Proceedings of the 1990 Winter Simulation Conference
Osman Balci, Randall P. Sadowski, Richard E. Nance (eds.)

HIERARCHICAL MODELING IN A GRAPHICAL SIMULATION SYSTEM

Robert F. Gordon
Edward A. MacNair

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

Kurtiss J. Gordon
James F. Kurose

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

ABSTRACT

Hierarchical performance modeling allows the modeler to create models either in a top-down process, where he or she creates a high-level model and progressively fills in the details replacing stubs with detailed submodels or in a bottom-up process, where he or she first builds a library of submodels (or uses an existing library of submodels) and then links them together to form the overall model. In reality, modelers use a combination of both techniques. Submodels can themselves contain submodels in any number of layers in the hierarchy.

To support this hierarchical performance modeling in a graphical system requires the functionality to create, view, edit and store the graphical networks of individual submodels, to graphically link the submodels to other modeling elements (including other submodel instances), to graphically replace submodels with other modeling elements, and to view the results, including animation, for the overall model and each submodel instance.

The RESearch Queueing Package Modeling Environment (RESQME), a graphical workstation environment for iteratively constructing, running and analyzing hierarchical models of resource contention systems, will be used to illustrate the hierarchical modeling methodology.

1. INTRODUCTION

Submodels in modeling play the same role as subroutines in a programming language. A submodel defines a portion of the model network. It can contain parameters and can be invoked any number of times by calling the submodel and assigning values to its parameters, thus forming a hierarchical model. Submodels can be used to clarify the structure of a model, to avoid duplication of effort within a model, to permit sharing of parts of models, to introduce variability in the model structure and, with decomposition, to solve the submodel separately and replace the submodel with a flow equivalent server.

Hierarchical modeling allows the modeler to create models either in a top-down procedure, where he or she creates a high-level model and progressively fills in the details replacing stubs with detailed submodels or in a bottom-up procedure, where he or she first builds a library of submodels (or uses an existing library of submodels) and then links them together to form the overall model. In reality, a combination of both techniques is desirable. Submodels can themselves contain submodels in any number of layers in the hierarchy. The result is structured, modular models that reuse code and share pre-tested model components.

This concept becomes even more powerful when combined with graphics. Graphics allows the modeler to view the hierarchy, to visually associate the submodels with objects in the modeler's domain, and to directly manipulate these objects in building the model structure. Graphics also provides the interface to analyze the results by viewing the performance measure charts for each submodel invocation, as well as the animation of jobs through the hierarchy.

Gordon et al. [1986], Concepcion and Schon [1986], and Thomasma and Ulgen [1988] discuss graphical, hierarchical systems. The RESearch Queueing Package Modeling Environment (RESQME) [Kurose et al. 1986; Gordon et al. 1986, 1987; Aggarwal 1989] is a graphical workstation environment for iteratively constructing, running and analyzing hierarchical models of resource contention systems. It is built on top of the RESearch Queueing Package (RESQ) [Chow et al. 1985; MacNair 1985; MacNair and Sauer 1985; Sauer and MacNair 1982; Sauer et al. 1982a, b, c] which provides the functionality to evaluate extended queueing networks, analytically or by simulation. RESQME is unique in that it uses a single, uniform, hierarchical graphical representation of the model throughout the entire modeling process.

In this paper, we describe the functionality provided in RESQME to graphically support hierarchical models through all aspects of the modeling process. Section 2 describes the hierarchical modeling methodology and how it is implemented in RESQME. Section 3 shows a simplified example to demonstrate the use of hierarchical modeling. We summarize the results and benefits in Section 4.

2. HIERARCHICAL MODELING

Submodels can be used effectively to clarify the structure of models, to ease repetition, to share common elements between models, to incorporate variability in the number of model elements and to perform model decomposition.

One way to approach the modeling of a complicated system is to identify major subsystems and structure the model around these subsystems. The subsystems can be defined first at an abstract level. The modeler can refine the top-level view of the system in a stepwise fashion by adding more and more details to the submodels and to their connections and perhaps defining submodels within them. This block structuring helps to clarify how the model functions and to determine the level of detail necessary for the application.

Another use of submodels is to replicate similar portions of a model. A submodel can be defined with parameters which can be assigned different values for each invocation. This ease of repetition simplifies the construction of many models.

Often, different models contain sections representing similar subsystems. For example, different assembly line models may use similar subsystems for robots, bake ovens, conveyor belt mechanisms. These subsystems can be placed in a library of submodels, and models can then be created by connecting these prefabricated modules in a "tinker-toy" fashion.

Submodels can also be used to provide a variable number of similar parts of a model. When the mechanism for invoking a submodel provides an array capability, the number of copies made can be a parameter of the model. By changing the parameter value, the number of queues, nodes, and chains in the model can be varied without recompiling the model.

Another benefit of submodels is for hierarchical decomposition. It is sometimes possible to hierarchically structure the model with submodels which can then be solved individually. Results from each submodel solution can be used to characterize a queue dependent rate server to replace the submodel. For models which cannot be solved analytically, decomposition may yield submodels and aggregate models which can all be solved analytically. Theoretical work which provides justification for using this approach can be found in Chandy, Herzog and Woo [1975] and Courtois [1975, 1978]. Even if the decomposed model does not produce submodels and aggregate models which can be solved analytically, decomposition could still provide speed-up benefits for the simulation, because the flow equivalent server requires only a single event in place of the many events required in the original submodel.

2.1 RESQME Implementation of Hierarchical Modeling

To provide a hierarchical modeling feature, a graphical system requires the functionality to create, view, edit and store the graphical networks of individual submodels, to graphically link the submodels to other modeling elements (including other submodel instances), to graphically replace submodels with other modeling elements, and to view the results, including animation, for the overall model and each submodel instance. We demonstrate how this functionality can be provided by describing the RESQME implementation.

The objects of a RESQME network diagram are icons which represent

1. fundamental modeling elements (such as an active queue, passive queue, job source),
2. invocations of subnetworks of these elements (submodels), and
3. chain links which connect the elements and submodel invocations to form the job routing.

The modeler is provided with tools to extend the fundamental modeling elements. The modeler can create a subnetwork of these elements and draw a user-created icon or use a generic submodel icon to represent that subnetwork. Placing a user-created icon on the network diagram creates an invocation of the submodel associated with it and causes that submodel (if it exists and is not already attached) to be attached to the the main model in a tree structure. This can also be accomplished by placing the generic submodel icon on the diagram and specifying its type by giving the name of the desired submodel to be invoked.

A submodel invocation has an iconic representation and a graphical representation of the submodel's network, and textual attributes, such as invocation name, submodel name, parameter names and parameter values for the invocation. Thus the submodel can be treated as a black box which the modeler can invoke, as long as the modeler knows its assumptions and can provide its parameter values for that invocation.

The graphics environment allows the modeler to pan, zoom, locate and layer up and down through the hierarchical model layers. The modeler can view and modify the higher-level model, as well as the corresponding details of all levels of submodels. A separate layer is provided for each submodel. Each submodel occupies a separate canvas, into which the modeler can view any portion of any submodel.

The modeler is presented with a menu-driven, iconic interface to build the model (Create/Edit Task), run the simulation (Evaluate Task), and examine the results (Output Analysis Task).

2.2 Create/Edit Task

In this task, the modeler can select icons from the icon palettes and place them on the modeling plane. The icon palettes contain both the elementary icons and the user-created submodel icons. When either type of icon is selected and placed on the modeling surface, the appropriate template appears for the modeler to supply the underlying textual information.

As the modeler builds the model by selecting the user-created icons or the generic submodel icon, a submodel tree structure is formed with the main model at the root and the submodels forming the branches. To view or modify a submodel represented by a user-created icon or the generic submodel icon, the modeler selects the "Layer Down" menu item from the screen management menu and points to the user-created object to see its details. The submodel represented by that icon is then displayed on the screen and can be edited in the same way as the main model. The modeler can traverse the multiple levels of the model's hierarchy by selecting the "Layer Up" or "Layer Down" menu items. The selection displays a choice in a pop-up menu of the names of the parent (sub)models in the hierarchy tree above the current layer when the "Layer Up" item is selected and the child submodels below when the "Layer Down" item is selected. Included in the choice when "Layer Down" is selected, are a "Library of submodels" entry and a "New" entry. The "Library of submodels" entry allows the modeler to select pre-packaged submodels to include in his model, and the "New" entry allows the modeler to create a new submodel.

Thus the modeler can build his model by linking together existing submodels from libraries and, if necessary, building new submodels in a bottom-up approach. Also, the modeler can replace elementary nodes (including the generic submodel node) with detailed submodels in a top-down approach. User-created icons can be created before, after or during the development of the associated submodel.

We will describe the key aspects for creating a new submodel here.

When a modeler selects "New" from the pop-up menu resulting from the "Layer Down" command, a blank canvas layer is provided on which to create a new submodel. Each submodel has a header which identifies its parameters (in the same manner as arguments are used in a subroutine). The parameters that the modeler specifies in the header for the submodel will appear as prompts for values in the invocation's attribute window when that submodel is invoked at a higher level. The modeler can use any of the icons, including user-created icons of other submodels, when drawing the subnetwork for the new submodel. In the attributes for each subnetwork chain, the modeler identifies the node names of the input port and the output port. RESQME identifies these in the subnetwork diagram by attaching an I and an O, respectively to these nodes.

The invocation of that submodel at the level above passes information to the submodel by providing values for the submodel parameters. These values are provided in the invocation's pop-up attribute window. When a job reaches an invocation, it is sent to the subnetwork through the input port, and jobs return to the next node on the path after the invocation when they reach the subnetwork's output port.

Actually, RESQME provides a number of ways to have jobs flow in and out of a submodel and also provides several ways to pass data to a submodel. First of all, as we just mentioned, each submodel has a unique input port and output port for each routing chain in the calling (sub)model. The input port and output port link the submodel chain to the corresponding routing chain in the (sub)model that invoked it. Additionally, jobs can be passed to the submodel through parameter nodes and fully-qualified path names. A generic icon is used to represent a parameter node. This node type associates by name any point in a submodel with a corresponding node in a higher level (sub)model. Whenever a job reaches the named node at the higher level, it is transferred to the point of the node parameter in the submodel. Similarly, whenever a job in the submodel reaches a node parameter, it is transferred to the corresponding node in the higher level (sub)model.

Lastly, using a fully-qualified path name, a modeler can specify in the routing the next node to visit which can be a node in any submodel. The fully-qualified path identifies the path through the invocation tree, uniquely specifying the desired node. Jobs traverse through that link directly to the specified node.

In addition to providing information to a submodel by assigning values to its parameters in an invocation, global, system, and job variables can be used to pass information among submodels. This information can be used, for example, for routing decisions, assignment statements, distribution means. As a result, decisions in one invocation can depend on activity anywhere else in the model.

The modeler can save the model and all its submodels and/or save a submodel in an individual file, in which case it will become part of the library of submodels. The modeler can change and save submodels within a model, while letting the library version remain unchanged, or can change the library version as well.

When using submodels, it is necessary to quickly switch between the higher-level view and the submodel details. Because of this, we store the complete model and its submodels in memory, so that layering between levels is very fast. There needs to be close coordination between editing in a submodel and corresponding changes to the main model. Changes, for example, in the parameters of the submodel are automatically reflected in the invocation's attributes. Large editing, such as deleting a submodel, or for that matter, a complete subtree is done in two keystrokes - one to delete and one to confirm.

2.3 Evaluate Task

In the Evaluate Task, the modeler provides the values for the model parameters and selects execute to evaluate the model. The underlying system, RESQ, expands the submodel invocations into a flat model and runs the simulation and generates the performance measures for all the objects.

2.4 Output Analysis Task

Selecting Output Analysis, the modeler can view the results at any level in the model hierarchy. The modeler does this by pointing to the icon(s) whose performance measures he wants to see. The corresponding performance measure names are displayed in a pop-up menu. The modeler can select any number of performance measures to plot and positions the resulting graph on the modeling surface along with the network diagram. Performance measures can be viewed for any object on the screen: elementary icons, user-created (invocation) icons, routing chains, the model itself. For example, pointing to an invocation icon allows the modeler to view all performance measures for all nodes in its subnetwork. Alternatively, the modeler can layer down to the detailed network diagram of that submodel invocation and view the performance measures of any specific node by pointing to it.

In the Create/Edit Task, we provide the modeler with the ability to traverse the submodel tree, whereas in the Output Analysis Task, we need to provide the modeler with the functionality to traverse an invocation (instances) tree. This is because, the modeler needs to access the performance measures associated with a specific instance of a submodel in the output analysis task.

For example, assume that there are two invocations, called "emerg1" and "emerg2", of a hospital emergency room submodel, called "emergency", and the submodel emergency has a passive queue named "doctor". The modeler has to be able to distinguish between the performance of the doctors in emergency room 1 from those in emergency room 2. Therefore, in the Output Analysis Task, we set up the menu items "Layer Up" and "Layer Down" to present the invocation tree to the user.

Similarly, when viewing the animation of jobs, the modeler needs to traverse the invocation tree. RESQME animates the jobs and tokens in each layer of the invocation tree. The modeler can move between the layers with the menu commands, to see the animation on any layer.

Alternatively, the modeler can trace the movement of a specific job as it traverses through the layers of the model. If the modeler selects this trace option, the screen view will automatically change to each layer, as the job moves through that layer of the model, i.e. as it reaches an invocation, an output port, a parameter node or a fully-qualified specification. The traced job is shown in a different color to distinguish it from the other jobs, which are also animated.

2.5 Icon Drawing

An icon drawing program is provided so the modeler can draw his or her own icons to represent individual submodels. As mentioned above, a generic submodel icon is also provided in RESQME as part of the elementary icons. The modeler links his or her drawing of an icon to a specific submodel by providing the name of the submodel. The drawing package provides edit commands and line, rectangle and circle elements to create the icon and place it in a palette. The resulting palette of user-created icons is given an application name and is added to the pages of elementary icons for that application, when it is being modeled.

The user-created icons are selected, manipulated, linked in the network exactly as the elementary icons. They have attributes determined by the submodel to which they are linked, and graphically, they are manipulated identically to the elementary icons. They can be rotated, zoomed, panned in the same way as the elementary icons.

3. HOSPITAL EXAMPLE

As a simple example to demonstrate the support provided for graphical hierarchical modeling, we present a model of an ambulance dispatching system containing a dispatching submodel, called "ambulance", and three invocations of a hospital emergency room submodel, called "emergency". Figure 1 illustrates the highest level of the model with two user-created icons, representing the ambulance and emergency room invocations. This specific invocation of the ambulance submodel is called vehicle. Patients enter the system at source nodes

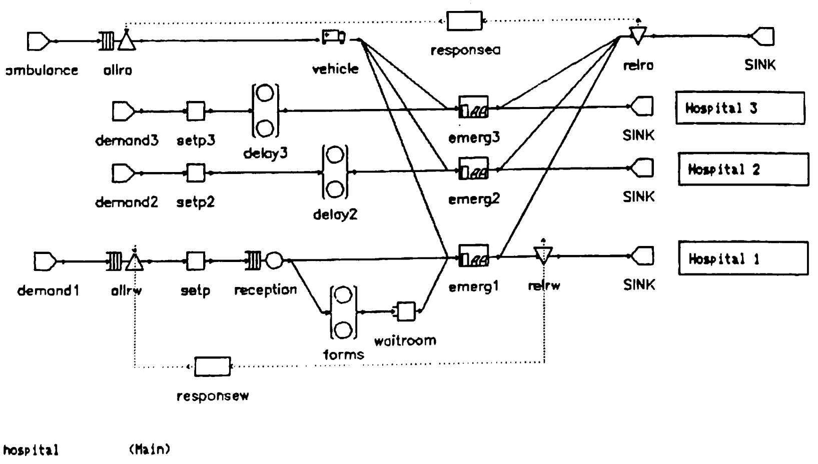

Figure 1. Ambulance dispatching system model

(demand1, demand2, demand3) where they are walk-ins to the emergency rooms in three hospitals in the city, and as demand for ambulances at source node, ambulance. Ambulances can take patients to any hospital. The model was developed in a top down fashion, where first active queues were used for ambulance transport time and time in the emergency rooms and then more details were added. Since there are multiple emergency rooms, a submodel was developed to be general enough to handle all three emergency rooms, using parameters for the number of nurses, the number of doctors, mean service time for walk-in patients, and mean service time for ambulance patients to differentiate the emergency rooms. Detail on the main model was started for the reception area for walk-ins for hospital 1, but is still shown as simply an active queue for the other two hospitals. This detail may later become a submodel that will be invoked for each hospital. Furthermore, the submodel for the emergency room may be included in other hospital models in a bottom-up development fashion.

If we layer down to the submodel which represents the ambulance dispatching, we see the layer of the model shown in Figure 2. The decision as to which emergency room to go to is based on global variables indicating the activity level of each emergency room and the specialty of that hospital and job variables identifying the requirements of the specific patient. The input and output ports are labelled. A global variable is set (in node dest1, dest2 or dest3) to indicate the chosen emergency room, and the appropriate transportation time is selected by the routing. At the main model level, that global variable is used to select the link to the appropriate emergency room invocation.

There are three invocations of the emergency room submodel in the main model diagram. Parameters for number of nurses, number of doctors and service times are used to differentiate the activity among these instances of the emergency room. Layering down to the emergency room submodel, we see the subnetwork diagrammed in Figure 3. The nurses and doctors are represented by passive queues. There are two chains, each with their input and output ports. One is for patients arriving by ambulance; the other is for walk-ins. Walk-in and ambulance patients share the same resources (nurses and doctors), although they require different service times and can have different priorities.

When viewing the animation, the modeler sees at the main level patients arriving from the four source nodes. The ambulance routing chain is shown in a different color than the walk-in routing as are the moving balls representing patients. This distinction is preserved through the submodels emphasizing the different classes of patients.

At any point in the animation, the modeler can select a particular patient (say from the ambulance source node) and follow that patient as it goes through the model. The patient

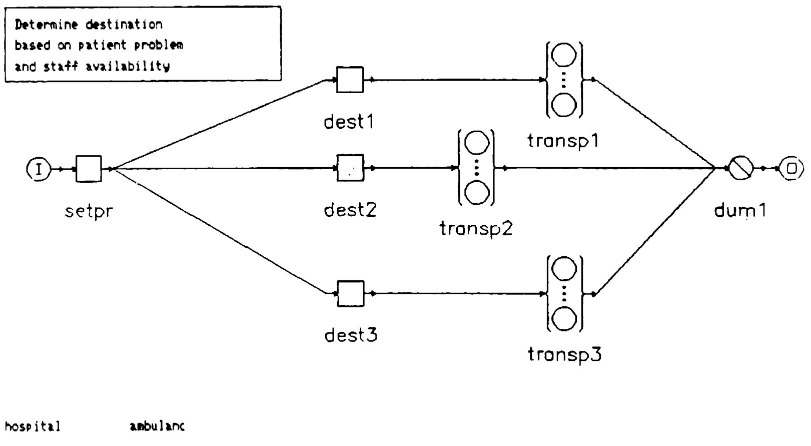
Determine destination based on patient problem and staff availability
setpr
dest1
dest2
dest3
transp1
transp2
transp3
dum1
hospital
ambulanc

Figure 2. Ambulance submodel

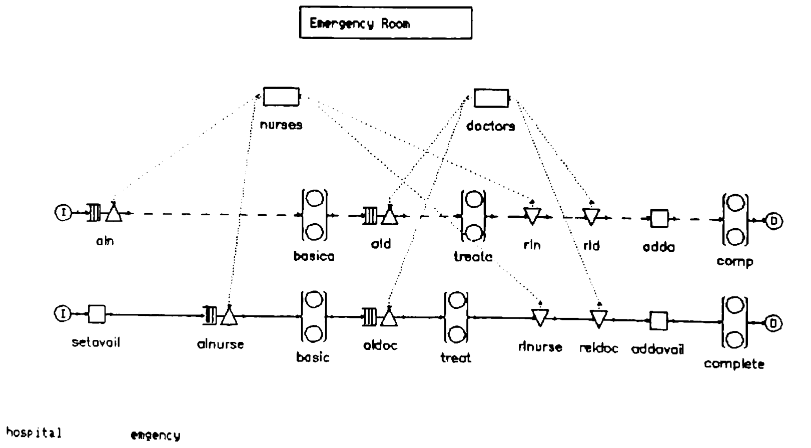

Figure 3. Emergency room submodel

icon will go to the vehicle invocation, layer down to this invocation showing the hospital selection (say hospital 1), then layer back up to the main level as it moves to the emergl invocation, then layer down into that invocation as it waits for nurses and doctors. Simultaneously, the other patients and the queues at each node will be dynamically changing.

4. SUMMARY

Graphics allow the modeler to interface with the computer based on a network diagram. Hierarchical modeling allows the modeler to build a model with reusable, pretested code in both a top-down and bottom-up approach. Together, graphics and hierarchical modeling provide the modeler with the ability to build and experiment with a model that is a visualization of objects in the modeler's application domain.

To provide this power, our modeling environment supports functions to create, view and manipulate both a submodel tree and an invocation tree. The former is used during model building, and the latter is used during output analysis and animation. The modeling environment provides a viewport into each layer of the tree to display its network, performance measures and animation and to edit each layer's contents as well as the tree itself. We also provide an iconic representation of each submodel layer. When placed on the modeling screen and given parameter values, it can be linked to other nodes in the network and creates a layer of the invocation tree.

ACKNOWLEDGEMENTS

We would like to thank Anil Aggarwal, Janet Chen, Paul Loewner and Geoff Parker for their work on RESQME and Charles Sauer for his design work of RESQ.

REFERENCES

Aggarwal, A., K.J. Gordon, J.F. Kurose, R.F. Gordon, and E.A. MacNair (1989), "Animating Simulations in RESQME," In *Proceedings of the 1989 Winter Simulation Conference*, E.A. MacNair, K.J. Musselman, and P. Heidelberger, Eds. IEEE, Piscataway, NJ, 612-620.

Chandy, K.M., U. Herzog, and L.S. Woo (1975), "Parametric Analysis of Queueing Networks," *IBM Journal of Research and Development 19*, 1, 43-49.

Chow, W.-M., E.A. MacNair, and C.H. Sauer (1985), "Analysis of Manufacturing Systems by the Research Queueing Package" *IBM Journal of Research and Development 29*, 4, 330-342.

Concepcion, A. and S. Schon (1986), "SAM - A Computer-aided Design Tool for Specifying and Analyzing Modular, Hierarchical Systems" In *Proceedings of the 1986 Winter Simulation Conference*, J.R. Wilson, J.O. Henriksen, and S.D. Roberts, Eds. IEEE, Piscataway, NJ, 504-510.

Courtois, P.J. (1975), "Decomposability, Instabilities and Saturation in Multiprogramming Systems" *Communications of the ACM 18, 5*, 371-377.

Courtois, P.J. (1978), "Exact Aggregation in Queueing Networks," *Proc. First Meeting AFCET-SMF*, Paris, 35-51.

Gordon, R.F., E.A MacNair, P.D. Welch, K.J. Gordon, and J.F. Kurose (1986), "Examples of Using the RESearch Queueing Package Modeling Environment (RESQME)," In *Proceedings of the 1986 Winter Simulation Conference*, J.R. Wilson, J.O. Henriksen, and S.D. Roberts, Eds. IEEE, Piscataway, NJ, 504-510.

Gordon, R.F., E.A MacNair, K.J. Gordon, and J.F. Kurose (1987), "A Visual Programming Approach To Manufacturing Modeling," In *Proceedings of the 1987 Winter Simulation Conference*, A. Thesen, H. Grant, and W.D. Kelton, Eds. IEEE, Piscataway, NJ, 465-471.

Kurose, J.F., K.J. Gordon, R.F. Gordon, E.A. MacNair, and P.D. Welch (1986), "A Graphics-Oriented Modeler's Workstation Environment for the RESearch Queueing Package (RESQ)," In *1986 Proceedings Fall Joint Computer Conference*, Dallas, 719-728.

MacNair, E.A. (1985), "An Introduction to the Research Queueing Package," In *Proceedings of the 1985 Winter Simulation Conference*, D.T. Gantz, G.C. Blais, and S.L. Solomon, Eds. IEEE, Piscataway, NJ, 257-262.

MacNair, E.A. and C.H. Sauer (1985), *Elements of Practical Performance Modeling*, Prentice-Hall, Englewood Cliffs, NJ.

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

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

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

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

Thomasma, T. and O. Ulgen (1988), "Hierarchical, Modular Simulation Modeling in Icon-based Simulation Program Generators for Manufacturing," In *Proceedings of the 1988 Winter Simulation Conference*, M.A. Abrams, P.L. Haigh, and J.C. Comfort, Eds. IEEE, Piscataway, NJ, 254-262.