AN EXTENSIBLE VISUAL ENVIRONMENT FOR CONSTRUCTION AND ANALYSIS OF HIERARCHICALLY-STRUCTURED MODELS OF RESOURCE CONTENTION SYSTEMS

K. Gordon, J. Kurose, R. Gordon, E. MacNair

COINS Technical Report 89-93

An Extensible Visual Environment for Construction and Analysis of Hierarchically-Structured Models of Resource Contention Systems

Kurtiss J. Gordon and James F. Kurose [1]

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

Robert F. Gordon and Edward A. MacNair

IBM Research Division
Thomas J. Watson Research Center
Yorktown Heights, NY 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 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 a single integrated graphical interface throughout all tasks of the modeling lifecycle. In this paper we briefly overview RESQME and then focus on two of its most important features: the construction and analysis of hierarchically-structured models and the ability to extend and customize the RESQME environment for domain-specific modeling via the use of user-defined modeling elements. A manufacturing model is developed in order to illustrate these capabilities.

[1]Supported in part by an equipment grant from the National Science Foundation CER DCR 8500332.

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) [21]. 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. A considerable amount of work is often required to build, modify, and maintain these programs and to communicate the model structure and model results to others. During the past decade, however, a number of efforts have been undertaken to help alleviate these problems by adopting the so-called visual interactive approach [17] towards performance modeling. At the heart of this approach is "an animated graphics representation of the system under investigation" [18]. A number of software packages which embody this approach are now also commercially available [5,9,10,11,12,13,14,26,29,38]; a recent survey of the capabilities of such packages can be found in [22]. Generally, capabilities are provided in these packages for graphically constructing and editing a performance model, graphically presenting and analyzing model output results, and displaying an animated rendering of the dynamics of the system being modeled.

The advantages of this visual interactive approach are said to be many; the advantages (as well as possible disadvantages) have been extensively discussed in the literature [2,4,18,25,20,27,33]. Some of the cited advantages are based on the belief that a visual representation of a model and its results is, in some sense, the most "natural" and hence there is no need in the visual approach to translate from the basic diagram to the syntax of a textual language, nor the need to translate from tables of output statistics to plotted results. Modeling speed and clarity of model representation are also cited advantages. Visual approaches which employ animation provide additional advantages as well. Models may be more easily debugged due to the modeler's enhanced ability to visually trace the movement of jobs through the model. The modeler may also obtain a qualitative understanding of complex phenomena by observing the evolution of the system "in the lab". Observing the evolution of the system offers the possibility of observing and discovering phenomena which otherwise would have been "washed out" in steady-state statistics. Moreover, animation provides an enhanced ability to communicate the model design and model results to others.

In this paper, we briefly describe the Research Queueing Package Modeling Environment (RESQME) [15,16,20], a graphical environment for constructing, solving, and analyzing the results of extended queueing network models of resource contention systems, and focus on what are the most novel features of RESQME:

- RESQME provides a **common graphical interface to all aspects of the modeling lifecycle.** In RESQME, a *single, uniform* graphical representation of a model is used throughout the *entire* modeling process, from model construction and editing, to model solution, to output analysis and animation; as far as the user is concerned, the picture *is* the model [19]. A model may be graphically constructed, edited, and documented. In the case of simulation, the solution may be animated using the same model diagram; performance results are also displayed, manipulated, and examined using this model diagram. The modeler can thus think in terms of the single graphical representation of the model, and need no longer be concerned with either translating his/her vision of the problem into the syntax requirements of a textual representation or with translating from one pictorial representation of the model (e.g., the model diagram, which specifies the model) to another (e.g., the artistic rendering of the system being modeled, which is used for animation). In the field of software engineering, the need for such a single, common representation of a program (via a "wide spectrum" language) through all tasks in the program development process has also been noted [36].

  The only other major efforts of which we are aware which integrate a common graphical interface throughout the modeling process are PAW [25] and XCELL+ [10]; our present effort has been influenced by these efforts (particularly the former) but differs from them primarily in the extensibility of RESQME and the support provided for creating, solving, and animating large, hierarchically structured performance models, the primary topics of this paper. RESQME's maintenance of a single graphical representation of a model contrasts with other visual approaches which also use graphics throughout the modeling lifecycle but employ different visual representations of the model in different tasks of the modeling process. For example, the graphical representation of a SIMAN model constructed using BLOCKS [34] is different from the graphical rendering used to display the model dynamics during an animated simulation [26]; similarly the SLAM II network graphically constructed using TESS is different from the "schematic facility diagram" used in a TESS animation [14]. Our approach of integrating a common visual representation of the model throughout the modeling lifecycle also contrasts with tools in which a graphical approach is used *only* in model construction (e.g., [5,35,37]), primarily for *post facto* animation of a model simulation (e.g., [26]), or requires the use of textual instructions in the underlying simulation language to control aspects of the graphics generation process (e.g., [30]).

  We believe the major advantage of maintaining a single graphical representation of the model is the ability to move through all tasks of the modeling process through a consistent model interface without the need to conceptually translate from one representational form of the model to another. Once the model is constructed, it can also be animated without further modeler effort, since it is this single visual representation of the model, and not a separate graphical rendering, that is animated. Finally, since the constructed performance model is directly animated, the use of animation as a model debugging aid is greatly facilitated.

  A potential disadvantage, however, is that during animation, the graphical model diagram is visually more primitive than the sophisticated 3-D renderings that can be created using the sketching capabilities provided by some of the animation packages (e.g., [14,26]). The importance of graphic design in visual interactive modeling is noted in [2]. With separate visual representation of the model and its animation, it is also easy to animate a single performance model in a variety of different ways, thus providing complementary visualizations of the dynamics of the simulated system.

- **Support for hierarchical modeling.** Many "real world" performance models can be quite large and complex. For example, RESQ users have built extended queueing network models with well over 4,000 nodes in the model. RESQME was designed to provide capabilities specifically for handling such large models. In particular, RESQME provides graphical support for the construction, solution, and animation of hierarchically-structured performance models. The hierarchical structure of a model is based on the use of submodels, which are parameterized templates of an interconnected set of RESQ modeling elements. The use of submodels was a key component of RESQ [31]; related structuring mechanisms have been adopted by later modeling languages as well [28].

  The importance and advantages of such hierarchical model structuring techniques have been noted [8]. It is often claimed (both within the modeling community as well as in the field of software engineering) that modular, well-structured models (particularly large ones) are more easily understood and more easily maintained (and hence tend to be more error-free). As discussed in section 3 of this paper, hierarchical structuring techniques also permit modular, pre-tested model fragments to be shared across many models. Graphics-oriented tools have recently begun to emerge for specifying and constructing such structured models [7,37]. RESQME is unique, however, in providing graphical support for hierarchically-structured models throughout the entire modeling process. We discuss hierarchical modeling in RESQME in section 3 of this paper.

- **Extensibility.** RESQME provides capabilities which permit the performance analyst to define higher-level modeling elements (by composing the lower-level RESQ modeling primitives into a submodel), to create new icons for these new modeling elements, and then to integrate these new constructs into RESQME. The extensibility of RESQME is thus closely tied with the use of submodels and, more generally, the development of hierarchically-structured, modular models [8].

  In the manufacturing domain [6], for example, a modeler might define modeling "primitives" corresponding to a manufacturing cell or assembly station; in the communication domain [32], 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 to build application-specific performance modeling packages. (We note that previous application-specific modeling packages have already been successfully built upon the text-only version of RESQ [3]). The ability to define and manipulate new modeling primitives is discussed in section 4 of this paper.

The remainder of this paper is structured as follows. In the following section we provide a brief introduction to RESQ, present an overview of the RESQME modeling environment, and develop a manufacturing performance model which will be used for illustrative purposes in the remainder of the paper. Section 3 describes the submodel facility of RESQME and the visual construction, solution, analysis, and animation of hierarchically structured models. Section 4 describes the definition and use of user-defined modeling objects in RESQME and illustrates how this capability of RESQME can be used to construct a customized, domain-specific, visual modeling environment on top of RESQME. Section 5 concludes this paper.

2. Overview of RESQME

2.1 The Research Queueing Package (RESQ)

We begin this section with a brief overview of the RESQ language, since it forms the foundation upon which RESQME is built and demonstrates the wide range of modeling capabilities that must be supported by the environment. For a more detailed discussion of RESQ, the reader is referred to [31,32,23].

The RESQ modeling language provides the modeler with a rich set of high-level modeling primitives with which performance models of resource contention systems may be *specified* and *evaluated*. At the highest level, a RESQ model consists of

- a population of jobs,
- a set of queues,
- a set of nodes,
- rules specifying how the jobs circulate among the nodes and queues,
- additional modeling constructs, and
- information concerning the solution of the model.

Queues typically represent resources in the system being modeled; jobs represent the objects in this system which require the use of these resources. RESQ provides two type of queues: *active* queues and *passive* queues. Active queues have servers which provide service to a job. An active queue might be used to represent, for example, a computer's central processing unit, a communication link in a network, or an assembly station in a flexible manufacturing system. A job arrives at an active queue, waits until it is selected for service (according to the user-selected service discipline), receives service, and eventually leaves the queue. Passive queues allow for a more general notion of resource use and provide a natural mechanism for modeling phenomena such as simultaneous resource use and blocking. A passive queue consists of a pool of tokens, where each token typically represents a distinct unit of some allocatable resource (e.g., a buffer or a block of memory). Associated with each passive queue is a set of nodes, at which a job may queue while waiting for the requested number of tokens to become available and allocated to the job, or at which a job may deallocate, create, or destroy tokens associated with the pool of tokens. A job may hold tokens from a passive queue as it visits other nodes or (active or passive) queues in the model.

Additional nodes are provided by RESQ in order to model phenomena other than strict resource contention. For example, if there are external arrivals or departures to/from the system being modeled, *source* and *sink* nodes (respectively) can be used to model these effects. *Set* nodes permit values to be assigned to variables associated with the simulation. *Split* nodes allow a job to create an independent copy of itself. *Wait* nodes cause a job to be delayed until a specified Boolean condition becomes true. These and other additional auxiliary nodes are described in [31,32,23]. 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). 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. RESQ also provides numerous modeling constructs in addition to those discussed above, such as symbolic constants and parameters, arrays of nodes, and submodels. The submodeling capability of RESQME is discussed in section 3 of this paper.

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 (see, e.g., Chapter 6 in [21]).

2.2 The RESQME Environment

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. The unique features of RESQME were outlined in the introduction to this paper, and will be discussed in more detail in sections 3 and 4. RESQME also contains many of the basic capabilities found in other visually-oriented modeling environments [22]. In this section, we briefly overview the functionality in RESQME; additional details can be found in [1,15,16,20].

RESQME runs on a workstation consisting of an IBM personal computer with an all-points-addressable graphics display, a pointing device (e.g., a mouse) and a communications adapter connecting the workstation to a host computer. It is written in the C programming language, and the device-independent graphical support is provided by VDI. As with most visual modeling environments, the RESQME modeler controls the modeling process by selecting menu items and directly manipulating objects on the display.

Figure 1 shows the RESQME display with the main area providing a viewport into a portion of the model. The lower part of the screen contains the commands needed to complete the modeling process, and the right vertical menu contains the commands to manipulate the screen.

Information about the model is provided at two levels. At the most visible level is the graphical view—the network diagram, its elements and submodels, animated job flow, and output charts. At a second level, there is attribute information for each graphical object. For example, a queue icon object has attributes consisting of its name, type of queue, queueing discipline, service time, etc. Similarly, an output chart object has attribute information defining its chart type (line, bar, histogram), colors, numerical values. The network diagram has attribute information, such as the model name, solution method, parameter names, run control limits. The attribute information appears in a pop-up window whenever the modeler chooses to look at it. The important point is that some information is easier to specify or more meaningful to display in a graphical form, while other information is more appropriate in a textual form. The combined graphics and text define the model.

Interaction with RESQME divides broadly into three tasks which comprise the modeling process. During any of these tasks, the modeler can pan, zoom, and locate objects by name on the model diagram. During the Create/Edit task, a model is constructed by picking icons representing the RESQ modeling primitives from a palette and placing them on the modeling "canvas". Textual attributes associated with an icon are then specified in a context-sensitive pop-up window (i.e., a window whose contents dynamically vary according to responses entered in the window). Job flow among the elements within a model is specified simply by connecting the icons using the pointing device. Since most models contain one or more branch points—icons from which a job may go to one of several other icons (typically based either on a probabilistic decision or on some simulation-dependent condition)—we placed considerable emphasis on allowing routing from one-to-one, one-to-many, many-to-one, and many-to-many with a minimum of effort. The attribute pop-up window is used to specify the routing conditions.

The second task in the experimental process is to evaluate the model. RESQME allows the modeler to provide sets of parameter values for each desired run of the model and then to execute the series of runs on the host computer. The host execution is done in the background as far as the modeler is concerned, so that the modeler can continue to work at the workstation on this or other models while the host is processing the model. The concurrent processing frees the modeler to continue working without having to wait for the simulation run to be completed. This cooperative processing takes advantage of the host for the computation-intensive execution of the model and the workstation for the interactive graphics.

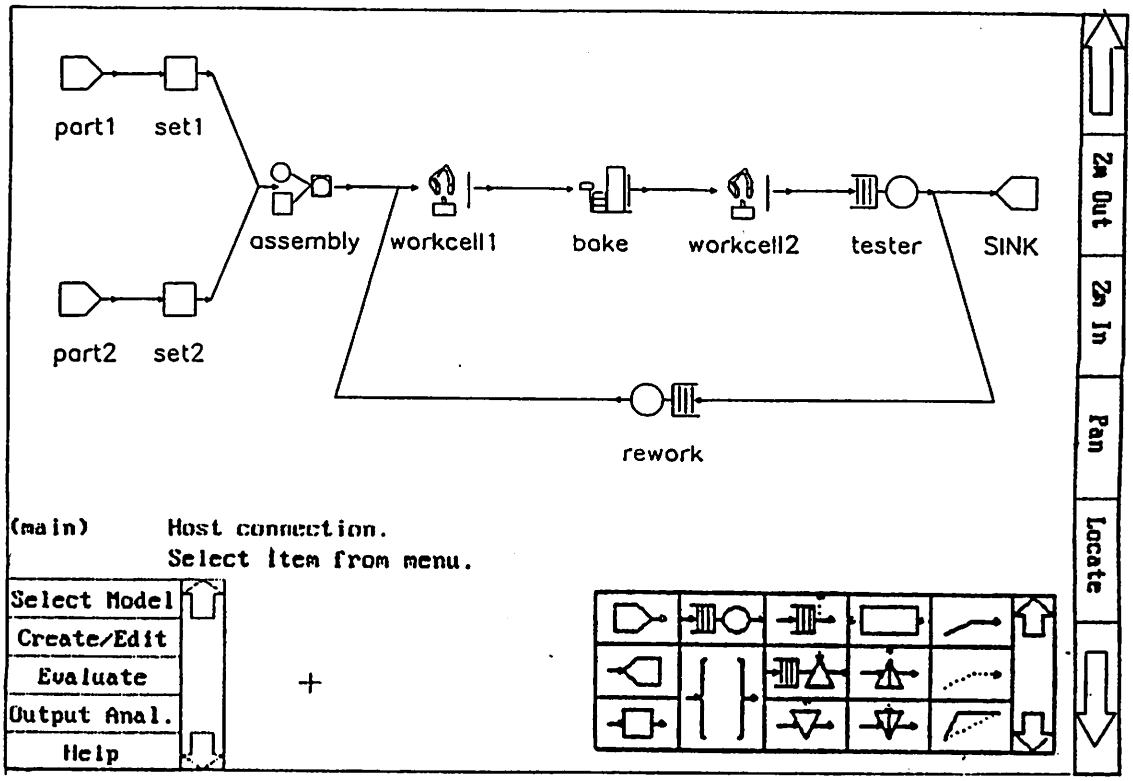
The final task is the Output Analysis phase, which again uses the model diagram as the primary interface to the model. RESQME supports the output analysis task by providing a general-purpose plotting package to graph the collected performance results. Performance results related to a specific icon or routing chain are selected simply by pointing to that icon or chain and then selecting the performance measure(s) of interest from the pop-up window of available measures. The modeler can analyze performance measures from one model run or across runs, for one node or for many nodes, and with a number of different plotting options. Animation of the job and token flow resulting from a simulated run is also available in a post-processing mode, again with the single graphical representation of the model used for animation purposes. Animation in RESQME is described in [1].

Finally, we note that RESQME was specifically designed to be used by both novice and experienced users. The novice modeler can play through tutorial scripts which demonstrate (via animated example) various aspects of the system. We have built a large number of tutorials using a record and playback facility in RESQME. The tutorials provide a mechanism for teaching new users about simulation methodology, the use of RESQME, and the structures of RESQ. They also provide the expert with a mechanism to design customized, animated tutorials in order to, for example, document or explain a model that has been developed. For the expert, RESQME contains features which help expedite the modeling process. For example, there are "modes" of operation which permit easy repetition of the same task, library capabilities, journaling functions, and confidence-interval run control. For both novice and expert, there are libraries of submodels that can be used to construct models, default values for many items, and scrollable pre-defined responses for many prompts.

2.3 A RESQ Manufacturing Model

The model of a manufacturing application discussed in this section is used throughout the remainder of the paper to illustrate various aspects of RESQME and will be referred to extensively in sections 3 and 4. This is not a model of an actual system, but rather a simple, illustrative model. Although this is a simpler system than is typically used in the field, it contains many of the modeling elements found in actual models and thus will provide a useful vehicle for explaining some of the modeling elements of RESQ.

In the hypothetical system represented by our model, two types of parts arrive at an assembly station to be merged. The merged parts go to a robotic workcell, undergo a baking operation, go to another robotic workcell, and then to a testing station. Those parts that fail the test undergo some rework and are routed back to the first robotic workcell. The good parts leave the model



(main) Host connection.
Select item from menu.

Figure 1: Flow of parts

at the SINK. The model is constructed hierarchically, with submodels representing the processing that takes place at the assembly, workcells, and baking operations. The flow of parts through the model is shown below in Figure 1 by the solid lines.

The two types of parts enter at the nodes labeled PART1 and PART2. These nodes are source nodes which generate parts according to a specified interarrival time distribution. Each part type then goes to a set node (SET1 and SET2 in the model diagram). Set nodes are used to execute assignment statements. At SET1 and SET2, the parts are assigned a part type number. From this point, each part flows into the assembly process.

The assembly process is depicted by a submodel which merges two types of parts. The submodel has parameters for the number of each part type to be merged and each part's identification number. Figure 2 displays the paths that parts follow for the assembly process. The dotted lines in the figure represent the flow of tokens at passive queues. A submodel has a primary entry point and a primary exit point; the node labeled DIN is the primary entry point for the merge submodel. Node DIN is a dummy node, which is used here to provide a convenient place at which jobs can make routing decisions. One type of part follows the upper path, and the second type follows the lower path. The set nodes SETADD1 and SETADD2 count the number of each part in the next group of parts to be merged. The remaining processing in the submodel provides a place for jobs to wait until all of the parts for the next assembly are ready to be put together.

The type of synchronization required by this model is typically accomplished in RESQ by using

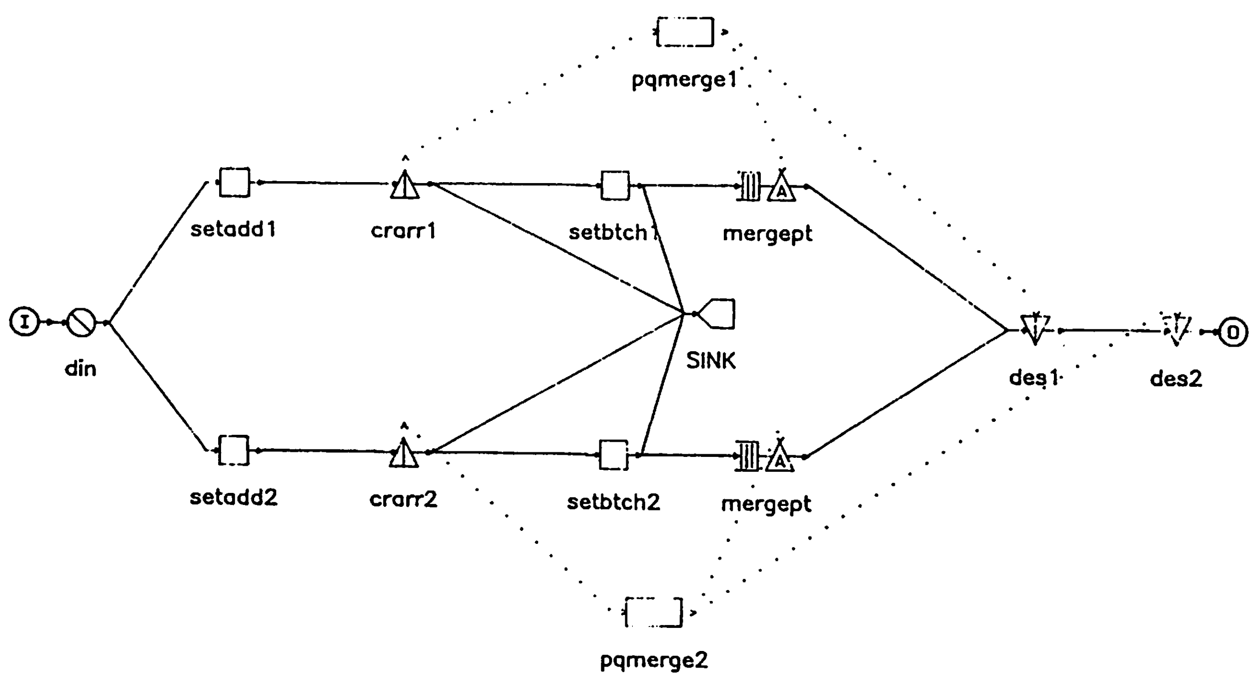


Figure 2: Merge submodel

a passive queue. Recall that a passive queue normally has a pool of a finite number of elements called tokens to be allocated to jobs. The jobs wait at an allocate node to be allocated the requested number of tokens from the pool. The jobs then hold onto the tokens while they visit other nodes until they eventually release the tokens back to the pool or destroy them. New tokens may also be created to increase the number of tokens available. In our model, after incrementing the count of the number of parts for the next assembly, each part type creates a token indicating that another part is ready to be assembled. If the part is the first part in an assembly, the part goes to a set node, SETBTCH1 or SETBTCH2, where the assembly number is incremented. The part then waits at an allocate node, MERGEPT, until all parts for the next assembly have arrived. If the part is not the first part in an assembly, it is sent to a SINK node to leave the model. Only the first part in each assembly will leave the submodel and, when it leaves the submodel, it represents the entire assembly. The MERGEPT node is a special type of node known as an AND allocate node. AND allocate nodes belong to multiple passive queues. The first part in each assembly waits at the AND allocate node until all parts in an assembly have arrived. One token for each part in the assembly is allocated to the part and then destroyed at nodes DES1 and DES2. The assembled job then leaves the merge submodel and goes to the first workcell.

WORKCELL1 and WORKCELL2 are invocations of the robot submodel, which is based on a paper by Medeiros and Sadowski [24]. The flow of jobs is depicted in Figure 3. There is an input staging area where the part is oriented, a robot which moves the part to a machine for processing,

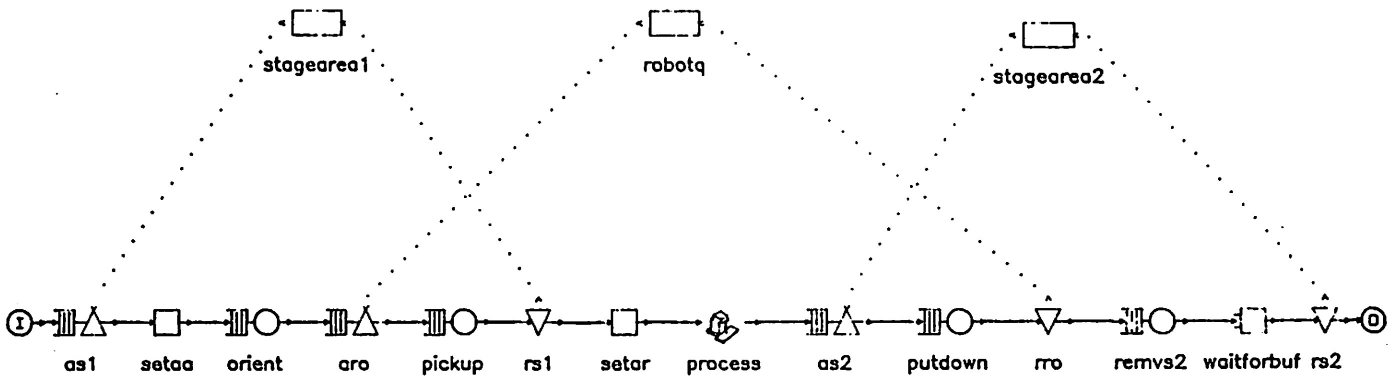


Figure 3: Robot submodel

stays with the part, and moves it to an output staging area. A passive queue is used to represent the staging area, in which the parts are oriented one at a time. The robot is also modeled as a passive queue. The oriented parts wait at the ARO allocate node until the robot is available. The robot then picks up the oriented part, releases the orientation station, and stays with the part as it goes to the processing step represented by the failure submodel. Here we see an example of a submodel nested within a submodel. After the part has completed the processing step, the robot puts it down at the output staging area if it is free. If the output staging area is occupied, the robot waits with the part. When the robot puts the part at the output staging area, it becomes free to pick up the next oriented part. The part at the output staging area takes some time to be removed and then waits if the buffer at the bake operation is full. By checking on the availability of buffer space, we are modeling the pull method of moving parts through the system. When a buffer is available, the part leaves the output staging area.

The failure submodel is a simple process step. If the machine is up, the parts undergo processing and leave. There is a special controlling job which represents failures. When the controlling job is at the UP node, the machine is up and running. When it goes to the DOWN node, it gains control of the PROCESS machine and any part in process is interrupted (and requeued for service) until the machine is repaired; this is illustrated in Figure 4.

From WORKCELL1 the parts flow to the BAKE operation. The bake operation is a submodel which accumulates batches of parts until there are enough to fill the bake oven. Then all parts

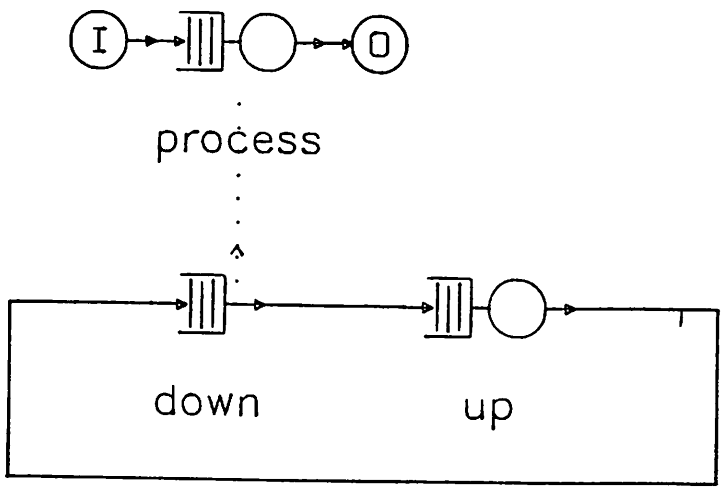


Figure 4: Failure submodel

in a batch are baked simultaneously. Figure 5 shows how the batching is accomplished. Parts in each batch are counted. The last part in each batch causes all parts in the batch to be released to the PROCESS step which accomplishes the baking. This is implemented with a passive queue by holding all parts in a batch at the BATCHWAIT allocate node until the last part creates enough tokens at STARTBATCH for all parts in the batch to progress. When parts finish the process step, they check the buffer availability at WORKCELL2. When a buffer is free at WORKCELL2, the part goes to another copy of the robot submodel.

3. Hierarchical Modeling in RESQME

Realistic models of complex systems can be made more tractable by providing an environment that supports modularity. System models can be built in such an environment by selecting and linking preprogrammed modules together in a bottom-up approach. Models can also be described top-down in such an environment as a network of black boxes whose details are further developed in stages. Furthermore, different users of the system model may want to view it at different levels of detail, and the modeler will find it conceptually useful to visualize the system at different levels of detail. Modules can be replaced by other modules to analyze alternative formulations. RESQME supports these multilevel modeling concepts with hierarchical model definitions.

The conceptual unit in the hierarchy is a *submodel* definition—a collection of RESQME primi-

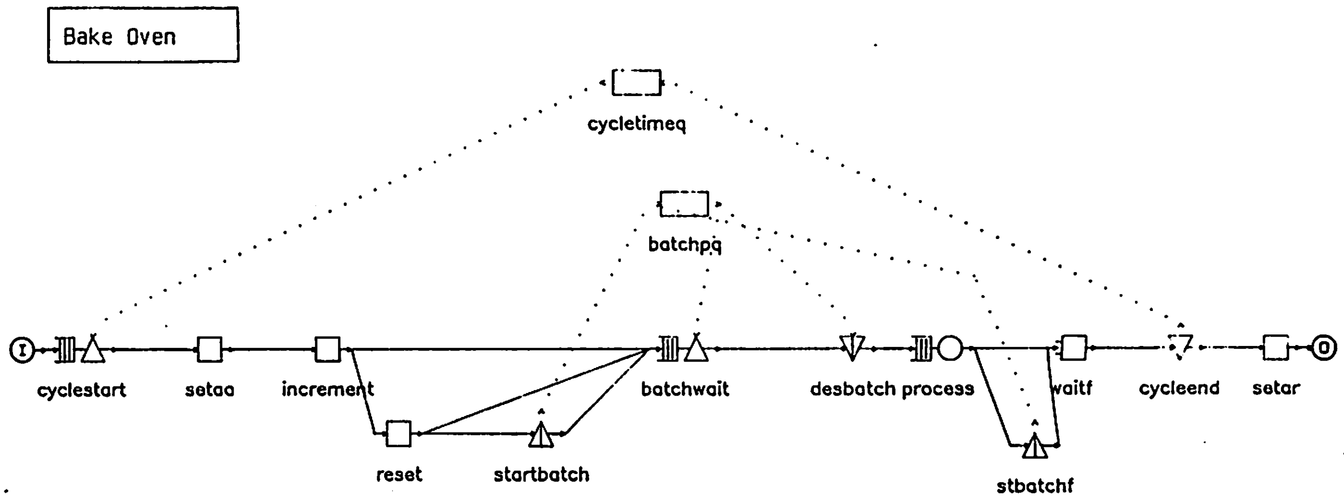


Figure 5: Bake submodel

tives and/or other submodels and the routing connecting them to one another and to the input and output connections of the submodel. In RESQME, each submodel is defined on its own modeling canvas, allowing the full modeling surface to be used for each submodel and for each submodel to have individual zooming sizes. Each submodel has attributes that identify its type (submodel name) and its parameter names. The model's submodels are related to one another as the nodes of a tree.

The submodel construct in RESQ provides a means for defining a parameterized template of an interconnected "subnetwork" of nodes and queues. This submodel may then be *invoked* any number of times (in much the same way that a macro may be invoked any number of times in a programming language) to create multiple instances of that subnetwork. Rules which determine the lexical scope of the names and expressions used within a submodel are also similar to those governing macro expansion in programming languages. Submodels provide an important mechanism for constructing modular, hierarchically-structured models of large and complicated systems and were an integral part of the initial design of the RESQ language [31]; a related structuring concept (the "station") has also been introduced in another text-based modeling language [34]. RESQME provides explicit graphical support for such hierarchical modeling throughout the modeling process. This is in contrast to programs such as SAM [7], BLOCKS [34], and the work of Thomasma and Ulgen [37], which provide differing amounts and type of support for *specifying* hierarchically-structured models, and to the "schematic facility diagram" [14] which permits an animated rendering of the system

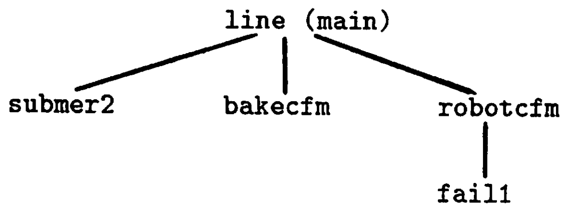


Figure 6: Submodel tree

being modeled (as opposed to the model definition itself) to be hierarchically constructed.

Our example model of an assembly line is organized as a hierarchy tree of three levels. The submodel tree for the example model is shown in Figure 6. This model may have been originally constructed in this hierarchical fashion, because there may have existed submodels for the different physical workstations on the assembly line, and the modeler then built the line by selecting and sequencing these prefabricated modules Alternatively, the modeler may have begun with a flat model and, while adding detail, may have seen natural groupings or repeating sequences (such as the two robot workcells) that would be more meaningful to encapsulate into a submodel. In any case, we show the resulting hierarchical model which provides increasing levels of detail as the modeler expands the submodel invocations by moving down the levels of the tree.

At the root (main) level, the network consists of nodes for the two sources of jobs, the set nodes that assign the values to each type of job, the tester and the rework active queues and the sink for completed jobs. This level of the model also contains submodel invocation nodes. Any node (including invocation nodes) is placed on the modeling surface by selecting it from the icon palette with the mouse and pointing to the desired position on the modeling surface. An attribute pop-up window is automatically displayed then to allow the entry of textual attribute information for that node. For the set node, for example, the textual information would prompt for the assignment statement, whereas for the submodel invocation node, the textual information would prompt for the submodel type and the specific parameter prompts for that submodel type. In all cases, the modeler can fill in the values for the textual attributes at any time before running the model. The nodes at a given level are connected in the routing by selecting the job chain icon and pointing to the nodes in sequence to form the desired network. Probabilities and conditional statements used in routing definitions are entered in the attribute pop-up window associated with each routing chain.

The modeler can also create and save submodels for insertion into other models. The saved submodels comprise a library of definitions, from which the modeler can recall a desired submodel for insertion into the hierarchy tree of another model. This feature allows for the sharing and reusing of pre-tested model "building blocks." An obvious advantage of this approach is that collections of submodels designed for a particular application domain can greatly reduce the modeling effort by allowing the modeler to build upon previous work. It is also possible to create submodels which describe the same component or subsystem at different levels of accuracy or detail or, perhaps, which model differently designed components performing the same function in different versions of the system under study; these library definitions then serve as interchangeable building blocks in the construction of the model. In our example, we show submodels for continuous flow workcells. These could be replaced, for example, by related submodels that have infinite buffers and no wait nodes to model a "push" system instead of the illustrated "pull" system. Other submodel variations can be accomplished with parameters.

The modeler traverses the different layers of the hierarchical model with two screen-management menu items. The "Layer Down" menu item of the screen-management menu, allows the modeler to traverse down the various levels of the hierarchy tree, in order to view, add or delete submodels at any lower level of the tree. Similarly, the "Layer Up" menu item allows the modeler to move up the hierarchy tree.

When the "Layer Down" menu item is selected from within the Create/Edit task, the user is provided with a list of the submodels that are children of the currently displayed submodel of the tree, as well as the menu items "New" (which is used to create a new submodel) and "Library" (which is used to select a submodel from the library.) If the user either points to a submodel name in the list or to an invocation node in the diagram, the modeling area would then display that submodel diagram, making it the "current" node of the submodel tree. If "New" is selected, the user is provided with a new modeling area and allowed to build a submodel which would be attached at this point of the hierarchy tree. If "Library" is selected, a list of submodels in the library is displayed; any one of which when selected would be attached as a child of the current node of the hierarchy tree, and its diagram displayed. The user can delete subtrees by selecting the delete menu item and pointing to the "Layer Down" menu item. After verification, the subtree rooted at the current submodel is removed.

Selecting the "Layer Up" menu item from within any task, provides the user with all the ancestor levels above the current level. Selecting the desired ancestor moves the user up the hierarchy tree and displays that selected submodel's diagram.

In the Create/Edit task, the modeler traverses and edits the submodel tree (Figure 6) as described above. However, a different tree is needed when analyzing the results, i.e., either viewing the performance measures or animating the flow of the jobs and tokens. In the Output Analysis task, the modeler needs to be able to view the results for each specific *invocation* of a submodel and consequently the layer menu items traverse an *invocation* tree in the Output Analysis task (as opposed to the submodel tree in the Create/Edit and Evaluate tasks.) The user interaction is similar, except that the menu shown when the layer command is selected displays invocations names (as opposed to submodels when in the Create/Edit or Evaluate tasks.)

In our example model from the previous section, at the main level, there is one invocation of submodel SUBMER2 called ASSEMBLY, one invocation of submodel BAKECFM called BAKE, and two invocations of submodel ROBOTCFM called WORKCELL1 and WORKCELL2. In turn, each invocation of ROBOTCFM contains an invocation, called PROCESS, of the submodel FAIL1.

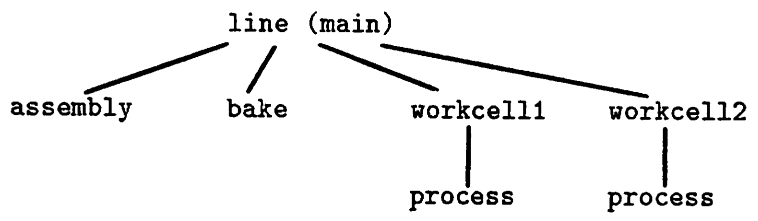


Figure 7: Invocation tree

The corresponding invocation tree is shown in Figure 7.

Each invocation is uniquely identified by its path name, e.g., WORKCELL1.PROCESS is distinguished from WORKCELL2.PROCESS. In the Output Analysis task, the user can display the performance measures for the nodes of a specific invocation, such as the mean queue length at the ROBOTQ node of WORKCELL2 (queue length of WORKCELL2.ROBOTQ) or the mean service time of the UP node of the invocation PROCESS within invocation WORKCELL1 (service time of WORKCELL1.PROCESS.UP).

To select performance measures for a given node in a given invocation, the modeler layers to the invocation containing that node and points to the desired node. A pop-up menu lists the performance measures associated with that instance of the node. The modeler can then select any number of the performance measures to form a chart to plot. It is also possible to point to any other node to add its performance measures to the same chart or to plot one performance measure against another. The charts are kept in memory along with the network for each invocation and thus, as the modeler layers between invocations, both the visual network diagrams and their selected associated performance measure charts are displayed.

It is also useful in some modeling situations to be able to have an array of invocations, the number of elements of which can vary. RESQME supports this concept with a node array icon which, when combined with another icon (in this case, an invocation icon), produces an arbitrarily sized array of these icons. A single icon is used to graphically represent the array of icons; to layer to a specific element of the invocation array, the modeler can either select it by name from the resulting pop-up window of invocations by choosing "Layer Down" or by pointing to the node and then specifying the desired array element by number.

The animation subtask in the Output Analysis task also requires a means of traversing the invocation tree. The animation shows the movement of jobs and tokens and the change in queue lengths of the nodes in the model and in the submodel invocations. RESQME can display the animation for any selected invocation or, at the modeler's option, it can trace a given job as it moves up and down through the invocations of the model. In the latter case, the animation will automatically layer to each invocation that the job visits. During this tracing of a given job, the status at queues and the movement of other jobs and tokens for each displayed invocation are also shown.

4. Extending RESQME into a Customized Modeling Environment

In this section, we describe how the performance modeler may extend the RESQME modeling environment by defining higher-level modeling elements. This can be accomplished by composing the system-defined RESQME modeling primitives or other existing submodel definitions into new submodels, creating new icons for these new modeling elements, and integrating these new elements into the set of available modeling constructs. The modeler can then use these higher-level objects as well as the RESQ modeling primitives in constructing a performance model.

By defining an appropriate set of such high-level constructs, a modeler can thus use RESQME as a base system upon which to build an application-specific performance modeling environment. We note that extending RESQME in this manner (i.e., by using the system-supplied modeling primitives to "build up" a higher level modeling environment) is quite different from another way in which modeling languages and environments can be extended—by "dropping down" [22] into some underlying modeling or programming language (e.g., the manner in which RESQ can drop down into PL/I, Siman and Slam can drop down into Fortran [22], and Witness can drop down into See Why [13], to program at a lower level a construct which is difficult to model using the basic low-level primitives of the language).

As previously mentioned, RESQME has a generic invocation icon that can be used to invoke any submodel. When this icon is selected and placed on the modeling surface, the modeler must supply the desired submodel's name to form the association. However, a user also can create customized icons by drawing a picture for each new icon and then specifying the submodel it should represent (automatically invoke). The higher-level object (i.e., the newly-created customized icon) contains the parameter names of its associated submodel in its attribute list. Each instance of the higher-level object can thus invoke its associated submodel with different argument values.

The purpose of these higher-level modeling elements is to make it possible for the modeler to work directly with the modeling constructs which closely represent the objects in a specific problem domain. The provision of such application-specific objects reduces even further the effort required from the modeler to translate real-world problems into and out of the descriptive form mandated by modeling software. In the assembly line example, higher-level objects have been created to represent a robot, a bake oven, an assembly unit and general machine failure. The modeler can work at the level of these building blocks and provide the parameter values, such as the robot speed and the batch size for the bake oven, to construct the model.

RESQME provides capabilities to draw new icons, to associate them with submodels (thereby creating higher-level objects, and to add them to the palettes of system-defined RESQ icons). The modeler creates a new object by (1) creating a submodel (or selecting an existing submodel), and (2) drawing the new icon and linking it to the submodel. The two steps can be done in either order. We provide an icon-drawing package which allows the modeler to draw icons (thus creating the internal icon structure used by RESQME) by using line, circle, and polyline elements, and to edit these elements with move, copy, and delete commands. Existing icons can be selected as a base for new icons and modified, or the modeler can start with a blank drawing box. The modeler links a specific icon to a submodel by providing it the name of the submodel. The icon drawing package allows the modeler to group the resulting icons into user-created icon palettes for a given model or application area. These palettes become additional icon palette pages in RESQME.

If the modeler selects a model with user-created icons, those icons will be displayed in icon palettes in addition to the two built-in palettes of system-defined icons. When the modeler selects a user-created icon and places it on the modeling surface, RESQME checks whether the submodel is already in the submodel tree structure of the model. If it is not, RESQME will search for the submodel description on disk, and if it exists, will attach it to the model tree structure. If it does not exist, the modeler must, at some later time, layer to a new modeling surface and create the new submodel definition. The underlying textual attributes for the user-created icon are displayed based on the submodel to which it is linked.

The higher-level objects are integrated in RESQME to behave and be manipulated exactly as the RESQ primitives. They respond identically as the RESQ primitives through all tasks of the experiment process, from selecting and placing them on the modeling surface, to connecting them in the network, to combining them to form other higher-level objects, to layering down to their submodel definitions, to selecting their performance measures, and to animating their flow of jobs and tokens.

5. Conclusion

In this paper, we have briefly described the Research Queueing Package Modeling Environment, and presented a manufacturing example to illustrate some of the RESQME modeling elements. Our primary focus has been on the most distinctive features of RESQME: its hierarchical modeling capabilities and its extensibility. The hierarchical modeling capability of RESQME provides the means for graphically constructing and analyzing modular, well-structured performance models models. It also enables the user to easily replicate similar model sections. The higher-level, user-defined modeling elements permit a user to create a customized modeling environment specifically tailored for modeling within a particular application domain.

REFERENCES

[1] Aggarwal, A., Gordon, K., Kurose, J., Gordon, R. and MacNair, E., "Animating Simulations in RESQME," IBM Research Report RC14680 (1989), IBM Corp, Yorktown Heights, New York. Also to appear in *1989 Winter Simulation Conference.*

[2] Bell, P., "Visual Interactive Modeling in Operational Research: Successes and Opportunities," *Journal of the Operational Research Society,* Vol. 36 (1985) pp. 975–982.

[3] Bharath-Kumur, K. and Kermani, P., "Performance Evaluation Tool (PET): An Analysis Tool for Computer Communication Networks", *IEEE J. on Selected Areas in Communications,* Vol. SAC-2 (1984), pp. 220–226.

[4] Binnie, J. and Martin, D., "The Role of Animation in Decision Making," *Proc. 1988 Winter Simulation Conf.*, IEEE Press, Piscataway, NJ (1988), pp. 272–276.

[5] Browne, J., Neuse, D., Dutton, J. and Yu, K., "Graphical Programming for Simulation of Computer Systems," *Proceedings of the 18th Annual Simulation Symposium,* (1985), pp. 109–126.

[6] Chow, W., MacNair, E. and Sauer, C., "Analysis of Manufacturing Systems by the Research Queueing Package," *IBM Journal of Research and Development,* Vol. 29 (1985), pp. 330–342.

[7] Concepcion, A. and Schon, S., "SAM - A Computer-aided Design Tools for Specifying and Analyzing Modular, Hierarchical Systems," *Proc. 1986 Winter Simulation Conf.*, IEEE Press, Piscataway, NJ (1986), pp. 504–510.

[8] Concepcion, A. and Zeigler, B., "DEVS Formalism: A Framework for Hierarchical Model Development," *IEEE Trans. on Software Engineering,* Vol. 14 (1988), pp. 228–241.

[9] Conway, R. and Maxwell, W., "XCELL: A Cellular, Graphical Factory Modeling System," *Proc. 1986 Winter Simulation Conf.*, IEEE Press, Piscataway, NJ (1986), pp. 160–163.

[10] Conway, R., Maxwell, W., McClain, W. and Worona, S., *Users Guide to XCELL+ Factory Modeling System,* Scientific Press, Redwood City, CA, (1987).

[11] Cox, S., "GPSS/PC Graphics and Animation," *Proc. 1988 Winter Simulation Conf.*, IEEE Press, Piscataway, NJ (1988), pp. 129–135.

[12] Davis, D., and Pegden, C., "Introduction to Siman," *Proc. 1988 Winter Simulation Conf.*, IEEE Press, Piscataway, NJ (1988), pp. 61–69.

[13] Gilman, A, and Watremez, R., "A Tutorial on See Why and Witness," *Proc. 1988 Winter Simulation Conf.*, IEEE Press, Piscataway, NJ (1988), pp. 129–135.

[14] Grant, M. and Starks, D, "A Tutorial on TESS: The Extended Simulation Support System," *Proceedings of the 1988 Winter Simulation Conference,* IEEE Press, Piscataway, NJ (1988), pp. 136–140.

[15] Gordon, R., MacNair, E., Welch P., Gordon, K., and Kurose, J., "Examples of Using the RESearch Queueing Package Modeling Environment (RESQME)," *Proceedings of the 1986 Winter Simulation Conference,* IEEE Press, Piscataway, NJ (1986), pp. 494–503.

[16] Gordon, R., MacNair, E., Gordon, K., and Kurose, J., "A Visual Approach to Manufacturing Modeling," *Proceedings of the 1987 Winter Simulation Conference,* IEEE Press, Piscataway, NJ (1987), pp. 465–471.

[17] Hurion, R., "The Design, Use, and Required Facilities of an Visual Interactive Computer Simulation Language to Explore Production Planning Problems," Ph.D. Thesis, University of London (1976).

[18] Hurion, R., "Visual Interactive Modeling," *European Journal of Operational Research,* Vol. 23 (1986), pp. 281–287.

[19] Kirkpatrick, P. and Bell, P., "Simulation Modeling: A Comparison of Visual Interactive and Traditional Approaches," *European Journal of Operational Research,* Vol. 39 (1989), pp. 138–149.

[20] Kurose, J., Gordon, K., Gordon, R., MacNair, E. and Welch, P., "A Graphics-Oriented Modeler's Workstation Environment for the RESearch Queueing Package (RESQ)," *1986 Proceedings ACM/IEEE Fall Joint Computer Conference,* IEEE Press, Piscataway, NJ (1986), pp. 719–728.

[21] Lavenberg, S. (ed.), *Computer Performance Modeling Handbook,* Academic Press, New York (1983).

[22] Law, A. and Haider, S., "Selecting Simulation Software for Manufacturing Applications: Practical Guidelines and Software Survey," *Industrial Engineering,* Vol. 31 (1989), pp. 33–46.

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

[24] Medeiros, D.J. and Sadowski, R.P., "Simulation of Robotic Manufacturing Cells: A Modular Approach," *Simulation,* Vol. 40, No. 1 (1983), pp. 3–12.

[25] Melamed, B, and. Morris, R., "Visual Simulation: The Performance Analysis Workstation," *IEEE Computer,* Vol. 18, No. 8 (1985), pp. 87–94.

[26] Miles, T., Sadowski, R. and Werner, B., "Animation with Cinema," *Proc. 1988 Winter Simulation Conf.,* IEEE Press, Piscataway, NJ (1988), pp. 180–187.

[27] O'Keefe, R., "What is Visual Interactive Simulation: (and is There a Methodology for Doing it Right?)," *Proc. 1987 Winter Simulation Conf.,* IEEE Press, Piscataway, NJ (1987), pp. 461–464.

[28] Pegden, C., *Introduction to SIMAN,* Systems Modeling Corp., State College, PA (1986).

[29] Roberts, S.,"Simulation Modeling and Analysis with Insight: a Tutorial," *Proc. 1988 Winter Simulation Conf.,* IEEE Press, Piscataway, NJ (1988), pp. 461–464.

[30] Russell, E., "SIMSCRIPT II.5 and SIMGRAPHICS: A Tutorial," *Proc. 1988 Winter Simulation Conf.,* IEEE Press, Piscataway, NJ (1988), pp. 115–124.

[31] Sauer, C., MacNair, E. and Salza, S., "A Language for Extended Queueing Network Models," *IBM Journal of Res. and Dev.,* Vol. 24 (1980), pp. 747–775.

[32] Sauer, C., MacNair, E. and Kurose, J., "Queueing Network Simulations of Computer Communication," *IEEE Journal on Selected Areas in Communications,* Vol SAC-2 (1984), pp. 203–219.

[33] Schruben, L., "Using Simulation to Solve Problems: A Tutorial on the Analysis of Simulation Output," *Proc. 1987 Winter Simulation Conf.,* IEEE Press, Piscataway, NJ (1985), pp. 40–42.

[34] Systems Modeling Corp., *The Siman Simulation Language Reference Guide,* Systems Modeling Corp., State College, PA (1987).

[35] Sinclair, J., Doshi, K. and Madala, S., "Computer Performance Evaluation with GIST: A Tool for Specifying Extended Queueing Network Models," *Proceedings of the 1985 Winter Simulation Conference,* IEEE Press, Piscataway, NJ (1985), pp. 290–300.

[36] Smith, D., Kotik, G. and Westfold, S., "Research on Knowledge-Based Software Environments at Kestrel Institute," *IEEE Transactions on Software Engineering,* Vol. SE-11 (1985), pp. 1278–1295.

[37] Thomasma, T. and Ulgen, O., "Hierarchical, Modular Simulation Modeling in Icon-based Simulation Program Generators for Manufacturing," *Proc. 1988 Winter Simulation Conf.*, IEEE Press, Piscataway, NJ (1988), pp. 254–262.

[38] White, D., "PCModel and PCModel/GAF - Screen Oriented Modeling," *Proc. 1988 Winter Simulation Conf.*, IEEE Press, Piscataway, NJ (1988), pp. 164–167.