8-17-1990

# RESQME and Stand-Alone Simulation on a Workstation

Robert F. Gordon Ph.D.
*Molloy College*, rfgordon@molloy.edu

Paul G. Loewner

G J. Burkland

J-C Chen

Edward A. MacNair

Modeling & Simulation Conf
Manufacturing Symposium
1990

RC 16037 (#71232) 8/17/90
Computer Science 7 pages

# Research Report

## RESQME and Stand-Alone Simulation on a Workstation

G. J. Burkland, J-C. Chen, R. F. Gordon,
P. G. Loewner and E. A. MacNair

IBM Research Division
T. J. Watson Research Center
Yorktown Heights, NY 10598

# RESQME and Stand-Alone Simulation on a Workstation

G.J. Burkland, J-C. Chen, R.F. Gordon,
P.G. Loewner, and E.A. MacNair
IBM Research Division
IBM Thomas J. Watson Research Center, U.S.A.
Yorktown Heights, New York 10598
BURKLAN, JCCHEN, RGORDON, LOEWNER, MACNAIR at YKTVMH
TL 863-7503, 7751, 7170, 7564, 7561

## ABSTRACT

The Research Queueing Package Modeling Environment (RESQME) provides a graphical environment for constructing and solving extended queueing network models of manufacturing systems, for plotting graphs of results and for viewing animations of models. The modeling environment can be run entirely on a workstation or optionally can execute large simulations on a host system using cooperative processing.

In this paper we give a brief introduction to RESQME and to the RESQ modeling elements. We demonstrate how to use the package by constructing a simple model of part of a manufacturing line and solve this model to produce charts of performance measures and an animation which shows how the jobs flow through the system.

By having the simulation available for use on the workstation and cooperatively on the host, RESQME provides a unique tool for understanding the performance of manufacturing systems. A user can do most of the model debugging locally on the workstation and make short pilot runs to get a feeling for the amount of resources necessary to make more realistic experiments on the host. Then long runs which investigate large parts of the parameter space can be done cooperatively on the host.

Whether the model is solved on the workstation or on the host, the graphics environment provides the same user interface to all of the underlying files. The processor where the model is solved is transparent to the user. In all cases, the user has easy access to plots of results and to the animation of the model diagram.

## INTRODUCTION

The RESearch Queueing Package Modeling Environment (RESQME) provides a graphical interface for constructing and solving extended queueing network models of manufacturing systems, computer systems, communication networks, and other contention type systems. In this paper we give a brief description of RESQME and the RESQ modeling elements that are used in building the models. Additional information on RESQME and RESQ can be found in Aggarwal et. al. 1989, Chow et. al. 1985, Gordon et. al. 1986, 1987, 1990b, Kurose et. al. 1986, MacNair and Sauer 1985, and Sauer et. al. 1980, 1982, 1986. A simple example is presented to illustrate how to use the modeling environment to construct and solve a model. The package allows models to be solved either on the workstation or on a host. The choice of where to solve the model is left up to the user at run time and requires nothing more of the user than to select the desired processor in an options window. The user can progress from one processor to another as his or her requirements from the model changes. After the model is solved, the results are available on the workstation so that the performance measures can be quickly viewed graphically and an animation of the model can be displayed.

## THE GRAPHICAL ENVIRONMENT AND MODELING ELEMENTS

RESQME provides a graphical environment for creating, editing, evaluating, and analyzing models. It is menu driven with three main tasks:

1. Create/Edit
2. Evaluate
3. Output Analysis

The Create/Edit task enables the user to build a new model or change an existing model. Parameters and other symbolic names can be defined while creating the model. Icons, representing various modeling elements, can be selected from a palette and linked together to define the flow of jobs through the model. As these icons are placed in the graphics area, their textual attributes are specified in pop-up windows. The simulation run length, a confidence interval method, and the desired accuracy of the results are also specified. Additional information about the graphical environment is provided to the user through on-line, context-sensitive help windows and dynamic tutorials.

The Evaluate task is used to assign values to parameters and to submit multiple simulation runs with different sets of parameter values. The user can choose to evaluate the model locally on the workstation or on a host system by selecting the appropriate switch. When the solution is completed, the results consisting of all performance measure means and selected distributions are read into memory for immediate access.

The Output Analysis task allows the user to select the performance measures to plot, to change the way he or she views the data, to plot a graph or display a table of numbers, and to view an animation. This is all provided from the same visual representation of the model without requiring any additional work from the user. Performance measure graphs and tables are provided by pointing to the desired object on the model diagram and then selecting from the resulting performance measures for that object, whether the object is a chain, a node, a queue, or the whole model. Animation is provided by simply selecting the animation menu item.

The modeling elements of RESQ include a rich set of primitives for constructing realistic models of complicated systems. Service centers can be defined with one or more servers, various queueing disciplines for scheduling jobs, to support different types of jobs with their associated service time distributions and priority levels, and the servers can serve at specified service rates. An open model which has jobs that enter and leave can have sources and sinks representing the various entry and exit points. Resources that have a finite number of elements that have to be allocated to jobs, like buffers, operators, robots, positions on a conveyor mechanism, can be represented by passive queues. A continuous flow manufacturing system can be implemented with wait nodes and passive queues. Variables can be assigned values at set nodes to, for example, distinguish jobs at different stages of a multi-loop process. Jobs can be split into multiple copies which are either related or unrelated, as may be required to handle batching operations. The related jobs will be reunited at

some point in the model. The model can be constructed hierarchically using submodels. The submodels can contain parameters so that different copies of the submodels can be assigned different values for the parameters. Information can be carried by jobs, by system, global and local variables and by parameters, so that decisions in a CIM environment, based on activities at different parts of the manufacturing line, can be simulated.

## AN EXAMPLE

The manufacturing model discussed in this section is taken from the one discussed in Gordon et. al. 1990a. The model is hierarchically structured. The top level is shown in Figure 1. 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. It might be a portion of a more complicated manufacturing line that is in an early stage of design or can be separated from the full line to examine in detail and so may be analyzed on the stand-alone workstation.

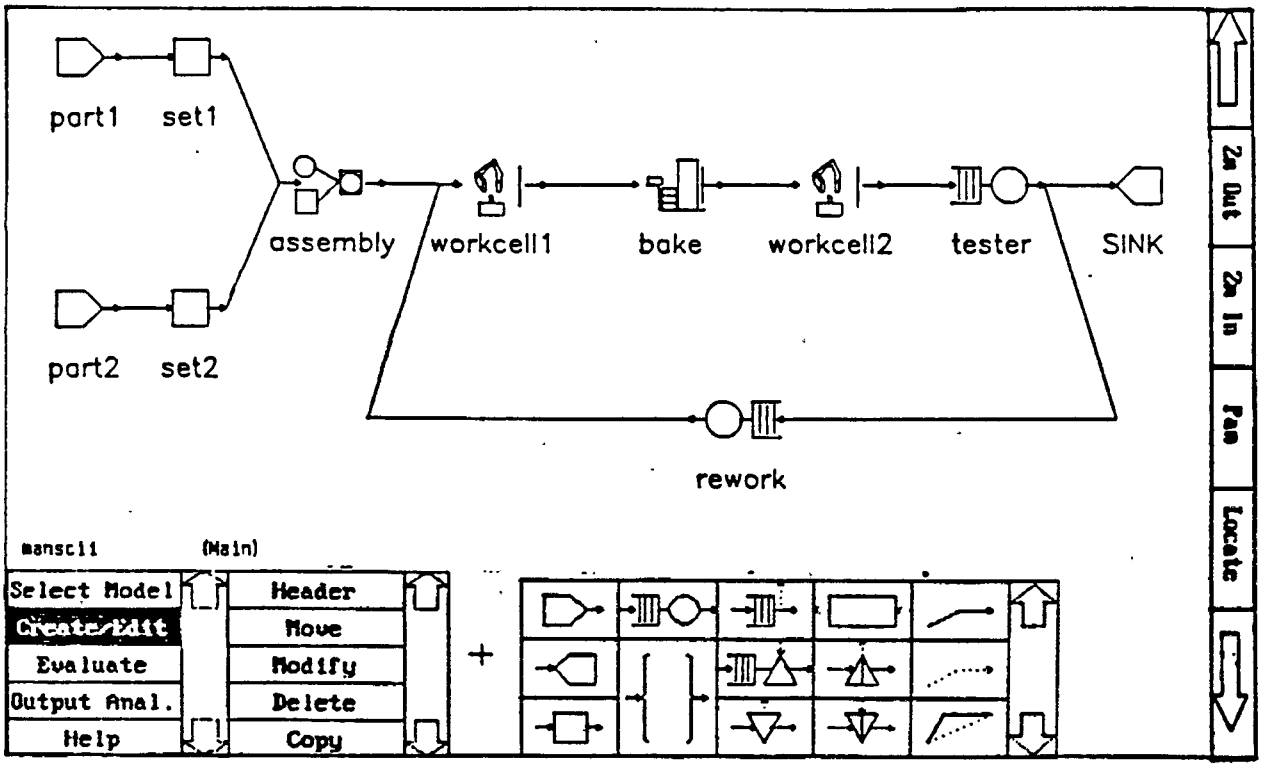



**Figure 1. RESQME screen with model**

Two types of parts are to be assembled at the **assembly** process. The assembled 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 parts that pass the test leave the model at the *SINK*. The decision as to which path is taken is specified in an attribute pop-up window, using conditional or probability expressions. Suppose it should go to SINK with probability .90, the portion in the pop-up window that describes this is as follows:

```
...
tester -> SINK  ; .90
tester -> rework; .10
...
```

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 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 create parts according to a specified interarrival time distribution. Each part then goes to a **set node** (***set1*** and ***set2*** in the model diagram). Set nodes are used to assign values to variables. At ***set1*** and ***set2***, the parts are assigned a part type. From this point, each part flows into the ***assembly*** process.

The ***assembly*** process is represented by an **invocation** of a **submodel**, called ***ASSEMBLE***, 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 from or to the passive queue. A submodel has a primary entry point and a primary exit point; the node labeled ***din*** is the primary entry point for the ***ASSEMBLE*** submodel. ***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.

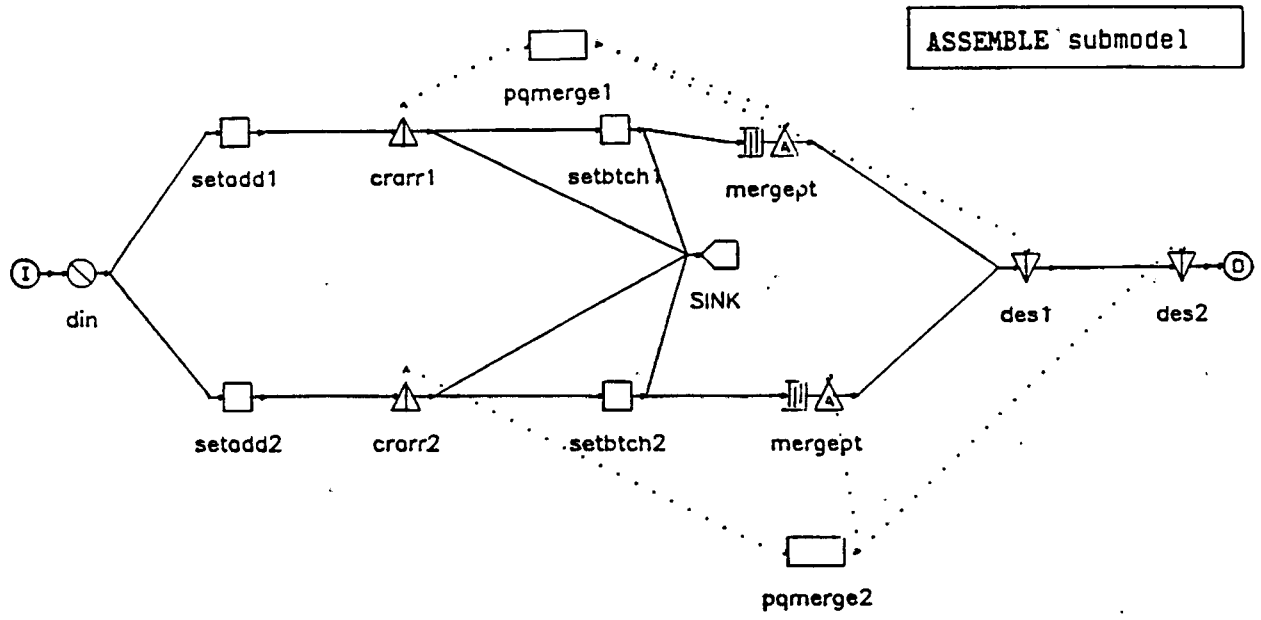



**Figure 2. ASSEMBLE submodel**

The type of synchronization required by this submodel is typically accomplished in RESQ by using a **passive queue**. 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 visits a **create node** (***crarr1*** or ***crarr2***), where it creates a token indicating that another part is ready to be assembled. If the part is the first part of its type in a particular assembly, the part goes to a **set node**, ***setbtch1*** or ***setbtch2***, where the assembly number is incremented. If the first part of the other type for the same assembly has not

yet arrived, the part then waits at an **allocate node**, ***mergept***, until all parts for that 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. (Thus, most of the parts will flow from ***crarr1*** or ***crarr2*** to ***SINK***. The corresponding branches from ***setbtch1*** or ***setbtch2*** to ***SINK*** are used only for the first part of the later part type in each assembly.) Only the first part in every 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 and are represented in RESQME by multiple icons with the same name. 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 the **destroy nodes** ***des1*** and ***des2***. The assembled job then leaves the ***ASSEMBLE*** submodel and goes to the first workcell.

The icons ***workcell1*** and ***workcell2*** are **invocations** of the ***ROBOCELL*** **submodel**, which is based on a paper by Medeiros and Sadowski 1983. The flow of jobs is depicted in Figure 3. There is an input staging area where the part is oriented, a machine to process the part, and an output staging area where the processed part waits until it can leave the submodel. There is also a robot which moves the part from the input staging area to the machine, stays there with the part, and then 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.** An oriented part waits 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 ***process*** **invocation** of the ***JOBPROC*** **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: the part leaves the output staging area only when a buffer is available to hold it. This is an example of one **invocation** requiring information about another **invocation**; such information is made available here through a global variable.

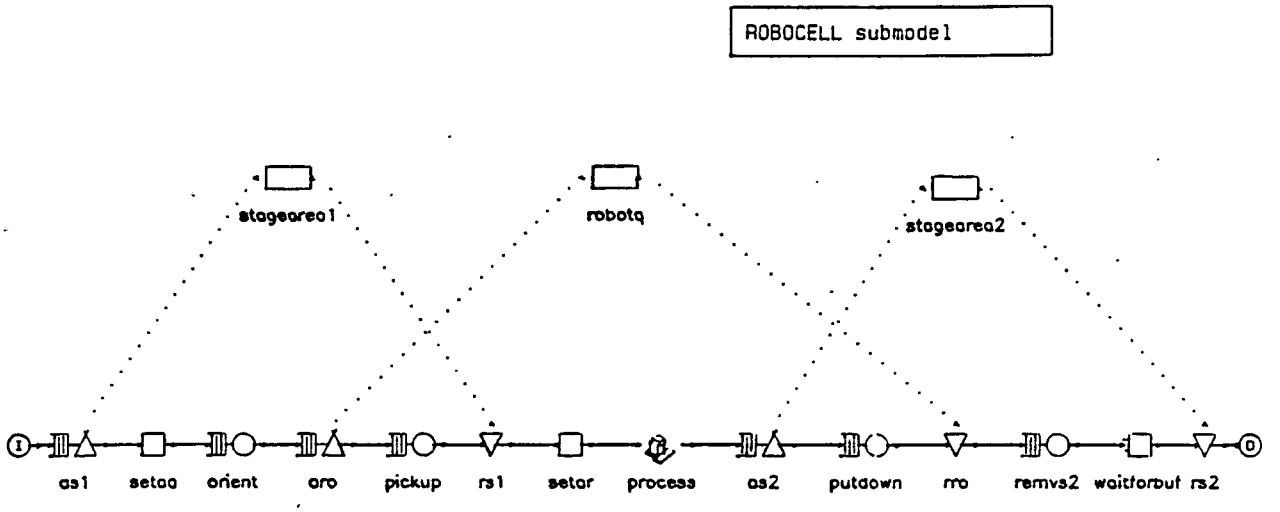
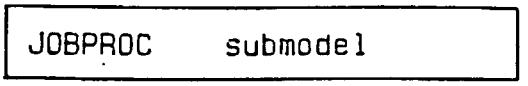



**Figure 3. ROBOCELL submodel**

The ***JOBPROC*** **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 of the machine which processes the parts. 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 (i.e.,

at the server of the *process* queue) is interrupted and re-queued for service until the machine is repaired; this is illustrated in Figure 4 on page 5.

From *workcell1* the parts flow to the *bake* operation. The *bake* **invocation** is a copy of a **submodel** (*OVEN*) which accumulates batches of parts until there are enough to fill the bake oven. Then all parts 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 this second **invocation** of the ***ROBOCELL*** submodel.



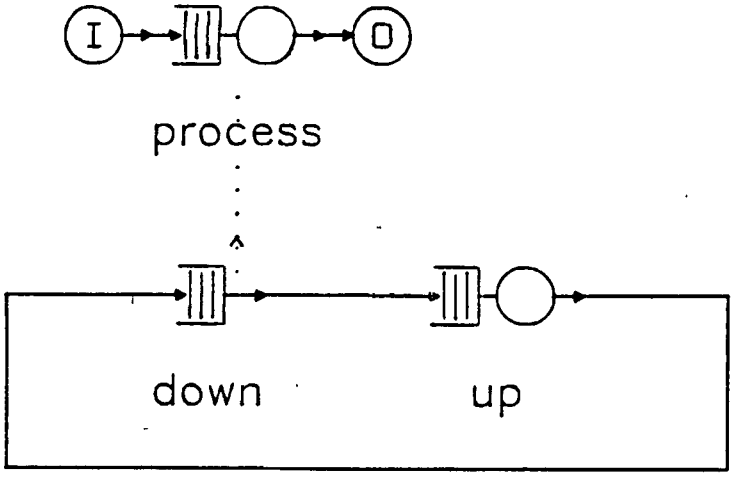




**Figure 4. JOBPROC submodel**

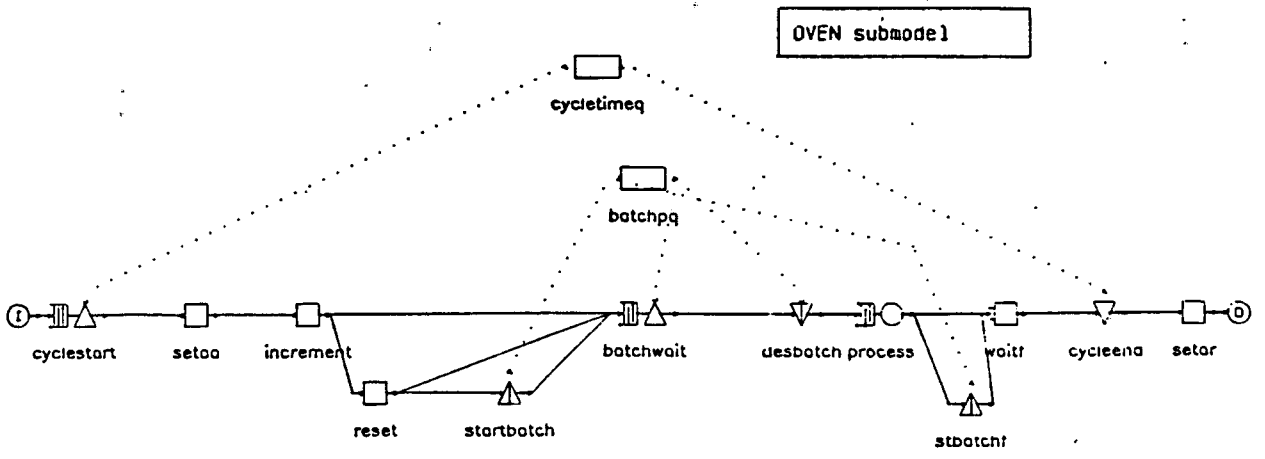



**Figure 5. OVEN submodel**

## STAND-ALONE OR COOPERATIVE PROCESSING FOR SOLUTION

The user has the choice of solving the model either on the workstation or in a cooperative processing mode on a host. The user can freely switch between these processors without having to change the model. Having the simulation available on the workstation is very convenient for making short pilot runs and for debugging the model. After the model is working correctly, the results from the pilot runs can be used to estimate the length of longer runs. When final experiments are to be performed, they can be run on the host.

Independent of where the model is solved, all of the performances measures are available on the workstation for rapidly producing graphs. Figure 6 shows some sample results for the model described in the previous section. The QL & QT bars are displayed with different colors.

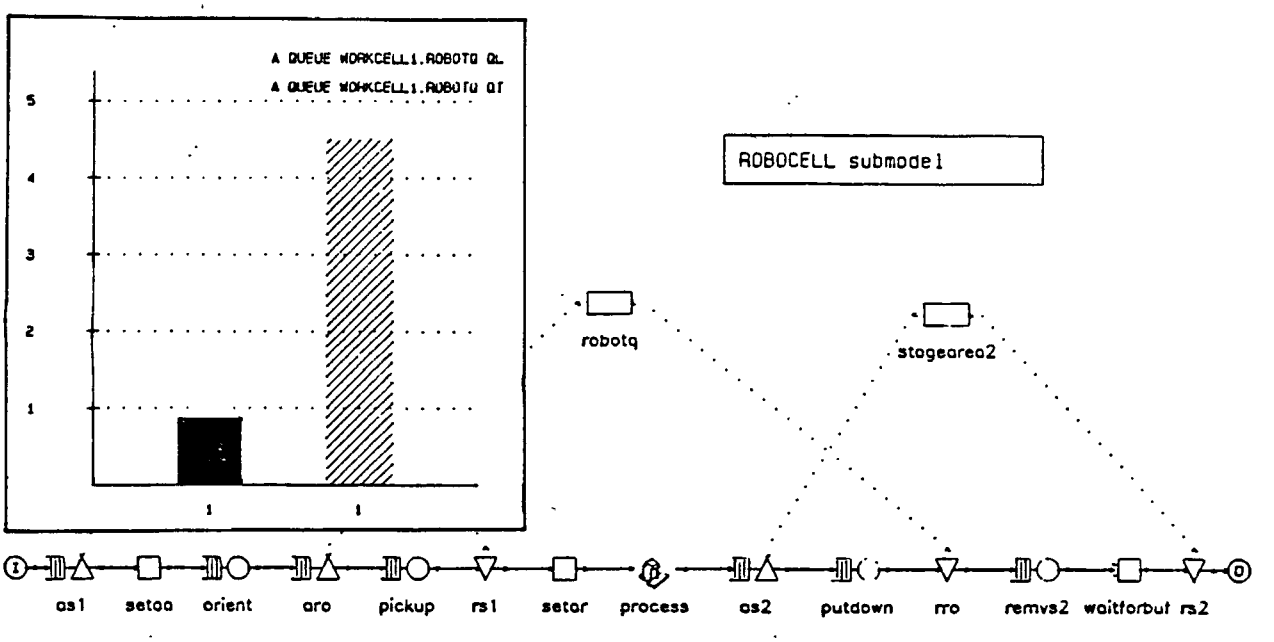



**Figure 6. Output Analysis: display of performance statistics**

After the simulation is finished, the analyst can also view the animation of the model diagram. This will show jobs moving between nodes and tokens being allocated to and released from jobs. See Aggarwal et. al. 1989 for more information related to animation.

## SUMMARY

RESQME provides a graphical environment for constructing, solving, and analyzing extended queueing network models. It supports a rich underlying modeling paradigm and provides a single integrated graphical interface for use throughout the modeling lifecycle. The stand-alone version provides additional support for obtaining quick solutions and for debugging.

## ACKNOWLEDGEMENT

The authors wish to thank Ben Antanaitis, Howard Jachter, Jack Servier, Daniel Souday and Peter Welch for their many suggestions which helped improve the RESQME package. We are also indebted to Jim Kurose and Kurt Gordon for their intimate involvement with RESQME for many years. We thank Anil Aggarwal, Al Blum, Diana Coles and Geoff Parker for their work in implementing RESQME. We would also like to thank our users for their ideas and feedback that we tried to incorporate in the RESQME interface.

## REFERENCES

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

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

Gordon, K. J., Kurose, J. F., Gordon, R. F. and MacNair, E. A. (1990a). An Extensible Visual Environment for Construction and Analysis of Hierarchically-Structured Models of Resource Contention Systems. Accepted by *Management Science.*

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

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

Gordon, R. F., Loewner, P. G., MacNair, E. A., Gordon, K. J. and Kurose, J. F. (1990b). The RESearch Queueing Package Modeling Environment (RESQME) OS/2 Guide for Version 3.1. IBM Research Report, Yorktown Heights, New York.

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

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

Medeiros, D. J. and R. P. Sadowski (1983). Simulation of Robotic Manufacturing Cells: A Modular Approach. *Simulation*, 40, 1, 3-12.

Sauer C., MacNair, E., Salza, S. (1980). A Language for Extended Queueing Network Models. *IBM Journal of Research and Development* **24**, 747-755.

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

Sauer, C. H., Blum, A. M., Loewner, P. G., MacNair, E. A. and Kurose, J. F. (1986). The Research Queueing Package Version 2: CMS Reference Manual. IBM Research Report RA-139, Yorktown Heights, New York.