, *stimulus* and *overwriting stimulus buffer*. Neither of the two extensions has any meaning when applied to the *pool* or the *constant*. Both these protocols have the non-destructive reading property so items cannot be removed to expose the items behind. Also the concept of an unconsumable void has no useful purpose. In spite of this fact, we are going to consider these *basic protocols* as one-member families. The reason of this consideration is the new extensions not included in *DIA*. Let's define the new notion of *protocol variant*. Even the importance of the *capacity* extension, not only the buffering (or void values) can be considered as the unique feature to build *soft real-time* components. We define new interaction protocols as *variants* of the members of a *protocol family* taking into account, e.g., the item *classification*, the scheduling (*priority*), the *composition* or the *bi-directionality*. In case the *composition* and *bi-directional* extensions, *simple* interaction protocols are connected to build *composite* elements as higher-level protocols. To recap, it is possible to define *protocol taxonomy* of *soft real-time* components starting from the families defined by Simpson in [24], but adding much more elements with the new extensions.

## 4 Protocol Taxonomy

There are four *protocol families* that correspond to the four *basic interaction protocols*. Any of these families includes one or more components and its *variants*. We distinguish *simple protocols*, where only a component is used, from *composite protocols*, where more than one component is used to build new protocols.

### 4.1 Simple Interaction Protocols

**Channel Family.** *Rendezvous*. This is a *channel* of capacity zero. It is to denote the meeting of two processes for the purpose of intercommunicating information. In terms of the basic classification scheme, the destructive reading of data and the non-destructive writing of data are now theoretically simultaneous. The temporal implications are that both processes must be requesting to communicate before data can be transferred. The void value counterpart is known as *directional handshake*. There are four variants for *rendezvous* protocol depending on the simultaneity of the writing and reading events: *Open*, *Half Open*, *Half Closed* and *Closed.* In the first case there is simultaneous access by the writer and the reader all times. The second and the third are two combinations of holding up one process whilst the other completes the action. The last one only is possible with specialised hardware due the restricted synchronisation between two processes.

*Bounded Buffer*. This is a *channel* of capacity equal or greater than one. A *bounded buffer* has the temporal properties of the *channel*, and in addition can store a series of intermediate items in a FIFO queue. Its purpose in an operational system is to smooth the flow of message data that is being processed at a variable rate, without the possibility of data loss. However, there is now the consequential hazard that the writer can be held up should the buffer become full (see section 2). The void value counterpart is known as either *bounded stimulus buffer*, if the buffer capacity is greater than one, or *dataless channel* for capacity one. There are a lot of *channel* variants, some of them have already been studied:

*Basic Channel*. This is a *bounded buffer* modified to have a permanent consumer.

*Random Channel*. This is bounded buffer modified to have a random consumer. In order to maintain the coherence of the buffering; mean inter-arrival rate of consumptions must be greater or equal to the mean inter-arrival rate of productions.

*Multi-class Channel*. This is a *bounded buffer* modified to have more than one class of intermediate data. Different data classes have different time constraints, e.g. mean inter-arrival rate of productions, mean inter-arrival of consumptions, storage and/or retrieval mean times.

*Priority Channel*. This is a *multi-class channel* modified to have more than one *priority* class of items. The scheduling could be *preemptive* or non-*preemptive*. In *preemptive priority* strategies the data item is stopped during its service as soon as a higher-priority item arrives. In that case, first the higher-priority data item is served, after which the service of the lower-priority data item that was originally being serviced is either resumed or restarted (these are also two different variants)

**Signal Family.** *Flash Data*. This is a *signal* modified to have capacity zero. This reflects the property that an item will only be passed if the reader is waiting for it at the time the writer completes its insertion of the new item; otherwise the item is lost because there is nowhere to retain it. In terms of the basic classification scheme, the item is either destroyed by the writer (if there is no reader waiting) or destroyed by the reader (if the reader takes it), so maintaining the zero buffering condition. The void value counterpart is known as *prod.*

*Overwriting Buffer*. This is a *signal* modified to have a capacity greater than one (in the case of capacity one, the term *signal* is more used). The *overwriting buffer* has

the important property that, when the writer attempts to insert an item in an already full buffer, the oldest unread item is overwritten and the writer is able to continue. The void protocol counterpart is known as either *overwriting stimulus buffer*, for capacity greater than one, or *stimulus* for the capacity one case. The *overwriting buffer* has similar protocol variants of the *bounded buffer* and their corresponding names.

**Pool Family.** As we stated above, the *pool* is the unique member of its family. However there are two very interesting variants:

*Basic Pool*. This is a *pool* where readers and writers are different classes of tasks. They can obtain different service time and can arrive with different inter-arrival rate.

*Priority Pool*. This is basic pool modified to have *non-preemptive priority*. The *priority pool* is also a data storage as the *basic* pool, but writer tasks have *non-preemptive* priority over the reader tasks. Therefore, reader tasks are overtaken by writer tasks, when they are waiting for the *pool* service. However, if a number of readers are already in service, an arriving writer that always holds higher priority does not preempt them. Inside the reader and the writer classes the queueing discipline is FIFO (or FCFS).

### 4.2 Composite and Bi-Directional Interaction Protocols

Previous subsection was devoted to a set of protocols, grouped into families, for describing one-way interaction between two processes of a *soft real-time system*, i.e. *bilateral unidirectional interaction* protocols or *simple* protocols. S*imple* protocols provide a wide variety of forms of sharing data. However, the set of *simple* protocols can be extended in order to build new protocols with these elementary components. The main extensions are the *composition* and the *bi-direction*. The components of these two protocol extensions are also variants of the four *basic interaction families*.

**Bi-directional Protocols.** The *simple* protocols model a *peer-to-peer* relationship where the interaction is flowing in one direction from writer to reader. The extension to *bi-directional* protocols model a *client-server* relationship in which a data value (or void) is transmitted as before from client to server, but there is now an explicit and separate response in the reverse direction. Such protocols are asymmetric, with the client using a single operation to transmit parameters and receive results, and with the server using two separate operations to receive the parameters and transmit the results. These bi-directional protocols are also known as *response* protocols. These *response* protocols are made up from *channel* and *dataless channel* protocols. The combination of two *channel*s reflects the fact that data and voids can never be lost, so they are variants of the *channel family*:

*Remote Function Call*. This is a combination of two *channels*. The client process transmits data through one *channel* and waits for the server process to take these data, carry out an action, and then return the results through other *channel*. The corresponding combination of two *dataless channels* (void) is known as *Remote Thread Invocation*.

*Remote Data Send*. This is a combination of a *channel* and a *dataless channel* (void). No result is expected but there is an explicit acknowledgement.

*Remote Data Fetch*. This is the reverse of the *remote data send*. It provides a means of requesting a value from another process.

Other combinations are possible with other *simple* protocols but are not appropriate in this text, either because imply destructive writing or due apparent synchronisation problems because some data (or void) could be lost.

**Composite Protocols.** *Bi-directional* is not the unique form of interaction with two *simple* protocols. In fact, *composite* protocols can hold the *peer-to-peer* relationship through a *queued* connection. The number of *simple* protocols involved in the interaction could also not to be limited to only two elementary components. This extension is known as *composition* and allows unlimited combinations of components. Some usual *composite channels* are now described:

*Communication Channel*. This is a combination of two *bounded buffers*. The *composition* is *unidirectional*; the data passes from a *basic channel* to a *random channel* in *queued* connection through a communication server. Figure 3 represents the queueing network of this *channel*.

*Communication Channel with supplementary load*. This component is identical to the previous one, but there is a supplementary load, which uses the same communication service. The load that is added introduces a greater delay between the sender and the receiver. This means that the proposed queueing model is identical to the one of the *communication channel* but there will be also a source generating data of a different class and sending them through the communications service. Once this another class of data has been communicated, it will be routed outside the component.

*Protocol Pool*. This is a *communication channel* modified to include acknowledgement of the consumption. A *dataless channel* (void) is used to acknowledge. Despite it is a combination of three *channels*, the functional behaviour for the process waiting for the acknowledgement is a high-level *pool* and the interaction is *bilateral bi-directional*.

## 5 Performance Modelling of Interaction Protocols

An initial performance study of *simple channels* can be found in [12]. In [13] simulation, exact and approximate analytical models of *basic*, *random* and *composite channels* are presented. *Composite channels*, including *communication channel with supplementary load* and *protocol pool* models were proposed in [14]. Performance analysis for *multi-class channels* and *priority channels* can be found in [15] and [16], respectively. In [17] queueing models for *basic* and *priority pools* are proposed. Any of these *interaction protocol* analyses are based on the *semaphore queue* paradigm created initially for modelling window flow control mechanisms in typical local or wide area networks [6]. In fact, this technique could also be applied to any simple asynchronous communication between processes sharing a buffer, and this is the incipient motivation of this chapter.

The application of the *semaphore* queue paradigm to solve window control mechanism problems provided us the initial motivation for analysing *interaction protocols* as elementary components of a *soft real-time system*. The modelling of these components through queueing networks shares the same features:

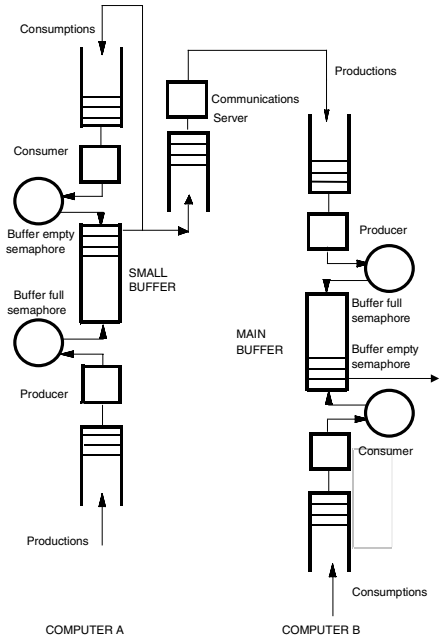



**Fig. 3.** Simulation model of a communication *channel* as composition of two *simple channels*, a *basic* and a *random channel*, through a communication server

- a memory to transfer data among processes.
- the finiteness of that memory.
- a blocking mechanism of the system when memory is full or when no more processes are allowed to access the component.
- an undetermined number of customers trying to access to this queueing system.
- the possibility of connecting components either in the same or in a different layer of a *soft real-time system* design.
- the asynchronous communication among processes interacting with the component in the same layer.

These characteristics lead us to study the different models that can be obtained from a given single component. In every component several alternatives of modelling were considered. A simulation model representing the dynamic behaviour (functional) of a given component; by this way, good performance and behaviour were obtained but execution was too long if obtaining small confidence intervals around the results is required. Sometimes a simple analytical model with exact solution was proposed. In this last case, the functionality of the model had to be relaxed in order to reach some kind of analytical computation, although its execution time was the fastest and performance computation exact. Between these two opposite cases, there was the possibility of finding some analytical approximation, in which some numerical

exactitude is given up in front the exact algorithms in order to approach to the simulated behaviour achieving a quite good approximation with a quick execution.

The aim of this chapter is to provide approximate analytical procedure for *interaction protocols*. These approximations are analysed using the *decomposition-aggregation* method [4] by means of *Single-server Semaphore queue* (*SsSq*) and *Multi-server Semaphore queue* (*MsSq*) paradigms modified accordingly to the functional behaviour of the given component. In next section, we are going to establish the theoretical foundations of the *SsSq* and the *MsSq*.

### 5.1 Single server Semaphore queue (*SsSq*)

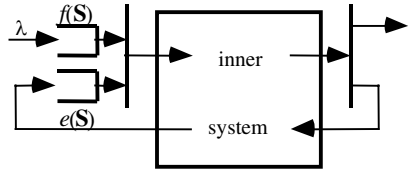



**Fig. 4.** The Single-server Semaphore queue (*SsSq*) model

The queueing network shown in figure 4 can model a classical *semaphore* queue. The queueing network consists of the service sub-network, which will be referred to as the inner system, and a *semaphore* station. The *semaphore* station *S* consists of an input queue *f*(*S*), referred to as the customer queue, and a token queue *e*(*S*), referred to as the resource queue. Customers arrive at the system in a *Poisson* fashion at the rate of $\lambda$. An arriving customer *join*s the input queue if it finds other customers waiting in the queue. If it finds the queue empty, then it requests a token from the token queue. If there is a token available in queue *e*(*S*), the customer takes a token and then it moves into the inner system. However, if the token queue is empty, the customer waits in the input queue *f*(*S*) until a token becomes available. A customer that enters the inner system *join*s the service sub-network. After receiving service, the customer departs from the queueing network and its token *join*s the token queue. Since a customer cannot enter the inner system without a token, the total number of customers in the inner system may not exceed *C*, the maximum number of tokens. If there are tokens in *e*(*S*), then that means that there are no customers in the input queue. On the other hand, if there are customers in the input queue, then *e*(*S*) is empty.

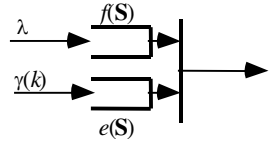



**Fig. 5.** The input queue and the token queue belonging to a *semaphore*. At the instant that each queue contains a customer, both instantaneously depart and merge into a single customer (*join*)

In figure 4, symbols commonly used in *Petri* nets are introduced in order to describe the *fork* and *join* operations. In particular, the *join* symbol of figure 5 depicts the

following operation: at the instant that queues $f(S)$ and $e(S)$ contain a customer, both instantaneously depart from their respective queues and merge into a single customer. The *fork* symbol (figure 6) depicts the following operation. A customer arriving at this point is split into two siblings. These two symbols are used for descriptive convenience.

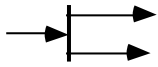


**Fig. 6.** The *fork* operation: a customer arriving at the transition is split into two siblings

An exact analysis of the queueing system depicted in figure 4 is rather difficult. Therefore, it is proposed to analyse it using hierarchical *decomposition* and *aggregation* [8]. The basic modelling strategy consists of *decomposing* the queueing network model into an *aggregated* or *flow-equivalent* sub-network and the *designated* or *complement* sub-network. Then, the original queueing network (figure 4) is short-circuited to make a shorted closed queueing sub-network of the inner system and the remaining *semaphore* sub-network of figure 5. This way, the *flow-equivalent* sub-network of the inner system is analysed in isolation to obtain the throughput across for various customer populations (*decomposition* step). Finally, a load dependent server can replace the *flow-equivalent* sub-network in the original queueing network to form an equivalent queueing network. For any given number of customers, the service rate of the load dependent server is equal to the throughput of the *flow equivalent* sub-network (*aggregation* step). In particular, the system shown in figure 5, the *complement* sub-network, is analysed assuming that the arrival process at queue $e(S)$ is described by a state dependent arrival rate $\gamma(k)$ obtained from the isolated analysis of the *flow-equivalent* sub-network of the inner system.

This queueing system depicts the *semaphore* operation described above. The arrival process at queue $f(S)$ is assumed to be *Poisson* distributed and there are $C$ tokens. It is also assumed that the *inter-arrival* times available at queue $e(S)$ are *exponentially* distributed with a rate $\gamma(k)$, where $k$ is the number of outstanding tokens, i. e. $C - k$ is the number of tokens in queue $e(S)$. The state of the system in equilibrium can be described by the tuple $(i, j)$, where $i$ is the number of customers in queue $f(S)$ and $j$ is the number of tokens in queue $e(S)$. The rate diagram associated with this system is shown in figure 7, and corresponds to a *Single-server Semaphore queue* (*SsSq*).

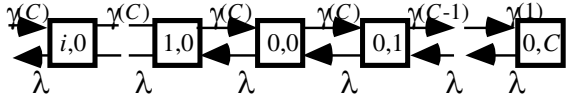



**Fig. 7.** Flow rate diagram of a S*ingle-server Semaphore queue* (*SsSq*) with $\lambda$ arrival rate and $\gamma(k)$ state dependent service rate, respectively

It is noticeable that the *SsSq* system studied by means of the proposed approximation becomes a *birth-and-death* process with an arrival rate $\lambda$ and a state dependent service rate $\gamma(n)$ if $n \leq C$, and $\gamma(C)$, if $n > C$, where $n$ is the number of customers in this queue. The random variables, $i$ and $j$, are related to $n$ as the number of customers waiting in

the queue, as follows: $i$ = max (0, $n$ - $C$) and $j$ = max (0, $C$ - $n$), which is equivalent to the number of customers in the buffer. The solution of this system is obtained by a direct application of classical *birth-and-death* process results. Thus,

$$p(i,0) = \rho^i p(0,0)\,, \tag{1}$$

$$p(0,j) = \frac{\Pi(j)}{\lambda^j} p(0,0)\,, \tag{2}$$

where $\rho = \lambda/\gamma(C)$ and

$$\Pi(j) = \left\{ \begin{matrix} \prod_{k=0}^{j-1} \gamma(C-k) & j > 0 \\ 1 & j = 0 \end{matrix} \right\}. \tag{3}$$

The probability $p(0,0)$ is chosen so that the equilibrium state probabilities add up to 1:

$$p(0,0)^{-1} = \frac{1}{1-\rho} + \sum_{j=1}^{C} \frac{\Pi(j)}{\lambda^j}\,. \tag{4}$$

However, all these expressions have been obtained assuming that $\gamma(k)$ is known. This can be approximately obtained by studying the closed *flow-equivalent* sub-network. Therefore, the throughput of this network can be computed for different values of $k$, the number of customers, where $k$ = 1, 2,…, $C$. Finally, this throughput is set equal to the arrival rate $\gamma(k)$ of tokens at the token queue $e(S)$. The solution existence condition is $\lambda < \gamma(C)$, where $\gamma(C)$ is the maximum throughput and $\rho < 1$.

### 5.2 Multi-server Semaphore queue (*MsSq*)

On the other hand, a *Multi-server Semaphore queue* (*MsSq*) S is identically constituted by its input queue $f(S)$ and its token queue $e(S)$, where there are $C$ tokens available. A customer that enters the inner system receives the required service. Nevertheless, the service rate obtained depends on the number of the customers inside the inner system. In fact, the main difference between a classical *Single-server Semaphore* and a *Multi-server* one is the *concurrent* service of the customers when they are in the inner system. After service completion the token *join*s the token queue and the customer departs from the *Multi-server Semaphore* queue. Since a customer cannot enter the inner system without a token, the total number of customers obtaining the required service *simultaneously* may not exceed $C$, the maximum number of tokens. The arrival process at queue $f(S)$ is assumed to be *Poisson* distributed and there are $C$ tokens. It is also assumed that the *inter-arrival* times, available at queue $e(S)$, are *exponentially* distributed with the rate $\varepsilon(k) = k\gamma(k)$. The state of the system in equilibrium can be described by the tuple ($i$, $j$), where $i$ is the number of customers in queue $f(S)$ and $j$ is the number of tokens in queue $e(S)$. The rate diagram associated with the *Multi-server Semaphore* queue (*MsSq*) is shown in

figure 8. Thus, the state dependent service rate of a *SsSq*, $\gamma(k)$, is weighted by the number of servers available in parallel and this result is $\varepsilon(k)$.

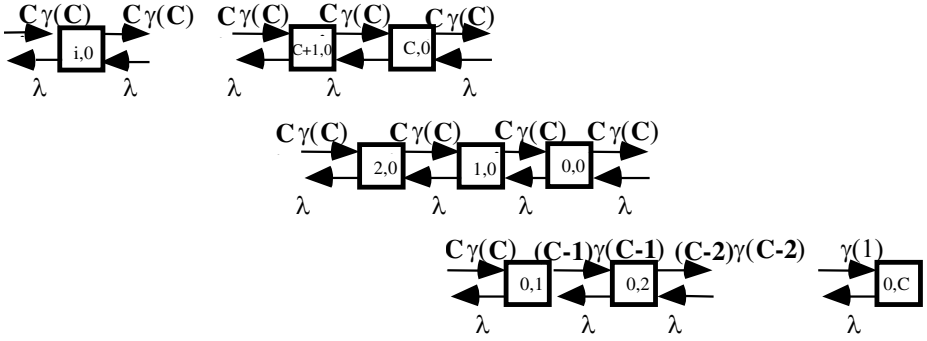



**Fig. 8.** Flow rate diagram of a *Multi-server Semaphore queue* (*MsSq*) with $\lambda$ arrival rate and $\varepsilon(k) = k\gamma(k)$ state dependent service rate, respectively

The solution existence conditions is expressed as $\lambda < \varepsilon(C) = C\gamma(C)$, the corresponding maximum throughput of *S*. Under these assumptions, it can be computed the following steady-state probabilities from the global balance equations:

$$p(i,0) = \frac{(C\rho)^i}{C^i} p(0,0) = \rho^i p(0,0) \quad \forall i, 0 \le i \le C , \tag{5}$$

$$p(i,0) = \frac{C^C \rho^i}{C^C} p(0,0) = \rho^i p(0,0) \quad \forall i \ge C , \tag{6}$$

$$p(0, j) = \frac{\Pi(j)}{\lambda^j} p(0,0) , \tag{7}$$

where $\rho = \lambda/\varepsilon(C) = \lambda/[C\gamma(C)]$ and

$$\Pi(j) = \left\{ \begin{array}{ll} \prod_{k=0}^{j-1} \varepsilon(C-k) & j > 0 \\ 1 & j = 0 \end{array} \right\} . \tag{8}$$

Consequently, $p(0,0)$ is given by the normalisation equation as follows:

$$p(0,0)^{-1} = \frac{\rho^C}{1-\rho} + \sum_{j=0}^{C-1} \frac{(C\rho)^j}{C^j} + \sum_{j=1}^{C} \frac{\Pi(j)}{\lambda^j} . \tag{9}$$

Formulas 5 and 6 are equivalent when the marginal probabilities from $i = 0$ to $C$-1 and beyond are computed. In fact, there would be have no problem to join the first two summands of formula 9 into only one. Since a *SsSq* is an extension of an *M/M/1*, at least in the simplest case, a *MsSq* is an extension of a *M/M/C* and that is the reason

to distinguish the first $C$ terms of the infinite part in the *MsSq* (central section in figure 8). By this way, the calculation of some performance measure, e.g. the mean queue length of a *MsSq*, have to take into account the shift of C terms due the extended states and consequently, the corresponding weight of every marginal state probability, from $i = 0$ to $C$-1.

Nevertheless, all these expressions have been calculated again assuming that $\gamma(k)$ is known. Therefore $\gamma(k)$, the throughput of this network, can be computed for different values of the number of customers, where $k = 1, 2, \ldots, C$. As we remarked above, the corresponding rates $\varepsilon(k)$ are given by

$$\varepsilon(k) = k\gamma(k) . \qquad \textbf{(10)}$$

### 5.3 Resolution Considerations Using Semaphore Queues

The approximate analysis of different components in the *protocol taxonomy* was solved by application of *semaphore* queues. Given the requirements about the arrival-departure flows, their statistical distributions and the scheduling at the queue representing the model of interest, the first and unforgettable step would be to determine if a set of *semaphore* queues with minor changes would be applicable. This problem is summarised in how to obtain the corresponding flow rates of tokens, i.e. the different $\gamma(k)$ for any of the *semaphore* queue. In other words, the system shown in figure 5, is analysed assuming that the arrival process at queue $e(S)$ is described by a state dependent arrival rate $\gamma(k)$ obtained from the isolated analysis of the *flow-equivalent* sub-network of the inner system. In some cases, these rates are the throughput of tokens calculated in a BCMP sub-network [9]. As example, Figure 9 shows the queueing network of a *communication channel* as composition of a *basic channel* and a *random channel* (see the simulation model in figure 3). Both *simple protocols* could be approximately solved with the application of the *SsSq* technique. The composite model builds a *communication channel* by substitution of the *simple protocols* by state dependent service queues.

However, there are models that not fit into this ideal. Then the resolution procedure needs to compute these rates either with an approximation for non-BCMP queues or even through the resolution of the isomorphic Markov chain of a reachability graph of an Stochastic *Petri* Net. There are also models of protocol variants where the *memory-less* property is not applicable. In these last cases, the resolution comes from the computation of the mean queue length through classical approximations combined with the *semaphore* queue techniques.

## 6 Conclusions and Future Work

This chapter has described the possibility of building approximated analytic models of typical *soft real-time* design components known as *interaction protocols*. The

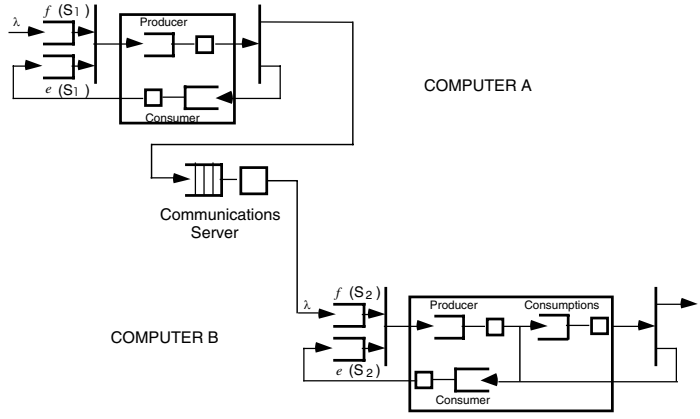



**Fig. 8.** A communication *channel* as result of composing two *Single-server Semaphore queues*

*interaction protocols* serve as intercommunication data areas among processes. These protocols are grouped into four families namely, *channel*, *pool*, *signal* and *constant*. Each *protocol family* is distinguished from the other for its destructive or non-destructive writing or reading of the intermediate data that the processes are sharing. *Protocol family* members have been extended varying not only the *capacity* of buffering but also the *classification* and the *scheduling* of the intermediate data. Some higher-level *protocol variants* have been proposed with *composition* and *bi-direction*. In *soft real-time systems*, several tasks, attempting to access a common resource may be excluded while one task has possession. This effect is generally given as example of what queueing networks cannot model well. By the use of the *semaphore queues*, we have developed approximate analytical models in order to avoid the need of using a simulation with a reasonable degree of accuracy.

The initial *taxonomy* of *interaction protocols* that has been proposed, can be included by the software developers in performance libraries in order to *complement* system design methodologies as *MASCOT*. These performance libraries would include also simulation models to implement a mixed performance model of the system with the corresponding performance tool. Therefore, queueing performance models of *soft real-time* components could complement the whole design of a large *soft real-time system*.

Also it has been given some clues to generalise the approximated analytical modelling and performance evaluation of new *protocols* of the performance *taxonomy*. Therefore, new *interaction protocols*, with buffering or shared variables, will be solved from the study of *simple* protocols. The successive application of the decomposing technique combined with some iterative, will increase the number of modelled components as available *building blocks* for developing new performance tools for *soft real-time systems*.

This chapter should be extended in order to increment the approach alternatives to solve analytically all the *interaction protocols* that could appear in the whole system design. Since the analytical approximations to solve most of the *interaction protocols* are based on *semaphore* queue modelling, another open task is the completion and classification of this paradigm of queues varying those parameters that the Kendall's notation depicts.

**Acknowledgements**. The authors gratefully acknowledge the inspirer of this chapter, *Hugo R. Simpson* from *Matra British Aerospace Dynamics Ltd.*, who provided the main rationale and specification of interaction protocol families, the bi-directional extension and the definition of the *DIA* architecture. Despite his interesting research is not addressed to performance issues, he also indirectly gave us new ideas and open questions for our future research.

## References

1. Bate, G.: Mascot 3 an Informal Introductory Tutorial. Software Engineering Journal, May (1986) 95-102
2. Bause, F., Kritzinger, P.S.: Stochastic Petri Nets. An Introduction to the Theory. Advances in Computer Science, Verlag Vieweg, Wiesbaden, Germany (1996)
3. Bolch, G., Greiner, S., de Meer, H., Trivedi, K.S.: Queueing Networks and Markov Chains. Modeling and Performance Evaluation with Computer Science Applications. John Wiley & Sons, New York (1998)
4. Conway, A.E., Georganas, N.D.: Queueing Networks-Exact Computational Algorithms. In: Schwetman, H. (ed.): A Unified Theory Based on Decomposition and Aggregation. Computer Systems, MIT Press, Cambridge, Massachusetts (1989)
5. Cooling, J.E.: Real-Time Software Systems. An Introduction to Structured and Object-oriented Design. International Thomson Computer Press (1997)
6. Fdida, S., Perros, H.G., Wilk, A.: Semaphore Queues: Modeling Multilayered Window Flow Control Mechanisms. IEEE Transactions on Communications, March, 38(1990)3, 309-317
7. Gomaa, H.: Software Design Methods for Concurrent and Real-time Systems. In: Habermann, N. (ed.): The SEI Series in Software Engineering. Addison-Wesley, Reading, Massachusetts (1993)
8. Harrison, P.G., Patel, N.M.: Performance Modelling of Communication Networks and Computer Architectures. McGettrick, A.D. (ed.): International Computer Science Series. Addison-Wesley, Wokingham, England (1993)
9. Haverkort, B.R.: Performance of Computer Communication Systems. A Model-Based Approach. John Wiley & Sons, Chichester, England (1998)
10. Jackson, K.: Language Design for Modular Software Construction. IFIP Congress Proceedings (1977) 577-581
11. Jackson, K., Llamosí, A., Puigjaner, R.: A Comparison between Two Methods for Large Real-Time Systems Design. Complement Project Document UIB67-1.0, November (1993)
12. Juiz, C., Puigjaner, R.: Improved Performance Model of a Real Time Software Element: The Producer-Consumer. Proceedings of the Second International Workshop on Real-Time Computing Systems and Applications, Tokyo, Japan, IEEE Computer Society Press, October (1995) 174-178
13. Juiz, C., Puigjaner, R.: Performance Modeling of Data Transfer Channels in Soft Real-Time Systems. In: Ni, L., Znati, T.F. (eds.): Proceedings of the Conference on Communication Networks and Distributed Systems Modeling and Simulation. San Diego, CA, USA, Society for Computing Simulation, January (1998) 101-106
14. Juiz, C., Puigjaner, R., Jackson, K.: Performance Evaluation of Channels for Large Real-Time Software Systems. Proceedings of the International Conference and Workshop on Engineering of Computer Based Systems, Jerusalem, Israel, IEEE Computer Society Press, March (1998) 69-76
15. Juiz, C., Puigjaner, R.: Performance Analysis of Multiclass Data Transfer Elements in Soft Real-Time Systems. Proceedings of the International Conference and Workshop on Engineering of Computer Based Systems, Nashville, TN, U.S.A., IEEE Computer Society Press, March (1999) 152-158

16. Juiz, C., Puigjaner, R., Perros, H.G.: Performance Analysis of Multi-Class Data Transfer Elements in Soft Real-Time Systems using Semaphore Queues. In: Gelenbe, E. (ed.): System Performance Evaluation: Methodologies and Applications. CRC Press, Boca Ratón, Florida (2000) 275-289
17. Juiz, C.; Puigjaner, R.: Queueing Analysis of Pools in Soft Real-Time Systems. In: Haverkort, B.R. et al. (eds.): TOOLS 2000. LNCS, No 1786, Springer-Verlag, Berlin Heidelberg (2000) 56-70
18. Pyle, Y., Hruschka, P., Lissandre, M., Jackson, K.: Real-time Systems, Investigating Industrial Practice. In: Tully C., Pyle I. (eds.): Wiley Series in Software Based Systems. John Wiley & Sons, Chichester, England (1993)
19. Peterson, J.L.: Petri Net Theory and the Modeling of Systems. Prentice Hall, Englewood Cliffs, New Jersey (1981)
20. Simpson, H.R.: The Mascot method, Software Engineering Journal, May (1986) 103-120
21. Simpson, H.R.: A Data Interaction Architecture (DIA) for Real Time Embedded Multi Processor Systems. Proc. in Computing Techniques in Guided Flight, Royal Aeronautical Society, Boscombe Down, U.K. (1990)
22. Simpson, H.R.: Real Time Networks in Configurable Distributed Systems. Proc. Int. Workshop on Configurable Distributed Systems, London, U.K. (1992)
23. Simpson, H.R.: Layered Architecture(s): Principles and Practice in Concurrent and Distributed Systems. Proc. Int. Conf. and Workshop on Engineering of Computer-Based Systems, Monterey, CA, U.S.A, IEEE Computer Society Press (1997) 312-320
24. Simpson, H.R.: Real-time Network Architecture. Proc. World Conf. on Integrated Design and Process Technology, Berlin, Germany (1998)

# A Performance Engineering Case Study: Software Retrieval System*

José Merseguer, Javier Campos, and Eduardo Mena

Dpto. de Informática e Ingeniería de Sistemas, University of Zaragoza, Spain
{jmerse,jcampos,emena}@posta.unizar.es

**Abstract.** This chapter presents a case study in performance engineering. The case study consists of a Software Retrieval System based on agents. The system is modelled in a pragmatic way using the Unified Modeling Language and in a formal way using stochastic Petri Nets. Once the system has been modelled, performance figures are obtained from the formal model. Finally, some concluding remarks are obtained from our experience in the software performance process.

## 1 Introduction

The common tasks of retrieving and installing software using Internet are provided by several sites (e.g., Tucows [3], Download.com [1], and GameCenter [2]). From a user point of view, the process of selecting and downloading software could become costly and sometimes slow, therefore the performance of these kind of services becomes *crucial*.

This chapter describes the performance study of a Software Retrieval System (SRS hereafter) proposed in [11]. This SRS, as those mention above, allows Internet users to select and download new pieces of software. The main differences between it and the others are that the SRS has been designed for wireless network systems (to satisfy mobile computer users) and it has been developed using mobile agents technology [16].

The chapter is organised as follows. In Section 2, the SRS is specified. In section 3, the SRS is modelled using the *Unified Modeling Language* (UML) and *Petri nets* (PN). In section 4 a performance analysis of the system is addresed. Finally, some concluding remarks are given.

## 2 System Specification

Mobile agents are intelligent and autonomous software modules that can move themselves from one *place* to another, carrying with them their whole states. A place is a context, within an agent system,[1] where an agent can execute [15].

* This work has been developed within the project TAP98-0679 of the Spanish CICYT.

[1] An agent system is a platform that can create, interpret, execute, transfer and dispose agents.

R. Dumke et al. (Eds.): Performance Engineering, LNCS 2047, pp. 317–332, 2001.

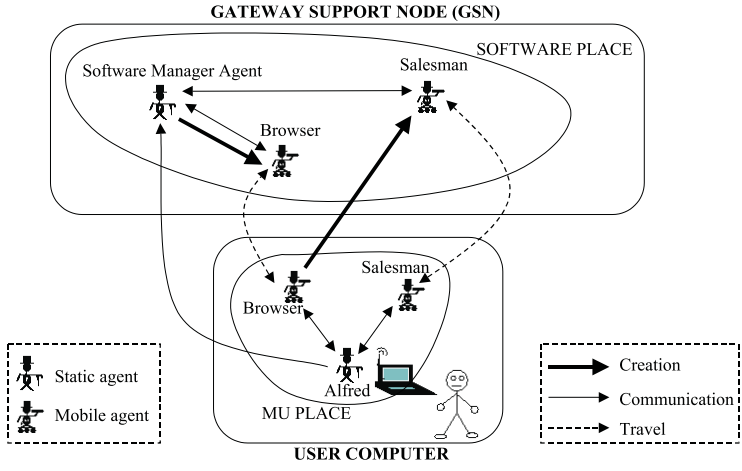



**Fig. 1.** Architecture for the Software Retrieval System

Some of the advantages of the use of mobile agents, related to accessing remote information, are the following:

- They encapsulate communication protocols.
- They do not need synchronous remote communications to work.
- They can act in an autonomous way and carry knowledge to perform local interactions at the server system instead of performing some remote procedure calls.
- They can make use of remote facilities and perform specific activities at different locations.

The main components that are involved in the SRS, with emphasis on the agents defined and how they interact, are shown in Figure 1. In the following we briefly describe how these agents interact when the user wants to retrieve software:

1. *Alfred*, an agent specialized in user interaction, allows the user to express her/his information needs. It resides in the *Mobile Unit (MU)* place.
2. Alfred communicates with the *Software Manager agent*, residing in the Software place, which obtains a catalog with the available software.
3. The Software Manager creates a *Browser agent*, a specialist in helping users to select software. This agent moves to the MU place carrying a catalog of software (see [12] for more details).
4. The Browser interacts with Alfred in order to select the wanted piece of software. The user can request a refinement of the catalog; in that case, the Browser will travel back to the *Gateway Support Node* (GSN) or, depending on the concrete refinement, it will request the information to the Software Manager. This process is repeated until the user selects the name of the software that s/he wants to install on her/his computer.

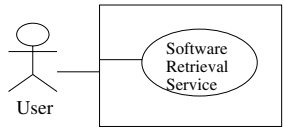



**Fig. 2.** Use Case diagram

5. Then, the Browser creates a *Salesman agent* on the GSN, which is on charge of performing all the tasks previous to installing the software on the user node, i.e., e-commerce, downloading of the software, etc.

Finally, the main features of the SRS are pointed out:

- Automatic generation (without user intervention) of a catalog of available software [12].
- The location and access method to retrieve remote software are transparent to users.
- The system maintains up to date the information related to the available software.
- The system is able to help the users to find software even if they are inexperienced users.

## 3 Modelling the Software Retrieval System

In the following, we are going to model the SRS as a previous step to obtain performance figures. First, it will be modelled using UML, but as we explain in section 5 it is not possible to obtain performance figures from UML diagrams. Therefore, in this section, we will model the system using PNs, which will be obtained from the UML diagrams. PNs will allow us to obtain performance figures for the system.

### 3.1 Modelling Using UML

The first step in the performance study of the SRS conveys in the development of the UML [4] diagrams that model the system. A *use case diagram*, a *sequence diagram* and several *statechart diagrams*, one for each agent present in the system, will be modelled; all them represent the dynamic view of the system.

In order to express the system load, the diagrams have been enriched following the notation presented in [14,13]. It must be noted that for this case study all the system load can be represented in the dynamic view of the system, avoiding the necessity to develop a static view, which could be appropriately represented using a class diagram.

Figure 2 shows the only use case needed to describe the dynamic behaviour of the system. We can see in it the unique actor which interacts with the system, the "user". The use case is described in the following.

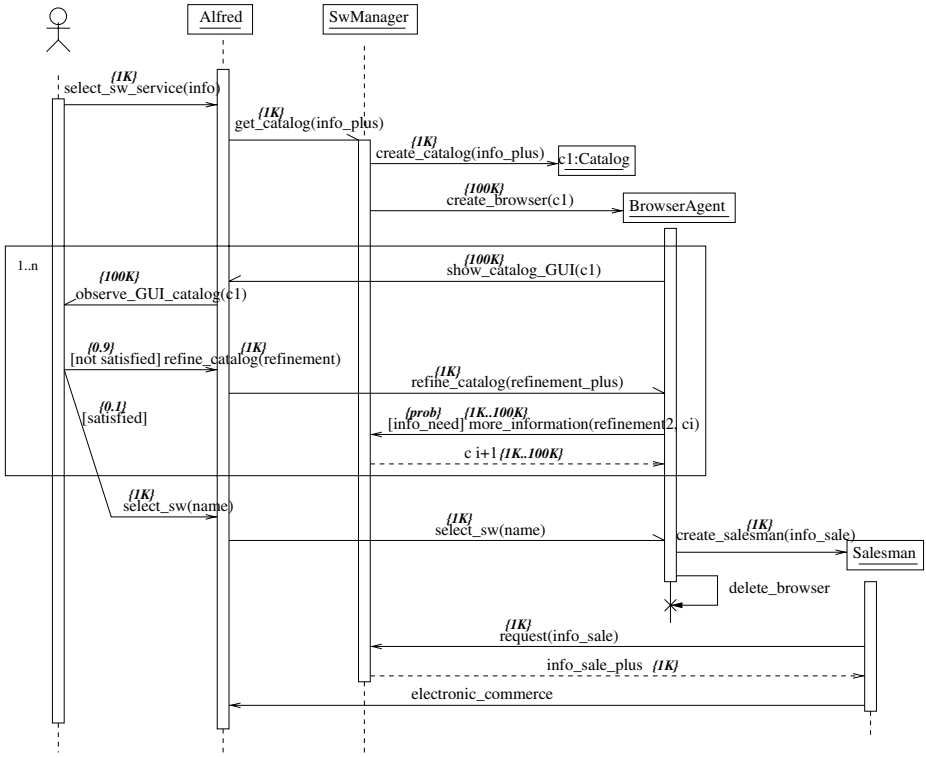



**Fig. 3.** Sequence diagram for the Software Retrieval System

**Software Retrieval System Use Case Description.**

– Principal event flow: the user requests Alfred for the desired software. Alfred sends the request to the Browser, who obtains a catalog with the available software. The Browser gives the catalog to Alfred, who shows it to the user. If the user is satisfied with the catalog then s/he selects the software, in other case s/he can ask for a refinement. This process could be repeated as many times as necessary until the user selects a concrete piece of software.

The *sequence diagram* for the SRS, that describes in detail the use case, is shown in Figure 3. The diagram represents the messages sent among agents, their load and the probability associated to the guards.[2]

As it has been told, a *statechart* has been developed for each agent in the system. They represent the life of the agents as well as the load of their events, the time spent by their activities and the probabilities associated to the guards.

In the following a detailed description of the statecharts is given:

[2] It must be pointed out that these values have been obtained from our knowledge of the *problem domain*.

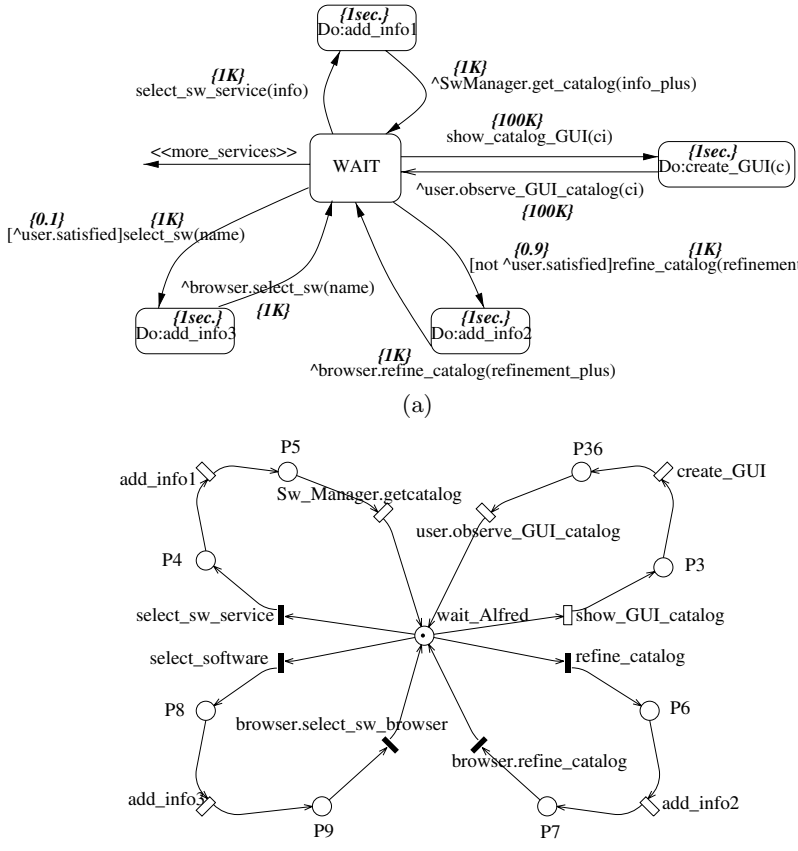



**Fig. 4.** (a) Statechart for Alfred, (b) SWN for Alfred

**Alfred Statechart.** Alfred is always present in the system, no creation event is relevant for our purposes. Its behaviour is typical for a server object. It waits for an event requesting a service (select_sw_service, show_catalog_GUI, refine_catalog or select_sw). For each request it performs the necessary activities and it returns to its wait state to serve another request. Figure 4(a) shows Alfred's statechart. The stereotyped transition $\ll more\_services \gg$ means that Alfred may attend other services that are not of interest here.

**User Statechart.** In Figure 5(a), the behaviour of a user is represented. The user is in the wait state until s/he activates a select_sw_service event. This event sets the user in the waiting_for_catalog state. The observe_GUI_catalog event, that could be sent by Alfred, allows the user to examine the catalog in order to look

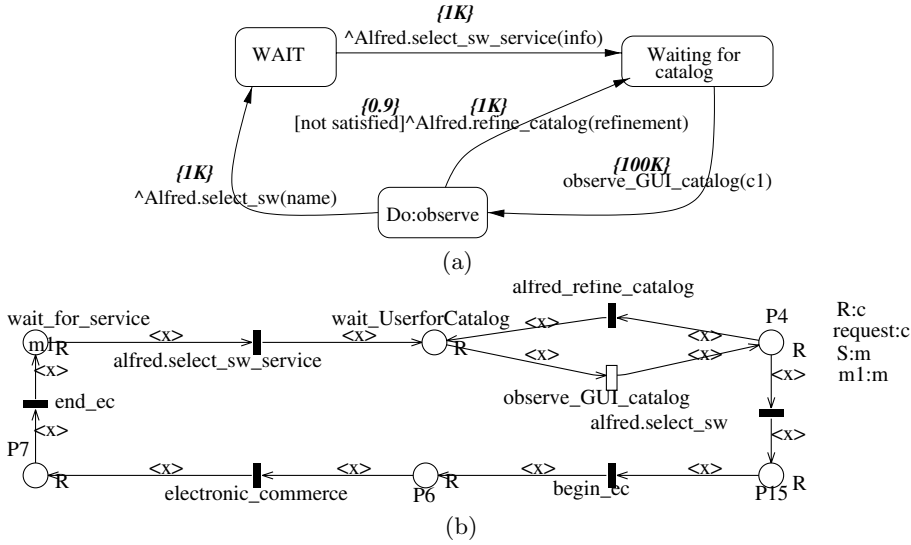



**Fig. 5.** (a) Statechart for the user, (b) SWN for the user

for the desired software. If it is in the catalog, the user sends the select_sw event to Alfred, in other case s/he sends the refine_catalog event.

**Software Manager Statechart.** Like Alfred, the Software Manager behaves as a server object. It is waiting for a request event (more_information, get_catalog, request). When one of them arrives, it performs the necessary activities to accomplish it. Figure 6(a) shows its statechart diagram. It is interesting to note the actions performed to respond the get_catalog request: first, the catalog with the available software is created, after that, the browser is created.

**Salesman Statechart.** The Salesman's goal is to give e-commerce services, as we can see in Figure 7(a). After its creation it asks the Software Manager for sale information. With this information the e-commerce can start. This is a complex task that must be described with its own use case and sequence diagram which is out of the scope of this case study.

**Browser Statechart.** The statechart diagram in Figure 8(a) describes the Browser's life. It is as follows: once the Browser is created it must go to the MU place, where it invokes Alfred's shows_catalog_GUI method to visualize the previously obtained catalog. At this state it can attend two different events, refine_catalog or select_sw. If the first event occurs there are two different possibilities: first, if the Browser has the necessary knowledge to solve the task, a refinement action is directly performed; second, if it currently has not this knowledge, the Browser must obtain information from the Software Manager,

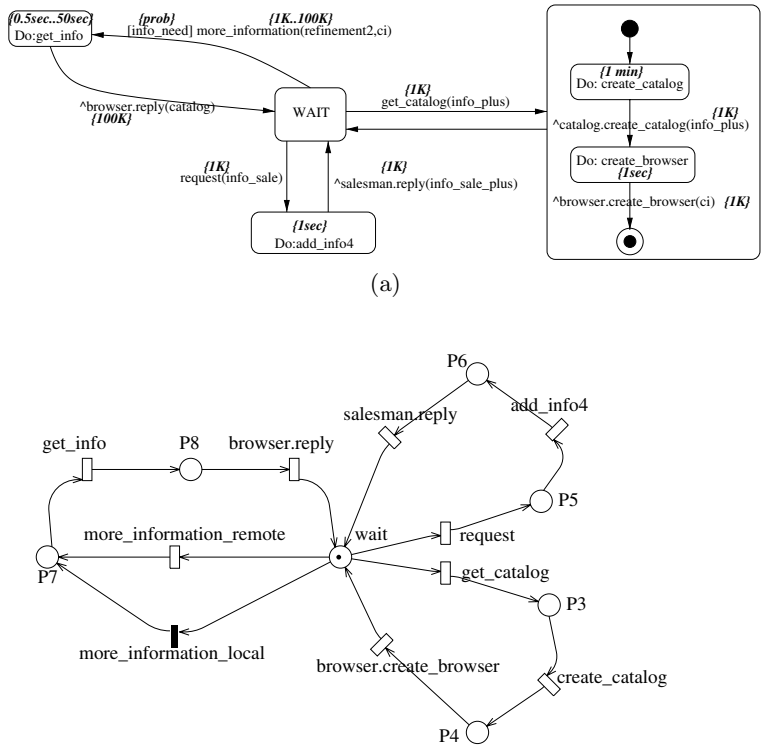



**Fig. 6.** (a) Statechart for the Software Manager, (b) SWN for the Software Manager

by sending a more_information request or by travelling to the software place. If the select_sw event occurs, the Browser must create a Salesman instance and die.

### 3.2 Modelling Using Stochastic Petri Nets

The UML diagrams previously modelled are expressive enough to accomplish with different implementations. For example, in the system specification we did not specify how many majordomos should attend requests, how many concurrent users can use the system, etc. Several situations can arise, such as:

1. One user request served by one majordomo (no concurrency).
2. Several user requests served by one majordomo.
3. Several user requests served by many majordomos, one per request.

These kind of questions can be satisfactorily solved using a formal language, such as PNs, to model the system.

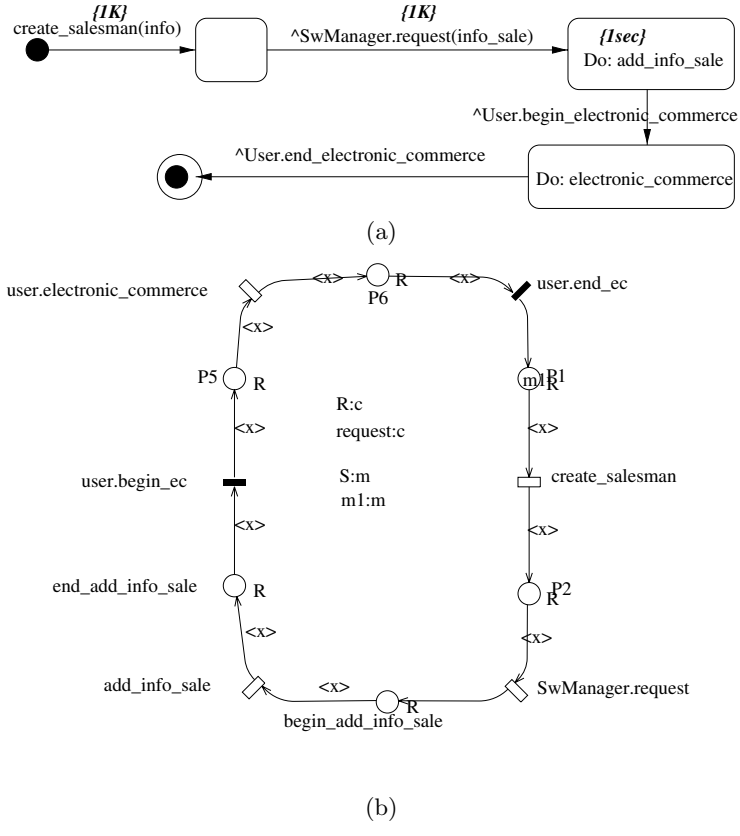
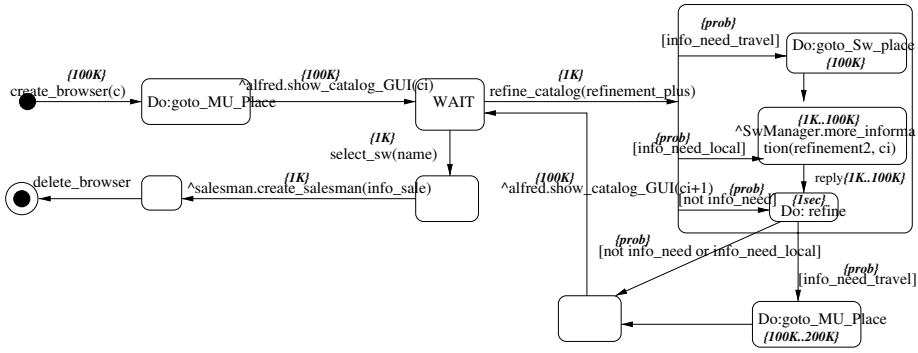



**Fig. 7.** (a) Statechart for the Salesman, (b) SWN for the Salesman

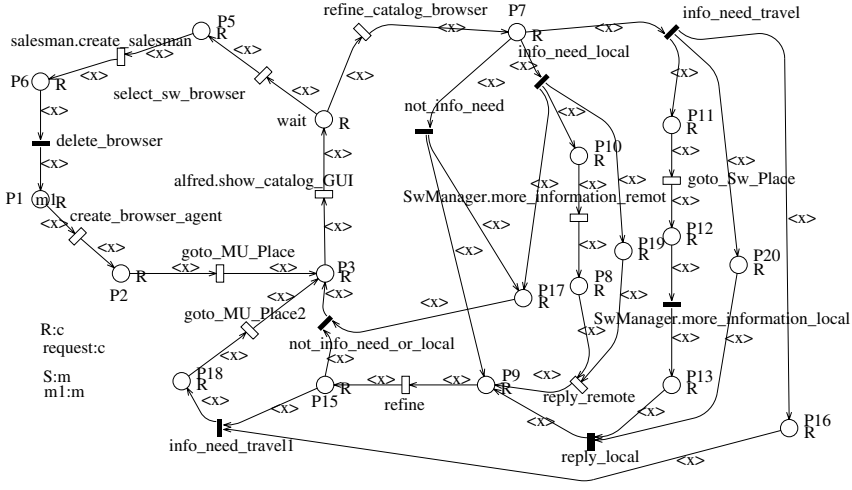
Therefore, once a statechart has been modelled, a PN (concretely a stochastic well formed coloured Petri net (SWN) [5]) is obtained from it (called *component net*). It must be noticed that the statechart and its component net provide the same information. The reasons to maintain both formalisms are given in section 5.

The component net is obtained following the *transformation rules* given in [14]. The most important facts are the following:

- Each token in the PN represents an object.
- Each state of the STD is represented by a place, with the same name, in the PN.
- For each transition in the STD arriving at a state, there will be an input transition in the PN for the place which represents the state. The name of the transition in the PN will be the same as the event that labels the transition in the STD.
- For each transition in the STD exiting from a state, there will be an output transition in the PN from the place which represents the state. The name




(a)




(b)

**Fig. 8.** (a) Statechart for the Browser, (b) SWN for the Browser

of the transition in the PN will be the same as the event that labels the transition in the STD.

– Guards in the STD are translated into a subnet including immediate (firing in zero time) transitions.

The component nets corresponding to Alfred, the user, the Software Manager, the Salesman and the Browser statecharts are shown in Figures 4(b), 5(b), 6(b), 7(b) and 8(b).

From the component nets, and considering the sequence diagram, a *complete* PN for the system is obtained. It represents the behaviour of the whole system. This net is shown in Figure 9. The complete PN has been obtained by synchronizing the component nets and following the *transformation rules* given in [14].

Moreover, it must be noticed that, for every message in the sequence diagram, there are two transitions with the same name in two different component nets, the net representing the sender and the net representing the receiver. Basically, the transformation rules state that:

- If the message has no wait semantics (half arrowhead in the sequence diagram), then an extra place will appear in the complete net to model a communication buffer between the two component nets.
- If the message has wait semantics (full arrowhead in the sequence diagram), then only one transition appears in the complete net, that represents the fusion of the transitions in the two component nets.

## 4 Performance Analysis of the SRS

Once the system has been modelled, the performance analysis can be addressed. It is accomplish in this section.

### 4.1 Performance Assumptions

It will be of interest to study the system *response time*. There are two possible bottlenecks that can decrease the system performance. First, the trips of the Browser from the MU place to the Software place (and way back) in order to obtain new catalogs. Second, the number of user requests for catalog refinements.

In order to develop the performance tests the following realistic scenarios have been considered:

1. When *the Browser needs a new catalog* (under request of the user) there are several possibilities:
   - The Browser has enough information to accomplish the task or it needs to ask for new information. It is measured by the not_info_need transition. We have considered an "intelligent Browser" which does not need information the 70% of the times that the user asks for a refinement.
   - When the Browser needs information to perform the task, it may be requested by a *remote procedure call* (RPC) (represented in the net system by the info_need_local transition) or it may travel through the net to the Software place (represented in the net system by the info_need_travel transition) to get the information and then travel back to the MU place. In this case, we have considered two scenarios. First, the 30% of the times the Browser performs a RPC, therefore the 70% of the times it travels through the net. Second, the opposite situation, the 70% of the times it performs a RPC, therefore the 30% of the times it travels through the net.
2. To test the *user refinement request*, we have considered two different possibilities. An "expert user" requesting a mean of 10 refinements, and a "naive user" requesting a mean of 50 refinements, until s/he finds the wanted software.

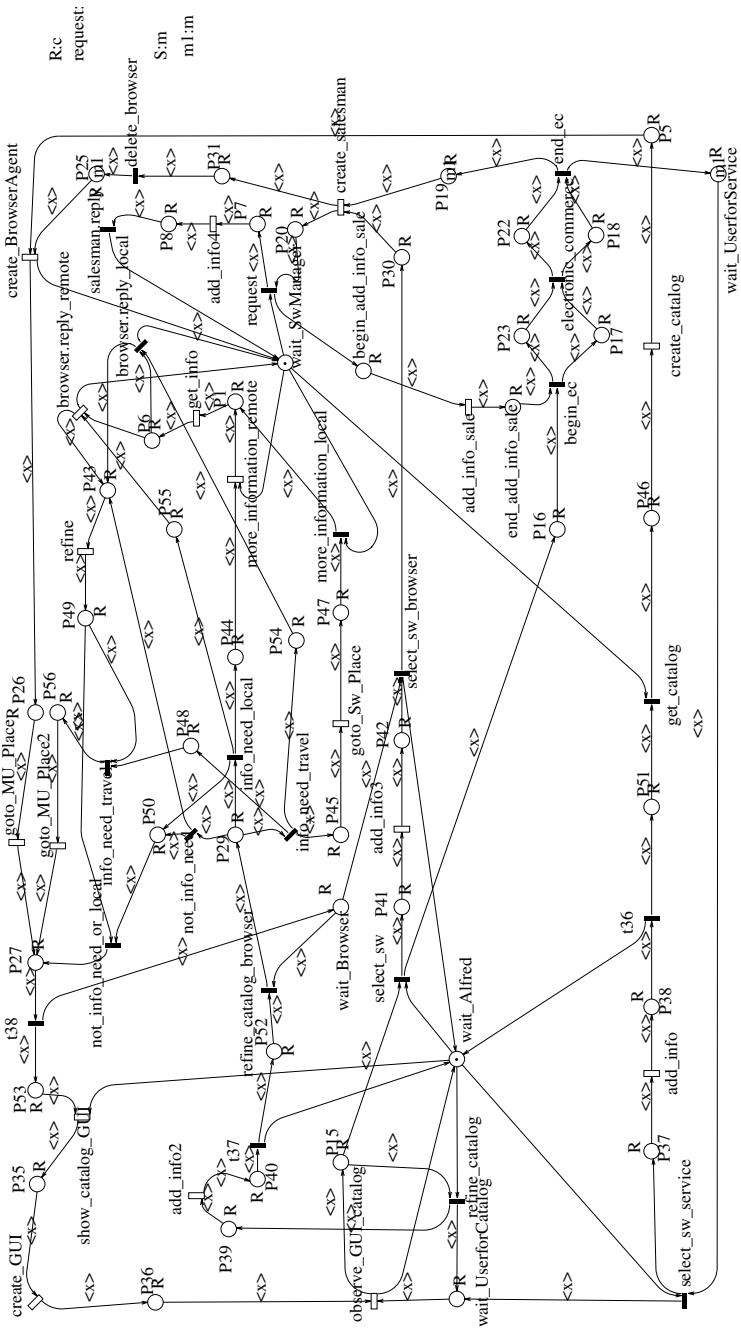



**Fig. 9.** The Petri net for the whole system

3. *The size of the catalog* obtained by the Browser can decrease the system performance. We have used five different sizes for the catalog: 1 Kbyte, 25 Kbytes, 50 Kbytes, 75 Kbytes and 100 Kbytes.
4. With respect to the speed of the net, two cases have been considered: a net speed of 100 Kbytes/sec. ("fast" connection speed) and a net speed of 10 Kbytes/sec. ("slow" connection speed).

### 4.2 Performance Results

Now, we present the results for two possible cases:

1. One user request served by one majordomo (no concurrency).
2. Several user requests served by one majordomo.

In both cases the interest is to study the system *response time*, as we said previously. The performance figures were obtained by analysing the net in Figure 9 with the tool *GreatSPN* [6]. To obtain the response time, first the throughput of the select_sw_service transition, in the net system, is calculated by computing the steady state distribution of the isomorphic *Continuous Time Markov Chain* (CTMC); finally, the inverse of the previous result gives the system response time.

**One user and one majordomo.** Figure 10(a) shows the system response time (in minutes) assuming "fast" connection speed, an "expert user" and an "intelligent Browser". One of the lines represents a probability equal to 0.7 to travel and 0.3 to perform a RPC, the other line represents the opposite situation. We can observe that there are small differences between the RPC and travel strategies. Such a difference is due to the round trip of the agent. As the agent size does not change, this difference is not relevant for the global system performance. Thus, we show that the use of mobile agents for this task does not decrease the performance.

Figure 10(b) shows system response time, supposing "fast connection", "intelligent Browser" and "naive user". The two solutions still remain almost identical.

Someone could suspect that there exist small differences because of the net speed. So, the net speed is decreased to 10 Kbytes/sec. In Figures 10(c) and 10(d) it can be seen how the differences still remain non significant for a faster net speed.

**Several user requests served by one majordomo.** Figure 11 represents a test for an "intelligent Browser", an "expert" user, a probability for RPC equal to 0.7 and equal to 0.3 to travel. Now, we tested the system for a different number of requests ranging from 1 to 4. Observe that when the number of requests is increased, the response time for each request increases, i.e., tasks cannot execute completely in parallel. Alfred and the Software Manager are not duplicated with simultaneous requests. Thus, they are the bottleneck for the designed system with respect to the number of concurrent requests of the service, and the impact of such bottlenecks can be evaluated using our approach.

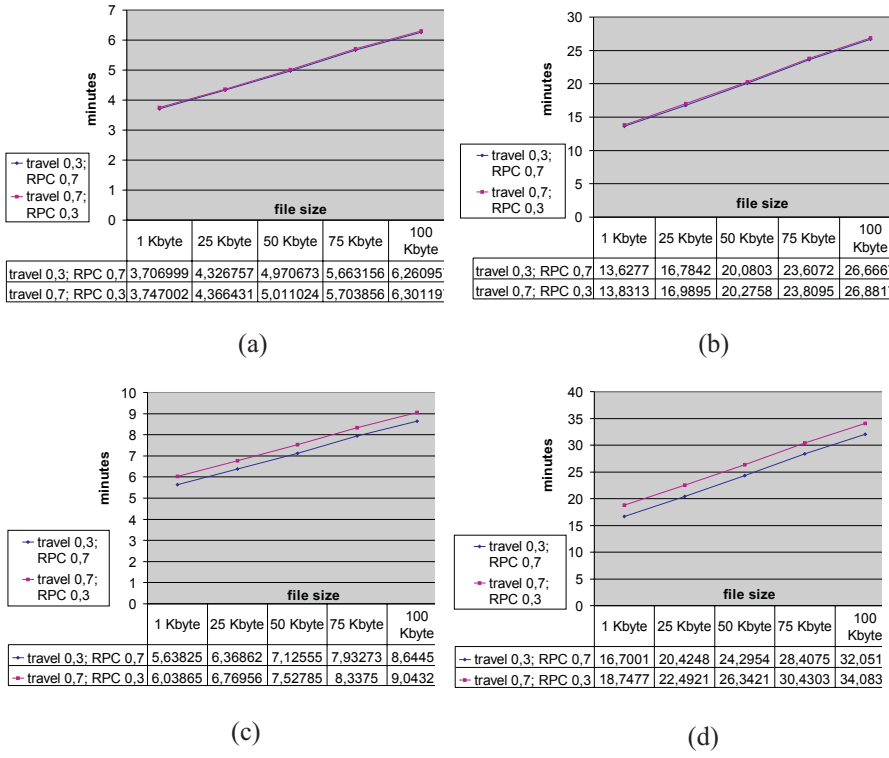



**Fig. 10.** Response time for different scenarios with an "intelligent Browser". (a) and (b) represent a "fast" connection speed, (c) and (d) a "slow" connection speed; (a) and (c) an "expert user", and (b) and (d) a "naive user"

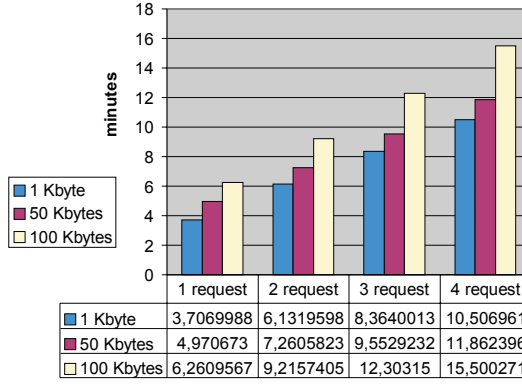



**Fig. 11.** Response time for an "intelligent Browser", an "expert user", a "fast" connection and different number of requests (1-4) and catalog size (1K, 50K, 100K)

## 5 Concluding Remarks

Along the Performance Engineering process several choices must be done. In the following, we stress some of them:

**Why to use UML to model system requirements?** Fundamentally because in the last years UML has become the standard notation to model software systems, widely accepted by the software engineering community. Unfortunately, UML lacks of the necessary expressiveness to accurately describe performance features. There have been several approaches to solve this lack [19,20,17,14]. In this chapter the last one has been followed.

**Why to use two modelling languages –UML and PNs– during the performance process?** The reasons to maintain both formalisms are given by the advantages and disadvantages they provide.

A disadvantage of PNs is that they do not show the system load (message size, guards probability, activities duration) as clear as UML diagrams do, this information is hidden in the definition of the net transitions. Besides, much work has been done in the software engineering area in developing methodologies [18, 9] for object oriented design, from which UML take profit and PNs do not.

PNs have advantage over UML because they can be used to obtain performance figures, due to the underlying mathematical model; even more, there exist software tools that automatically obtain them [6,21]. Moreover, PNs express concurrency unambiguously, UML do not.

Finally, it can be said that the UML diagrams are considered in this chapter as the documentation part of the system, being useful for the system analyst to express in a easy way the system requirements, including performance requirements. And PNs are considered as the mathematical tool which represent the performance model of the system.

**Why to use PNs to represent the performance model instead of other mathematical formalism?** As an alternative to stochastic Petri nets (SPNs), several performance-oriented formalisms can be considered as underlying mathematical tools for computing performance indices of interest. Among them, Markov chains (MCs) [7], queuing networks paradigm (QNs) [10], and stochastic process algebras (SPAs) [8]. For all of them, translation algorithms can be devised from the time-annotated UML diagrams using the general rules exposed in [14].

The main drawback of plain MCs is their poor abstraction level (each node of the underlying graph model represents a state of the system). Concerning SPAs, even if they are compositional by nature, thus specially adequated for the modelling of modular systems, the present lack of efficient analysis algorithms and software tools for performance evaluation would make them difficult to use in real applications. The use of SPNs rather than QNs models is justified because

SPNs include an explicit primitive for the modelling of synchronization mechanism (a *synchronizing transition*) therefore they are specially adequated for the modelling of distributed software design. Even more, a vast amount of literature exists concerning the use of PNs for both validation of logical properties of the system (e.g., liveness or boundedness) and quantitative analysis (performance evaluation).

## References


[1] *CNET Inc., 1999. http://www.download.com.*

[2] *CNET Inc., 1999. http://www.gamecenter.com.*

[3] *Tucows.com inc., 1999. http://www.tucows.com.*

[4] G. Booch, I. Jacobson, and J. Rumbaugh, *OMG Unified Modeling Language specification*, June 1999, version 1.3.

[5] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad, *Stochastic well-formed coloured nets for symmetric modelling applications*, IEEE Transactions on Computers **42** (1993), no. 11, 1343–1360.

[6] G. Chiola, G. Franceschinis, R. Gaeta, and M. Ribaudo, *GreatSPN 1.7: GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets*, Performance Evaluation **24** (1995), 47–68.

[7] E. Cinlar, *Introduction to stochastic processes*, Prentice-Hall, Englewood Cliffs, NJ, 1975.

[8] H. Hermanns, U. Herzog, and V. Mertsiotakis, *Stochastic process algebras as a tool for performance and dependability modelling*, Proceedings of IEEE International Computer Performance and Dependability Symposium, IEEE CS-Press, April 1995, pp. 102–113.

[9] I. Jacobson, M. Christenson, P. Jhonsson, and G. Overgaard, *Object-oriented software engineering: A use case driven approach*, Addison-Wesley, 1992.

[10] K. Kant, *Introduction to computer system performance evaluation*, Mc Graw-Hill, 1992.

[11] E. Mena, A. Illarramendi, and A. Goñi, *A software retrieval service based on knowledge-driven agents*, Cooperative Information Systems CoopIS'2000 (Eliat, Israel), Opher Etzion, Peter Scheuermann editors. Lecture Notes in Computer Science, (LNCS) Vol. 1901, Springer, September 2000, pp. 174–185.

[12] E. Mena, A. Illarramendi, and A. Goñi, *Automatic ontology construction for a multiagent-based software gathering service*, Proceedings of the Fourth International ICMAS'2000 Workshop on Cooperative Information Agents (CIA'2000), Springer series of Lecture Notes on Artificial Intelligence (LNAI), Boston (USA), July 2000.

[13] J. Merseguer, J. Campos, and E. Mena, *A pattern-based approach to model software performance*, Proceedings of the Second International Workshop on Software and Performance (WOSP2000) (Ottawa, Canada), ACM, September 2000, pp. 137–142.

[14] J Merseguer, J Campos, and E. Mena, *Performance evaluation for the design of agent-based systems: A Petri net approach*, Proceedings of the Workshop on Software Engineering and Petri Nets, within the 21st International Conference on Application and Theory of Petri Nets (Aarhus, Denmark) (Mauro Pezzé and Sol M. Shatz, eds.), University of Aarhus, June 2000, pp. 1–20.

[15] D. Milojicic, M. Breugst, I. Busse, J. Campbell, S. Covaci, B. Friedman, K. Kosaka, D. Lange, K. Ono, M. Oshima, C. Tham, S. Virdhagriswaran, and J. White, *MASIF, the OMG mobile agent system interoperability facility*, Proceedings of Mobile Agents '98, September 1998.

[16] E. Pitoura and G. Samaras, *Data management for mobile computing*, Kluwer Academic Publishers, 1998.

[17] R. Pooley and P. King, *The unified modeling language and performance engineering*, IEE Proceedings Software, IEE, March 1999.

[18] J. Rumbaugh, M. Blaha, W. Premerlani, E. Frederick, and W. Lorensen, *Object oriented modeling and design*, Prentice-Hall, 1991.

[19] G. Waters, P. Linington, D. Akehurst, and A. Symes, *Communications software performance prediction*, 13th UK Workshop on Performance Engineering of Computers and Telecommunication Systems (Ilkley), Demetres Kouvatsos Ed., July 1997, pp. 38/1–38/9.

[20] M. Woodside, C. Hrischuck, B. Selic, and S. Bayarov, *A wide band approach to integrating performance prediction into a software design environment*, Proceedings of the 1st International Workshop on Software Performance (WOSP'98), 1998.

[21] A. Zimmermann, J. Freiheit, R. German, and G Hommel, *Petri Net Modelling and Performability Evaluation with TimeNET 3.0*, Proceedings of the 11th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation, Lecture Notes in Computer Science, Vol. 1786, Springer, 2000, pp. 188–202.

# Performance Management of SAP® Solutions

Thomas Schneider

SAP AG
D-69190 Walldorf, Germany
Thomas.Schneider@SAP.com

**Abstract.** A critical success factor for companies today is the high performance of their core business processes, such as supply chain management, customer relationship management, or e-business applications. Companies with more than 30,000 business installations worldwide rely on SAP[1] solutions to run their business processes. Starting with a case study of a typical business process, the technical factors are described that ensure the high performance of an SAP installation, for example, multi-layer architecture, scalability, buffer handling, lock/enqueue management, and hardware sizing.

## 1 Introduction

When the directors of a company speak of the "performance" of an SAP System, they are thinking of the Cost of Ownership, that is, the costs of hardware, software, personnel, and development work that are associated with the implementation and operation of the system. "Performance" for them means comparing these costs with the extent to which the SAP System increases the profitability of the company's business operations.

When IT managers speak of the "performance" of an SAP System, they mean the SAP installation's availability, administrability, and error tolerance.

This chapter, however, limits itself to the technical aspects of optimizing the SAP System's response time and throughput. The term "technical performance" refers to the ability of a data processing system to fulfill a customer's requirements as to response times and data throughput. These requirements might specify, for example, a throughput of 10,000 invoices per hour, or a response time of less than one second for processing customer orders. Performance in this sense is not an absolute property of a system, but is always relative to the requirements placed on that system. [1]

This chapter describes the prerequisites that enable a SAP System to meet a customer's performance requirements.

---

[1] "SAP" and mySAP.com are trademarks of SAPAktiengesellschaft, Systems, Applications and Products in Data Processing, Neurottstrasse 16, 69190 Walldorf, Germany. The publisher gratefully acknowledges SAP's kind permission to use its trademark in this publication. SAP AG is not the publisher of this book and is not responsible for it under any aspect of press law.

R. Dumke et al. (Eds.): Performance Engineering, LNCS 2047, pp. 333–348, 2001.

### 1.1 A Case Study

To make the idea of performance requirements more concrete, consider a typical example of a business process from the mySAP Supply Chain Management (SCM) solution, consisting of the following processing steps. Assume that the company sells medicine goods.

**Step 1:** Sales orders are received daily by telephone in a call center. At peak load times, for example at 10 a.m., the SAP System owner expects *n* employees to book as many as *m* orders. To enable the employees to attain this goal, the owner of the SAP System requires that there be an average dialog response time of *t* milliseconds.

**Step 2:** The owner of the SAP System promises his or her customers that all orders that are received by *u* o'clock will be shipped to reach the customer by the following morning. To achieve this goal, all deliveries must leave the warehouse by *v* o'clock. For the SAP System, only a limited time frame remains to process the customer orders in a collective run that automatically creates the appropriate deliveries.

**Step 3:** All deliveries to be sent out are packed using a production line technique. As the articles are packed, the shipment is confirmed. This triggers the SAP System to immediately create the delivery note and the invoice, which are printed and then added to the package further along the production line. The package is then sealed and shipped. The owner of the SAP System requires that document creation and printing take no more than two minutes, since otherwise the production line must be stopped and the packaging process interrupted.

**Conclusion from this case study:** For dialog applications such as the sales order processing, optimizing the response time is the highest priority. The toughest demands on an SAP installation are made by companies such as call centers, whose employees book sales or service orders received by telephone. For large SAP installations of this kind it is not uncommon for there to be 4,000 active users and over 600,000 dialog steps per hour, with average response times of significantly less than one second. Application benchmarks however show that more than 18,000 concurrent users can be handled by the SAP technology in an benchmark environment.

For non-dialog, automated processing (for example, through interfaces or background processing), a customer's highest priority is to achieve maximum throughput despite maintaining a relatively high level of capacity utilization in order to keep hardware costs low. In this context, the greatest demands are made on SAP Systems by, for example, retail chains whose stores simultaneously transfer bulk sales data via interfaces to the central system (in an SAP Retail procedure known as "Point of Sales Inbound Processing"). Large companies with mySAP SCM or mySAP Retail solutions use their SAP System to process over a million items (in sales documents, invoices, and so on) daily. Similar mass-data throughput is achieved by banks, insurance companies, telecommunication and utilities companies during accounts rendering (using mySAP Financials).

## 1.2 Performance Tuning

Based on this case study, we can identify the different areas of performance tuning that can be devided into business process tuning and technical tuning.

**Business process tuning.** Business process tuning starts with the analysis of the business process. A core business process is the program-technical mapping of a commercial business procedure. In the case study above, the core business process consisted of creating the following documents: a customer order, a delivery with completion confirmation, the delivery note and invoice. In addition, there would normally be internal follow-on processes in Accounts and Controlling. Business process tuning usually begins with a workload analysis-finding out which transactions and programs place a heavy load on the system and should, if possible, be optimized. Performing a workload analysis means asking the following questions:

- Which programs or transactions consume the most system resources?
- Which parts of the programs cause unnecessary database load through too many database accesses, accesses that take too long, or accesses that consume too much memory or CPU? From which programs do the statements originate?

The first task in business process tuning is to ensure that SAP standard functions work efficiently. Normally, there are many ways of realizing business processes in the SAP System. While these ways are equivalent from a business perspective, they differ in their technical implementation, and some are dramatically less efficient than others. The less efficient realizations ultimately result in a high response time for the user. As these implementations are configured during the Customizing phase, it is here that you can influence the subsequent performance of the SAP System.

Another area of application tuning is optimizing the ABAP code. This type of optimization is particularly useful for customer-developed programs, customer modifications to SAP standard software objects, and the customer's use of user exits. SAP's open platform strategy allow customers to modify SAP standard coding or to write individual customer extensions to SAP standard functions. For customer programs and extensions the performance optimization is essential. This includes the performance testing of all programs with representative data. Performance optimization should be considered early on, during the SAP implementation phase; that is, during the customizing of customer-developed programs, and before implementing customer modifications of SAP software objects. If these implementation tasks are outsourced, the consultant performing them must be accountable not only for the resulting functionality, but also for its effects on performance.

Business process tuning is usually done by a team that consists of the business process owner, application and development consultants.

**Technical tuning.** The technical tuning involves configuring all SAP System components so that the load placed on the system by users can be optimally processed and does not cause performance bottlenecks. The components for technical tuning are the operating system, the database, the SAP work processes, SAP buffers, memory management, and the network. Technical tuning is usually done by system administrators, database administrators and technical consultants.

Before going into details of performance optimization in chapters 2 and 3, we want to describe briefly the new challenges that arise from the Internet.

## 1.3 Excursus: From SAP R/3® to mySap.com®

With the increasing significance of the Internet, a paradigm shift is occurring in the world of business software: The software is no longer targeted at highly specialized employees, but rather at Internet or Intranet users.

The classical approach of process automation in SAP R/3 is based on having highly specialized users who access their Enterprise Resource Planning (ERP) systems from installed SAP GUIs in fixed locations. However, this need for a specialized user who must be trained to use the software is increasingly disappearing. Instead, users can gain direct access to the ERP systems of relevant companies through the Internet or Intranet. In many companies today, for example, employees can use the Intranet to book their work hours, absences, travel expenses, and so on, thus performing activities that previously had to be performed by central users. Customers today increasingly order products directly over the Internet and no longer by the less direct means of writing a letter, sending a fax, or making a telephone call to an order center. Experts speak of a popularization or even democratization of ERP Software.

As one commentator put it: "SAP R/3 is made for clerks, but we aren't talking about clerks anymore". With the change from R/3 to mySAP.com, SAP implements the paradigm shift within its software development both through a consistent expansion of its Web technology, and through a stronger focus on application components.

Since SAP R/3 Release 3.1, you can access R/3 directly from a Web browser in conjunction with the SAP Internet Transaction Server and a Web server. With R/3 Release 4.6, R/3 attains Web capability throughout its entire range of functionalities.

The mySAP Workplace provides a uniform method of access to all systems in a system landscape. A user logs on once to the Workplace, and the Workplace Server constructs a "portal page", comprising all the starting points a user requires in his or her daily work. From here, the user can perform all daily tasks, without worrying about which systems he or she needs to log on to. The Workplace lets you create portal pages for employees, as well as for customers and partners.

In the Internet age, it is vital for companies to be able to open their systems to the outside world. Using electronic marketplaces - a further main focus of the mySAP.com initiative - you can hold auctions over the Internet, negotiate prices with other companies, and transact business with suppliers and customers. You can choose between open marketplaces, which are open to all customers, for example, via the Internet, and closed marketplaces, to which only members of a single organization or trading partnership have access, for example, oil traders participating in a spot market.

SAP collectively refers to its products from an application perspective using the terms "mySAP Application Areas" and "mySAP Application Components". In addition to familiar R/3 products such as SAP FI, SAP SD, and SAP HR, there are now New Dimension Products such as SAP Customer Relationship Management (CRM), SAP Advanced Planner & Optimizer (APO), SAP Business-to-Business Procurement (BBP), and SAP Business Information Warehouse (BW).

The expectations of "new" users as to the user friendliness and performance of IT applications are significantly higher than the corresponding expectations of the classical employee. The latter is dependent on "their" IT system, and tends to accept this system and tolerate minor bugs or performance weaknesses if, in general, the system expedites daily work. Internet users, however, are different: If the applications provided in the Internet do not function simply and efficiently, the user can immediately change to a competitor company and do his or her purchasing there.

Similarly, the Internet age brings with it heightened expectations as to how simply and effectively you can perform IT-system administration and monitoring.

## 2 Parallel Processing – Requirements on System Configuration and Hardware

The SAP System has a multi-layer client/server architecture that is the basis for its scalability.

### 2.1 SAP Client/Server Architecture

Figure 1 shows SAP's multi-layer client/server architecture.

**Presentation Layer.** Users perform data input and output using SAP's user interface (SAP GUI) or a Web browser. A presentation server is normally set up as a Personal Computer (PC). If the PC's hardware conforms to the recommendations current for the respective SAP Release, no further tuning is required on this layer.

**Internet Layer.** Occasional users (Internet or Intranet users) log on to the SAP System from the SAP GUI for HTML. This type of logon has the advantage of not requiring a special GUI program at the front end. All input and output screens are displayed in a Web browser in HTML format. Communication between the Web browser and the SAP System are enabled using a Web server and the SAP Internet Transaction Server (ITS).

**Application Layer.** After a user has entered data and pressed "Enter", the presentation server sends this data to an application server as a user request.

*SAP Work Processes.* User requests are processed by SAP work processes on the application server. One important tuning task is configuring the type and number of

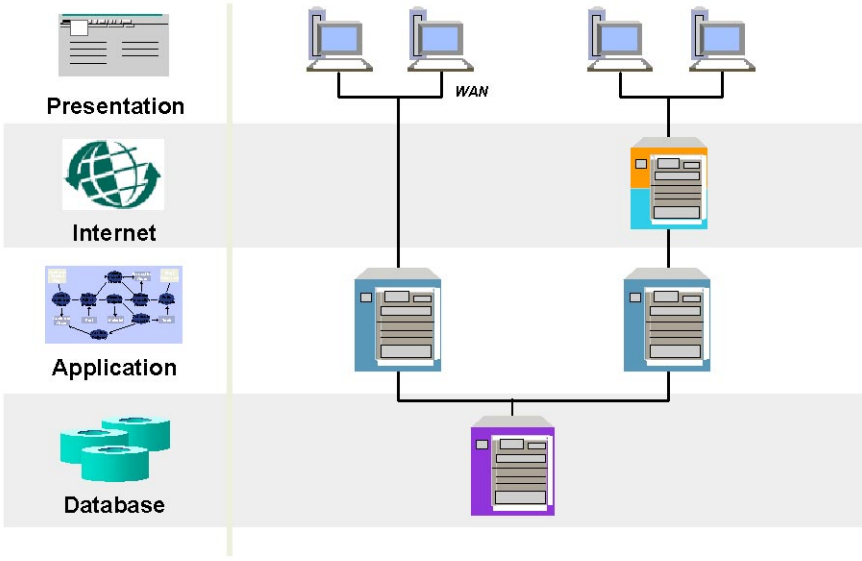



**Fig. 1.** In SAP's multi-layer client/server architecture, the presentation layer consists of the front ends, and is where the users perform data input and output using SAP's user interface (SAP GUI) or a Web browser. The actual data processing occurs on the application servers. The database server is used for saving and accessing data. If a Web browser is used as front end, communication between the Web browser and the application servers are enabled by the Internet layer

work processes on each application server. Typically, the SAP work processes are configured so that, on average, 5 to 10 active users share one SAP dialog work process. This assumes that the users need around 10 times as long to enter data in the screen and interpret the results, as the SAP System needs in order to process the user requests. Therefore, the average number of free work processes available in the system should be sufficiently large to ensure that users' queries are processed without delay. If users start one or more programs such as complex reports, the affected work processes may be occupied for several minutes. This may mean that the number of remaining work processes is not sufficient to process the queries of other users, which could cause wait times. The SAP System does not have any way of prioritizing users. If a bottleneck occurs, all users, regardless of their corporate role or the urgency of their request, must get in the queue and wait their turn.

In addition to the dialog requests for online or dialog transactions, an SAP System also processes background requests, update requests, and print requests. Each of these request types is processed by a distinct type of SAP work process. Appropriate tuning enables the workload to be distributed to optimally spread the load across the system.

*SAP Table Buffering.* Every SAP application server has various buffers, including table buffers, in which data is stored that is needed by users at runtime. This data is

known as global data since it is available to all work processes on the application server. When the data is in the buffer, it does not have to be obtained from the database, because buffers enable direct reads from the main memory of the application server. Accessing a buffer rather than the database two advantages:

- Buffer accesses are normally between 10 and 100 times faster than accesses to the database.
- Database load is reduced, a fact that becomes increasingly important as the system grows in size.

For each table, the developer or consultant can decide whether and how the table will be buffered. When the SAP System is delivered, all tables already have default buffering settings. However, to optimize runtime performance, you may need to change some of these settings depending on the usage of your SAP System. For customer-created tables, the responsible developer decides the buffering settings.

Different buffering guidelines apply to the following different types of data:

- Transaction data: The amount of transaction data in an ERP system tends to grow significantly over time and may reach several megabytes or even gigabytes. This volume of data is too large for any buffer. Examples of transaction data include sales orders, delivery notes, goods movement, and goods reservations.
- Master data: Tables with master data grow slowly over time and can reach sizes of several hundred megabytes. Therefore, as a rule, master data is not buffered. A second argument against buffering tables that contain master data is that master data is normally accessed through diverse types of selections and without necessarily using the primary index key. To optimize these accesses collectively, you should use secondary indexes instead of buffering. Examples of typical master data are materials, customers, and suppliers.
- Customizing data: Customizing data individualizes the company's business processes in the ERP system. Normally, Customizing tables are small and are rarely changed after the start of production. This, plus the fact that they are frequently read, makes them very suitable for table buffering. Examples of Customizing data include definitions and descriptions of client systems, company codes, plants, and sales organizations.

SAP buffering in includes a buffer synchronization process whereby a change to an entry in a buffered table on one SAP application server is updated in the table buffers of other SAP application servers. This is of particular interest in system where SAP applications are distributed over different application servers.

**Database Server.** If data is required to process a user request and this data is not yet in the application server's main memory, the data is read from the database server.

Database tuning is divided into three tasks:

- Optimizing database parameter settings such as those for the size of database buffers.

- Optimizing the hard-disk layout of the database to distribute the workload as evenly as possible across the hard disks. This distribution of the workload avoids wait situations when writing to or reading from the hard disk.
- Optimizing excessively long-running SQL statements known as expensive SQL statements.

Optimizing applications that are run in conjunction with a database begins with optimizing database design, and proceeds with the subject of the present chapter-optimizing the programming of SQL statements within those applications.

During application programming, SQL statements are often written without sufficient regard to their subsequent performance. Expensive (long-running) SQL statements slow performance during production operation, resulting in large response times for individual programs and placing an excessive load on the database server. Frequently, you will find that 50% of the database load can be traced to a few individual SQL statements.

As the database and the number of users grow, so do the number of requests to the database and the search effort required for each database request. This is why expensive or inefficient SQL statements constitute one of the most significant causes of performance problems in large installations. The more a system grows, the more important it becomes to optimize the SQL statements.

**Network.** The speed of transmission and the throughput between the application servers and the database server across the network are very important as they influence the performance of the entire SAP System. The SAP System is designed so that the volume of data flowing between the presentation server and the application server remains as low as possible. Typically, this data volume is only a fraction of the data volume flowing between the application server and the database server.

## 2.2 Scalability

As client-server systems, SAP Systems are scalable. Vertical scalability means that the software components of all layers can be installed either centrally on one server, or distributed over several servers.

Within one client-server layer, the workload can be distributed across several logical instances that may run on different servers. This is termed horizontal scalability, and is the technique by which the presentation layer is typically distributed on PCs or terminal servers. The application layer is realized using SAP instances. The Internet layer is realized using ITS instances and Web server instances. In principle, some database systems allow you to parallelize the database layer by setting up multiple database instances, but in practice this technique is rarely used.

Experience shows that performance problems in large SAP installations with more than 1000 concurrent users are most often caused by bottlenecks in the database server. Therefore, tuning the database becomes increasingly important as the size of the system increases. When a system has been running in production operation for some time, most tuning settings will have been satisfactorily optimized and require no further change. These settings include buffer settings, load distribution, and so on. By

contrast, the tuning of expensive SQL statements becomes increasingly important as the database data volume grows, and is an ongoing tuning process.

### 2.3 Benchmarks

SAP and its hardware partners have developed and performed Standard Application Benchmarks to test and to prove the scalability of SAP systems. The benchmark results provide basic sizing recommendations for customers by testing new hardware, system software components, and Relational Database Management Systems (RDBMS). They also allow comparisons of different system configurations. The SAP Standard Application Benchmarks have been available since R/3 Release 1.1H (April 1993) and are now available for numerous components. The benchmarking procedure is standardized and well-defined. It is monitored by the SAP Benchmark Council that consists of representatives of SAP as well as hardware, logo, and technology partners involved in benchmarking. Originally introduced to strengthen quality assurance, the SAP Standard Application Benchmarks can also be used to test and verify scalability, concurrency and multi-user behavior of system software components, relational database management systems, and business applications. All performance data relevant to system, user, and business applications are monitored during a benchmark run and can be used to compare platforms and as basic input for sizing recommendations. [2, 3]

### 2.4 Sizing Methods

Hardware sizing is the calculation of how much hardware capacity is required by an SAP System. It includes calculating the required amount of main memory, the required CPU capacity, and the required hard-disk sizes. In all operating systems, you can allocate more memory than is physically available. To enable users to enjoy maximum system performance, the amounts of physical and virtually allocated memory should not be disproportionate to one another.

To perform sizing, SAP's hardware partner companies require details regarding the demands users will place on the SAP System. Based on the practical requirements of these users and benchmark values, hardware partners create a proposal specifying the required hardware dimensions. In an SAP implementation project, the customer obtains proposals from several alternative hardware partners. See Figure 2 for an illustration of how hardware is sized. [3, 4]

Sizing is performed in accordance with detailed guideline values based on research into the memory requirements of users and transactions. These guidelines change continually and are therefore not included here.

To simplify and standardize sizing for small- and medium-sized SAP Systems, SAP's Internet site SAPNet offers the Quick Sizer, a program that enables customers and partners to make a rough assessment of your hardware needs (CPU, main memory, and disk capacity) based on published benchmarks based on the Standard SAP Application Benchmarks. The Quick Sizer performs the following types of sizing, which differ according to the kinds of information you must supply to the program:

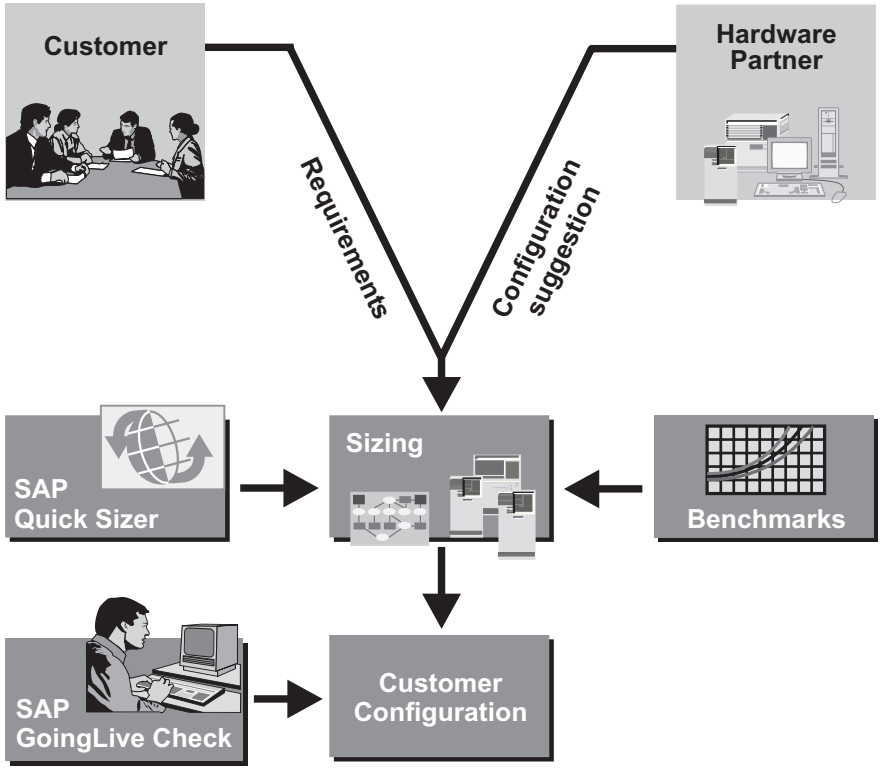



**Fig. 2.** The hardware sizing procedure involves the implementation project team, the hardware partner and SAP

- User-based sizing: The customer supplies the number of users in the various SAP modules.
- Document-based (quantity-structure based) sizing: The customer supplies the quantity structure-that is, the number of business documents required to be processed in particular time frames. These documents can include customer orders, deliveries, manufacturing orders, or printed documents. This type of sizing has the advantage that it reflects the data volume of background and interface processes, as well as the daytime distribution of document volumes.

Normally, the Quick Sizer uses a combination of both of the above types of sizing. However, either type of sizing is valid only when applied to standard SAP System software, as it cannot take into account system load caused by customer-developed ABAP programs.

## 3 Parallel Processing – Locking Constrains

To achieve the required throughput, processes should be parallelized. In order to do this, you need sufficient hardware and a suitable system configuration (see "Hardware

Requirements and System Configuration"). In addition, there are numerous points to keep in mind from a software perspective.

In an ERP system, many users can simultaneously cause reads of the same data in a database table. However, changes to database data must be restricted so that only one user can change a particular table row at one time. This is done by locking some or all of the table during each change operation. Within the SAP application layer, enqueues play a similar role to that of locks in the database layer.

To understand the role played by locks (database locks or SAP enqueues) in a common business process, consider a travel booking. The different components of the booking, such as flights, hotels, and bus or boat transfers, are all interdependent. The person making your booking uses the all or nothing principle: If you can't take the flight, you won't need the hotel room, and so on.

Since the availability of the different components is usually checked chronologically, you want to be certain that no other user makes a change to items in the sequence before the entire booking is completed. This is enabled by locks, which thus preserve data consistency.

Both database locks and SAP enqueues have the same ultimate purpose of preserving data consistency, but they are based on different technologies and used in different situations. They are closely related to transactions and logical units of work (LUWs).

The amount of time that a process holds an SAP enqueue or a database lock is termed the critical path. If other processes require these critical resources during the same time, they are denied and must wait. If a lock is not released quickly enough, this results in what is termed serialization, and restricts the degree of parallelization and therefore the maximum attainable throughput as shown in figure 3.

The following sections explain two exemplary lock problems, namely, the locks affecting availability checks and those affecting number range assignment.

### 3.1 ATP Check

A materials availability check is used in the R/3 Logistics modules to check the availability of materials required-for example, in sales orders, supply and distribution, or production. The particular availability check being discussed in this chapter is based on ATP (Available-to-Promise) logic.

**A Case Study.** A major PC manufacturer receives customer orders for custom-built PCs, and builds and delivers them accordingly using a "configure-to-order" scenario.

To guarantee the availability of materials required, a materials availability check is performed during sales orders creation. This availability check is based on Available-to-Promise (ATP) logic. During an availability check, system performance is affected by the following factors:

- Locks: The material whose availability is being checked must be locked by an enqueue. When the lock is in place, it may block other users who need to deal with the material, especially if the lock remains for a long time or the material is required frequently.

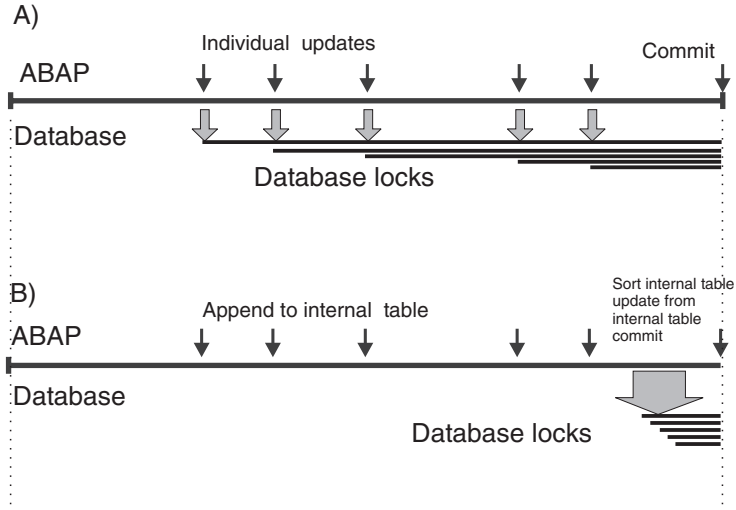



**Fig. 3.** Upper part (non-optimal lock-programming): If locks are set too early in a process step, the respective business object is locked in the database. Other processes that want to access this object have to wait for the release of the lock at the end of the process step (database commit or rollback). This locking situation leads to serialization and prevents the system from running processes in parallel. Lower Part: All necessary information are stored in (shared) main memory first. Database change operations that require locking are condensed to the very end of the process step thus leading to reduced lock times

- Read and calculation effort: An availability check is used to ensure that a material will be available at a specific time in the future. As part of the check, planned inward movements prior to that time are added to the current stock, while planned outward movements are subtracted. Since the tables containing this plan data can by very large, reading them and calculating availability can cause a high runtime for the availability check.

In the example of the PC manufacturer, assume that 36,000 availability checks are required per hour for sales enquiries, sales orders, and production planning. The manufacturer has 250 individual components that are affected by the availability check.

Assume that one particular component (material IC12345) is included in half of all assembled PCs. This means that the availability of material IC12345 must be checked 18,000 times per hour. Since each check locks the component, the lock may only be held very briefly, so that no wait situations result. Since there are 18,000 locks per hour, the maximum average lock time is 3,600 / 18,000 = 0.2 seconds. These considerations form the framework for implementing the availability check function.

**How the ATP Server Works**. As of SAP Release 4.0, the availability check may be conducted using a dedicated SAP instance, the ATP Server, which is an service running on an SAP System. It stores availability check data in a buffer in shared memory to dramatically reduce the number of accesses to the database tables that store the information on the material requirements and reservations.

An availability check with ATP Server uses a special SAP technique called locking with quantities. Instead of relying on exclusive enqueues, which allow only one user to lock a material, this technique uses shared enqueues - enqueues that can be concurrently set by several users on the same object. This allows several users to check the availability of a material at the same time.

Increasingly, companies implement multiple coupled SAP Systems. In these complex system landscapes, SAP's Advanced Planner & Optimizer (APO) Server enables company-wide (or even inter-company) materials management and inventory planning, as well as real-time decision support. The APO Server is an independent installation within the component-oriented Business Framework architecture. The performance of the APO Server is based on a series of new technologies and functions-for example, memory-based, LiveCache processing for data objects, which enables data sharing between application servers and therefore the implementation of task-specific application servers. APO is offered as an independent installation. The APO Server provides many forms of cross-system support, such as the availability check function. The APO Server works by providing quick, multi-staged checks on product and resource availability, and considers customer or plant preferences as well as approved product substitutions. It provides real-time results and availability simulations, the basis for a new performance dimension in the control and use of company-wide resources.

### 3.2 Database Locks on Number Range Objects

To use or change the data of business objects in an ERP system, you need to be able to access individual database records. To enable accesses that are so specific, each new record must receive a unique key as it is created.

The main part of this key is a serial number assigned using the system of number ranges. Examples of these serial numbers include order numbers or material master numbers. SAP number range management is based on a database table and ensures that previously assigned numbers are not issued a second time.

The current number level of a number range is the number that is to be assigned to the next new document. A database table stores the current number level, as well as the name of each number range-for example, MATDOCUMENT.

Legal and business requirements make it imperative that the same number is never assigned twice. In addition, in certain cases - for example, for accounting documents - number assignment must constitute a series without gaps. This means that, between the highest and lowest numbers of the series, there are no numbers that are not assigned.

Both of these requirements are realized through the following procedure. If a program needs a number for a new document-for example, from the number range MATDOCUMENT -it proceeds as follows:

1. The program locks the number range MATDOCUMENT and reads the current number level from the database table that holds the current number levels. To set the lock, the SQL statement SELECT FOR UPDATE is applied to the line of table corresponding to the number range.
2. The program increases the number range level by one, by updating database table accordingly.

3. The number range in the database remains locked until the program completes its database logical unit of work (LUW) by performing a commit or a database rollback. If an error occurs before the lock is released, the document cannot be created, and the change in the table is rolled back-that is, the previous number level is returned. This ensures that numbers are assigned chronologically and without gaps.

Bottlenecks occur when many numbers are requested from a particular number range in a short period of time. Since the number range is locked in the database from the time of the initial reading of the current number level to the time of the database commit, all business processes competing for number assignment must wait their turn, and this limits transaction throughput.

**A Case Study.** In a SAP Retail System, the cash register data of retail stores is centrally collected through the POS (Point of Sales) interface. There is a daily volume of around 500,000 items to be processed in an available time frame of 3 hours.

If we assume a processing time of 500 milliseconds per item, this results in a total processing time of 500,000 x 500 milliseconds = 250000 sec = 69 hours. In order to perform the same amount of work in 3 hours, 25 parallel processes are required.

**How Number Range Buffering Works.** Two solutions to this lock problem are provided by SAP.

If it is *not* necessary that numbers are assigned in a sequence, number ranges can be buffered in main memory. When an SAP system is shut down, the remaining numbers in the buffer-that is, the numbers that have yet to be assigned-are lost. This causes a gap in number assignment. As a result of the separate buffering of numbers in the various SAP instances, the sequence in which numbers are assigned across the entire SAP System is not numerical. In other words, a document with a higher number may have been created before a document with a lower number.

If it is necessary that numbers are assigned in a sequence, the following solution is possible: Instead of managing the number range for a particular type of document centrally in a single row of the number range database table, number intervals are selected from the table and managed in several rows with different primary keys. It is necessary to configure as many buffers as processes run in parallel. This procedure ensures that numbers are assigned in a sequence.

## 4 Skills, Methods, and Service Strategies in SAP Performance Management

This final section of this chapter concerns the tools and techniques in daily system operation that are needed to ensure optimal performance for an SAP system landscape. It explains the monitoring infrastructure delivered with the SAP Basis System, as well as important SAP offerings in the area of service and training.

**Continuous System Monitoring.** On-going system monitoring ensures that all components are available and function with a high level of performance. If components are unavailable or performance is poor, an alarm is triggered. Continuous monitoring can be automated using alert monitors, but may also involve periodic manual monitoring.

Included in the SAP Basis System, there is a full range of administration and monitoring tools. To perform continuous system monitoring, a central alerting monitor SAP's Computer Center Management System (CCMS) allows you to define one SAP System from the system landscape as the central monitoring system from which to monitor all sub-components: Databases, SAP instances, ITS, Web server, and so on.

**Error and Bottleneck Analysis.** If continuous system monitoring or a user in the system detects errors, unsatisfactory response times, or throughput problems, the system administrator or consultant has to perform an error analysis or bottleneck analysis. Analysis tools are part of SAP's Computer Center Management System (CCMS) that is shipped with the SAP System. [1]

**IT Reporting.** Periodically (weekly or monthly), the system administrator or consultant should produce an IT Report that covers the most important performance indicators, such as system downtimes, logged-on users, response times, hardware load, database capacity utilization, and statistics related to error messages.

To simplify and standardize IT reporting, SAP offers the EarlyWatch Alert Service free of charge. When using this service, the SAP System is configured to send a package of system performance data once weekly to SAP, where a standardized Service Report is produced that can be used as the basis of the IT reporting. At the same time, this keeps SAP informed about the performance of the customer's systems, and enables SAP's Service & Support Organization to provide recommendations for avoiding and solving problems.

**Performance Forum.** For large SAP implementation projects, it is necessary to set up a performance forum to enable regular meetings between people who represent the various aspects of performance optimization.

The long-term plan for performance optimization should include a regular performance analysis, based on workload analysis, to identify bottlenecks and their causes. This regular analysis is normally performed by people with experience in technical operation, who also know which functions in business processes are performance-critical. They should make a list of the transactions and programs that require special monitoring, and use this list to steer the analysis in the right direction

If you do not attempt to steer the optimization process, you run the risk that your optimization team will improve the performance of "exotic" controlling programs (rather than time-critical business processes), because the employees in these areas complain louder than the rest. In practice, it is when bottlenecks occur, such as during order entry for sales, that business is disrupted. These types of bottlenecks should be analyzed as a top priority.

Technical and application-oriented considerations should lead to a defined and prioritized monitoring strategy. This is how you can ensure that the optimization measures you take are exactly the ones that will ultimately work.

## References

1. Schneider, T.: SAP R/3 Performance Optimization. Sybex, San Francisco (1998)
2. For more information about SAP benchmarks and sizing methods, you can check the homepage of SAP http://sap.com/solutions/technology/. Then choose “Implementation”, “Sizing”
3. To view all certified benchmarks online, you can check the homepage of ideas international (http://www.ideasinternational.com/). The company provides up to date information about benchmarks
4. Derek P.: ERP Server Sizing – Getting It Right. Gardner Group, Stamford, CT (2000)

# Author Index