A Graphics-Oriented Environment for Solving Large, Complex Extended Queueing Network Models

July 26th, 1988

Kurtiss J. Gordon and James F. Kurose

Department of Computer and Information Science
University of Massachusetts
Amherst, Mass. 01003

Robert F. Gordon and Edward A. MacNair

IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598

A GRAPHICS-ORIENTED ENVIRONMENT FOR SOLVING LARGE, COMPLEX EXTENDED QUEUEING NETWORK MODELS

Kurtiss J. Gordon and James F. Kurose
Department of Computer and Information Science
University of Massachusetts
Amherst, Mass. 01003

Robert F. Gordon and Edward A. MacNair
IBM Thomas J. Watson Research Center
Yorktown Heights, New York 10598

ABSTRACT

The development of models for evaluating the performance of resource contention systems, such as manufacturing systems, computer systems, and communication networks, is often a difficult and complex task. This effort can be very costly in terms of time required to build, verify, modify and maintain the models, as well as the work required to evaluate the performance measures of the models. This modeling effort can be dramatically reduced by the use of appropriate software tools. The Research Queueing Package Modeling Environment (RESQME) provides a graphical environment for constructing, solving, and analyzing the results of extended queueing network models of resource contention systems. It supports a rich underlying modeling paradigm previously developed in the Research Queueing Package (RESQ) and provides an integrated graphical interface throughout all phases of the modeling lifecycle. In this paper we describe the guiding principles behind RESQME and illustrate how these principles are embodied in its capabilities. Particular emphasis is placed on the manner in which RESQME supports the modeling of large and complex real-world systems, and on the way its graphical modeling capabilities can be extended into diverse, application-specific domains.

1. Introduction

Manufacturing lines, communication networks, and computer systems are examples of systems which are sufficiently complex that carefully developed models are needed in order to understand system performance. In these systems, *contention* for the use of system resources is the primary factor affecting performance. Thus, queueing network models are typically employed for modeling system behavior. These models can be broadly divided into "traditional" queueing network models (solved analytically) and "extended" queueing network models (solved through simulation) [14]. In both cases, the performance model contains queues and "nodes" (which represent the system resources), and jobs (which represent the objects in the system which contend for the use of these resources). The purpose of the model, then, is to analyze the effect of contention on the flow of jobs through the system.

The development of both simulation and analysis programs represents a significant software development effort and can be very costly in terms of the effort required to build, modify, and maintain these programs. In the case of simulation, a modeler who programs a simulation "from scratch" faces the myriad, low-level details of discrete event simulation such as random number generation, maintenance of event lists, event process, and statistics gathering. In the case of analysis, coding, testing, and validation of numerical software are required. Fortunately, a growing number of software tools have recently become available to free the modeler from this low-level effort. A modeler using these tools can thus concentrate his efforts on developing, evaluating, and validating models (rather than low-level software) and on more thoroughly exploring the system design space.

Traditionally, the notion of an "appropriate" software tool has meant a declarative language in which a performance model may be specified and then evaluated. General purpose programming languages such as PASCAL and FORTRAN and general purpose simulation languages such as GPSS, SIMULA, and SIMSCRIPT have often been used in the performance evaluation process. More recently, special-purpose modeling languages such as RESQ (Sauer, MacNair, and Kurose [18,19,20,21], Chow, MacNair, and Sauer [3], MacNair and Sauer [15]), PAWS (IRA [12]), and STEP-1 (Tripathi, Agrawala, Abrams, Ramakrishnan, and Singhal [25]) have been developed. These modeling languages provide the modeler with a rich set of modeling elements which can be easily used to construct complex models. At this level of abstraction, performance models can be easily and rapidly constructed, evaluated, and verified. The modeler's productivity and effectiveness thus increase dramatically.

With the recent widespread availability of relatively inexpensive, graphics-oriented workstations, the development of a performance evaluation workstation environment has become possible. At the heart of such an environment is the notion of a graphical representation of a performance model (Melamed and Morris [16]). We believe a graphical representation is the most natural form in which to express a performance model. Other work expressing this belief has appeared in Browne, Neuse, Dutton, and Yu [1], Gilbert and Kleinöder [4], Healy [8], Pegden, Miles, and Diaz [17], Sinclair, Doshi, and Madala [22] and Standridge, Vaughan, and Sale [23,24]. Indeed, it is interesting to note

the amount of textual information needed on a (sub)model diagram, a powerful and flexible approach towards specifying job routing, host execution of the model, generalized output graphics, and context-sensitive forms which minimize the effort needed to specify the textual attributes associated with a model icon. These features will be discussed in more detail in the following sections.

- **Extensibility.** RESQME provides capabilities which permit the performance analyst to define higher-level modeling elements (by composing the lower-level RESQ modeling primitives within a submodel), create new icons for these new modeling elements, and then to integrate these new constructs into RESQME. For example, in the manufacturing domain, a modeler might define modeling "primitives" corresponding to a manufacturing cell, or assembly station; in the communication domain, new modeling elements might be defined for a particular type of link or for a statistical multiplexer with a complex queueing discipline. The new modeling constructs may then be graphically manipulated in exactly the same manner as the pre-defined, lower-level RESQME modeling elements. They can also be combined with other higher-level modeling elements as well as with the lower-level RESQ elements to form new higher-level elements. Thus, RESQME can be used as a base system upon which application-specific performance modeling packages can be built. (We note that previous application-specific modeling packages have already been successfully built upon the text-only version of RESQ [2]). The ability to define and manipulate new modeling primitives is discussed in section 3.4.

  Since much of RESQME is data-driven, it can also be easily customized and used as the basis for a general interface environment for any network-based decision-making package. For example, the format and field definitions in the textual attribute panel definitions (discussed in section 3) and the pictorial icon representations are stored in files which can be easily modified without changing the internal structure of RESQME. The output analysis routines are also generic in nature rather than being intimately tied to the internal storage of performance data in RESQ. Thus, for example, RESQME could be used with very little effort to construct and manipulate models of Stochastic Petri Nets.

The remainder of this paper is structured as follows. In the following section we provide a brief introduction to RESQ and an overview of the RESQME modeling environment. In Section 3 we describe how RESQME can be used to construct large and complex performance models. Sections 4 and 5 discuss the solution of these models and the analysis of the output results within RESQME. Section 6 contains a discussion of initial user reactions and outlines some of our current and future activities. Section 7 concludes this paper.

In RESQME, each type of queue or node is represented by a graphical icon; each icon then has one or more associated attributes (e.g., a service time distribution must be associated with a queue). As discussed shortly, these attributes are specified in RESQME using context-sensitive forms, which minimize the need for a modeler to know the syntax of the underlying RESQ language.

A *chain* specifies the route or path that a class of jobs will follow through the queues and nodes in a model. These routing decisions may be fixed, probabilistic, or dependent on the current state of the simulation (e.g., dependent on the number of jobs at a given queue). In an *open* chain, new jobs may be created at source nodes, and jobs may leave the chain at a sink node. In a *closed* chain, an essentially fixed number of jobs circulates among the nodes in the chain. As discussed in Section 3, RESQME provides a sophisticated line-drawing facility for specifying job routing.

RESQ also provides numerous modeling constructs in addition to those discussed above, such as symbolic constants and parameters, and arrays of nodes. Another important modeling construct is the *submodel*, which may be used to define a parameterized template of an interconnected "subnetwork" of nodes and queues. This submodel may then be *invoked* several times (in much the same way that a macro may be invoked several times in a programming language) to create multiple instances of that subnetwork. Submodels provide an important mechanism for constructing modular, hierarchically-structured models of large and complicated systems. RESQME provides explicit graphical support for such hierarchical modeling.

The final component of a RESQ model concerns the solution itself. RESQ can both simulate and, in some cases, analytically solve for the various performance measures requested by the performance modeler. In the case of simulation, the modeler may specify information regarding the desired length of the simulation run, a method for generating confidence intervals, tracing parameters, and other simulation-related information. Resource utilization, throughput, average queue length, queueing time, and queue length distribution are typically requested performance measures. Several methods are provided by RESQ for generating confidence intervals, including "independent replications", the "spectral" method, and the "regenerative" method [18,19,20]. The ability to graphically generate and analyze performance results in RESQME is discussed in section 5.

2.2 An Overview of RESQME

Here we present an overview of the RESQME environment. In doing so, we discuss our design goals and outline the RESQME implementation.

Our design goals were:

- to provide a "natural" environment for modeling and analysis of large and realistic systems which are amenable to an extended queueing network type of approach,
- to give the modeler access to all the functionality of a large decision-support package (RESQ), and
- to make the modeling process easy and productive.

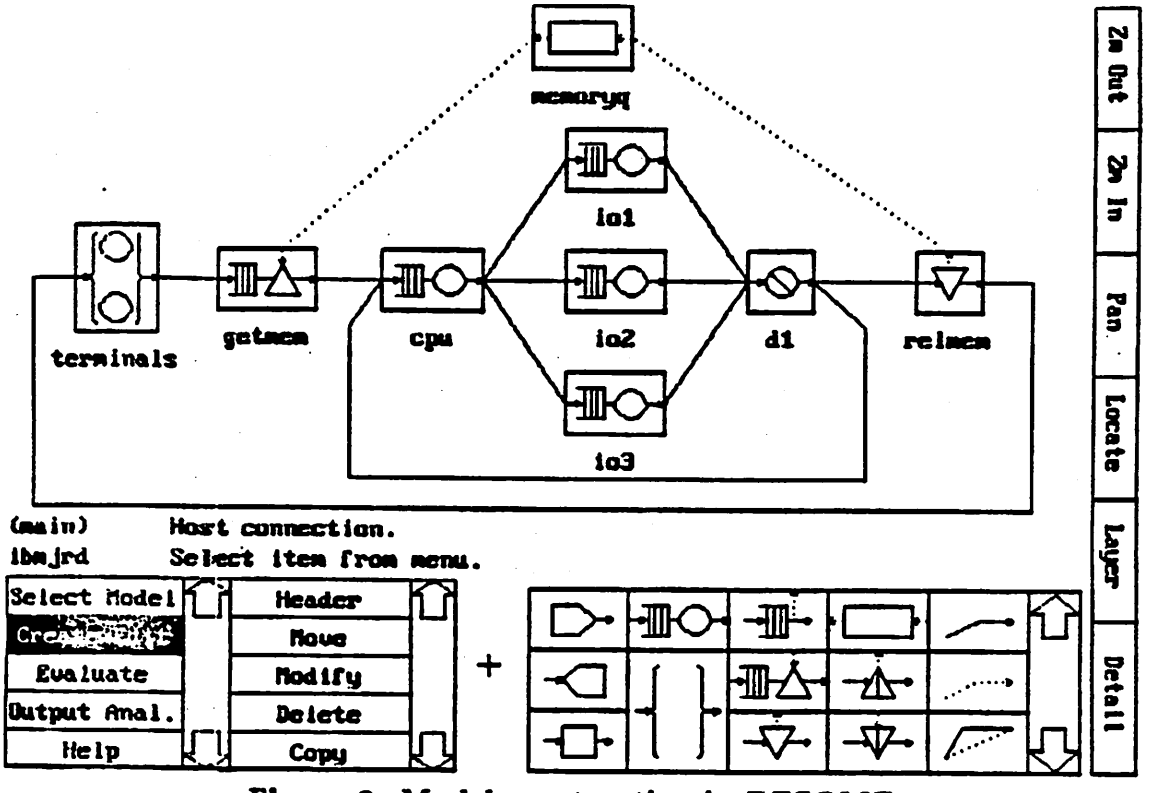

Figure 2: Model construction in RESQME

each of these three tasks in detail. At this point, we describe functionality which is common to, or transcends, the individual tasks.

Uniform access is provided to all of these modules by a high-level graphical command dispatcher. The control of the three tasks is handled by the Main Menu, the lower leftmost menu in Figure 2. The modeling area, or modeling "canvas," is shown above the menu. Selecting a task in the Main Menu causes the appropriate Task Menu to appear for selection. The tasks can be selected and repeated in any order until the modeler completes the analysis. In Figure 2, a completed "central server" model [14] is shown in the modeling area; and the Edit/Create Task menu and one page of the icon palette are also shown.

A vertical "screen manager menu" is also shown at the right of Figure 2. This menu contains commands for manipulating the modeler's view of the performance model and is available at all points in the modeling process. The modeler can pan, zoom, and locate objects by name on the model diagram. As described later, the modeler can also change the graphical view to another layer, or submodel, of a structured model definition.

Information about the model is specified at two levels. At one level is the graphical specification — the network diagram, its components, and the output charts. At a second level, there is information consisting of the underlying attribute data for the graphical objects. For example, a queue icon has attribute data consisting of its name, type of queue, queueing discipline, service time, etc. Similarly, a chart has attribute data consisting of its type (line, bar, histogram), color, and contents. The network diagram has attribute information consisting of the model name, solution method, parameter names, and run control limits. The underlying attribute data specification itself

is presented with one workstation interface, making the PC-to-host connection transparent. The workstation is connected to the mainframe using the communication package PC/VM Bond [10].

3. Model Construction

3.1 Specifying a Model

In RESQME, the performance modeler constructs a model by (1) selecting icons to represent the system resources and placing them on the modeling canvas in meaningful relative positions, (2) specifying attributes for each icon so that it represents the behavior of the corresponding system resource, and (3) interconnecting the icons on the canvas with paths which represent the possible flows of jobs and control in the system. Different modelers may wish to perform these actions in different sequences or may mix these sequences, building up resources and routing in the model diagram section by section. RESQME accommodates all such strategies, as well as the inevitable breaks in strategy which arise from false starts, editing, and debugging.

The modeler enters an icon into the model by picking it from the icon palette and placing it in the modeling area of the screen. For ease in later attachment of routing connections, the icon can be placed on the modeling canvas in any of the four cardinal orientations. When the icon is placed, RESQME displays a context-sensitive form (described below) for specifying the icon's attributes. The modeler may fill in attributes at this point, or may escape and leave them to be filled in later. Figure 2 shows a snapshot of RESQME during the model creation process. The queue icons used to represent the CPU, memory, terminals, and I/O devices in the central server model are shown on the modeling canvas.

Once an icon has been placed on the modeling canvas, the modeler can MOVE, DELETE, or COPY the icon, or MODIFY its attributes. (These commands are shown on the second command menu in the lower left of Figure 2.) COPYing an icon from the modeling canvas differs from ADDing a new icon from the palette in that all the attributes of the copied icon except for its name are replicated in the new icon. If the modeler MOVEs an icon which has routing connections associated with it, the connections stay with the icon in its new position and orientation. Commands such as COPY, MOVE, DELETE, and MODIFY are *modal* in nature in that, once chosen, the command remains highlighted on the menu, and the modeler can perform the same operation on several icons in sequence without re-selecting the command. The use of modes has proven extremely useful in reducing the number of keystrokes needed in editing a model.

The icon attributes (such as its name, associated distribution (if any), priority information, etc.) are specified textually in RESQME using context-sensitive *forms*, which help reduce the need for the modeler to know the syntax of the underlying RESQ language. Figure 3 shows such a form. Attributes are entered in fields following each colon (":"). The various commands available for manipulating the form are indicated at the bottom of the figure. The forms are context-sensitive in the sense that the specification of certain attributes may cause the remainder of the form to change dynamically. For example, if a priority queueing discipline is chosen in Figure 3 (by scrolling

next connection. Depending on the geometrical arrangement of the icons, a straight line may not be the ideal representation for the routing path. RESQME permits the modeler to tack down the path at several intermediate points before picking the "to" icon. (The model in Figure 2 contains tackpoints in several of its routes.) Although these tackpoints have no semantic value in the model, they can be moved or deleted just like icons. It is also possible to insert a tackpoint into an existing line segment.

Most models contain one or more branch points—an icon from which a job may go to one of several other icons, based either on a probabilistic decision or on some simulation-dependent condition. To aid in the specification of such "1–n" routing, RESQME allows the modeler to build up a list of "to" nodes by double-clicking on the first icon in the list, single-clicking on the second through $(n-1)$th, and double-clicking on the last one. The same procedure can be used to build up a list of "from" nodes for convergent (n–1) or parallel (n–n) routing. Upon completion of a 1–n or n–n link, all n of the "to" nodes become "from" nodes, and the modeler may continue with n–n or n–1 routing. Figure 2 shows an example of 1–n and n–1 routing. RESQME also permits tackpoints in 1–n, n–1, and n–n routes. When the modeler puts down tackpoints between the lists of "from" and "to" nodes, the routes converge to the first tackpoint, a single path runs from tackpoint to tackpoint (a single line thus representing n different actual paths in compressed form), and the routes diverge from the last tackpoint. The n–1, 1–n, and n–n routing, tackpoints, and compressed routing have proved particularly valuable in specifying the job routing in large models.

As the modeler graphically specifies the routing, RESQME also presents a textual confirmation of the routing, complete with default probabilities for the routing control. When the modeler finishes the graphical part of the routing for a particular chain, there is an opportunity to edit these probabilities or change them to Boolean predicates. It is also possible to reenter any chain to insert additional routes, to MODIFY the routing predicates, or to MODIFY the line style or color of the chain links. The modeler can also DELETE individual links or an entire chain.

3.3 Hierarchical Modeling Using Submodels

Systems in the real world can be large and complex. This complexity can arise from a large number of components in the system as well as from detailed structure within individual components. In either case, the modeler will find it conceptually useful to visualize the system at different levels of detail. RESQME supports this multilevel modeling concept with hierarchical model definitions. The conceptual unit in the hierarchy is a *submodel* definition—a collection of RESQME primitives and/or other submodels and the routing connecting them to one another and to the input and output connections to the outside of the submodel.

Graphically, each submodel is defined on its own modeling canvas. This single definition may be parameterized, and can be invoked as many times as the modeler wishes. Since the model can be constructed in a multilayered hierarchy by the nesting of submodels, a model's submodels are thus related to one another as the nodes of a tree. The LAYER menu command in RESQME (shown in the vertical screen manager menu in Figure 2) provides the functionality for traversing the tree

the submodel is already in the tree structure of the model. If it is not, RESQME will search for the submodel description, and if it exists, will attach it to the model structure. If it does not exist, the modeler can layer to a new modeling canvas and create it. The underlying text attributes for the user-created icon are displayed based on the submodel to which it is linked. The attributes include the submodel type and any parameter variables. The modeler provides the parameter values for each instance of the higher-level object.

The modeler can view the underlying submodel network at any time by selecting the DETAIL item from the screen manager menu and pointing to the desired user-created icon. RESQME will then display the submodel network. The modeler can also view any of the submodels in the model structure and move back to the root by using the screen manager menu item LAYER.

3.5 Data Structure Support for Large Models

It must be clear at this point that RESQME models contain both graphical and textual information, that large models will contain large amounts of both kinds of information, and that (in order to be useful) RESQME needs to make it easy for the modeler to switch back and forth between graphics and text management while keeping the context of the model in the modeler's view.

The graphics management contains two classes of functions: icon-oriented and screen-oriented. The icon-oriented functions include adding, deleting, moving, and copying individual icons and the individual line segments which comprise the routing paths. The screen-oriented functions include zooming in and out, panning, and locating a named icon. If we think of the model diagram existing in its own coordinate frame, independent of the monitor screen, then the icon-oriented functions alter the model diagram while the screen-oriented functions merely alter the window through which the modeler views the diagram.

The efficient implementation of these functions imposed an interesting combination of requirements on the data structures for the graphical elements of RESQME. The icon-oriented functions require efficient searching of the data on the basis of 2-dimensional coordinates. This requirement suggested some form of tree, ideally one which is easy to maintain in a reasonably balanced condition with frequent insertion and removal of data items. The screen-oriented functions, on the other hand, require efficient scanning of a rectangular region of the coordinate space to retrieve all of the data in the region. This requirement suggested some form of a linked list in order to obviate the need for repeated searches.

Our solution was to use a combination of both data structures. All of the information needed to draw the graphical elements and find the textual information associated with them is contained in a linked list. In addition, there is an index tree with pointers to the elements of the linked list. Actually, the icons and the line segments have separate lists and separate index trees, because they have different data structures. The icon list is organized according to a key built up by concatenating a coarse version of the vertical and horizontal coordinates of the center of each icon. This gives a zoned or "fat raster" scan of the data, which is the same organization used in many catalogues of

Selecting the EVALUATE main menu item causes the EVALUATE task menu to be displayed. This menu provides the modeler with the commands to enter parameter values, execute and monitor the host run. A model can then be executed with different parameter values in a series of experiments.

The communication with the host is transparent to the modeler. It is presently accomplished with calls to the communication package PC/VM Bond through a cable-connected RS-232 port. Selecting the EXECUTE item uploads the model files to the host and issues the host command to run the model. In this case, it is the command to run RESQ with the model files. This, of course, could be revised to run another decision-support system. The host execution gives us the computing power to run large, realistic models.

The host execution is done in the background, so that the modeler can continue to work at the workstation on this or other models while the host is processing the model. This cooperative processing takes advantage of the host for the computation-intensive execution of the model and the workstation for the interactive graphics.

The modeler can check on the progress of the run at any time. When the model results are ready, the workstation will sound a beep to notify the modeler, and continue to do so periodically until the modeler acknowledges the message.

5. Model Output Analysis

RESQME supports the output analysis task by providing a general-purpose plotting package to graph the results. This task is integrated with the other tasks of RESQME again using the model diagram as the interface. The plotting package is general in that it can plot the output from any modeling program as long as the output is put in the form: performance measure identification followed by a vector of x and y values.

When the modeler selects the Output Analysis menu item from the Main Menu, a Task Menu appears with the commands to access the performance measures from the host, specify the content and the form of the plot, and plot the resulting chart. Figure 4 shows the RESQME display with this task menu and an example plot of performance measures.

The modeler can analyze performance measures from one model run or across runs, for one node or for many nodes. He can plot many results on one chart or on different charts. Whenever confidence intervals are produced by a simulation, they are automatically displayed on the charts for those performance measures. The specific numerical data for each performance measure are also available as the underlying textual attributes associated with the chart contents.

Several functions are made available to the modeler to help analyze the data. For example, the modeler can produce a short run using one of the confidence interval methods of RESQ. The modeler can then select performance measures from this pilot run and ask for a confidence interval projection. A plot is then displayed that estimates the required run lengths to meet a range of desired confidence interval widths at a desired percent. In addition, the modeler can apply

name to choose performance measures associated with the whole model or to a routing link to select performance measures associated with a route. To handle the volume of performance measure data for large models, the output data are stored on disk in an indexed file. Only when the modeler selects a given performance measure is it brought into the PC memory.

Our windowing system allows the charts to share the modeling surface with the model diagram. Charts can overlap the model diagram and other charts. The charts can be positioned, moved, shuffled, and removed by direct manipulation. Each submodel layer has its associated charts which are independent of the charts on other layers of the model.

The model diagram, then, is the basis for the selection of output as well as the interface for model creation, and the modeler can view the output and the model diagram simultaneously. The modeler can explore changes based on his analysis of the output by directly modifying the model diagram, re-evaluating the model, and then viewing the next version of the performance measures.

6. Discussion

We began distributing a prototype version of RESQME in March 1987 and currently have sent it to over ninety users throughout IBM. In the previous sections, we have highlighted those capabilities which we believe make RESQME an easy-to-use performance modeling environment; the initial user feedback we have received has borne out these beliefs, particularly with respect to the importance of maintaining a single graphical interface to the model throughout the modeling process.

Interestingly, an unexpected advantage of the graphical interface noted by users is that it provides a constantly up-to-date picture of the flow of jobs through the model. When building large and complex models, it is particularly important to know the current job flow. It is also evident that this flow changes considerably during the modeling process; as a result, most hand-drawn diagrams (previously used with textual versions of RESQ) were eventually not kept current. Hence, users commented that the graphical interface helps reduce the number of errors made while modifying the model. The capability of obtaining a hardcopy of the diagram was also mentioned as a valuable feature. It was also noted that the model diagram was a help in communicating the model structure and underlying behavior to other people who have not been involved in the model-building process.

The cooperative processing between the workstation and host has also proven particularly effective. One weakness of a strictly host-based modeling environment is that a heavily-loaded host system can slow down the modeling process, particularly when dealing with large models. RESQME permits simultaneous processing on both the local workstation and the host. Even when a large model is being evaluated, the workstation is free. This permits the modeler to work on other models or to view performance measures from prior runs of the model being evaluated. In essence, the host session behaves like a batch machine which does not interfere with the interactive session on the workstation.

can point to any desired node in the model diagram to specify performance measures which are then displayed in graphical and tabular forms. RESQME allows the modeler, through a consistent interface on the PC, iteratively to create the model, view the results of the analysis or simulation, revise the model based on the output, and compare results for families of models.

Acknowledgements

We would like to express our continuing thanks to Peter Welch for his support and encouragement of this work, to Richard Gilbert and to Mark Giampapa for their technical advice, and to Anil Aggarwal, Al Blum, Gary Burkland, Diana Coles, Paul Loewner, Geoff Parker and David Stein for their many suggestions and implementation work which are helping to improve the package. We would also like to express our appreciation to We-Min Chow, Darcy Hulseman, Howard Jachter, Peter Manson, and Sarah Spedden for their suggestions after using early versions of the system. In conversations with them they have contributed many ideas for future directions of RESQME. We would also like to acknowledge the continuing interest of Charles Sauer in RESQ. His contributions to RESQ continue to be of tremendous benefit. We are grateful to the many other colleagues and RESQ users who have helped improve RESQ over the years.

REFERENCES

[1] J. C. Browne, D. Neuse, J. Dutton and K.-C. Yu, "Graphical Programming for Simulation of Computer Systems," *Proceedings of the 18th Annual Simulation Symposium*, Tampa, FL, 1985, 109–126.

[2] K. Bharath-Kumur and P. Kermani, "Performance Evaluation Tool (PET): An Analysis Tool for Computer Communication Networks", *IEEE J. on Selected Areas in Communications*, Vol. SAC-2, No. 1, (Jan., 1984), pp. 220–226.

[3] W.-M. Chow, E. A. MacNair and C. H. Sauer, "Analysis of Manufacturing Systems by the Research Queueing Package," *IBM Journal of Research and Development*, 1985, 330–342.

[4] R. S. Gilbert and W. B. Kleinöder, "CNMGRAF - Graphic Presentation Services for Network Management," *Proc. 9th Data Communications Symposium*, Whistler Mountain, BC, 1985, 199–206.

[5] R. F. Gordon, E. A. MacNair, P. D. Welch, K. J. Gordon and J. F. Kurose, "Examples of Using the RESearch Queueing Package Modeling Environment (RESQME)," *Proceedings of the Winter Simulation Conference*, Washington, DC, December 1986, 494–503.

[6] R. F. Gordon, E. A. MacNair, K. J. Gordon and J. F. Kurose, "A Visual Approach to Manufacturing Modeling," *Proceedings of the Winter Simulation Conference*, Atlanta, December 1987, 465–471.

[7] R. F. Gordon, E. A. MacNair, K. J. Gordon and J. F. Kurose, "Higher Level Modeling in RESQME," *Proceedings of the European Simulation Multiconference 1988*, Nice, France, June 1988.

[24] C. R. Standridge, D. K. Vaughan and M. L. Sale, "Presenting Simulation Results with TESS Graphics," *Proceedings of the 1985 Winter Simulation Conference,* San Francisco, 1985, 237–243.

[25] S. Tripathi, A. K. Agrawala, M. Abrams, K. K. Ramakrishnan and M. Singhal, "STEP-1: A User Friendly Performance Analysis Tool," *Proceedings of the International Conference on Modelling Techniques and Tools for Performance Analysis,* Paris, May 16–18, 1984.