THE EVOLUTION OF THE RESEARCH QUEUEING PACKAGE

Charles H. Sauer

IBM Entry Systems Division
Austin, Texas 78758

Edward A. MacNair

IBM Thomas J. Watson Research Center
Yorktown Heights, New York 10598

Queueing networks are important as performance models of systems where performance is principally affected by contention for resources. The Research Queueing Package (RESQ) is a well known software package for definition and solution of queueing network models. This paper examines the history of the several versions of RESQ and why RESQ has evolved as it has. Successes, failures and relationships to other research and development efforts are described. The paper assumes the reader has some familiarity with queueing network models and queueing network software.

1. INTRODUCTION

1.1. Queueing Network Models

Queueing models have been used for decades in studying the performance of manufacturing lines, communication networks and similar systems. In the 1960's there were dramatic insights into queueing network models of communication networks and computer systems (particularly those of Kleinrock [24] and Buzen [9]) and product form solutions of queueing networks (particularly those of Jackson [22] and Buzen [9]). In turn, in the 1970's and early 1980's there has been substantial additional progress in the understanding and application of queueing network models. This paper assumes the reader is sufficiently familiar with this work that a review is not necessary. For general discussion of queueing network models, see the recent textbooks in the area (e.g., Sauer and Chandy [48], Lavenberg *et al* [27], Lazowska *et al* [30]) and the queueing network issues of *Computing Surveys* (September 1978) and *Computer* (April 1980).

1.2. Queueing Network Software

For queueing network models to be used effectively, appropriate software is necessary to construct models and to obtain solutions for models. This paper traces the evolution and design issues in one of the earliest and most influential packages of queueing network software, the Research Queueing Package (RESQ). An appendix shows a frequently used example model constructed and solved using the current version of RESQ. The bibliography at the end of the paper includes most of the previous papers and technical reports on RESQ, except for those restricted to IBM internal use. Included in the bibliography are two prior retrospectives [51,54], a monograph [53], user manuals [55,56,57], and a product availability notice [52].

The product version of RESQ is available to IBM customers in the U.S., Puerto Rico, Europe, the Middle East and Africa. Some of the RESQ characteristics described in this paper are considered experimental and are not included in the product version of RESQ. Similarly, this paper speculates about future development of RESQ; this should not be taken as indication of future additions to the product version of RESQ.

There are many other packages of queueing network software. The bibliography includes references for most of the well known packages, but does not attempt to include references for all of the packages that have been developed. A few of these are discussed to the extent they relate to RESQ.

1.3. Significance of RESQ

RESQ success and importance is due to the multiple roles it plays and the synergism between these roles.

- The primary role of RESQ is as a modeling package for practical performance evaluation work. RESQ has been used throughout IBM on a wide variety of modeling problems, including not only computer/communication system modeling, but also manufacturing [35], emergency building evacuation and others. RESQ is currently installed on over 130 IBM internal computing systems.

- RESQ is a flexible and convenient tool for research in performance evaluation methodology. Areas covered in this research include simulation methodology, e.g., output analysis, stopping rules, etc., and approximate solution methods, e.g., development of new methods, empirical validation of methods, etc.

- RESQ is an effective vehicle for making available new results in performance methodology, both those developed using RESQ, e.g., in the areas just cited, and those developed without using RESQ, e.g., new algorithms for exact solution of queueing networks.

- The RESQ concepts themselves are research topics, e.g., the appropriate definition of queueing network extensions for effective modeling of general classes of systems, the development of languages for expressing definitions of queueing networks, etc.

All of these roles will be discussed further in the remainder of the paper, as appropriate. The remainder of the paper is largely chronological, first addressing work prior to the first RESQ prototype, then the prototype, the first widely used version, the current version and future possibilities.

2. PRE-RESQ SOFTWARE AND CONCEPTS

2.1. ASQ, QAL, QSIM, APLOMB (and the Origin of the Passive Queue)

Keller developed the ASQ (Arithmetic Solution of Queues) package [23] based upon Buzen's algorithms for queueing networks with product form solutions [9] and Chandy's generalization of the class of networks with product form solution [11]. ASQ provided an interactive dialogue for definition and solution of networks, making it possible to produce results with much less effort than with conventional simulation languages. ASQ was a great inspiration to Sauer and other students at the University of Texas at Austin, for it both demonstrated the power of a such a tool and pointed out the opportunity to develop more powerful tools, particularly for networks without product form solutions.

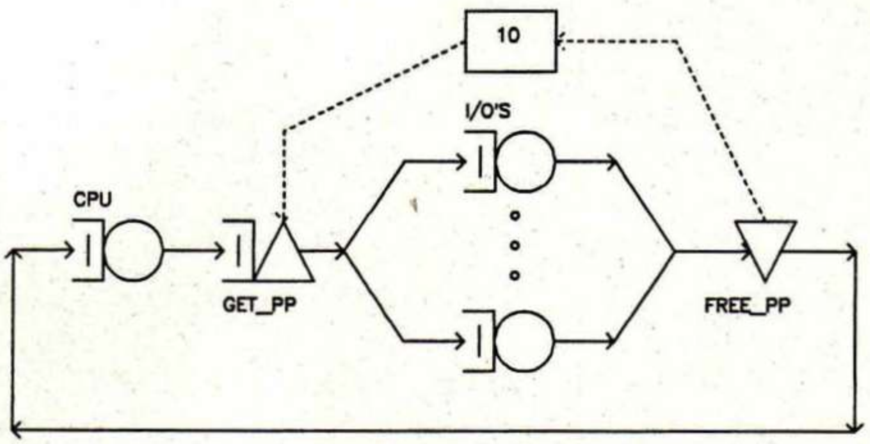

Figure 1. Central Server Model with Peripheral Processors

One of the primary limitations of product form networks was (and is) that a job could only hold one resource at a time. As part of a planned dissertation, F. Palacios-Gomez was seeking the solution of the model shown in Figure 1. The model was intended to represent the CDC 6600 in use at the University of Texas. The model was based on Buzen's central server model [9], but included representation of acquisition of one of the 6600's peripheral processors to perform I/O, and the subsequent release of that processor. (The 6600 CPU cannot perform I/O and is dependent on the peripheral processors for non-computational functions.)

Foster, Sauer and Waggoner generalized the construct used in the Palacios-Gomez model, and termed it a "passive server." (Passive as opposed to the "active" servers of traditional queueing models.) The passive server was a key concept in QAL (Queueing Analysis Language) which they proposed as a general language, analogous to a programming language, for specification of queueing models [19]. Though the definition of passive servers had flaws, the concept would be refined and, with the name "passive queue," become a key aspect of the success of RESQ.

QAL was intended to be implemented with a variety of solution techniques, but the only implementation was the QSIM simulation program of McGehearty [19]. There were three principal problems with QAL. First, there was little direct consideration given to non-simulation implementations and even simulation implementation was difficult. Second, QAL provided little support for representing distinct classes of jobs. Third, because QAL was designed as a language and not an interactive dialogue it required greater understanding on the part of the user.

While still at the University of Texas, Sauer began development of the simulation program APLOMB [42]. This program was initially intended to be a vehicle for empirical validation of approximate solution methods for queueing networks, not an interesting work in its own right. APLOMB provided the regenerative method for estimation of confidence intervals [17] and a sequential stopping rule to determine run lengths [29]. (The name was chosen because the program handled a difficult problem, simulation output analysis, with aplomb.)

After joining IBM, Sauer continued to develop APLOMB, with the intent that it be used for hybrid simulation of SNA networks [61]. New constructs, similar to those of QAL, were added to the program, but the construct definitions were chosen much more carefully than with QAL. Special care was taken to allow efficient implementation, to allow the use of the regenerative method and to provide more unified definitions [43]. For example, the QAL definition of passive servers considered two types of passive servers, reusable and consumable, which were combined into a single, more useful definition.

This implementation of APLOMB was in Fortran 66. There was no user interface as such, but rather a model would be defined by assigning values to variables and calling subroutines. Model simulation was effected by calling an event handling subroutine, and model results were obtained from program variables after return from that routine.

2.2. *IQNA*

One of the other fundamental limitations of product form queueing networks is that FCFS queues must have exponential service time distributions. Chandy, Herzog and Woo developed a general iterative approach for approximate solution of networks with general service time distributions at FCFS queues [13]. Woo and MacNair refined this approach for solution of networks with several types of jobs and with priority scheduling. They implemented this approach in a PL/I program, IQNA (Iterative Queueing Network Analysis).

2.3. *QNET4*

Reiser and Kobayashi developed a computational algorithm similar to Buzen's which handled a more general class of product form networks, including mixtures of open and closed routing chains [39]. The QNET4 program of Reiser [37] incorporated this algorithm. QNET4 was implemented in APL and provided interactive dialogues for definition, solution, listing and modification of models. The user interface of QNET4 would become the basis for the initial RESQ user interface.

Another significant aspect of QNET4, especially from a RESQ perspective, was that QNET4 broke from the tradition of previous software and literature which considered jobs belonging to classes which are "global" in the sense that jobs remain in the same class when they move from queue to queue unless they explicitly change class. QNET4 introduced the concept of local classes explicitly partitioned into routing chains; a queue has one or more local classes for each routing chain of jobs which may visit the queue. Jobs change class each time they leave a queue but never change from one chain to another. (Global classes must also be partitioned into routing chains for non-simulation solutions. This is often overlooked.) The two representations of job classes are of equivalent generality. However, the local class representation is often more convenient for the user and simplifies implementation of solution methods [27,48].

Unlike APLOMB and IQNA, QNET4 was distributed to IBM locations outside of Yorktown and became popular among IBM performance analysts. However, it was not able to handle the general network characteristics considered by the other two programs.

3. THE PROTOTYPE VERSION OF RESQ

By late 1975, it was quite evident that APLOMB needed a convenient user interface and that QNET4 needed to be able to deal with queueing networks which violated product form conditions. Since QNET4 had a significant IBM user community, it was natural to extend the QNET4 model definition dialogue to support the generalizations supported by IQNA and to support the new constructs in APLOMB, e.g., passive queues.

Initially, the objective was to develop a prototype system quickly. The existing code of APLOMB, IQNA and QNET4 was used essentially intact. Since these three programs used three different languages, Fortran, PL/I and APL, respectively, communication between the dialogues in APL and the APLOMB/IQNA solution portions was through files. Using the CMS stack facilities of VM/370, it was possible to give the appearance of a single program.

Besides the obvious problems that result when programs are patched together this way, there were two main problems with this prototype. First, the code was more installation dependent than anticipated, and it was troublesome to move the object code to other installations. Second, and more significant, was the implicit assumption that the user would construct small models, typically on the order of ten queues. There were several unfortunate consequences of this assumption, some of which were not rectified until the second version of RESQ. For example,

- On basis of principle, it was decided that the model definition dialogue should not include solution method specific characteristics. As part of the model solution dialogue, when using simulation, the user specified characteristics such as initial state, regeneration state and run length criteria. Though marginally acceptable for small models, this made it unnecessarily tedious to solve larger models for a range of parameter values.

- Rather than identify elements symbolically, the elements (e.g., queues, classes, allocate/release nodes for passive queues) were identified numerically, and the user had to specify the numbers of each type of element at the beginning of the model definition dialogue.

- There was no "batch" mode of model definition and solution -- the user had to use the interactive dialogues.

- There was no facility for macro definition of submodels.

Some of these problems could be alleviated by using the procedure level interface to the solution packages, but that approach does not provide the convenience such a tool should provide, nor does it solve all of the above problems.

In addition to these general problems, there were problems with the underlying technology of each of the solution programs. QNET4 suffered from the now well known numerical problems of the "convolution" algorithm [48]. IQNA used heuristic methods which failed entirely on some problems. The regenerative method used in APLOMB for confidence interval analysis required considerable sophistication to be used for general models.

In spite of these problems, the prototype was useful both for practical modeling problems and for research in modeling methodology. For example, several empirical studies in simulation output analysis were conducted using the prototype [28,29]. For a more thorough overview of the prototype version, see Reiser and Sauer [41] and Sauer, Reiser and MacNair [60].

4. RESQ1

4.1. Implementation and Support

The prototype was considered a success, but did not realize the potential of such software because of the above problems. Late in 1976, work began on what became known as RESQ1, a well integrated implementation of essentially the same function as the prototype. From an implementor's view, the natural implementation language was PL/I. There already existed a PL/I version of QNET4, IQNA was written in PL/I, and the dialogue components could be fairly mechanically rewritten in PL/I.

However, there were two formidable problems, the effort required to re-implement APLOMB in PL/I, and the insistence on support in APL from many users and potential users. The first problem was addressed by constructing a translator in SNOBOL to translate Fortran, as used in APLOMB, to PL/I. The elapsed time between beginning work on the translator and getting a running PL/I version of APLOMB was approximately two weeks, and this achievement was a great relief to those who anticipated a much, much larger effort. The second problem was addressed by redundancy: though the complete system was implemented in PL/I, the user interface was also implemented in APL.

A RESQ Workshop was held in Yorktown in the Spring of 1977, to announce the availability (within IBM) of RESQ1. This initial Workshop began an annual tradition of bringing together both experienced and potential users for tutorials, presentations on applications and exchanges on planned development and suggested improvements. A periodic RESQ Newsletter was also initiated. Besides the capabilities of the package itself, a major factor in the ensuing popularity of RESQ1 within IBM was the consultation and other support made available to users.

RESQ1 came into use at IBM laboratories throughout the world. Users began to recognize that RESQ was applicable to a wide variety of modeling problems, not just computer and communication system modeling. However, usage of the non-simulation components of RESQ became a small part of the overall usage; the dominance of the simulation mode continues today.

Most of the reports dealing with RESQ1 remain IBM proprietary. In addition to the previously cited papers dealing with the prototype, a report by Sauer and MacNair [49] gives an extensive set of examples using RESQ1.

4.2. Problems Solved and Unsolved

The simulation specific information was made part of the model definition dialogue, and many minor improvements and extensions were made, but RESQ1 still suffered from most of the "thinking small" problems of the prototype. The interactive dialogues were extended to allow suspension and later resumption of dialogue. The model definition component was modified to allow file input as well as terminal input, in the form of "dialogue files." But it remained awkward, at best, to construct large simulation models. (This is not to say that large models were not constructed -- some models were constructed with hundreds of queues, and even larger models were constructed using the procedural level interface.) In addition to the regenerative method for confidence intervals, the classical method of independent replications was added to the program in a "scaffolded" manner.

5. RESQ2

5.1. The Implementation Plan

With the evident inherent limitations of the RESQ1 programs, there was the natural inclination to begin anew with the design and implementation process. However, this inclination was tempered by reality: First, it was necessary to provide an easy migration path for users of the existing programs. So the user interface could not be changed radically, and there had to be a way to convert an existing model to use a new implementation. Second, as a Research project, simply reimplementing to "do it right" could not be justified. Rather, substantially new value had to be provided to justify a major implementation effort.

In the spring of 1977, a comprehensive plan for substantial additional development was completed. The key aspect of this plan was that a new, compiler-like translator was to be constructed to support the user interface.

- The language to be accepted by the translator looked much like a transcript of an interactive dialogue in RESQ1. Thus learning the language would be easy for a RESQ1 user.

- Though the system would be oriented toward "batch" translation of model definitions, the same translator would serve as an interactive prompter.

- "Templates," would allow macro definition of both queues and entire subnetworks. (Queue templates are called "queue types" and subnetwork templates are called "submodels.") Libraries of templates would be supported. In addition to the translator, a macro expansion processor would be implemented as a "front end" of the solution system.

- Symbolic identification of network elements would be fully supported.

- Arrays of network elements, including macro invocations of submodels, would be supported.

- General expressions to be evaluated during simulation would be supported. (Previously, a limited set of dynamically evaluated expressions had been supported.)

- The definitions of network elements were to be generalized significantly. In particular, considerable flexibility was to be added to the passive queue, as discussed below. Fission nodes, which allow a job to create related jobs, and fusion nodes, which allow related jobs to reunite as one, would be extended to allow multiple generations of related jobs.

- In addition to normal macro invocation of submodels, "substitutions," would be supported. Substitutions would allow heuristic, hierarchical solutions of models with the potential of greatly reduced computational requirements.

- IQNA would no longer be supported because of low usage and because it would be largely superceded by the substitution support.

- New numerical solution methods, e.g., Mean Value Analysis [40], would be supported.

- The method of independent replications would be added as an integral part of the simulation support, and other methods would be added as appropriate.

Sauer, MacNair and Salza [59] describes the strategy in more detail.

5.2. The Implementation of RESQ2

With one exception, the above plan was followed closely, resulting in Version 2 of RESQ. Empirical evidence [5] indicates that the substitution support would be of considerably less practical value than anticipated. Thus there is no longer a plan to support substitutions in RESQ.

A batch only version of the translator and an initial version of the macro expansion processor were ready in the spring of 1980, along with many of the planned extensions to the simulation component. The integrated batch/interactive version of the translator was first available that summer. Incremental development has continued since then.

Having the same translator capable of both "batch" and interactive modes has been remarkably useful in model construction because (1) in interactive mode, it is possible to immediately make revisions or corrections to prior dialogue by escaping to an editor to revise a transcript of the dialogue so far (a dialogue file) and to then continue in prompting mode after the (incomplete, edited) dialogue file has been reparsed and (2) revision of an existing model is possible in mixed mode by deleting portions of the existing dialogue file and using interactive mode for specification of revisions or additions. This mixed mode capability provides the "user friendliness" of interactive mode without losing the flexibility and efficiency of "batch" mode for development of significant models. More recently, a full screen, menu oriented mode has been added.

5.3. Passive Queue Extensions

Typically, passive queues are used for convenient representation of simultaneous resource possession. A job typically acquires tokens of a passive queue and holds on to them while visiting other queues (active and/or passive queues) and model elements. The job explicitly releases or destroys its tokens when it no longer needs them. A second major use of passive queues is to model mechanisms such as communication protocols and protocols for channel-device interaction. (Such usage may involve other RESQ elements. See Sauer and MacNair [53] for examples.)

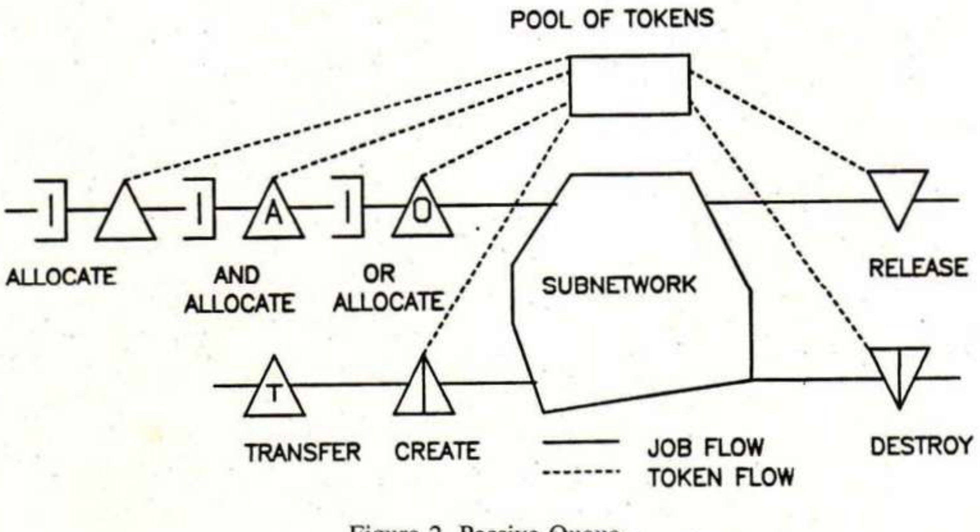

Figure 2. Passive Queue

The RESQ design assumes the tokens of a passive queue to be homogeneous. This is consistent with most applications and simplifies implementation. However, there are situations where the effect of heterogeneous tokens is desirable. (PAWS supports heterogeneous tokens explicitly [34].) The desired effect is obtained in RESQ2 through AND and OR allocate nodes. An AND allocate node indicates a job must receive simultaneous allocation from several different queues, and an OR allocate node indicates a job must receive allocation from any one of several passive queues.

RESQ2 also provides transfer nodes for passing tokens back and forth between related jobs (created by fission nodes). CHANGE allocate nodes allow a job to increase/decrease the number of tokens it holds from a given pool.

Support for preemptive priority scheduling at passive queues is a relatively recent addition to RESQ2. Appropriate definition of this discipline is difficult because of the implied side effects on other queues, both active and passive, when tokens are forcibly taken away from a job. The basic strategy is to put the job in a suspended state, not allowing it to move or receive resources, as soon as possible after the tokens are preempted.

5.4. Other RESQ2 Extensions

An implementation of Mean Value Analysis [40,62] has displaced QNET4 as the primary numerical solution component in RESQ2. An implementation of the Tree Convolution algorithm for product form networks [26] has also been added. The spectral method for confidence intervals [21] has been added for simulation analysis. Interactive simulation is now supported, in the sense that it is now convenient to continue a simulation run after examining results, either because one wants to see results at intermediate points in the run or because one is not satisfied with results at the planned run length or stopping condition.

6. FUTURE POSSIBILITIES

Incremental development of RESQ continues, but there are no clear plans for major enhancements of RESQ at this time. There are obvious possibilities worthy of consideration. One is to provide an interactive graphics interface. However, initial experience with a prototype [6] suggests that currently economical hardware is inadequate for the large models often constructed by RESQ users. Another possibility would be to develop a version designed for personal computers, now that these machines are approaching the capacities of the mainframes RESQ was initially developed for.

There is also the possibility of additional specialized, higher level modeling tools based on RESQ. One such tool, SNA/PET [4] already is in use and undergoing further development. Others, in areas such as manufacturing, seem reasonable possibilities.

ACKNOWLEDGEMENTS

M. Reiser and L.S. Woo made substantial contributions to the initial version of RESQ. J.F. Kurose and S. Salza helped define the syntax and semantics of the language used in the current version of RESQ and implemented the translator for that language. L. Berger and P. Loewner have made extensions in the model construction phase of RESQ. A. Blum, P. Heidelberger, E. Jaffe, K. Plochinski, P. Rosenfeld, S. Tucci and P.D. Welch all contributed to the definition and implementation of the numerical and simulation components of RESQ. K. Bharath-Kumar and P. Kermani have help spur further refinement of RESQ through their work on SNA/PET. We are indebted to numerous RESQ users for the many suggestions we have received, for the discussions that have helped crystallize our thinking and improve RESQ, and for the encouragement we have received in the development of this methodology and this tool. Finally, we would like to thank the many students at the IBM Systems Research Institute who have helped us in focusing our attention on the RESQ concepts.

BIBLIOGRAPHY

1. F. Baskett, K.M. Chandy, R.R. Muntz and F.G. Palacios, "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers," *JACM 22,* 2 (April 1975) pp. 248-260.
2. BGS Systems, "BEST/1 Product Description," BE77-010-2, Lincoln, Massachusetts, January 1977.
3. H. Beilner and J. Mater, "Simulation and Analytic Modelling of Computer System Performance Using the Software Tool COPE," *ECOMA 10 Conference Proceedings* (1982) pp. 179-183.
4. K. Bharath-Kumar and P. Kermani, "Performance Evaluation Tool (PET): An Analysis Tool for Computer Communications Networks," *IEEE Journal of Selected Areas in Communications,* (January 1984).
5. A. Blum, L. Donatiello, P. Heidelberger, S.S. Lavenberg and E.A. MacNair, "Experiments with Decomposition of Extended Queueing Network Models," IBM Research Report RC-10213, Yorktown Heights, New York (October 1983). To appear, *Proceedings International Conference on Modelling Techniques and Tools for Performance Analysis,* Paris, May 16-18, 1984.
6. A. Blum, E.A. MacNair and C.H. Sauer, "The Research Queueing Package: Graphics Developments," *Proceedings GI/NTG 83: Measurement, Modelling and Evaluation of Computer Systems,* University of Stuttgart (February 1983).
7. M. Booyens, P.S. Kritzinger, A. Krzesinski, P. Teunissen and S. Van Wyk, "SNAP: An Analytic Multiclass Queueing Network Analyzer," Technical Report ITR83-08-00 (September 1983) Institute for Applied Computer Science, University of Stellenbosch.
8. J.C. Browne, K.M. Chandy, R.M. Brown, T.W. Keller, D. Towsley and C.W. Dissley, "Hierarchical Techniques for Development of Realistic Models of Complex Computer Systems," *IEEE Proceedings 63,* 6 (June 1975) pp. 966-975.
9. J.P. Buzen, *Queueing Network Models of Multiprogramming,* Ph.D. Thesis, Harvard University, Cambridge, Mass. (1971). Garland Publishing, New York (1980).
10. J.P. Buzen, "Computational Algorithms for Closed Queueing Networks with Exponential Servers," *Communications of the ACM 16,* 9 (September 1973) pp. 527-531.
11. K.M. Chandy, "The Analysis and Solutions for General Queueing Networks," *Proc. Sixth Annual Princeton Conference on Information Sciences and Systems,* Princeton University (March 1972).
12. K.M. Chandy, U. Herzog and L.S. Woo, "Parametric Analysis of Queueing Networks," *IBM J. of Research and Development 19,* 1 (January 1975) pp. 43-49.
13. K.M. Chandy, U. Herzog, and L. Woo, "Approximate Analysis of General Queuing Networks," *IBM Journal of Research and Development 19,* (January 1975).
14. K.M. Chandy and D. Neuse, "Linearizer: A Heuristic Algorithm for Queueing Network Models of Computer Systems," *Communication of the ACM 25,* 2 (February 1982) pp. 126-133.
15. K.M. Chandy and C.H. Sauer, "Approximate Methods for Analyzing Queueing Network Models of Computer Systems," *ACM Computing Surveys 10,* 3 (September 1978) pp. 281-318.
16. M.A. Crane and D.L. Iglehart, "Simulating Stable Stochastic Systems Part II: Markov Chains," *JACM 21,* 1 (1974) pp. 114-123.
17. M.A. Crane and A.J. Lemoine, *An Introduction to the Regenerative Method for Simulation Analysis,* Lecture Notes in Control and Information Sciences, Vol. 4, Springer-Verlag, New York (1977).
18. G.S. Fishman, *Concepts and Methods in Discrete Event Digital Simulation,* Wiley, New York (1973).
19. D.V. Foster, P.F. McGehearty, C.H. Sauer and C.N. Waggoner, "A Language for Analysis of Queueing Models," *Proceedings Fifth Annual Pittsburgh Modeling and Simulation Conference,* University of Pittsburgh, April 1974.

20. G. Gordon, *The Application of GPSS V to Discrete System Simulation,* Prentice-Hall, Englewood Cliffs, N.J. (1975).

21. P. Heidelberger and P.D. Welch, "A Spectral Method for Confidence Interval Generation and Run Length Control in Simulations," *CACM 24,* 4 (April 1981) pp. 233-245.

22. J. R. Jackson, "Jobshop-Like Queueing Systems," *Management Science 10,* 1 (October 1963) pp. 131-142.

23. T.W. Keller, "ASQ Manual," Dept. of Computer Sciences Report TR-27, University of Texas, Austin, Tx., 1973.

24. L. Kleinrock, *Communication Nets: Stochastic Message Flow and Delay,* McGraw-Hill, New York (1964). Reprinted, Dover Publications (1972).

25. P.J. Kiviat, R. Villanueva and H. Markowitz, *The SIMSCRIPT II Programming Language,* Prentice-Hall, Englewood Cliffs, N.J. (1969).

26. S.S. Lam and Y.L. Lien, "A Tree Convolution Algorithm for the Solution of Queueing Networks," *CACM 26,* 3 (March 1983) pp. 203-215.

27. S.S. Lavenberg, editor, *Computer Performance Modeling Handbook,* Academic Press, Inc., New York (1983).

28. S.S. Lavenberg, T.L. Moeller and C.H. Sauer, "Concomitant Control Variates Applied to the Regenerative Simulation of Queuing Systems," *Operations Research 27,* 1, January-February 1979.

29. S.S. Lavenberg and C.H. Sauer, "Sequential Stopping Rules for the Regenerative Method of Simulation," *IBM J. of Research and Development 21,* 6 (1977) pp. 545-556.

30. E.D. Lazowska, J. Zahorjan, G.S. Graham and K.C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models,* Prentice-Hall, (1984).

31. E.A. MacNair and C.H. Sauer, "The Research Queueing Package: A Primer," *Proceedings of SHARE60,* (February 1983) pp. 29-37.

32. J. McKenna, D. Mitra and K.G. Ramakrishnan, "A Class of Closed Markovian Queuing Networks: Integral Representations, Asymptotic Expansions, and Generalizations," *The Bell System Technical Journal 60,* 5 (May-June 1981) pp. 599-641.

33. D. Merle, D. Potier and M. Veran, "A Tool for Computer System Performance Analysis," *Performance of Computer Installations,* Ferrari, D. (editor), North-Holland (1978) pp. 195-213.

34. D. Neuse, K.M. Chandy, J. Misra and R. Berry, "Simulation Tools in Performance Evaluation," *CPEUG 81,* (Computer Performance Evaluation Users Group), San Antonio, Texas (November 1981) pp. 331-334.

35. W.J. Oates, "Automated Manufacturing Analysis Using RESQ," IBM Technical Report TR 08.152, Lexington, Kentucy (September 1982).

36. Quantitative System Performance, "MAP User Guide," (1983).

37. M. Reiser, "QNET4 User's Guide," IBM Research Report RA-71, Yorktown Heights, New York (1975).

38. M. Reiser, "Interactive Modeling of Computer Systems," *IBM Systems Journal 15,* 4 (1976) pp. 309-327.

39. M. Reiser and H. Kobayashi, "Queueing Networks with Multiple Closed Chains: Theory and Computational Algorithms," *IBM J. of Research and Development 19,* 3 (May 1975).

40. M. Reiser and S.S. Lavenberg, "Mean Value Analysis of Closed Multichain Queueing Networks," *JACM 27,* 2 (April 1980) pp. 313-322.

41. M. Reiser and C.H. Sauer, "Queueing Network Models: Methods of Solution and their Program Implementation," in K.M. Chandy and R.T. Yeh, editors, *Current Trends in Programming Methodology, Volume III: Software Modeling and Its Impact on Performance,* Prentice-Hall (1978) pp. 115-167.

42. C.H. Sauer, "Simulation of Generalized Queueing Networks," *Proceedings 1975 Summer Computer Simulation Conference,* San Francisco, July 1975.

43. C.H. Sauer, "Characterization and Simulation of Generalized Queueing Networks," IBM Research Report RC-6057, Yorktown Heights, New York, May 1976.

44. C.H. Sauer, "Passive Queue Models of Computer Networks," *Computer Networking Symposium,* Gaithersburg, Maryland (December 1978). IEEE Catalog No. 78CH1400-1.

45. C.H. Sauer, "Computational Algorithms for State-Dependent Queueing Networks," *ACM Transactions on Computer Systems 1,* 1 (February 1983) pp. 67-92.

46. C.H. Sauer, "Approximate Solution of Queueing Networks with Simultaneous Resource Possession," *IBM J. of Research and Development 25* (1981) pp.894-903.

47. C.H. Sauer and K.M. Chandy, "Approximate Solution of Queueing Models," *IEEE Computer 13,* 4 (April 1980) pp. 25-32.

48. C.H. Sauer and K.M. Chandy, *Computer Systems Performance Modeling,* Prentice-Hall, Englewood Cliffs, NJ (1981).

49. C.H. Sauer and E.A. MacNair, "Computer/Communication System Modeling with Extended Queueing Networks," IBM Research Report RC-6654, Yorktown Heights, New York (July 1977).

50. C.H. Sauer and E.A. MacNair, "Simultaneous Resource Possession in Queueing Models of Computers," *Performance Evaluation Review 7,* 1 and 2 (1978), pp. 41-52.

51. C.H. Sauer and E.A. MacNair, "Queueing Network Software for Systems Modeling," *Software-Practice and Experience 9,* 5 (May 1979) pp. 369-380.

52. C. H. Sauer and E.A. MacNair, "The Research Queueing Package Version 2: Availability Notice," IBM Research Report RA-144, Yorktown Heights, New York (August 1982).

53. C.H. Sauer and E.A. MacNair, *Simulation of Computer Communication Systems,* Prentice-Hall, Englewood Cliffs, NJ (1983).

54. C.H. Sauer, E.A. MacNair and J.F. Kurose, "The Research Queueing Package: Past, Present and Future," *Proceedings 1982 National Computer Conference* (1982) pp. 273-280.

55. C.H. Sauer, E.A. MacNair and J.F. Kurose, "The Research Queueing Package Version 2: Introduction and Examples," IBM Research Report RA-138, Yorktown Heights, New York (April 1982).

56. C.H. Sauer, E.A. MacNair and J.F. Kurose, "The Research Queueing Package Version 2: CMS Users Guide," IBM Research Report RA-139, Yorktown Heights, New York (April 1982).

57. C.H. Sauer, E.A. MacNair and J.F. Kurose, "The Research Queueing Package Version 2: TSO Users Guide," IBM Research Report RA-140, Yorktown Heights, New York (April 1982).

58. C.H. Sauer, E.A. MacNair and J.F. Kurose, "Queueing Network Simulations of Computer Communication," *IEEE Transactions on Communications COM-32,* 1 (January 1984).

59. C.H. Sauer, E.A. MacNair and S. Salza, "A Language for Extended Queueing Networks," *IBM J. of Research and Development 24,* 6 (November 1980) pp. 747-755.

60. C.H. Sauer, M. Reiser and E.A. MacNair, "RESQ - A Package for Solution of Generalized Queueing Networks," *Proceedings 1977 National Computer Conference* (1977) pp. 977-986.

61. C.H. Sauer, L. Woo and W. Chang, "Hybrid Analysis/Simulation: Distributed Networks," IBM Research Report RC-6341, Yorktown Heights, New York, June 1976.

62. S. Tucci and E.A. MacNair, "Implementation of Mean Value Analysis for Open, Closed and Mixed Queueing Networks," *Computer Performance 3,* 4 (December 1982) pp. 233-239.

APPENDIX

Effective use of RESQ is based on constructing diagrams representing queueing network models. Figures A.1 and A.2 depict a hypothetical computer system model. (This network is similar to networks used as computer system models since the mid sixties.)

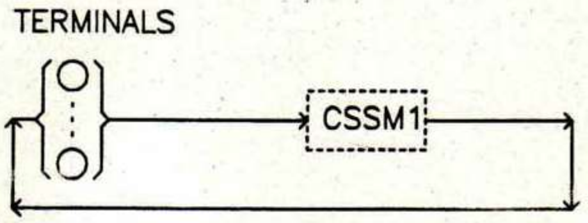

Figure A.1. - Terminals and Submodel

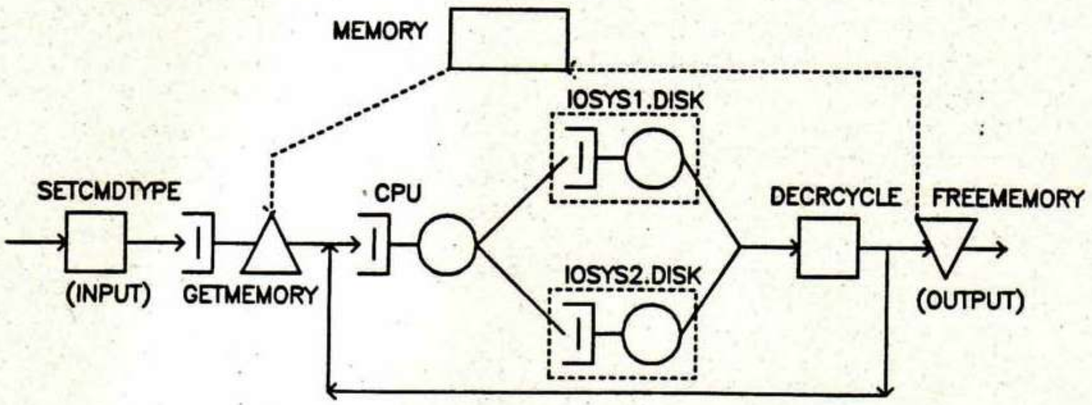

Figure A.2. - Computer System Submodel

The SETUP command invokes the RESQ prompter/translator for definition or revision of a model. If a RESQ user realizes a semantic error was made in some previous portion of the dialogue, he or she may temporarily suspend the dialogue, correct the error with an editor and then resume the dialogue at the point of suspension. A transcript (a "dialogue file") of a model definition dialogue is kept for the user by the translator. The user may edit this transcript and then have it translated again, with or without additional interactive dialogue.

We will now give an example of a possible SETUP dialogue for the model represented by Figures A.1 and A.2, assuming RESQ is used with CMS. As we present the dialogue we will make arbitrary assumptions about system characteristics previously left unspecified. The example is presented as if a scrolling terminal is used, to simplify formatting of this document. In our examples, upper case characters will correspond to prompts from RESQ components and lower case will generally be used for replies from the user.

```
setup
MODEL:csm /*model name - Computer System Model*/
RESQ2 Translator V2.04 (01/19/82)  Time: 13:56:12  Date: 01/29/82
MODEL IS CSM
   METHOD:simulation
   NUMERIC PARAMETERS:thinktime users
   NUMERIC PARAMETERS:
   NUMERIC IDENTIFIERS:userframes /* a constant */
      USERFRAMES:50
   NUMERIC IDENTIFIERS:
```

Each RESQ job has a vector, JV, used to store job specific information. In our example system we assume that there are three types of commands which may be issued by terminal users. JV(1) will be used to store the command type, and JV(2) will be used to count the number of CPU-I/O cycles for a particular command.

```
   MAX JV:how  /* request for help message */
ENTER AN ARITHMETIC EXPRESSION FOR THE EXTENT OF THE JV VECTOR
   MAX JV:2 /*1: command type, 2: cycle count*/
```

The second major section of dialogue is for definition of queues. First we may define a "queue type", a macro definition of a queue dialogue. We indicate here that we choose not to define a queue type by giving a null reply. We will illustrate definition and invocation of a user defined queue type later in the dialogue.

```
   QUEUE TYPE:
```

Next we define individual queues.

```
   QUEUE:terminalsq
      TYPE:is  /*Infinite Server*/
      CLASS LIST:terminals
         SERVICE TIMES:thinktime
      CLASS LIST:
   QUEUE:
```

The third major section of dialogue is for definition of additional nodes not belonging to queues. "Nodes" in RESQ are functional elements in the routing, including the class just defined for the terminals. No more nodes appear outside of the submodel of Figure A.1, so we give null replies to the prompts for names of these nodes.

```
   SET NODES:
   FISSION NODES:
   FUSION NODES:
```

The fourth major section of dialogue is for definition of submodels. The submodel definition dialogue closely parallels the dialogue for model definition, including subsections corresponding to those sections we have already seen. There will also be a routing subsection corresponding to the model routing section which follows submodel definition and invocation.

```
   SUBMODEL:cssm /*Computer System Submodel*/
      NUMERIC PARAMETERS:pageframes
      NUMERIC PARAMETERS:
      NODE PARAMETERS:
```

Routing "chains" are used to define the routing among nodes of a network. A submodel must have at least one chain parameter in order to connect the nodes inside of the subnetwork with nodes outside of the subnetwork.

```
CHAIN PARAMETERS:interactiv
CHAIN PARAMETERS:
NUMERIC IDENTIFIERS:cmdtype cyclecount
   CMDTYPE:1 /*JV(1) to be used to indicate command type*/
   CYCLECOUNT:2 /*JV(2) to be used to count CPU-I/O cycles*/
NUMERIC IDENTIFIERS:cpiocycles(3) pageneed(3)
   CPIOCYCLES: 8 15 50
   PAGENEED:  20 24 30
NUMERIC IDENTIFIERS:cputime
   CPUTIME:.025 /*mean time in seconds*/
NUMERIC IDENTIFIERS:
QUEUE TYPE:
QUEUE:memory
   TYPE:passive
   TOKENS:pageframes
   DSPL:fcfs
   ALLOCATE NODE LIST:getmemory
      NUMBERS OF TOKENS TO ALLOCATE:pageneed(jv(cmdtype))
   ALLOCATE NODE LIST:
   RELEASE NODE LIST:freememory
   RELEASE NODE LIST:
   DESTROY NODE LIST:
   CREATE NODE LIST:
QUEUE:cpuq
   TYPE:ps /*processor sharing*/
   CLASS LIST:cpu
      SERVICE TIMES:cputime
   CLASS LIST:
QUEUE:
```

Set nodes are used to perform assignment statements in the sense of programming languages.

```
SET NODES:setcmdtype
   ASSIGNMENT LIST:jv(cmdtype)=discrete(1,.8;2,.15;3,.05), ++
                   jv(cyclecount)=cpiocycles(jv(cmdtype))
SET NODES:decrcycles
   ASSIGNMENT LIST:jv(cyclecount)=jv(cyclecount)-1
SET NODES:
FISSION NODES:
FUSION NODES:
```

The following submodel definition is very sparse, but could be embellished considerably without changing its subsequent invocations in the submodel cssm.

```
SUBMODEL:iosys
   NUMERIC PARAMETERS:
   NODE PARAMETERS:
   CHAIN PARAMETERS:interactiv
   CHAIN PARAMETERS:
   NUMERIC IDENTIFIERS:
   QUEUE TYPE:diskdef
      NUMERIC PARAMETERS:
      NODE PARAMETERS:servicecls
      NODE PARAMETERS:
```

```
      TYPE:fcfs
      CLASS LIST:servicecls
         WORK DEMANDS:.06
      CLASS LIST:
   END OF QUEUE TYPE DISKDEF
   QUEUE TYPE:
   QUEUE:diskq
      TYPE:diskdef /* invocation of above definition */
      SERVICECLS:disk
   QUEUE:
   SET NODES:
   FISSION NODES:
   FUSION NODES:
   SUBMODEL:
   INVOCATION:
```

We have not seen any routing chain definitions yet. The following definition is atypical because within the submodel there is only one node, "disk", and so no routing within the submodel will be defined. After giving the name of the chain, we indicate that this chain is to be completed in the external model, i.e., in the model invoking the submodel "iosys". We then indicate that "disk" is both the standard entry point and the standard exit point of the chain. In the invoking model we will refer to "disk" by the synonyms "input" and "output". The colon prompt (":") is for a routing transition, as we will see below.

```
   CHAIN:interactiv
      TYPE:external
      INPUT:disk
      OUTPUT:disk
      :
   CHAIN:
END OF SUBMODEL IOSYS
SUBMODEL:
INVOCATION:iosys1
   TYPE:iosys
   INTERACTIV:interactiv
INVOCATION:iosys2
   TYPE:iosys: interactiv  /* shorthand positional form*/
INVOCATION:
```

These invocations create two subnetworks with the characteristics of submodel IOSYS. (In this case the subnetworks consist only of a single queue each.)

The following chain definition is more typical than the previous one. After declaring the standard entry point to be the set node setcmdtype and the standard exit point to be the release node freememory, we define the routing among the nodes of the subnetwork.

```
CHAIN:interactiv
   TYPE:external
   INPUT:setcmdtype
   OUTPUT:freememory
   :setcmdtype->getmemory->cpu->iosys1.input iosys2.input;.5 .5
   :iosys1.output iosys2.output->decrcycles
   :decrcycles->cpu freememory;if(jv(cyclecount)>0) if(t)
```

```
    INITIAL STATE DEFINITION-
    CHAIN:interactiv
       NODE LIST:terminals
       INIT POP:users
    CHAIN:
```

The simulation run will end when the first of the following limits are reached.

```
    RUN LIMITS-
       SIMULATED TIME:3600
       EVENTS:50000
       QUEUES FOR DEPARTURE COUNTS:cssm1.memory
          DEPARTURES:500
       QUEUES FOR DEPARTURE COUNTS:
       NODES FOR DEPARTURE COUNTS:
    LIMIT - CP SECONDS:5
    TRACE:no
END
 NO FATAL ERRORS DETECTED DURING COMPILATION.
```

The EVAL command invokes dialogue for model solution (e.g., simulation). This dialogue prompts the user for parameter values, performs the solution and then provides the user with performance measures requested by the user. For our example model, we might have the following dialogue with the EVAL command.

```
eval
RESQ2 EXPANSION AND SOLUTION PROGRAM.
MODEL:csm
RESQ2 VERSION DATE: JANUARY 29, 1982 -  TIME: 17:00:35  DATE: 01/29/82
THINKTIME:16
USERS:25
```

Once the parameter values are specified, the model definition is complete and macro-expansion of the submodel definitions is performed. Then solution commences. When simulation ends, we get one or more messages indicating why simulation stopped, an error message or a message indicating no errors were detected, and a summary of the simulation run.

```
RUN END: CPU LIMIT
NO ERRORS DETECTED DURING SIMULATION.
                  SIMULATED TIME:      245.77480
                        CPU TIME:           5.25
               NUMBER OF EVENTS:           7461
```

Then we are prompted "WHAT:" meaning "What performance measures do you want to see?". A reply of "all" results in a display of all measures normally provided. Instead of "all", we give now give codes of particular measures and names of particular elements of interest. (A reply of "how" would provide a tutorial listing all codes.)

```
WHAT:nd(cssm1.memory)
INVOCATION     INVOCATION     ELEMENT        NUMBER OF DEPARTURES
     .          CSSM1         MEMORY         324
```

```
      CHAIN:
   END OF SUBMODEL CSSM
   SUBMODEL:
```

Following is the invocation of the submodel representing the entire computer system, with values for the numeric and chain parameters.

```
   INVOCATION:cssm1
      TYPE:cssm
      PAGEFRAMES:userframes
      INTERACTIV:interactiv
   INVOCATION:
```

A chain in the model proper will be either open, if there are to be provisions for external arrivals and departures, or closed, if jobs are fixed within the chain (as in our example).

```
   CHAIN:interactiv
      TYPE:closed
      POPULATION:users
      :terminals->cssm1.input
      :cssm1.output->terminals
      :
   CHAIN:
```

This completes definition of the model proper. The remaining dialogue section pertains to the specifics of simulation solution.

Many performance measures are gathered by the simulation by default. However, gathering of distributions of these measures for all appropriate network elements can be expensive in both time and memory, so distributions are only gathered when requested. The queueing time for the memory queue, defined as the time of arrival at the allocate node to departure from the release node, will be the response time seen by terminal users.

```
   QUEUES FOR QUEUEING TIME DIST:cssm1.memory
      VALUES:1 2 3 4 5 6 7 8
   QUEUES FOR QUEUEING TIME DIST:
   QUEUES FOR QUEUE LENGTH DIST:cssm1.memory
   MAX VALUE:users
   QUEUES FOR QUEUE LENGTH DIST:
   NODES FOR QUEUEING TIME DIST:
   NODES FOR QUEUE LENGTH DIST:
```

RESQ provides three methods for estimating confidence intervals for performance measures, and two of these three methods also provide for run length control based on the confidence intervals. In this example we will not illustrate confidence interval estimation or associated run length control.

```
   CONFIDENCE INTERVAL METHOD:how
CONFIDENCE INTERVAL METHODS ARE: REGENERATIVE, REPLICATIONS, SPECTRAL
                                 OR NONE
   CONFIDENCE INTERVAL METHOD:none
```

For closed routing chains (and open chains which are not initially empty) we must specify where the jobs of the chain are to be placed initially.

```
WHAT:qt(cssm1.memory)
INVOCATION     INVOCATION     ELEMENT        MEAN QUEUEING TIME
               CSSM1          MEMORY         2.81971
```

A null reply to "WHAT:" terminates the examination of performance measures.

```
WHAT:
```

We are then given the opportunity to extend the simulation run. We may increase any of the run limits we specified before and let the simulation run until one of the new limits is reached. In the following we increase the limit on CPU time. (This example was run on a model 3033 processor.)

```
CONTINUE RUN:yes
LIMIT - SIMULATED TIME:how
LARGER VALUE THAN  3.600E+03 OR NULL TO KEEP THAT VALUE
TRY AGAIN-
LIMIT - SIMULATED TIME:
LIMIT - EVENTS:
LIMIT - CSSM1.MEMORY DEPARTURES:
LIMIT - CP SECONDS:10
```

When the simulation reaches one of the new limits, we see the old termination message followed by a new one and a new summary of the simulation run. We then receive the "WHAT:" prompt again.

```
RUN END: CPU LIMIT
RUN END: CSSM1.MEMORY DEPARTURE LIMIT
NO ERRORS DETECTED DURING SIMULATION.
                 SIMULATED TIME:       366.25098
                       CPU TIME:            8.10
               NUMBER OF EVENTS:           11528

WHAT:nd(cssm1.memory)

INVOCATION     INVOCATION     ELEMENT        NUMBER OF DEPARTURES
               CSSM1          MEMORY         500

WHAT:qt

INVOCATION     INVOCATION     ELEMENT        MEAN QUEUEING TIME
                              TERMINALSQ     14.82022
               CSSM1          MEMORY         2.95095
               CSSM1          CPUQ           0.03118
CSSM1          IOSYS1         DISKQ          0.07692
CSSM1          IOSYS2         DISKQ          0.07245

WHAT:

CONTINUE RUN:yes
LIMIT - SIMULATED TIME:
LIMIT - EVENTS:
LIMIT - CSSM1.MEMORY DEPARTURES:
1000
```

```
LIMIT - CP SECONDS:20
RUN END: CPU LIMIT
RUN END: CSSM1.MEMORY DEPARTURE LIMIT
RUN END: CSSM1.MEMORY DEPARTURE LIMIT
NO ERRORS DETECTED DURING SIMULATION.
                  SIMULATED TIME:      753.42139
                        CPU TIME:          16.29
                NUMBER OF EVENTS:          23054

WHAT:qt(cssm1.memory)
INVOCATION     INVOCATION     ELEMENT        MEAN QUEUEING TIME
               CSSM1          MEMORY         2.78478

WHAT:
CONTINUE RUN:no
```

Having terminated both the performance measure dialogue and the simulation, we are now given the opportunity to define a new set of parameters and start a new run.

```
THINKTIME:
EXPANSION FINISHED.
```

A transcript of the dialogue with the EVAL command is also available, e.g., for printing.