NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilming. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SFC 1970, c. C-30, et ses amendements subséquents.
A SPREADSHEET TOOL
FOR MODELLING AND SOLVING
QUEUING NETWORKS
(QUEST)

by

Guy Montreuil

A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfillment of the requirements
for the degree of
Master of Engineering

Ottawa-Carleton Institute for Electrical Engineering
Faculty of Engineering
Department of System and Computer Engineering
Carleton University
Ottawa, Ontario, Canada
May 16, 1990
© 1990, Guy Montreuil
NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.


AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.
The undersigned hereby recommend to the Faculty of Graduate Studies and Research acceptance of the thesis,

A SPREADSHEET TOOL FOR MODELLING AND SOLVING QUEUING NETWORKS (QUEST)

submitted by Guy Montreuil in partial fulfilment of the requirements for the degree of Master of Engineering

______________________________
C.M. Woodside
Dr. C.M. Woodside, Thesis Supervisor

______________________________
Dr. S.A. Mahmoud, Chairman, Department of Systems and Computer Engineering

Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University
May 16, 1990
Abstract

Computer performance analysis is an important tool for planning and operating computer systems, and is becoming more important as systems become more interconnected and complex. A major tool for performance analysis is the set of models known as queuing networks, which are used for all kinds of capacity and delay studies. Experience shows that the key to using queuing networks is good software tool for editing, calibrating, and solving them (either analytically or by simulation), and to provide a simple and convenient interface to the analyst. The nature of this interface is very important and a variety of textual and graphical approaches exist. This thesis explores the use of a new kind of interface based on spreadsheet editing and formatted tabular data entry, which provides rapid and intuitive presentation of the essentials of the model. The QUEuing analysis Spreadsheet Tool (QUEST) had the goal of providing all the user capability of the existing QNAP2 package, which is based on a programming language. Various strategies were adopted to overcome the intrinsic limitations of the spreadsheet approach, particularly the limited space in each table entry. Quest in fact generates well-structured QNAP2 source code, solves the model using the existing solution tools, and reports the results within the spreadsheet. The advantages of QUEST are rapid and
intuitive presentation of information and the automated prevention of many bugs in the model, through built-in constraints on the model definition.
Acknowledgements

I would like to thank my supervisor, Dr. C.M. Woodside for the advice, guidance and encouragement which he has given me during this project.

Finally, special appreciation goes to my wife Carole and to my son Alexandre, to whom this thesis is dedicated.
becomes saturated and no increase in the arrival rate of customers can be handle successfully. The throughput bounds are the smallest arrival rate at which any service center becomes saturated. The service center that saturates is called the bottleneck of the system.

We introduce this notation:

\[ U_i \quad : \text{utilization at server } i \]
\[ \lambda_r \quad : \text{customer throughput for class } r \]
\[ \bar{x}_{ir} \quad : \text{mean service time for class } r \text{ in station } i \]
\[ y_{ir} \quad : \text{mean visit ratio for class } r \text{ in station } i \]
\[ i \quad : \text{station index} \]
\[ r \quad : \text{class index} \]

i.e. \[ \sum_{r} \bar{x}_{1r} \] means the sum, for all classes \( r \), of the mean service time at server 1.

We now define the utilization law:

\[ U_i \leq 1 \quad \text{for all service centers} \quad (1) \]

\[ U_i = \sum_{r} (y_{ir} \cdot \bar{x}_{ir} \cdot \lambda_r) \quad (2) \]

then from (1) and (2):

\[ \sum_{r} (y_{ir} \cdot \bar{x}_{ir} \cdot \lambda_r) \leq 1 \quad (3) \]
4.5.4- The bottleneck analysis module ...................  66
4.5.5- The simplex evaluation module ....................  68
4.5.6- The visit ratio to probabilities module ............  69
4.5.7- The code generator module .........................  70
4.5.8- The solve and get results module ...................  74
4.6- Special cell entries provided for iteration ...........  74
4.6.1- Example of iteration ................................  76
4.7- Design issues in developing QUEST ....................  80
4.7.1- Cell data structure ..................................  81
4.7.2- Workspace management ...............................  84
4.7.3- Modularity ..........................................  87
4.7.4- Cell movement .......................................  87
4.8- QUEST capabilities compare to PAW ....................  87
4.8.1- Requirements ........................................  87
4.8.2- Model definition and modification ..................  88
4.8.3- Model types that can be defined .....................  89
4.8.4- Solution techniques and results .....................  89
4.8.5- Utilities ............................................  90

5- Performance analysis using QUEST

5.1- The model structure used by Lazowska ..................  91
5.1.1- Model description ...................................  92
5.1.2- The baseline case description .......................  95
5.1.3- The baseline case study ..............................  98
5.1.4- Effect of SUN workstation enhancements ............ 103
5.2- The departmental SUN network model ................... 104
5.2.1- The SUN network description ......................... 104
5.2.2- The baseline analysis of the model .................. 111
5.3- Example of iteration for an approximation model ..... 114
5.3.1- Model description ................................... 116

6- Conclusion

6.1- What was retained from QNAP2 .......................... 124
6.2- What was improved .................................... 125
6.3- What was lost ......................................... 125
6.4- Major goals .......................................... 126
6.5- Further work ........................................ 128

References
1- Introduction

The new computers today are more complex, more rapidly changing and more essential to society. The consequence is an intensifying need for tools and methods that aid in modeling the behavior of these systems. Such tools and methods are necessary to provide answers to the questions of capacity planning and performance management.

Queueing network modeling [LZGS1984] [SC1981] is widely used to answer these questions. It is an approach to computer system modeling in which the computer system is represented as a network of queues which is evaluated analytically [LZGS1984] or by a simulation [WSB1975] of the model. The evaluation involves using a software package to formally describe the system and to solve a set of equations derived from the network of queues. There are on the market some very good queueing network analysis tools like QNAP2 [SIM] or PAW [ATT1986].

This project uses QNAP2 because it happened to be available for the study. QNAP2 can be seen as a system for describing, manipulating and solving queuing network models. It includes a specification language which is used for the description of the model under study, the control of the solution, some reporting features and the choice among several solution modules or solvers, imple-
menting different solution techniques.

Although the QNAP2 language is excellent in many ways, and gives great flexibility to the analyst, it does have some weaknesses.

One problem is the fact that models are described within QNAP2 as a sequence of statements in a linear fashion (1 dimension). Because QNAP2 is one dimensional, making changes to a large model, for instance adding a class, causes difficulties. When a class is added, the analyst has to define the class at the beginning of the program, and then he has to go through all the stations one by one and add the required information to define the new class for example its service time at each station, its routing from station to station and the initial number of customers. All this information is scattered in the program, and it is easy to forget some item. One goal of this present work is to bring these changes into a relationship which is easy to navigate, by using a two-dimensional layout in which stations are represented by vertical columns of information, and classes by horizontal arrays.

Another problem is the fact that to use QNAP2, the analyst has to write and debug a program. This programming can take a substantial time. One goal of the present work is to reduce the debugging by entry validation and by organization of the data.
A graphical interface is another type of tool used to model queuing networks. The model is defined by positioning on the screen (2 dimensions) different types of icons (queues) and by linking them with arcs (routing). These tools have some inconveniences. They run only on special terminals and they are generally not portable. They degrade toward "spaghetti" as the number of stations in the model increases. One goal of this project is to have a tool which runs on ordinary terminals.

This project looks for an alternative interface. This interface should:

- describe queuing network models in a 2 dimensional fashion.
- be portable and run on a regular text terminal.
- show the model parameter in a natural format.
- provide the user with the full QNAP2 functionality to create, modify and solve queuing network model.

This new high level interface will take the form of a spreadsheet.

There are many reasons why the spreadsheet approach is considered:

- queuing network models fall naturally into a tabular format with resources (eg: devices) in one dimension and classes (kind of users) in the other.
- spreadsheet editors are very powerful for handling
data that fits their tabular format.
- spreadsheet editors run on standard text terminals.
- spreadsheet editors are portable.

Spreadsheets are a very successful tool for manipu-
ulating tabular information and have been adapted in this
research to drive a general purpose analysis package.

The QUEuing analysis Spreadsheet Tool (QUEST) is
the resulting tool. A user can define, solve and inves-
tigate a queuing network model within QUEST. QUEST can
be seen as the sum of the spreadsheet editor power (cell
linking, ...) added to the solution techniques power of
the external solver package QNAP2.

QUEST has been implemented using QNAP2 (a product
of Simulog and INRIA), the public domain spreadsheet VC
and the standard screen library Curses.

The author’s contribution in this thesis is:
- the complete performance spreadsheet design.
- extensive modifications and redesign in the
  spreadsheet part (originally VC).
- the design and implementation of the new perfor-
  mance functions (over 800) lines of code), reported
  in chapter 4.
- the testing of the software by solving many queuing
  network models, reported in chapter 5.

The thesis is divided into six chapters. In the
remainder,
  - chapter 2 covers general concepts of performance modeling.
  - chapter 3 introduces the different types of data used in defining performance models, in the project.
  - chapter 4 covers the QUEST design and implementation.
  - chapter 5 covers a substantial performance study that was conducted using QUEST.
  - chapter 6 is the thesis conclusion.
2- General concepts

2.1- Queuing theory:

In a queuing network model, a computer system is seen as a network of queues which is evaluated analyti-
cally or by a simulation. A network of queues is a set of service centers or stations, which represents system resources (CPU, I/O, ...), and customers which represent jobs or programs. Analytic evaluation uses certain algorithms, such as mean value analysis, to solve effi-
ciently a set of equations generated by the network of queues and its parameters. Simulation uses the repre-
sentation of the model to mimic the system behavior. Both are fully described in several books, such as those of Lazowska [LZGS1984], Sauer and Chandy [SC1981] or Lavenberg [LAV1983].

Figure 2.1 illustrates a single server queue. Customers arrive at the service center, wait in the queue if the server is busy, receive service from the server and leave the station. This single server and its cus-
tomers form the simplest queuing network model. More generally, a queuing network is a collection of service centers and customers joined by transfer paths (routing). We can consider two types of networks: open and closed. An open queuing network is characterized by one or more sources of customer arrivals and one or more
Figure 2.1 - A small queuing network

Figure 2.2 - A small computer model
links that absorb jobs leaving the network. In a closed queuing network, jobs neither enter or leave the network. Customers are present in the system initially and do not need to be generated. These two kinds of behavior can coexist together in the same system, giving a mixed queuing network.

Defining a queuing network model representing a particular computer system is easily done because of the close relationship that exists between the parameters of the model and the parameters of the computer system. For example, terminals in a computer system can be modeled as a number of customers and the average "think time" that each customer spends using the terminal. The description of a model is done by describing each service center in the model and by describing the behavior of customers in the system. Figure 2.2 shows a small computer model and introduces the probability notation used to describe the customer routing.

For service centers, we have to know some important characteristics if we want to evaluate the system performance. The most important parameters are:

- the type of the service center represents the kind and the number of servers present in the station. The kind can be: a single, multiple or infinite server, a source, a resource, etc.
- the scheduling discipline of the service center
describes how customers are ordered in the station queue. A common discipline is first come first served. (FCFS)

- the service time is the time that a customer spends in a particular station. The service time is generally described by a probability distribution function such as exponential. The inverse of the service time represents the rate at which customers are served.

- the routing describes how customers behave between stations. The routing can be given in two different formats. The first is to describe customers movement by a probabilistic behavior. In this case, the probability \( P_{ij} \) gives the probability for a customer to go from station \( i \) to station \( j \). The second is to give the frequency with which a customer visits a particular station relative to a reference station. This is called a visit ratio.

In general, there are two approaches to evaluate queuing network models: analytical and simulation. Two classes of errors may be introduced during a model analysis: modeling error and solution error. The modeling error is generally present in both evaluation methods and can be attributed to assumptions made to simplify the real system into a workable model. The solution error is the precision at which the model will be
solved. This error may be large when using simulation, but is zero for analytic solutions.

The solution process gives results like utilization, throughput and response time. The utilization represents the fraction of the time a station is busy. The throughput is the rate at which the customers are passing through a particular station or system. The response time is the time spent by a customer in the station or system.

Algorithms, such as mean value analysis, have been implemented for the computer and are both stable and fast (at least for models with only a few servers). These algorithms, for evaluating queuing network models, constitute the lowest layer of a queuing network software package. This layer is accessed by higher levels like a programming language or a graphical interface.

The second approach involves calculating values of performance using a simulation [WSB1975] [PRI1986] model. The principal strength of simulation is its flexibility. There are no restrictions on the type of system that can be simulated, so a computer system can be represented with as much detail as we want. The principal problem with simulation is its relative cost. Simulation models are generally expensive to evaluate, because running a simulation requires a long time, especially when small confidence intervals are desired.
Generally, in a computer system, jobs are not all the same. A word processing job does not require the same amount of CPU time as a compilation job. For modeling the differences between these customers, we have to introduce the concept of classes. A class of customers is characterized by the same behavior in the queuing network model and within a class the customers are not distinguishable. Parameters for a multiple class model, such as service time, priority and routing, are class dependent. Station type and scheduling discipline are given for each station and they are not class dependent.

Two interesting values computed by analytical or simulation methods are utilization and throughputs. They are calculated for each station (averaged over all classes) or they could be calculated for each class in the model. The utilization represents the fraction of the time a service center is busy. The throughput represents the rate at which customers circulate in the network.

Three points must be considered in applying queuing networks for computer system modeling:

- simplicity:

There is a possibility to identify and eliminate irrelevant details in a model. For example, if a system has a large number of identifiable classes,
we may be interested in the performance of only one of them. In this case we may choose to employ an approximate aggregated model with two classes, one representing the class that we are interested in and the other representing the rest of the system. This approach will not be exact, but it will give a good idea of the system behaviour.

- measurement:

A queuing network model requires a small number of parameters. Measurements, available on the computer, provide a large volume of data and interpretation and reduction of these data [SV01976] is required from the analyst to get proper inputs for the model.

- ease of evaluation:

A subset of general queuing network models can be evaluated efficiently because they need the assumption of analytic evaluation (separable or product form) [SC1981]. If we stay in this subset, by making some compromises (like first come first served as a scheduling discipline) in the representation of certain computer system characteristics, then the evaluation is relatively straightforward and fast. Otherwise we get into approximation or simulation.

Queuing network models have become an important
tool in the design and analysis of computer systems. This is due to the fact that, in general, they have achieved a good balance between result accuracy and solution efficiency [LZGS1984].
2.2- Bottleneck analysis:

Bottleneck analysis is most useful in the process of finding the capacity of a system in terms of the number of customers it can handle. This kind of analysis typically involves the consideration of a large number of possible models. Often a single resource (such as the CPU or a disk) is the major concern and the rest of the system can be considered fast enough to match the power of this resource. This technique can also be used to estimate the potential performance gain of alternative modifications to an existing system.

With very little computation, a bottleneck analysis [LZGS1984] can determine upper bounds on system throughputs as a function of the system workload intensity. The system bottleneck can be defined as the device (resource or server) which constrains the throughput.

There are several characteristics of the bottleneck analysis technique that make it interesting and useful:

- the application of the technique provides a good insight into the major factors affecting the performance of computer systems. Particularly, the critical influence of the system bottleneck is highlighted and quantified.
- the upper bounds can be computed quickly, even by hand. This approach is suitable as a first step in
modeling to eliminate inadequate alternatives at an early stage of study.

In contrast to this technique, more powerful techniques, like mean value analysis, require more computation and are almost impossible to perform by hand.

For an open queuing network, the throughput bounds indicate the maximum possible arrival rate of customers that the system can process successfully. If the arrival rate exceeds this bound, a queue of unprocessed customers grows continually as jobs arrive. In this case, we say that the system is saturated. The throughput bounds give the arrival rate that separates feasible processing from saturation. For a closed queuing network, the throughput bounds indicate the maximum possible system throughput of customers cycling in the network.

The key to determine the throughput bounds is the utilization law. A simple statement of this law is that a station can not be used more than at its full capacity. This means that a utilization of 1 is the maximum utilization for a particular station. To determine the throughput bounds we simply state that as long as all service centers are not saturated (utilization less than or equal to 1), an increased arrival rate can be accommodated by the system. When any of the service centers becomes saturated (utilization=1), the entire system
becomes saturated and no increase in the arrival rate of customers can be handle successfully. The throughput bounds are the smallest arrival rate at which any service center becomes saturated. The service center that saturates is called the bottleneck of the system.

We introduce this notation:

\[ U_i \quad : \text{utilization at server } i \]
\[ \lambda_r \quad : \text{customer throughput for class } r \]
\[ \bar{x}_{ir} \quad : \text{mean service time for class } r \text{ in station } i \]
\[ y_{ir} \quad : \text{mean visit ratio for class } r \text{ in station } i \]
\[ i \quad : \text{station index} \]
\[ r \quad : \text{class index} \]

\[ i.e. \quad : \sum_r \bar{x}_{ir} \text{ means the sum, for all classes } r, \text{ of the mean service time at server } i. \]

We now define the utilization law:

\[ U_i \leq 1 \text{ for all service centers} \quad (1) \]

\[ U_i = \sum_r \left( y_{ir} \cdot \bar{x}_{ir} \cdot \lambda_r \right) \quad (2) \]

then from (1) and (2):

\[ \sum_r \left( y_{ir} \cdot \bar{x}_{ir} \cdot \lambda_r \right) \leq 1 \quad (3) \]
We now apply the bottleneck analysis to a simple queuing network (Figure 2.3):

<table>
<thead>
<tr>
<th>station</th>
<th>class1 CPU intensive</th>
<th>class2 I/O intensive</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>2 \cdot 0.5 \cdot \lambda_1 + 6 \cdot 0.1 \cdot \lambda_2 \leq 1</td>
<td>\text{(4)}</td>
</tr>
<tr>
<td>DISK</td>
<td>1 \cdot 0.5 \cdot \lambda_1 + 5 \cdot 0.2 \cdot \lambda_2 \leq 1</td>
<td>\text{(5)}</td>
</tr>
</tbody>
</table>

and from (4) and (5):

<table>
<thead>
<tr>
<th>station</th>
<th>class1 CPU intensive</th>
<th>class2 I/O intensive</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>\lambda_1 + 0.6 \cdot \lambda_2 \leq 1</td>
<td>\text{(6)}</td>
</tr>
<tr>
<td>DISK</td>
<td>0.5 \cdot \lambda_1 + \lambda_2 \leq 1</td>
<td>\text{(7)}</td>
</tr>
</tbody>
</table>

If we use 2 axes, one representing \lambda_1 and the second representing \lambda_2, we can trace 2 lines from equations (6) and (7). The space defined by these two lines and the two axes represent all possible throughputs that the system can handle (Figure 2.4). Outside that region, the system is saturated. This means that if the input rate coordinate (\lambda_1, \lambda_2) is outside of the bounded region, a queue of unprocessed customers will grow continually.

If we suppose that the input rate \lambda_1 is known and is fixed to 0.2, we may want to find the input rate \lambda_2 that will saturate (U_1 = 1 or U_2 = 1) the system.
Figure 2.3 - A computer model example

<table>
<thead>
<tr>
<th></th>
<th>TERM</th>
<th>CPU</th>
<th>DISK</th>
<th>$P_2$</th>
<th>$P_1$</th>
<th>$P_3$</th>
<th>$P_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>CPU intensive</td>
<td>18 sec.</td>
<td>0.500 sec</td>
<td>0.500 sec</td>
<td>1.0</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>service</td>
<td>I/O intensive</td>
<td>18 sec.</td>
<td>0.100 sec</td>
<td>0.200 sec</td>
<td>1.0</td>
<td>1/6</td>
<td>5/6</td>
</tr>
</tbody>
</table>

Figure 2.4 - Throughput bounds
When $U_1 = 1$ or $U_2 = 1$:

<table>
<thead>
<tr>
<th>station</th>
<th>class1 CPU intensive</th>
<th>class2 I/O intensive</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>0.2</td>
<td>$0.6 \cdot \lambda_2 = 1$</td>
</tr>
<tr>
<td>DISK</td>
<td>0.5 \cdot 0.2</td>
<td>$\lambda_2 = 1$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\lambda_2 \leq 1.33$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\lambda_2 \leq 0.90$</td>
</tr>
</tbody>
</table>

The maximum input rate $\lambda_2$ is 0.90 and we can say that the disk is the bottleneck of the system when $\lambda_1 = 0.2$.

Suppose we want to improve the system and to do so we replace the CPU by a new one with double the speed of the old one. We can calculate the gain in throughput by using equations (3) and we find the following results:

- for CPU $\lambda_2 \leq 3.00$
- for DISK $\lambda_2 \leq 0.90$

We can see on Figure 2.3 that even if we double the speed of the CPU the gain in the throughput bound is zero. This is due to the fact that the system is disk bounded. An improvement on a device that is not the bottleneck of the system will not increase the system throughput. This kind of analysis is useful to find what device should be upgraded to increase the efficiency of a system.

In summary, bottleneck analysis is useful because:
- it is simple to apply and it is a quick method to obtain a rough feeling of the behavior of queuing network models.
- it reveals the critical influence of the bottleneck
of a service center. Changes to the system that do not affect the bottleneck center, do not alter the efficiency of the system.

- it is useful in the early stage of system design. Bottleneck studies can cut down rapidly the number of alternatives of the system models.
2.3- Queuing network analysis packages and QNAP2:

There are, on the market, some very good queuing network analysis packages like Best/1 [BGS] or QNAP2 [SIM].

Most queuing analysis packages can be understood in terms of the structure illustrated in Figure 2.5. There is a sequence of levels, each transforms inputs received from the above level into suitable outputs for the level below until the algorithms level has been reached where the model is solved. We will describe the major components of a queuing network analysis package.

The job of the lowest level (algorithms for resolution of queuing network model), in Figure 2.5, is to solve the model. Given a queuing network model defined by its customers description, service center description, and service demand, this level computes performance measures. This collection of routines can be based on exact, approximate or simulation algorithms.

The "language translator" level is responsible for analyzing the model definition coming from the user interface and to produce a set of outputs which are used by the lower level to solve the model. This level reads in a model description (program) written in a high level language, analyses the program and reports any error if necessary. When no error is detected, it generates outputs used in the solution phase by the lower level.
Figure 2.5 - General queuing analysis packages .structure
An important attribute of a queuing network analysis package is the convenient expression of performance models. The "user interface" level in Figure 2.5 links the user to the solvers via a convenient interface, where the analyst can run or debug a program. This interface can be seen as a high level language interpreter. This interpreter permits performance studies to be conducted efficiently. The feature provided by this level and the level below is a general language which is used by the analyst to program the package to study a particular model.

The "high level interface" is a program built on the top of the third level. For example it could be a translator which automates the labor involved in translating measurement data into a queuing network model inputs for the user interface level. A graphical interface is also a good example of such program. It is used to define a queuing network model using icons. There is a common attribute to these kind of interfaces; they use sophisticated ways to define a model and they generate a model representation which will be processed by the queuing network analysis package.

All these levels do not need to be present; simple queuing network analysis packages might consist only of the first level. However, the higher levels are important to the professional computer system performance
analyst.

Research efforts in the field of modeling and evaluation have resulted in a large number of solution techniques for queuing network models. These methods, like exact mean value analysis [LZGS1984], differ by their cost of operation and by the limitations they impose on the model. Each method realizes a certain compromise between cost and accuracy. It is important to be able to have access to these different techniques in order to select the one which represents the best compromise for the problem.

There are good reasons (speed or accuracy) to include more than one solution technique in a single analysis tool. In the absence of specific aids, the application of several solution methods, on a given model, is a long task. The analyst will have to restrict his attention to one method, the one he knows the best. It is better to provide computer systems designer with a tool which facilitates the practical use of these various methods. This is the main objective of general queuing network analysis packages.

2.3.1- QNAP2:

QNAP2 can be described as a system for describing, manipulating and solving queuing network models. It includes a specification language which is used for the
description of the model under study, the control of the solution, some reporting features and the choice among several solution modules or solvers, implementing different solution techniques.

The QNAP2 language allows the user to describe the following items:

- the network configuration:
  A queuing network consists of a set of stations through which circulate customers according to given routing rules. The customers may be distributed into several classes characterizing different routing behaviors and different processing in the stations.

- the processing done by each station:
  The processing may be described by a single time duration defined by its probability distribution or by a complex algorithm using built-in QNAP2 functions.

- the network solution control:
  The solution control specifies the initialization of parameters of the model, the activation and sequencing of the solution, and the editing of the results.

  The first level (algorithms for solving) in QNAP2 includes these different techniques:

  - discrete event simulation which is a complete simu-
lation program and can be used to solve any model.
- exact analytic solvers which include convolution and mean value analysis. These algorithms are restricted to a subset of queuing network models.
- approximate analytical solvers which include iterative methods, diffusion approximations and heuristic approaches.

Any of these solution techniques can be used easily by the analyst and yields standard performance results (utilization, throughputs,...) which characterize the stationary behavior of the network. Analytical methods may be applied only under restricted conditions for a subset of queuing network models. These conditions are automatically checked by QNAP2 before any treatment. Simulation is the only general solution technique which may always be used, but it may be costly to use in terms of processing requirement.

The second level (language translator) takes a model description program and transform it into a suitable form which can be used by the first level for solution.

The QNAP2 programming language is made of 2 levels: a control language and an algorithmic language. The control language is used to declare the model and to control the solution phase. It includes:
- a type declaration statement which is used to
declare every variable needed to define and solve the model.

- a "station" statement which is used to define a service center. In a station block the user defines everything needed to describe a service center like station type, scheduling discipline, . . .

- some other statements used to initialize model parameters, to change some solution parameters, etc.

The algorithmic language includes roughly the same features found in Pascal. It performs data items management for integer, real, string, array, lists, queue, etc. It includes I/O facilities that gives access to the screen, the keyboard, and files. The language is block oriented and supports procedures. The flow of control is assured by if-then-else blocks, while loops, for loops and by procedure calls. It includes many functions which can be divided into two groups: mathematical and statistical.

Figure 2.6 presents an example of QNAP2 language defining the model shown on Figure 2.3.

Although the language features are powerful they become cumbersome with a complex model. They permit a user to define, solve and investigate a model with a variety of solvers. The programming (building, typing and debugging programs), involved during the analysis, requires a certain amount of time which may be consider-
&
//declare/
queue  TERM,
     CPU,
     DISK;
&
class  CPU_INT,
       IO_INT;
&
&
/station/
    name = TERM;
    type = server, infinite;
    init(CPU_INT) = 5;
    init(IO_INT) = 5;
    service(CPU_INT) = exp(10);
    service(IO_INT) = exp(18);
    transit(CPU_INT) = CPU, 1.000000;
    transit(IO_INT) = CPU, 1.000000;
&
&
/station/
    name = CPU;
    type = server, multiple(1);
    sched = ps;
    service(CPU_INT) = exp(0.5);
    service(IO_INT) = exp(0.2);
    transit(CPU_INT) = TERM, 0.500000
                   , DISK, 0.500000;
    transit(IO_INT) = TERM, 0.166667
                   , DISK, 0.833333;
&
&
/station/
    name = DISK;
    type = server, multiple(1);
    sched = fifo;
    service(CPU_INT) = exp(0.08333);
    service(IO_INT) = exp(0.06667);
    transit(CPU_INT) = CPU, 1.000000;
    transit(IO_INT) = CPU, 1.000000;
&
&
&
/exec/
    begin
        solve();
    end;
/end/

Figure 2.6 - A QNAP2 program example
able when the model gets more complex. In this case a high level interface (level 4) may be useful to decrease the time spent in programming and in the same time increase the productivity of the analyst.

A weakness of QNAP2 is the difficulty to modify a model. For example, if a class has to be added, many statements have to be inserted at many places in the code. This becomes a real problem as the model gets complex.

Another problem with QNAP2, as a programming language, is the amount of programming time needed to solve a particular model. This time can be divided in three parts: a design part, a typing part and a debugging part.

In the design part the analyst transforms a queuing network model and its parameters into a sequence of QNAP2 statements. This transformation is generally straightforward. After this step, the analyst has to write a second block of code which will produce the solution of the model, using QNAP2 solvers, and will get back the desired results.

The typing part consists simply of the time required to key in the program produced by the design part, using the syntax supported by QNAP2. This syntax is quite strict and special attention is needed during this part to avoid syntax errors during the solution.
The debugging part may not be needed if the first two parts have been done methodically. If the typing has been done rapidly, we will probably have to correct only some syntax errors or minor typing errors. But if the design part was not done correctly, the debugging part may be very long. We may have to rebuild a part of the program or maybe the entire program.

To avoid unnecessary waste of time, a high level interface is very appropriate.
2.4- Spreadsheets:

Our new fourth level interface will take the form of a spreadsheet. A spreadsheet is defined as an array of cells organized into rows and columns. Each cell can hold a piece of information. In order to access a cell, the spreadsheet's columns are labeled A, B, C, ... and its rows are labeled 0, 1, 2, ... . Using these labels, we can refer to each cell by its column-row location or cell address. To understand how spreadsheets work we introduce VC, a public domain spreadsheet calculator running under UNIX [BOU1983].

VC is based on rectangular tables, in much the same style as Visicalc or Lotus 1-2-3. When it is invoked, it presents the user with an empty table of cells (Figure 2.7). Each cell may contain a label string or an expression or maybe both. The expression may be a constant or it may compute a value based on other entries (cells) using mathematical functions.

When VC is running, the screen is divided into four regions. The top line is for entering commands. The second line is for messages from VC. The third line and the first four columns of the screen show columns of letter codes and row numbers respectively. The rest of the screen forms a window looking at the table. The screen has two cursors: a cell cursor (cell pointer indicated by a < symbol) and a character cursor (indi-
Figure 2.7 - An empty spreadsheet table.
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASSES</td>
<td>TERM</td>
<td>CPU</td>
<td>DISK</td>
</tr>
<tr>
<td>14</td>
<td>utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CPU intensive</td>
<td>0.000</td>
<td>0.247</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>IO intensive</td>
<td>0.000</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>ALL_CLASSES</td>
<td>0.000</td>
</tr>
<tr>
<td>17</td>
<td>throughputs</td>
<td>CPU intensive</td>
<td>0.247</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td>IO intensive</td>
<td>0.246</td>
</tr>
<tr>
<td>19</td>
<td></td>
<td>ALL_CLASSES</td>
<td>0.493</td>
</tr>
<tr>
<td>20</td>
<td>response</td>
<td>CPU intensive</td>
<td>18.000</td>
</tr>
<tr>
<td>21</td>
<td></td>
<td>IO intensive</td>
<td>18.000</td>
</tr>
<tr>
<td>22</td>
<td></td>
<td>ALL_CLASSES</td>
<td>18.000</td>
</tr>
</tbody>
</table>

The natural tabular format is the starting point for the creation of a spreadsheet editor for queueing network model.
very useful when many cells have to be moved around.
- miscellaneous commands

Examples are the "redraw screen" command and the "exit" command.

Expressions that can be entered in a cell may include other cells address. This kind of expression links cells together. When the content of a cell entry in the link changes, every linked cell is recalculated and updated. This linking possibility is a very powerful mechanism. Expressions have a fairly conventional syntax. Terms may be addresses, parenthesized expressions, negation terms and constants. Terms may be combined using many binary operators (+, -, *, ...). They can include also mathematical functions (sin, cos, min, ...).

A spreadsheet editor is very powerful for handling data that fits its table format. Any data that falls into a tabular format will be handled very efficiently by a spreadsheet editor.
3- Tabular format for different types of data

From the discussion in chapter 2, a natural data tabular format is essential for the use of a spreadsheet editor.

3.1- Tabular format in linear equations

Visit ratios, in a queuing network model, are generally used in analytical methods and in bottleneck analysis. It is convenient to have a method to transform a probability matrix into a visit ratios vector. This transformation involves linear equations [SB1980].

A general set of $m$ linear equations and $n$ unknowns can be written in the matrix form:

$$
\begin{bmatrix}
  a_{11} & a_{12} & \cdots & a_{1n} \\
  a_{21} & a_{22} & \cdots & a_{2n} \\
  \vdots & \vdots & \ddots & \vdots \\
  a_{m1} & a_{m2} & \cdots & a_{mn}
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2 \\
  \vdots \\
  x_n
\end{bmatrix}
=
\begin{bmatrix}
  b_1 \\
  b_2 \\
  \vdots \\
  b_m
\end{bmatrix}
$$

The difficulty is to find a way to express visit ratios and probabilities in the general format presented above.
From the identity:

\[
\begin{bmatrix}
  P_{11} & P_{12} & \cdots & P_{1n} \\
  P_{21} & P_{22} & \cdots & P_{2n} \\
  \vdots & \vdots & \ddots & \vdots \\
  P_{n1} & P_{n2} & \cdots & P_{nn}
\end{bmatrix}^T
\begin{bmatrix}
  y_1 \\
  y_2 \\
  \vdots \\
  y_n
\end{bmatrix} =
\begin{bmatrix}
  y_1 \\
  y_2 \\
  \vdots \\
  y_n
\end{bmatrix}
\]  

(8)

we can derive a set of linear equations where the variables to determine are visit ratios \( y_i \). Because \( y_i \) are ratios we can fix one of them, for instance \( y_1 \), to any number, let say 1, and we derive the others from that. We can derive from the identity (8) the set of linear equations (9) expressed in matrix format:

\[
\begin{bmatrix}
  P_{22} & P_{32} & \cdots & P_{n2} \\
  P_{23} & P_{33} & \cdots & P_{n3} \\
  \vdots & \vdots & \ddots & \vdots \\
  P_{2n} & P_{3n} & \cdots & P_{nn}
\end{bmatrix}
\begin{bmatrix}
  y_2 \\
  y_3 \\
  \vdots \\
  y_n
\end{bmatrix} =
\begin{bmatrix}
  -P_{12} \\
  -P_{13} \\
  \vdots \\
  -P_{1n}
\end{bmatrix}
\]  

(9)

In the set of equations (9), all \( P \) are known and we have fixed \( y_1 \) equal to 1. The set represents \( n-1 \) equations with \( n-1 \) unknowns and is therefore solvable.

3.2- Tabular format in linear programming:

The most common type of application in linear programming involves the general problem of allocating limited resources among competing activities in the best possible way (i.e. optimal). The variety of situations to which this description applies is diverse. One of
these situations is the allocation of computer resources to get the best throughputs.

Linear programming uses a mathematical model to describe the problem of concern. The word linear means that all the mathematical functions in the model are required to be linear functions. The word programming does not refer to computer programming; but it is essentially a synonym for planning. Thus linear programming involves the planning of activities to obtain an optimal result.

A remarkable efficient solution procedure, called the simplex method, is available for solving linear programming problems of even huge size [HL1986]. We can now formulate the general mathematical model for the problem of allocating resources to activities. This model is used to select the values for \( x_1, x_2, \ldots, x_n \) to maximize the function:

\[
z = c_1 \cdot x_1 + c_2 \cdot x_2 + \ldots + c_n \cdot x_n
\]

subject to restrictions:

\[
a_{11} \cdot x_1 + a_{12} \cdot x_2 + \ldots + a_{1n} \cdot x_n < b_1 \\
a_{21} \cdot x_1 + a_{22} \cdot x_2 + \ldots + a_{2n} \cdot x_n < b_2 \\
\vdots \\
a_{m1} \cdot x_1 + a_{m2} \cdot x_2 + \ldots + a_{mn} \cdot x_n < b_m \\
x_1 > 0, \quad x_2 > 0, \quad \ldots, \quad x_n > 0.
\]

This is a standard form for a linear programming problem [HL1986]. The function \( z \) being maximized is
called the objective function. The restrictions normally are called constraints. The $x_j$ variables are the decision variables.

3.2.1- Linear programming with throughput bounds

Clearly, a linear programming model is represented by a tabular format of a set of equations and therefore it can be defined (and solved) within a spreadsheet editor. But more interesting, equations used in the bottleneck analysis look very similar to linear programming equations. Indeed these equations represent a linear programming model and can be solved in the same way (using the simplex method).

For the example of the last chapter, from equations (6) and (7), we can set up a series of $n$ equations derived from the bottleneck analysis method (chap 2.2) that looks like the standard form of a linear programming problem:

\[
\begin{align*}
\lambda_1 + 0.6 \cdot \lambda_2 &< 1.0 \\
0.5 \cdot \lambda_1 + 1.0 \cdot \lambda_2 &< 1.0 \\
\lambda_1 > 0, & \quad \lambda_2 > 0
\end{align*}
\]

The goal is to maximize throughputs in the system. Therefore the objective function is the sum of all throughputs:

\[
z = \lambda_1 + \lambda_2
\]
In Figure 2.4, we can see the constraint lines. Their points of intersection are the keys to the analysis. These points of intersection are called corner-point-feasible solutions of the problem.

In this example, only four corner-point-feasible solutions exist for \((\lambda_1, \lambda_2): (0,0), (1,0), (0.571, 0.714), (0,1)\). From these points, we can find the optimal solution:

\[
\begin{align*}
\lambda_1 &= 0.571 \\
\lambda_2 &= 0.714 \\
z &= 1.285
\end{align*}
\]

The simplex method finds a sequence of such points, with increasing values of \(z\). It can be automated easily and the optimal solution for large system can be computed efficiently.

3.3- Tabular format in queuing network models

The description of a queuing network model uses a set of parameters which describe the behavior of customers in the model. As described earlier (chapter 2), two of these parameters (type and scheduling discipline) are not class dependent and therefore they have to be specified only for each station. The others are class dependent and they have to be specified for each station and each class. Figure 2.3 shows a simple queuing network
model. It contains only three stations and two kinds of customers circulate in the network, "CPU intensive users" and "I/O intensive users". We can arrange the parameters into a standard tabular format that will fit a 2 dimensional spreadsheet:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TERM</td>
<td>CPU</td>
<td>DISK</td>
</tr>
<tr>
<td>0</td>
<td>type</td>
<td>ser,1</td>
<td>ser,1</td>
</tr>
<tr>
<td>1</td>
<td>sched</td>
<td>fifo</td>
<td>ps</td>
</tr>
<tr>
<td>2</td>
<td>CPU_int</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>3</td>
<td>Init</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>serv_t</td>
<td>exp,18</td>
<td>exp,0.5</td>
</tr>
<tr>
<td>5</td>
<td>TERM</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>CPU</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>DISK</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>IO_int</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>9</td>
<td>Init</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>serv_t</td>
<td>exp,18</td>
<td>exp,0.1</td>
</tr>
<tr>
<td>11</td>
<td>TERM</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>CPU</td>
<td>.167</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>DISK</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Any queuing network model can be represented in a similar table. The format used demonstrates how easy it is to fit a queuing network model into a spreadsheet format.

The solution of a queuing network model can also be expressed in a tabular format as follows:
<table>
<thead>
<tr>
<th>CLASSES</th>
<th>TERM</th>
<th>CPU</th>
<th>DISK</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU intensive</td>
<td>0.000</td>
<td>0.247</td>
<td>0.124</td>
</tr>
<tr>
<td>IO intensive</td>
<td>0.000</td>
<td>0.147</td>
<td>0.246</td>
</tr>
<tr>
<td>ALL_CLASSES</td>
<td>0.000</td>
<td>0.394</td>
<td>0.370</td>
</tr>
<tr>
<td>CPU intensive</td>
<td>0.247</td>
<td>0.494</td>
<td>0.247</td>
</tr>
<tr>
<td>IO intensive</td>
<td>0.246</td>
<td>1.474</td>
<td>1.226</td>
</tr>
<tr>
<td>ALL_CLASSES</td>
<td>0.493</td>
<td>1.968</td>
<td>1.475</td>
</tr>
<tr>
<td>CPU intensive</td>
<td>18.000</td>
<td>0.745</td>
<td>0.744</td>
</tr>
<tr>
<td>IO intensive</td>
<td>18.000</td>
<td>0.153</td>
<td>0.288</td>
</tr>
<tr>
<td>ALL_CLASSES</td>
<td>18.000</td>
<td>0.302</td>
<td>0.364</td>
</tr>
</tbody>
</table>

The natural tabular format is the starting point for the creation of a spreadsheet editor for queuing network model.
4- The spreadsheet implementation

This chapter describes the different implementation problems encountered during the spreadsheet design such as:

- the spreadsheet layout for classes and service centers.
- the syntax supported to define a queuing network model.
- the special performance model commands.
- the software structure.
- the special iteration feature.

4.1- Design considerations

Many decisions were made at an early stage of the spreadsheet design. Those decisions were based on the major characteristic that the interface should possess. The following subsections describe some of the major design considerations.

4.1.1- Interface portability

The interface portability issue is very important. The interface should run on many types of computers and under many types of environments.

To address the portability issue, we decided to use only standard "C" functions. These standard functions comprises the standard calls defined in the book
by Kernighan and Ritchie [KR1978]. We also decided to use the curses library which is supported by many types of computers. No special graphic or system dependant functions should be used to prevent any compatibility problem between different types of architecture or different display types. By using curses and standard text output, the spreadsheet can run on many types of computers (VAX, SUN workstations, HP workstations, ...). It can also be accessed from a personal computer using a modem connection which is not possible for a graphical interface.

4.1.2- Spreadsheet power

The spreadsheet power resides in many specific features (cell linking, block editing, ...). The performance spreadsheet should not restrain the use of these features. For example, cell linking should be available for call containing model parameters. These features make the spreadsheet editor efficient for the creation or modification of a model (EG. use of copy block to duplicate a station).

4.1.3- Data arrangement

The parameters entered to define a model should be arranged to make the model modification and readability easy.
To meet this requirement, we decided to display the station parameters in a column using a set of rows for each class. This way it is easy to add a station (add a column) or to add a class (add a set of rows).

The model readability is excellent due to the fact that all parameters related to a station are located in one column and all parameters related to a class are located in a set of adjacent rows.

4.1.4- QNAP2 independent interface

The spreadsheet implementation should use the same terminology used in queuing network modeling. The parameters should be defined and used in the same way. The spreadsheet interface should be as independent as possible from QNAP2 programming language. The user should have to know only about queuing network theory and some basic spreadsheet operations. This insures a short training time.

The independence from the language also permits the use of other queuing network package to solve without having to add new commands in the spreadsheet interface. Only the code that is QNAP2 dependant will have to be modified to accommodate a new solution package.

4.1.5- QNAP2 features

The spreadsheet interface should retain as many
features as possible present in QNAP2 (eg. multiple classes, class chain, solution techniques, ...).

4.1.6- Interface completeness

The spreadsheet interface should be a complete environment. This means that all the functionality required to create, modify and solve queuing network models must be present. This includes presenting the results within the spreadsheet directly.

4.2- Service centers and classes organization

The first basic point in the design is to choose how to display classes and service centers. Three points have to be considered to make that decision. First the ordinary 2 dimensional spreadsheet layout uses 1 line per row and many characters per column. Second the screen displays more rows than spreadsheet columns and finally for n service centers, at least n rows are required to specify the probability matrix plus the rows needed to define the class dependent parameters.

The above three points show that more parameters are defined along the class axis than along the service center axis. Therefore it is logical, to get the best view of the model within the spreadsheet, to display the service centers horizontally and the classes vertically.

Two types of parameters are needed to define a
queueing network model. The first kind is the class independent parameters "the station type" and "the scheduling discipline". This type of parameter is specified only once for the service center. Since there are only two parameters of this nature, they are placed in the two first spreadsheet rows just above the class dependent parameters. All these class dependent parameters are grouped together for each class and each group is separated by the class name and a dashed line.

Figure 4.1 shows how the classes and service centers are displayed within the spreadsheet for the model presented in chapter 2 Figure 2.3.

It is important to note that the station names, the class names and the parameter names are part of the spreadsheet border and not part of the cell table. This means that the names remain visible when the cursor moves too far to the left or to the bottom of the table.

It is also important to note that when a new station or a new class is inserted in the model, the spreadsheet takes care of adding the necessary rows and columns needed to define the performance parameters.

4.3- Model definition

All parameters that have to be specified to define a model have to be entered in a spreadsheet cell. Therefore the cell content limitations have to be def-
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU_INT</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>ser,1</td>
<td>ser,1</td>
<td>ser,1</td>
<td></td>
</tr>
<tr>
<td>sched</td>
<td>fifo</td>
<td>ps</td>
<td>ps</td>
<td></td>
</tr>
<tr>
<td><strong>prio</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>init</td>
<td>5.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>servt-1</td>
<td>exp</td>
<td>18.000</td>
<td>exp</td>
<td>0.500</td>
</tr>
<tr>
<td>servt-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rate_svr</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>visit</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TERM</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>CPU</td>
<td>0.500</td>
<td>0.000</td>
<td>0.500</td>
<td></td>
</tr>
<tr>
<td>DISK</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td><strong>IO_INT</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>prio</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>init</td>
<td>5.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>servt-1</td>
<td>exp</td>
<td>18.000</td>
<td>exp</td>
<td>0.100</td>
</tr>
<tr>
<td>servt-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rate_svr</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>visit</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TERM</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>CPU</td>
<td>0.167</td>
<td>0.000</td>
<td>0.833</td>
<td></td>
</tr>
<tr>
<td>DISK</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td><strong>case 1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mservice</td>
<td>CPU_INT</td>
<td>18.000</td>
<td></td>
<td>0.500</td>
</tr>
<tr>
<td></td>
<td>IO_INT</td>
<td>18.000</td>
<td></td>
<td>0.100</td>
</tr>
<tr>
<td></td>
<td>ALL_CLASS</td>
<td>18.000</td>
<td></td>
<td>0.201</td>
</tr>
<tr>
<td></td>
<td>mbusypct</td>
<td>CPU_INT</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IO_INT</td>
<td>0.000</td>
<td></td>
<td>0.147</td>
</tr>
<tr>
<td></td>
<td>ALL_CLASS</td>
<td>0.000</td>
<td></td>
<td>0.395</td>
</tr>
<tr>
<td></td>
<td>mresponses</td>
<td>CPU_INT</td>
<td>18.000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IO_INT</td>
<td>18.000</td>
<td></td>
<td>0.153</td>
</tr>
<tr>
<td></td>
<td>ALL_CLASS</td>
<td>18.000</td>
<td></td>
<td>0.302</td>
</tr>
<tr>
<td></td>
<td>mthrput</td>
<td>CPU_INT</td>
<td>0.247</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IO_INT</td>
<td>0.246</td>
<td></td>
<td>1.474</td>
</tr>
<tr>
<td></td>
<td>ALL_CLASS</td>
<td>0.493</td>
<td></td>
<td>1.968</td>
</tr>
</tbody>
</table>

Figure 4.1 - Spreadsheet example
nowing these limitations will help to find how queuing parameters can be defined in the cell.

The cell, addressed within the spreadsheet by its column and row coordinate, can store two pieces of information: a value or an equation leading to a real value, and a string. The value or the equation value is a floating point number. The string is a sequence of ASCII characters as long as 255 characters. Those two storage areas, the value and the string, can be used to store the different parameters.

The model example introduced in chapter 2 Figure 2.3 will be used through this chapter to show how queueing network models are handled within QUEST. The spreadsheet representation of the model is given in Figure 4.1.

4.3.1. The "type" parameter

The "type" parameter defines the kind of service center that is being modeled. Four types of service centers can be defined: server, source, resource and semaphore.

A server may be regarded as a processor performing a service for customers. A source station works as an infinite producer of customers. A resource is a station consisting of a queue and one or more resource units. Each customer may request several resources and if the
request is not satisfied, the customer is blocked in the queue until the request can be satisfied. A semaphore station consists of a queue and a counter. The counter is the number of tokens available if positive, and the number of customers waiting if negative.

To define a "type", we need to specify the kind of service center and how many servers there are in the service center. To enter the definition in a cell, the 3 first letters of the "type" are used (eg: ser for server) and the number of servers can be entered in two ways: by placing a number in the cell value or by placing the number within the string beside the three letters abbreviation.

The special case of the infinite number of servers can be defined by placing "i" in the cell string.

A special syntax will be used to describe the different possible entries for a cell. The symbol OR represents a "OR", the symbol AND represents a "AND", the and the parenthesis are used for optional parameter and the indicates that the parameter can be entered as a list for multiple runs. The ## indicates that the parameter is a numerical value or an equation that when evaluated gives a numerical value.
To define the type we can use:

```
  label — [ser@res@soup@sesm]@[###]@[I]
```

```
  cell —
    label — [ser@res@soup@sesm]
    value — [###]
```

In Figure 4.1, two kinds of "type" are used to define the model. The infinite server (A0) is used to model the terminal center and the single server (B0 & C0) is used to model the processor and the disk.

The "type" parameter is not class dependent and it is placed in the first row of the spreadsheet; it is not repeated for each class.

4.3.2- The scheduling discipline

The scheduling discipline defines how the customers are handled in the queue. There are 4 general disciplines that may be used: First In First Out (FIFO), Last In First Out (LIFO), quantum (QUAN) and processor sharing (PS). Two parameters are optional and can be specified with the scheduling discipline: the priority (PRI0) and the preemption (PREMPT).

These four disciplines and the optional arguments are entered in the cell string using the abbreviated form shown within the parentheses.

To define the scheduling discipline we can use:

```
cell — label — [fifo@lifo@ps@quan]@[pri0]@[preem]`
```
In Figure 4.1, two kinds of scheduling are used: processor sharing (B1) and first in first out (A1 & C1). The scheduling discipline is not class dependent and it is placed directly below the "type" parameter in the second row. It is not repeated for each class.

4.3.3- The "priority" parameter

The "priority" parameter defines the priority of a certain class of customers. The priority is entered, as an integer, in the cell string or in the cell value.

To define the priority we can use:

```
label — [###]
```

```
value — [###]
```

The priority scheduling is not used for the model in Figure 4.1, so the "prio" lines are empty.

The priority is a class dependent parameter and it has to be repeated for all classes. Note that this parameter needs to be defined only if the priority argument has been selected in the scheduling parameter.

4.3.4- The initial number of customers

This parameter specifies the number of customers for each class and for each service center. The number is entered as an integer and represents the initial system state for a simulation; for an analytic solution only the total for each class is required. The number
is entered in the cell string or in the cell value.

To define the initial number of customers we can use:

\[
\begin{align*}
\text{cell} & \quad \text{label} \quad [\#\#]_n \\
\text{value} & \quad [\#\#]
\end{align*}
\]

We note that in Figure 4.1, the only customers present are in the terminal center (A4 & A14).

This parameter is class dependent and it has to be repeated for all classes.

4.3.5- The service time

The service time is the time spent by a customer in the station (not including queuing delay). This parameter is specified using the probability distribution followed by one or two floating point numbers representing the distribution parameters. Two rows are allowed for this parameter. Sometimes only one row is used and sometimes both rows are used depending on the type of distribution defined. The following distributions are the most commonly used:

- the Constant distribution (CST) needs one parameter to be defined: the constant load.
- the Exponential distribution (EXP) needs one parameter to be defined: the mean.
- the Hyper-exponential distribution (HEXP) needs two parameters to be defined: the mean and the squared
coefficient of variation.
- the Erlang distribution (ERL) needs two parameters to be defined: the mean and the number of stages.
- the Uniform distribution (UNI) needs two parameters to be defined: the starting and the ending point.
- the Normal distribution (NOR) needs two parameters to be defined: the mean and the standard deviation.

The service time can be entered in a cell using the following convention:

```
label — [cst@exp]@[###]n
```

```
label — [cst@exp]
```

```
value — [###]
```

```
label — [hex@erl@uni@nor]@[###]n
```

```
label — [hex@erl@uni@nor]
```

```
value — [###]
```

```
label — [###]n
```

```
value — [###]
```

```
cell1 — label — [serv_###]
```

The service time can be a much more complicated function, in this case a short program can be written in a file called "serv_mmm" where mmm is a number from 1 to 999. The program will be used during the solution phase as the service time for the station and the class.

In Figure 4.1, the cells A5 through C5 and the cells A15 through C15 show how the service time is
entered for the model.

The service time is a class dependent parameter and it has to be specified for each class.

4.3.6- The rate dependent server

Certain types of system do not serve customers at the same rate when the number of customers in the queue varies. This parameter is a list of numbers used to change the rate at which a customer is served. The service time defined earlier is divided by the first number in the list when one customer is present in the queue. The second number is used when two customers are waiting in the queue and so on.

The list is entered in the cell string and each number is separated using a coma.

cell — label — [###,###,...]

This parameter is not used for the example of Figure 4.1.

This parameter is class dependent and has to be specified for each class.

4.3.7- The visit ratio parameter

The visit ratio parameter defines how many times a customer visits a station relative to a reference station, per request as described in section 2.1. This parameter is a floating point value and can be entered
in the cell string or in the cell value.

\[
\text{cell} \quad \begin{array}{c}
\text{label} \quad [###]^n \\
\text{value} \quad [###]
\end{array}
\]

This parameter is not used for the model example. Instead, probability routing parameters are given.

This parameter is class dependent and it has to be specified for each class.

4.3.8- The routing parameter

The routing parameter defines how the customers circulate within the network. To define the routing, we have to specify the probability for a customer to go from a station to another station. The defined stations are displayed both horizontally and vertically to create enough room for the probability matrix for each class. This way the probability matrix is expressed in a natural fashion. This ensures a good readability of the routing parameters. The probability is entered directly in the cell value.

\[
\text{cell} \quad \begin{array}{c}
\text{label} \quad [\text{special class change syntax}] \\
\text{value} \quad [###]
\end{array}
\]

In Figure 4.1, the cell B9 is the probability for a customer in class CPU_INT to go from the terminal to the CPU. This parameter is class dependent and it has to be specified for each class.
4.3.9- Multiple runs

The multiple runs feature is used when many solutions are needed for many parameter values. A list of parameters, each separated with a comma, may be entered in the label of any cell and for each member of the list, a solution is generated. If many cells have lists, the first solution uses the first parameter in all lists, the second solution uses the second parameter and this process goes on until the end of the list is reached. Then the last parameter within the list is used until the longest list has been processed. Most of the parameters defined earlier can be entered as a list. For example, the initial number of customers can be entered as a list to investigate at what load the system saturates. This feature will be used later in this thesis.

4.3.10- Class changes

An important modeling tool is the ability for a customer to change class while it is circulating within the system. This ability is needed in many models for routing purpose.

It is important to introduce a new syntax permitting the probability to be entered with class changes. To express a class change, we need to specify the probability for a customer to go from one station in a cer-
tain class to another station into another class. Instead of specifying the probability alone we now have to specify a probability and an arrival class. We also have to give these two parameters for all classes. The class index (first class in the model is class 1, second is class 2, etc.) and the probability are entered in the cell label separated by a comma and placed within brackets (e.g. [1,.2]). Each pair of parameters within brackets are separated by a comma. For example the list [1,.2], [2,.8] means the customer has a probability of .2 to change into a class 1 customer and a probability of .8 to change into a class 2 customer.

4.4- Special performance model commands

A large number of special commands have been provided for performance modelling.

The general spreadsheet commands are accessible directly by pressing a combination of keys (e.g.: c for copy). On the other hand the commands used to define and solve a performance model are menu driven. Figure 4.2 shows all the available menu driven commands.

The different menu choices are displayed horizontally on the first line and a cursor indicates the present choice. The second line displays a short message that gives information about the command.
Figure 4.2 - Performance menu structure
4.1.1- The "model" command

The "model" command is the first step to define a model. When it is selected, the second menu level gives access to all commands needed to define and solve a model.

The "routing" command determines what parameters will have to be given to route the customer within the model.

The "workspace" command is used to add, rename, remove or reorder stations or classes. This command has been used for the model example to define the station names (TERM, CPU, DISK) and the class names (CPU_INT, IO_INT). The "method" command is used to select the set of solution techniques that will be used to solve the model. The "which" command is used to select the desired results (utilization, response time) and for which stations and classes they are requested. The "format" command is used to specify an output program that the user wants to append to the QNAP2 program generated by the spreadsheet. The "print" command prints the table on strips of paper that can be placed side by side to reconstruct the spreadsheet format. The "go" command is used to generate the QNAP2 program and to solve the model.
4.4.2- The "immediate" command

The "immediate" commands are used within the spreadsheet to get immediate results that are not calculated from the external package QNAP2. The "prob=>visit" command is used to get the visit ratio vector from the probability matrix. The "visit=>prob" command works the other way around. The "bottleneck" command is used to get the model throughput bounds. The "simplex" command is used to solve linear programming problems and the "lineq" command is used to solve a set of linear equations.

4.4.3- The "class_up" and "class_down" commands

The "class_up" and "class_down" commands are used to move the cursor one class up or one class down in the spreadsheet.

4.4.4- The "fill" command

The "fill" command fills empty cells within the model with default values. For example, if a probability cell is left blank, a zero will be placed in the cell or if a scheduling cell is empty then "fifo" will be entered in the cell.

4.4.5 The "help" command

The "help" command is used to get help on different
topics. The second menu level gives topics on which the user can get help.

When a topic is selected, an on line user manual is presented on the selected topic and it is possible to move through the different pages containing information on the subject, using the cursor.

4.5- Solver

This section describes the functions in QUEST which solve models.

4.5.1- General spreadsheet overview

The spreadsheet program is made of many modules and each of these modules has its own task to perform. Figure 4.3 shows the modules necessary to QUEST for performing its task.

The external module "user" represents the computer screen and keyboard. The "QNAP2 package" module represents the external software library used to solve the queuing network model.

The "user interface" module is the link between the user and the QUEST commands. It parses the user entry and sends to the appropriate module the entered command.

The "file I/O" module is responsible for all file manipulations like "PUT", "GET" and "PRINT". The "lexical analyzer and reevaluation" module is responsible to
Figure 4.3 - QUEST modules
keep all the cells updated. Each time a cell changes, it recalculates the entire cell table starting from the left to the right and going from the top to the bottom. It keeps recalculating the cells until no cell is changed within the table.

The "general command" module performs all the general spreadsheet commands like cursor movements, block operations (copy, delete), etc.

The "performance commands" module performs all the queuing network model related tasks like adding a station or choosing the solution technique. It also dispatches to other modules some commands that need more calculation or more work like the bottleneck analysis command. These commands are described in more details in the following sections using the model already defined in Figure 4.1.

4.5.2 The probability matrix to visit ratios module

This module uses the technique described in section 3.1 to set up a set of linear equations from the probability matrix. From that set of equations it derives a set of visit ratios that will be inserted directly in the model. To derive the set of visit ratios, the command uses the Gauss Jordan method (linear equation solver module).

When the command is performed on the model pre-
sented in Figure 4.1, the resulting spreadsheet (Figure 4.4) shows the visit ratios in row 8 and in row 18.

4.5.3 The linear equation solver module

Since the linear equation solver is needed to transform probabilities to visit ratios, the spreadsheet makes it available directly to solve any set of linear equations.

The linear equations are entered in the spreadsheet on the format presented in section 3.1. The "b" coefficients are placed beside the matrix. To solve the equations, the command "LINEQ" from the performance menu is chosen. Then the matrix location has to be entered as a set of two coordinates. The first coordinate gives the location of the top left corner of the coefficient matrix and the second coordinate gives the location of the bottom right corner of the matrix. The solution is displayed directly below the coefficient matrix. For example if we consider the following set of equations:

\[ \begin{align*}
  x_1 + 3 x_2 + x_3 &= 8 \\
  2 x_1^2 + 3 x_2 + 4 x_3 &= 3 \\
  4 x_1^2 + x_2 + 2 x_3 &= 2
\end{align*} \]

The equations can be entered in the spreadsheet and solved as shown on Figure 4.6. We see that the solution is \( x_1 = 1.90, \ x_2 = 2.80 \) and \( x_3 = -2.30 \).

This command gives an easy way to solve a linear equation system.
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>TYPE</td>
<td>SER</td>
<td>SER</td>
</tr>
<tr>
<td>1</td>
<td>SCHED</td>
<td>FIFO</td>
<td>PS</td>
</tr>
<tr>
<td>2</td>
<td>CPU_INT</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>3</td>
<td>PRI</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>4</td>
<td>INIT</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>5</td>
<td>SERV-1</td>
<td>EXP</td>
<td>0.500</td>
</tr>
<tr>
<td>6</td>
<td>SERV-2</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>7</td>
<td>RATE_SVR</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>8</td>
<td>VISIT</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>TERM</td>
<td>0.000</td>
<td>1.000</td>
</tr>
<tr>
<td>10</td>
<td>CPU</td>
<td>0.500</td>
<td>0.000</td>
</tr>
<tr>
<td>11</td>
<td>DISK</td>
<td>0.000</td>
<td>1.000</td>
</tr>
<tr>
<td>12</td>
<td>IQ_INT</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>13</td>
<td>PRI</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>14</td>
<td>INIT</td>
<td>1.000</td>
<td>0.000</td>
</tr>
<tr>
<td>15</td>
<td>SERV-1</td>
<td>EXP</td>
<td>5.000</td>
</tr>
<tr>
<td>16</td>
<td>SERV-2</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>17</td>
<td>RATE_SVR</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>18</td>
<td>VISIT</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>19</td>
<td>TERM</td>
<td>0.000</td>
<td>1.000</td>
</tr>
<tr>
<td>20</td>
<td>CPU</td>
<td>0.167</td>
<td>0.000</td>
</tr>
<tr>
<td>21</td>
<td>DISK</td>
<td>0.000</td>
<td>1.000</td>
</tr>
</tbody>
</table>

Figure 4.4 - Probability to visit ratio example

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>1</td>
<td>1.000</td>
<td>3.000</td>
<td>1.000</td>
<td>8.000</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2.000</td>
<td>3.000</td>
<td>4.000</td>
<td>3.000</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2.000</td>
<td>1.000</td>
<td>2.000</td>
<td>2.000</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1.900</td>
<td>2.800</td>
<td>-2.300</td>
<td>---</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.5 - Linear equation example
4.5.4- The bottleneck analysis module

This module uses the technique presented in section 2.2 to set up a linear programming problem (section 3.2.1). The linear programming problem is then solved using the "simplex evaluation" module (simplex method). This finds the maximum throughput that the system can handle.

As soon as the command is performed, the set of equations is built in a spreadsheet space below the model as we can see on Figure 4.6. The coefficient matrix is displayed in rows 24 and 25 columns B and C. The objective function is entered directly below the matrix on row 26. The solution is given below the objective function for each class and the maximum throughput for all classes is displayed beside the solution for each class. We remark that the solution derived earlier in section 3.2.1 gives the same results as the ones derived within QUEST.

Row 28 is used to calculate a class throughput when all the others are furnished. For example 0.2 was entered in cell B28 and the calculated throughput command was entered. The 0.90 was calculated and placed in the empty cell C28. The sum of all throughputs was calculated and entered in cell E28 and the throughput for each device was calculated and entered in cells F24 and F25, on the same row where the device names are given.
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>TERM</td>
<td>CPU</td>
<td>DISK</td>
<td>CPU</td>
<td>DISK</td>
<td>CPU</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>ser.1</td>
<td>ser.1</td>
<td>ser.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sched</td>
<td>fifo</td>
<td>ps</td>
<td>ps</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>prev</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>init</td>
<td>5.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>servt-1</td>
<td>exp</td>
<td>18.000</td>
<td>exp</td>
<td>0.500</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rate_svr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>visit</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>TERM</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>CPU</td>
<td>0.500</td>
<td>0.000</td>
<td>0.500</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>DISK</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>IQ_INT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>prev</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>init</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>servt-1</td>
<td>exp</td>
<td>5.000</td>
<td>exp</td>
<td>0.100</td>
</tr>
<tr>
<td>15</td>
<td>servt-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rate_svr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>visit</td>
<td>1</td>
<td>6</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>TERM</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>CPU</td>
<td>0.167</td>
<td>0.000</td>
<td>0.833</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>DISK</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>CPU_INT</td>
<td>IQ_INT</td>
<td>thruput</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>CPU</td>
<td>1.000</td>
<td>0.600</td>
<td>1.000</td>
<td>1.533</td>
</tr>
<tr>
<td>24</td>
<td>DISK</td>
<td>0.500</td>
<td>1.000</td>
<td>1.000</td>
<td>1.100</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>MaxThrput</td>
<td>0.571</td>
<td>0.714</td>
<td>1.286</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>Thruput</td>
<td>0.200</td>
<td>0.900</td>
<td>1.100</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 4.6 - Bottle neck analysis example**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.000</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>=</td>
</tr>
<tr>
<td>1</td>
<td>0.000</td>
<td>2.000</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>=</td>
</tr>
<tr>
<td>2</td>
<td>3.000</td>
<td>2.000</td>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
<td>=</td>
</tr>
<tr>
<td>3</td>
<td>3.000</td>
<td>5.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2.000</td>
<td>6.000</td>
<td>2.000</td>
<td>0.000</td>
<td>0.000</td>
<td>=</td>
</tr>
</tbody>
</table>

**Figure 4.7 - Linear programming example**
If we look at Figure 2.4, we remark that if we trace a straight horizontal line at a value of 0.2 for the CPU intensive class, the line crosses the device bound lines. The two intersections with the bound lines (0.9 and 1.33) are the numbers derived in row 24 and 25 column F minus the CPU intensive contribution which is 0.2.

4.5.5- The simplex evaluation module

Since the simplex evaluation method is needed for the bottleneck analysis, the spreadsheet makes it available directly to solve linear programming problems.

Each constraint equation is entered in the spreadsheet using the following syntax:

\[ a x_1 + b x_2 + c x_3 < d \]

\[
\begin{array}{cccc}
A & B & C & D & E \\
0 & a & b & c & < d \\
1 & & & & \\
\end{array}
\]

where column D could equally contain "=" or ">".

All the constraint equations are entered one after the other in the spreadsheet. Finally the function to maximize is entered as the last line of the system.

To solve the linear programming problem, the simplex command has to be selected in the menu and the problem location in the spreadsheet has to be entered.

The following example is entered and solved in the
spreadsheet:

\[
\text{Maximize } z = 3x_1 + 0.5x_2
\]

\[
\text{Subject to:}
\begin{align*}
1x_1 + 0x_2 + 1x_3 + 0x_4 + 0x_5 &= 4 \\
0x_1 + 2x_2 + 0x_3 + 1x_4 + 0x_5 &= 12 \\
3x_1 + 2x_2 + 0x_3 + 0x_4 + 1x_5 &= 18
\end{align*}
\]

Figure 4.7 shows how the example is entered and how the results are presented in the spreadsheet. We recognize the constraint equation in the row 0 to row 2 and the objective function in row 3. We remark that the column A represents the variable \( x_1 \), the column B represents the variable \( x_2 \) and so on. The solution is displayed directly below the objective function in the row 4 and the value for the objective function is displayed in cell G4. The optimal solution is:

\( x_1 = 2.00, x_2 = 6.00, x_3 = 2.00, x_4 \) and \( x_5 = 0 \).

4.5.6 The visit ratios to probabilities module

This module is used to derive a set of probabilities from a visit ratio vector. This means deriving \( n^2 \) probability values from \( n \) visit ratio values; clearly there will not be a unique solution. We simply seek one solution that corresponds to the visit ratios.

To do the derivation, a general queuing network model is used (Figure 4.8) and from this model a set of probabilities are derived from the visit ratios. Note that the method can be used only if the model is solved
Figure 4.8 - General queuing network model

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TERM</td>
<td>CPU</td>
<td>DISK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>type</td>
<td>ser.1</td>
<td>ser.1</td>
<td>ser.1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>sched</td>
<td>fifo</td>
<td>ps</td>
<td>ps</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>CPU_INT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>prio</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>init</td>
<td>5.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>servt-1</td>
<td>exp</td>
<td>18.000</td>
<td>exp</td>
<td>0.500</td>
</tr>
<tr>
<td>6</td>
<td>servt-2</td>
<td></td>
<td></td>
<td></td>
<td>0.500</td>
</tr>
<tr>
<td></td>
<td>rate_svr</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>visit</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>TERM</td>
<td>0.100</td>
<td>0.000</td>
<td>0.900</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>CPU</td>
<td>0.000</td>
<td>0.950</td>
<td>0.050</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>DISK</td>
<td>0.900</td>
<td>0.100</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>IO_INT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>prio</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>init</td>
<td>5.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>servt-1</td>
<td>exp</td>
<td>18.000</td>
<td>exp</td>
<td>0.100</td>
</tr>
<tr>
<td>15</td>
<td>servt-2</td>
<td></td>
<td></td>
<td></td>
<td>0.200</td>
</tr>
<tr>
<td>16</td>
<td>rate_svr</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>visit</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>TERM</td>
<td>0.100</td>
<td>0.000</td>
<td>0.900</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>CPU</td>
<td>0.000</td>
<td>0.317</td>
<td>0.683</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>DISK</td>
<td>0.180</td>
<td>0.820</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td>case 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>mservice</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>CPU_INT</td>
<td>18.000</td>
<td>0.500</td>
<td>0.500</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>IO_INT</td>
<td>18.000</td>
<td>0.100</td>
<td>0.200</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>ALL_CLASS</td>
<td>18.000</td>
<td>0.201</td>
<td>0.250</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>mbusycpu</td>
<td>CPU_INT</td>
<td>0.000</td>
<td>0.247</td>
<td>0.124</td>
</tr>
<tr>
<td>29</td>
<td>IO_INT</td>
<td>0.000</td>
<td>0.147</td>
<td>0.246</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>ALL_CLASS</td>
<td>0.000</td>
<td>0.395</td>
<td>0.369</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>mresps</td>
<td>CPU_INT</td>
<td>18.000</td>
<td>0.745</td>
<td>0.744</td>
</tr>
<tr>
<td>32</td>
<td>IO_INT</td>
<td>18.000</td>
<td>0.153</td>
<td>0.288</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>ALL_CLASS</td>
<td>18.000</td>
<td>0.302</td>
<td>0.364</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>mthruput</td>
<td>CPU_INT</td>
<td>0.247</td>
<td>0.494</td>
<td>0.247</td>
</tr>
<tr>
<td>35</td>
<td>IO_INT</td>
<td>0.246</td>
<td>1.474</td>
<td>1.228</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>ALL_CLASS</td>
<td>0.493</td>
<td>1.968</td>
<td>1.475</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.9 - Visit ratio to probability example
analytically. Generally the analytical algorithms do not use probabilities but instead internally use visit ratios. Therefore if a set of probabilities is derived that gives the correct visit ratios, the results calculated from the derived probabilities will be correct. This function is included because QNAP2 does not allow visit ratio routing.

The following steps are necessary to perform the transformation:

- reorder the stations in increasing order of visit ratio. This is to make sure that all probabilities will be less than one.

- set $p_{11} = .1$

- the sum of the probabilities of leaving station 1 is equal to 1. Therefore $p_{12} = 1 - p_{11}$.

- from the balance law we have:
  \[
  \begin{align*}
  \frac{y_1}{y_2} &= \frac{p_{21}}{p_{12}} \\
  p_{21} &= \frac{y_1 \cdot p_{12}}{y_2}
  \end{align*}
  \]

- $p_{23} = 1 - p_{21}$

- $p_{n-1,n} = 1 - p_{n-1,n-2}$

- $p_{n,n-1} = \frac{y_{n-1}}{y_n}$

- $p_{n,n} = 1 - p_{n,n-1}$

Once the probabilities are calculated they are placed back in the spreadsheet.

Figure 4.9 shows the probabilities derived from the example model. We remark that the probabilities are
completely different from the values in Figure 4.1, however they give the same visit ratios and hence when the model is solved, the results are identical.

4.5.7- The code generator module

The code generator uses the model defined in the spreadsheet to generate a QNAP2 program that obeys the QNAP2 modelling syntax described in [SIM].

The code generator module uses the station and class names from the model to name the stations and classes in the program. The program created is readable and it can be easily edited to modify certain parameters. When multiple runs are requested, a FOR LOOP is used to permit the multiple run feature.

The first part of program generated to solve the example model is presented in Figure 4.10.

The first declare section declares some general variables that can be used in the program if someone decides to edit the program or within a special service time file serv_mmm. The control statements are used to set some initial conditions used by the solver package. The second declare section declares the different variables needed to solve the model. The station statements are used to define the model stations and how they behave (service time, routing...). For more details on the language syntax see [SIM].
Figure 4.10 - The QNAP2 generated program
The solution portion is not presented here and contains only "Get", "Solve" and "Print" statements that are used respectively to get the parameters from the spreadsheet, to solve the model and to return the results to the spreadsheet.

The QNAP2 program is very readable and a person knowing the QNAP2 language syntax can easily edit the model and make some changes.

4.5.8- The solve and get results module

This module, after the program generation, starts the queuing network package and waits for the solution. When the solution is available, it places the results back in the spreadsheet as shown in Figure 4.1.

The solution is placed just below the model. The stations, for which the results were requested, are displayed horizontally. The results are given for all chosen classes and for all classes grouped together.

If a multiple run condition was entered by using parameter lists then a set of solutions is displayed. Each solution is preceded by a case number that corresponds to a specific run.

4.6- Specific cell entries provided for iteration

Some queuing models require an iterative solution, in which a model is evaluated then some of its parame-
ters are changed and then the model is evaluated again. This continues until a stopping condition is met. Iteration may be used in an approximate solution, as in [HT1982], or in an optimization process.

The iteration process is provided through two new functions: the "@val" and "@ite" functions. The "@val" function is entered in a cell label and contains, as argument, a cell address. To show the syntax, "@val(G2)" means that the cell containing the function will get the value of the cell G2, but from the previous iteration. It will get the value only after everything else has been updated in the spreadsheet, and the new value, placed in G2, will not initiate any further evaluation of the sheet. The second function, "@ite", is also entered in any cell label and contains, as arguments, a cell address and an integer which represents the maximum number of iterations. "@ite(F12,5)" means that cell F12 is the stopping condition and 5 is the maximum number of iterations. We describe, now, how the iteration process is accomplished. When "@ite(F12,5)" is entered:

- QUEST verifies if the maximum number of iterations has been reached and it verifies if the value in the stopping condition cell F12 is zero.
- if one or both of these conditions is met the iteration process stops, the total number of iterations
is placed as the value of the cell that contains "@ite(F12,5)" and the sheet is updated.
- if neither of the conditions is met, the model is solved and all cells are re-evaluated.
- repeat the 3 steps above.

4.6.1- Example of iteration

We can set up a small optimization problem for an iteration demonstration (Figure 4.11). A processor can access 2 disks that have two different access times. We want to find what are the probabilities to access each of these disks that will equalize their utilizations.

Figure 4.12 shows QUEST ready for the iteration process and it also gives the hidden links that are used within the process. We start the evaluation with a small probability of 0.15 for accessing disk2. This small probability gives us a small utilization of disk2 and a large utilization of disk1. At each iteration this probability will be incremented by a step of 0.05. Then during the process we will cover probabilities of 0.2, 0.25, 0.30, ... until the stopping condition is met, when the utilization of disk2 becomes bigger than the utilization of disk1. To increment the probability at each iteration we place, in cell F10, the function "@eval(g2)" where g2 initially contains 0.15. We place, in cell F9, the expression: F10 + 0.05, which computes
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>type</td>
<td>ser.1</td>
<td>ser.1</td>
<td>ser.1</td>
<td>ser.1</td>
<td>I/O intensity: p21 = 0.100</td>
<td>init num: 50.000</td>
<td>p23 = 0.750</td>
</tr>
<tr>
<td>1</td>
<td>sched</td>
<td>fifo</td>
<td>ps</td>
<td>fifo</td>
<td>fifo</td>
<td>p24 = 0.150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>IO_int</td>
<td>50 000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>pri</td>
<td>50 000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>servt-1</td>
<td>18 000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>servt-2</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>rate_svc</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>visit</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>TERM</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>CPU</td>
<td>0.100</td>
<td>0.000</td>
<td>0.750</td>
<td>0.150</td>
<td>@val(q2) 0.150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>DISK1</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>DISK2</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>cpu_int</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>1.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>servt-3</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>servt-4</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>rate_svc</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>visit</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>TERM</td>
<td>0.000</td>
<td>0.000</td>
<td>0.100</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>CPU</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>DISK1</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>DISK2</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>case 1</td>
<td>DISK1</td>
<td>DISK2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>mbmaxpct</td>
<td>0.773</td>
<td>0.309</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>cpu_int</td>
<td>0.014</td>
<td>0.028</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>ALL_CLASS</td>
<td>0.787</td>
<td>0.337</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 4.12 - QUEST ready for iteration**

77
<table>
<thead>
<tr>
<th></th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>type</td>
<td>set.1</td>
<td>serv.1</td>
<td>serv.1</td>
<td>serv.1</td>
</tr>
<tr>
<td>1</td>
<td>init</td>
<td>fifo</td>
<td>ps</td>
<td>fifo</td>
<td>fifo</td>
</tr>
<tr>
<td>2</td>
<td>IO_int</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>prior</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>servt-1</td>
<td>18.000</td>
<td>exp</td>
<td>0.005</td>
<td>exp</td>
</tr>
<tr>
<td>5</td>
<td>servt-2</td>
<td>rate erv</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>servt-3</td>
<td>rate erv</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>servt-4</td>
<td>rate erv</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>servt-5</td>
<td>rate erv</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>servt-6</td>
<td>rate erv</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>servt-7</td>
<td>rate erv</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>servt-8</td>
<td>rate erv</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>servt-9</td>
<td>rate erv</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>servt-10</td>
<td>rate erv</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>servt-11</td>
<td>rate erv</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>servt-12</td>
<td>rate erv</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>servt-13</td>
<td>rate erv</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>servt-14</td>
<td>rate erv</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>servt-15</td>
<td>rate erv</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>servt-16</td>
<td>rate erv</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>servt-17</td>
<td>rate erv</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>servt-18</td>
<td>rate erv</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>servt-19</td>
<td>rate erv</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>servt-20</td>
<td>rate erv</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>servt-21</td>
<td>rate erv</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>servt-22</td>
<td>rate erv</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>servt-23</td>
<td>rate erv</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>servt-24</td>
<td>rate erv</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>servt-25</td>
<td>rate erv</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 4.13 - QUEST after iteration**

78
the probability for the next iteration. Then we close
the loop by placing in cell G2 the cell address F9.
Each time the spreadsheet is updated the probability in
cell G2 will increase by 0.05.

The stopping condition is constructed in cell F12
as ( C28 - D28 > 0 ? 1:0 ). This means that if the uti-
lization of disk1 (C28) - the utilization of disk2 (D28)
is greater than 0, cell F12 gets the value 1. Otherwise
it is 0. Each time we do an iteration the probability
of going to disk2 will increase by 0.05. If this proba-
bility increases, the utilization of disk2 increases
too, and the iteration eventually stops. We start the
iteration process by placing "@ite(F12,10)" in cell G11.
We can trace the execution step by step:

- when "@ite(F12,30)" is entered, the iteration
  begins at once. The spreadsheet is re-evaluated.
  That gives us these cell values:
  G1 = 0.70 , G2 = 0.20 , F9 = 0.20 , F12 = 1
  Then "@val(G2)" is evaluated: F10 = 0.20 .
- since the value in cell F12 is not 0, the model is
  evaluated by QNAP2 and results are placed back in
  the same cell location: C28 = 0.740  D28 = 0.443.
- the spreadsheet cells are then re-evaluated giving
  G1 = 0.65 , G2 = 0.25 , F9 = 0.25 , F12 = 1; then
  "@val(G2)" is updated to 0.25 .
- since F12 is still not zero, steps 2 and 3 are
repeated.

We have to remember in the version shown here to decrease the value in G2 by 0.05 to correspond to the final solution. Figure 4.13 shows the last evaluated spreadsheet. We see that the disks are used roughly equally for probabilities of 0.6 for going to disk1 and 0.3, for going to disk2.

The time taken by the iteration to complete is the sum of the time needed to solve the model at each iteration step. Generally if an analytical solution is applied, the time to solve is relatively short (around 30 sec. for the preceding example).

The stopping condition used to stop the iteration is programmable using any of the spreadsheet functions.

There are many possible uses of iteration. With a little ingenuity, optimization strategies can be programmed, and cost data can be employed to evaluate the solution.

4.7- Design issues in developing QUEST

QUEST is a very substantial software design effort (over 8000 lines of code and major modifications of the original spreadsheet VC). In this section, different implementation issues are discussed to help understand the difficulties of the work.
4.7.1- Cell data structure

The basic spreadsheet component is the cell. Therefore the way the cell structure is defined is a crucial design point for the spreadsheet efficiency.

The cell structure contains the following 7 elements:

value
label
expression
flag
row
column
pointer to next cell

Each of these elements is now described.

- the value elements is used to hold the value calculated or entered in the cell. This element is a double floating point number.

- the label is a pointer to a character string storing a text string entered in the cell. The string is created using dynamic memory allocation.

- the expression is used to hold the formula (eg: A2 + B2) entered in the cell. This element is a pointer to an expression structure.

- The flag element is used to keep track of the current cell state. Several bits in this element are used to flag different cell states (eg: the label is right justified, the cell is valid, the cell is hidden). This element is an integer type in the
- the row and column elements represent the address of the cell in the table. The two elements are integer type.
- the last element is used to link together all the defined cells. The element is a pointer to a cell structure.

The first 4 elements in the list do not require more discussion. But the last 3 elements need more explanation.

A variable call "tbl" (which is an array of cell structure pointers) is defined in the spreadsheet. For example "tbl[0][0]" points to the cell A0 in the spreadsheet. When the program is started, there is no cell existing in the spreadsheet. When the user enters a value, then a cell structure is created (dynamic memory allocation). The "column" and "row" variables are set to reflect the cell position in the sheet. The appropriate "tbl" variable is updated to point to the cell. Two cell pointers (first and last) are set to point to the new created cell.

When a second cell is needed, the program creates a new cell. The "column" and "row" variables are set to match the cell position. The "tbl" variable is updated to point to the cell. The "pointer to the next cell" variable in the first cell structure is updated to point to the new created cell. Finally the "last" variable is
updated to point to the second cell which is now the last cell in the linked list of cell.

Each time a new cell is created, the same process is repeated. This process creates a linked list of cell and each created cell can be accessed either by following the linked list or by using the "tbl" variable. The two methods of accessing a cell are used in the spreadsheet and they are important in the spreadsheet efficiency.

The linked list method is used mainly during a spreadsheet recalculation. Each time a cell is changed, the spreadsheet has to be recalculated. During this process, we want to scan only the defined cells not the cells that can be addressed by the "tbl" variable (many thousands). Therefore we use the linked list to go through all the defined cells and update the cells that are affected by the change.

The "tbl" method is used to access the cell using its the coordinates. Many operations in the spreadsheet need to access a particular cell, for example printing the screen, inserting a row, or copying a block. So each time one of these commands is used, we do not want to go through all the defined cells to find the one that is needed for the command. We just want to use the coordinates.
4.7.2- Workspace management

Many commands, described in section 4.3, need to move data from one location to another in the spreadsheet (e.g. adding a station means adding a column). The data management, required by many commands, has been a major problem encountered during the spreadsheet implementation.

When a model is being defined within QUEST, the spreadsheet can be divided in 3 sections (Figure 4.14).

The section A is where all the model parameters are entered. The section B is generally used to maintain a set of values used in the model parameters via cell linking. These values are used to accelerate the model modification (one value can be linked to many model parameters and therefore by changing on cell, many parameters in the model are updated). The section C is used to present the results.

To illustrate the data management problem, the "add station" command will be described step by step.

- the spreadsheet asks for a station name. The name is then validated.
- a line has to be inserted in the probability matrix present for each class in the model. This means moving the section C down by x lines where x is equal to the number of classes.
- a line is inserted in each probability matrix mov-
Figure 4.14 - The spreadsheet sections
ing the appropriate parameters. The lines are inserted only in the model area (section A). The section B is not altered.

- A column has to be inserted to hold the parameters specific to the new station. This means that section B has to be moved one column to the right. The section C is not altered by this column insertion.

- The required space for the new station has been created. The spreadsheet has to do one last scan of all the cells defined in the sheet to update the links that point to a cell that has been moved. For example, the section B has been moved one column to the right, therefore all the links to that region have to be updated to reflect the section movement.

The difficulty with the data management is to move the right data to the right place and to keep all the links in the sheet valid and this for many types of commands (add or delete a station, move or copy a selected block).

The data management is completely transparent to the user. When a station is added, the user has only to fill the column and rows that have been created automatically.
4.7.3- Modularity

The spreadsheet is comprised of many modules (Figure 4.3). Each of these modules has a specific task to accomplish. The spreadsheet modularity makes changes to the program functionality or capability easier. For example the "code generator" module can be replaced or a new module can be added to make possible the use of a new queuing network package solver.

4.7.4- Cell movement

All the commands that move cells from one location to another (insert a row) do not create new cell. Only pointer copying is performed to insure maximum rapidity.

There were many implementation difficulties (not discussed in this thesis) that were solved during the spreadsheet development. These include model parsing for code generation, file parsing to get results from QNAP2 within the spreadsheet, error reporting.

4.8- QUEST capabilities compare to PAW

The Performance Analysis Workstation (PAW) is a visual modeling tool with simulation capability developed at AT&T Bell Laboratories [ATT1986].

4.8.1- Requirements

PAW runs on a bit-mapped display 5620 terminal in
conjunction with a host computer running under the UNIX operating system. A UNIX host is required for file operations because the dot mapped display 5620 has no peripheral memory. The display has 1 MB memory and is equipped with a keyboard and a three button mouse as a pointing device.

QUEST runs on many types of text terminal or graphic workstations like SUN, under the UNIX operating system. It also requires the QNAP2 package to solve the queuing network model.

4.8.2. Model definition and modification

PAW and QUEST user interfaces are both menu driven. The style of interaction with the user is object oriented (an icon on the screen represents a real object) in PAW. In QUEST, the interaction is "area" oriented (each column represents a station, each set of rows represents a class).

PAW enables a user to draw a queuing network model on the screen with a mouse, to parameterize the model by filling out forms from the keyboard and then to make animated simulation runs displaying traffic flows and evolution statistics.

The forms filled by the user are hidden and can be recalled one at the time by selecting an icon. This makes it difficult for the user to have a global view of
the model.

The model modification (eg. add a class) implies scanning all the defined icons and adding the parameters for the modification in the form.

The model definition in QUEST is accomplished in 2 steps. First the user has to define the stations and classes required by the model. Second he has to fill the model cells with the appropriate parameter values.

The model modification in QUEST (eg. add a class) implies filling an area of cells created automatically by the spreadsheet.

4.8.3- Model types that can be defined

PAW and QUEST allow many types of model to be defined. General scheduling disciplines, service times, types of service center, etc. can be used. Multiple class models and class changes can be entered.

PAW uses special node types to support class change, customer generation, shared resource. Whereas QUEST uses cell parameters to accomplish the same functionality.

4.8.4- Solution techniques and results

PAW uses only a Monte Carlo simulator to calculate the results. From the simulation, PAW generates different statistics on the model (server utilization,
throughput, response time, ...).

QUEST gives access to the solver library contained in QNAP2. This library is comprised of analytical solver (MVA, convolution, ...), approximate solver (approximate MVA, ...), and a simulator. QUEST presents the results calculated by QNAP2 (server utilization, throughput, response time, ...) directly within the spreadsheet.

4.8.5- Utilities

PAW and QUEST have all the necessary functions to store and recall queuing network models using disk files. PAW requires a special graphics printer to generate a hard copy of the model (screen graphic and forms). QUEST uses a much more common standard character printer to print the spreadsheet. The sheet is printed stripe by stripe and the stripes can be placed side by side to reconstruct the spreadsheet format.

QUEST provides the user with 3 important utilities that are not present in PAW. First, the spreadsheet cells can be linked together, to provide functionally linked model parameters. Second, a list of parameters can be entered in a cell which will generate a set of results for each list member (multiple run). Third, an iteration procedure has been implemented that permits to
solve iterative approximate models, thus greatly enlarging the applicability of the tool.

In summary, PAW represents an experiment directed toward graphics, but relatively impoverished in modeling power and solution power, and requiring a lot of special equipment, QUEST provides more features (except for the graphical definition and output) and uses a more standard execution environment.
5- Performance analysis using QUEST

This chapter shows how the spreadsheet tool QUEST can be used to conduct a substantial performance study of the System Engineering Department's network of SUN workstations in Carleton University. The study is divided into two major parts. The first part introduces a paper [LZCZ1986] which describes the approach to be used, and gives a general model structure. This part also determines the parameter values for the network. The second part uses the parameters derived in the first section to model within the spreadsheet, the departmental SUN network and to investigate its performance under load.

This chapter also describes a paper from Heidelberger and Trivedi [HT1982] which demonstrates how the iterative procedure can be applied.

5.1- The model structure used by Lazowska

The paper by Lazowska et al [LZCZ1986] studies the performance of single user workstations that access files remotely over a local area network. Three different systems are studied in the paper but only the one describing a SUN workstation system will be described here. The approach was to use the results of measurement experiments to define a queuing network performance model. This model is then used to check performance
under load and to evaluate design alternatives. In this thesis, the reference is used as a starting point for showing the capabilities of QUEST in describing and experimenting with a rather complex system. It is also used as a framework for modelling a closely related system at Carleton.

5.1.1- Model description

Figure 5.1 illustrates the model used in the reference. The model contains a workstation center, a local area network center, a file server center and a disk center.

The workstations are modeled as a single infinite server. The local area network and the disk are modeled as single servers (scheduling discipline is FIFO). The file server is modeled as a single server (scheduling discipline is processor sharing). One important assumption made by the authors is that Network contention should not be a performance problem for a 10 Mbits network and 100 active workstations in a software development environment. This assumption permits to model the network as a pure delay.

The model includes one customer for each client workstation, placed in the infinite server workstation center. Each customer cycles between its workstation center and the file server via the network, being served
Figure 5.1 - Layout of the model used in the reference

Figure 5.2 - The queuing network model
and encountering queuing delays due to competition from other customers.

The service demand is the average amount of service that one request requires at each service center. A request is defined here as the transfer of 4 Kbytes of data from the disk to the workstation and its associated processing at the various centers.

The associated processing has the following components:

- the service demand at the workstation, which is the "user mode" processing plus the processing required to transfer the data.

- the service demand at the network, which is the total transfer time of the multiple packets needed to carry out the transfer (4 Kbytes of data is transferred via the network using a standard packet size which could be different from 4 Kbytes).

- the service demands at the file server and the disk, which are the times required for those devices to perform the transfer.

The above service demands were measured (in the paper) in a series of controlled experiments. A key decision was whether to consider sequential or random access. The disk service demand is very sensitive to these two modes because of additional seek costs of random access. Measurements are interpreted on the
assumption that a seek is required, on average, once per two requests.

5.1.2- The baseline case description

The baseline case consists of measurements to find the parameters needed to conduct the study and to illustrate the cost of remote versus local file access. For local access, we consider only the workstation CPU and the disk; for remote access, we consider the workstation CPU, the network, the server CPU and the disk.

Table I gives the measured service demands drawn from the paper.

If we assume (derived from the paper) a 3:1 ratio of reads to writes, 106 milliseconds of "user mode" processing time for 4 Kbytes I/O and 4 Kbytes I/O per second, then results from Table I can be used to find the needed parameters.

<table>
<thead>
<tr>
<th>READ</th>
<th>Local</th>
<th>Remote</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client CPU</td>
<td>14.6 ms</td>
<td>43.8 ms</td>
</tr>
<tr>
<td>Disk</td>
<td>18.8 ms</td>
<td>18.8 ms</td>
</tr>
<tr>
<td>Server CPU</td>
<td></td>
<td>41.0 ms</td>
</tr>
<tr>
<td>Network</td>
<td></td>
<td>3.6 ms</td>
</tr>
<tr>
<td>Total</td>
<td>33.3 ms</td>
<td>107.1 ms</td>
</tr>
<tr>
<td>Elapsed</td>
<td>28.6 ms</td>
<td>69.4 ms</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>WRITE</th>
<th>Local</th>
<th>Remote</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client CPU</td>
<td>20.0 ms</td>
<td>47.5 ms</td>
</tr>
<tr>
<td>Disk</td>
<td>18.8 ms</td>
<td>18.8 ms</td>
</tr>
<tr>
<td>Server CPU</td>
<td></td>
<td>36.8 ms</td>
</tr>
<tr>
<td>Network</td>
<td></td>
<td>3.6 ms</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>38.8 ms</td>
<td>106.6 ms</td>
</tr>
<tr>
<td><strong>Elapsed</strong></td>
<td>23.8 ms</td>
<td>47.0 ms</td>
</tr>
</tbody>
</table>

Table I: Measured service demands (milliseconds)

The parameters in Table I are not the parameters needed to run a model; some work is needed to derive the service time and routing parameters as follows.

Since the request is the basic system modelling unit, the time to perform a transfer of 4 Kbytes is used to derive the different service times. The "think" time plus the "user mode processing" time at a workstation center is given directly in the reference as being 1000 ms. The service time \( x_{fs} \) at a file server center is calculated from Table I and from the read/write ratio, ie:

\[
x_{fs} = \frac{(3 \text{ reads} \times 41 \text{ ms/read} + 1 \text{ write} \times 36.8 \text{ ms/write})}{4}
\]

\( x_{fs} = 39.95 \text{ ms} \).

The other service times are directly taken from table I. They are

\( x_{ws} = 1000.0 \text{ ms per request} \)
\( x_{lan} = 3.6 \text{ ms per request} \)
\( x_{fs} = 39.95 \text{ ms per request} \)
"request" class customer into a "response" class customer. Then it goes back to the workstation center using the same resources. On the workstation center it changes back to a "request" class customer.

The service times derived earlier will be used in section 5.3 to analyze the departmental SUN network.

In the next sections, two case studies, from the paper, will be entered in the spreadsheet tool and results from the spreadsheet and from the paper will be compared. The purpose is to verify that the model in the paper has been properly understood and entered in QUEST, and check if some important details of their calculations were perhaps not fully reported in the paper.

5.1.3- The baseline case study

In this section, the baseline case model, defined in the previous section, is entered in the spreadsheet (Figure 5.3) and the effect of congestion is investigated. Also the results from the spreadsheet (Figure 5.3) and from the paper are compared to verify that our interpretation of the paper is correct.

From Figure 5.3, we remark that the model is defined between row 0 and row 21. The type of server, the scheduling discipline, the service time and the routing
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>type</td>
<td>ms 1</td>
<td>ms 1</td>
<td>ms 1</td>
<td>ms 1</td>
<td>ms 1</td>
</tr>
<tr>
<td>1</td>
<td>value</td>
<td>12</td>
<td>0.1</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>REQUEST</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>3</td>
<td>pipe</td>
<td>10.10.30.10</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>4</td>
<td>init</td>
<td>1.000</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>1.000</td>
</tr>
<tr>
<td>5</td>
<td>servt-1</td>
<td>200.000</td>
<td>3.000</td>
<td>5.000</td>
<td>3.000</td>
<td>6.000</td>
</tr>
<tr>
<td>6</td>
<td>servt-2</td>
<td>200.000</td>
<td>3.000</td>
<td>5.000</td>
<td>3.000</td>
<td>6.000</td>
</tr>
<tr>
<td>7</td>
<td>case_1</td>
<td>200.000</td>
<td>3.000</td>
<td>5.000</td>
<td>3.000</td>
<td>6.000</td>
</tr>
<tr>
<td>8</td>
<td>init</td>
<td>1.000</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>1.000</td>
</tr>
<tr>
<td>9</td>
<td>servt-1</td>
<td>200.000</td>
<td>3.000</td>
<td>5.000</td>
<td>3.000</td>
<td>6.000</td>
</tr>
<tr>
<td>10</td>
<td>servt-2</td>
<td>200.000</td>
<td>3.000</td>
<td>5.000</td>
<td>3.000</td>
<td>6.000</td>
</tr>
<tr>
<td>11</td>
<td>case_1</td>
<td>200.000</td>
<td>3.000</td>
<td>5.000</td>
<td>3.000</td>
<td>6.000</td>
</tr>
<tr>
<td>12</td>
<td>init</td>
<td>1.000</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>1.000</td>
</tr>
<tr>
<td>13</td>
<td>servt-1</td>
<td>200.000</td>
<td>3.000</td>
<td>5.000</td>
<td>3.000</td>
<td>6.000</td>
</tr>
<tr>
<td>14</td>
<td>servt-2</td>
<td>200.000</td>
<td>3.000</td>
<td>5.000</td>
<td>3.000</td>
<td>6.000</td>
</tr>
</tbody>
</table>

**Figure 5.3 - The baseline case model**
parameters are entered using the syntax rules defined in chapter 4. We remark that cell A4 gives a list of parameters which produces a list of solutions (multiple runs). It is important to note the class changes in cells C11 and A19.

Unlike a dedicated disk on a single user machine, the file server and the local area network are shared devices and accessing them is subject to queuing delays. The effect of these delays is to increase the remote file access time. We remark from Figure 5.4 that queuing delays can be significant (40 % of the total response time), even for 20 workstations. The queuing delays are due to congestion at shared resources.

Figure 5.5 graphs the utilization of different devices as the number of customers increases. We remark that at heavy load (large number of workstations), performance is governed by the most heavily utilized device (the file server) which limits throughputs and increases the response time. It is also clear, from that Figure, that the assumption made by the authors (that the network is not a performance problem) is valid because its utilization is very small (less than 10%).

Figure 5.6 is drawn from the paper and presents the same results as Figure 5.5 which was derived from the spreadsheet (the curves from Figure 5.5 has been added to the Figure 5.6 to facilitate the comparison). When
Figure 5.4 - Baseline model response time graph

Figure 5.5 - Baseline model utilization graph
Figure 5.6 - Baseline case utilization (paper)
the two Figures are compared, one can see that the curves obtained from the spreadsheet match very well the others obtained from the paper. This gives confidence that the parameters used in QUEST have been derived from the paper correctly.

5.1.4- Effect of SUN workstation enhancements

Since the paper was published, many improvements have been made to the SUN workstations and to the way files are transferred over the network. Here we will consider some new parameter values not considered in the reference.

The machines considered in the paper are built around a 68010 microprocessor. The newer model (as in the department network) uses a 68020 and a faster clock rate. A new more efficient mechanism, the network file system (NFS), has been introduced for transferring files.

Those improvements have the effect of reducing the service demand at the file server (the bottleneck). Since there are no measurements available on the department network that can be used to derive the queuing network parameters, in this thesis we will model the workstation as if they were the same type as the ones described in the reference. However an improvement factor could be used in further studies to model the system
more accurately.

The system will be defined in the same way the paper model has been defined earlier and the parameters derived from the reference in section 5.1.2 will be used. It is important to note that the goal beyond the investigation of the departmental SUN network is to demonstrate the different features of the spreadsheet tool and how the tool can be used in a modeling process.

5.2- The departmental SUN network model

This section first describes the departmental SUN network, and then uses the description to build a queuing network model that will be used to investigate the system limitations.

The study uses the parameters determined in the previous section and some new parameters are derived to complete the model.

5.2.1- The SUN network description

The network (Figure 5.7) as of the fall of 1989 is composed of 3 file servers (Sirius, Polaris and Antares) and each of these servers handles requests from a group of workstations. Most of the time, a workstation requests file services from its own assigned file server but occasionally it accesses another group’s file server.
Probabilities are given within QUEST in Figure 5.8

Figure 5.7 - The departmental SUN network
The first file server, Sirius (FS1), manages two disks (DISK1 & DISK2) and responds to requests from the following workstations:

<table>
<thead>
<tr>
<th>workstation</th>
<th>CPU</th>
<th>clock rate</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jupiter</td>
<td>68020</td>
<td>16.67 MHz</td>
<td>SUN-3/75</td>
</tr>
<tr>
<td>Alderon</td>
<td>68020</td>
<td>16.57 MHz</td>
<td>SUN-3/140</td>
</tr>
<tr>
<td>Krypton</td>
<td>68020</td>
<td>16.57 MHz</td>
<td>SUN-3/140</td>
</tr>
<tr>
<td>Saturn</td>
<td>68020</td>
<td>15.00 MHz</td>
<td>SUN-3/50</td>
</tr>
<tr>
<td>Uranus</td>
<td>68020</td>
<td>15.00 MHz</td>
<td>SUN-3/50</td>
</tr>
<tr>
<td>Neptune</td>
<td>68020</td>
<td>15.00 MHz</td>
<td>SUN-3/50</td>
</tr>
<tr>
<td>Terminus</td>
<td>68020</td>
<td>15.00 MHz</td>
<td>SUN-3/50</td>
</tr>
<tr>
<td>Dune</td>
<td>68020</td>
<td>15.00 MHz</td>
<td>SUN-3/50</td>
</tr>
<tr>
<td>Arcturus</td>
<td>68020</td>
<td>15.00 MHz</td>
<td>SUN-3/50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>workstation</th>
<th>CPU</th>
<th>clock rate</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Talos</td>
<td>68020</td>
<td>15.00 MHz</td>
<td>SUN-3/50</td>
</tr>
<tr>
<td>Metron</td>
<td>68020</td>
<td>15.00 MHz</td>
<td>SUN-3/50</td>
</tr>
<tr>
<td>Genesis</td>
<td>68020</td>
<td>20.00 MHz</td>
<td>SUN-3/60</td>
</tr>
<tr>
<td>Haven</td>
<td>68020</td>
<td>20.00 MHz</td>
<td>SUN-3/60</td>
</tr>
</tbody>
</table>

Each of these workstations is modeled as a customer in the workstation center #1 (WS1).

The second file server, Polaris (FS2), manages one disk (DISK3) and responds to requests from the following workstations:

<table>
<thead>
<tr>
<th>workstation</th>
<th>CPU</th>
<th>clock rate</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pyramid</td>
<td>68020</td>
<td>15.00 MHz</td>
<td>SUN-3/50</td>
</tr>
<tr>
<td>Vulcan</td>
<td>68020</td>
<td>16.67 MHz</td>
<td>SUN-3/140</td>
</tr>
<tr>
<td>Pluto</td>
<td>68020</td>
<td>16.67 MHz</td>
<td>SUN-3/140</td>
</tr>
<tr>
<td>Hoth</td>
<td>68020</td>
<td>20.00 MHz</td>
<td>SUN-3/60</td>
</tr>
<tr>
<td>Hendor</td>
<td>68020</td>
<td>20.00 MHz</td>
<td>SUN-3/60</td>
</tr>
<tr>
<td>Tantalus</td>
<td>68020</td>
<td>20.00 MHz</td>
<td>SUN-3/160</td>
</tr>
<tr>
<td>Vendikar</td>
<td>68020</td>
<td>16.67 MHz</td>
<td>SUN-3/75</td>
</tr>
<tr>
<td>Mercury</td>
<td>68030</td>
<td>20.00 MHz</td>
<td>SUN-3/80</td>
</tr>
<tr>
<td>Venus</td>
<td>68030</td>
<td>20.00 MHz</td>
<td>SUN-3/80</td>
</tr>
</tbody>
</table>
Each of these workstations is modeled as a customer in the workstation center #2 (WS2).

The third file server, Antares (FS3), manages one disk (DISK4) and handles requests from the following workstations:

<table>
<thead>
<tr>
<th>workstation</th>
<th>CPU</th>
<th>clock rate</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tatooine</td>
<td>68020</td>
<td>20.00 MHz</td>
<td>SUN-3/60</td>
</tr>
<tr>
<td>Homefree</td>
<td>68020</td>
<td>20.00 MHz</td>
<td>SUN-3/60</td>
</tr>
<tr>
<td>Trantor</td>
<td>68020</td>
<td>20.00 MHz</td>
<td>SUN-3/60</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>workstation</th>
<th>CPU</th>
<th>clock rate</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organia</td>
<td>68020</td>
<td>20.00 MHz</td>
<td>SUN-3/60</td>
</tr>
<tr>
<td>Caladan</td>
<td>68020</td>
<td>15.00 MHz</td>
<td>SUN-3/50</td>
</tr>
<tr>
<td>Delphi</td>
<td>68020</td>
<td>15.00 MHz</td>
<td>SUN-3/50</td>
</tr>
<tr>
<td>Gaia</td>
<td>68020</td>
<td>20.00 MHz</td>
<td>SUN-3/60</td>
</tr>
<tr>
<td>Dagobah</td>
<td>68020</td>
<td>16.67 MHz</td>
<td>SUN-3/140</td>
</tr>
</tbody>
</table>

Each of these workstations, except Dagobah, is modeled as a customer in the workstation center #3 (WS3). Dagobah has its own disk (DISK5) and requests file services from both the network and its local disk.

The network printers are not included in the model nor the network link to the VAX-11/750. Those components are not essential to the study because they are not involved with the basic request description. They could easily be added however.

The basic 4 Kbytes transfer is still accomplished in four 1 Kbyte transfer and the routing parameters are defined in the same way that was used in section
5.1.2.

As before, two classes (request & reply) are used in the model. A total of 8 classes are used to route properly the customers from the four workstation centers.

The last thing to be determined before the model can be defined, is the probability for a workstation customer to access its own file server. There is no data available to the author that can be used to determine that probability. It is known that a workstation uses its own file server for:
- all user files.
- paging.
- loading programs.

A workstation uses another group’s file server to run programs that are not accessible by its own file server and to access other user files that are not available through its file server. Since most of the time a workstation uses its own file server, the probability for a work station to access its own file server was arbitrarily set to 95%. This leaves a 2.5% probability to access each of the two remaining file servers.

Dagobah represents a special case because it has its own local disk. The probability to access the local disk has been arbitrarily set to 20%.
Table II gives the parameters needed to model the system.

<table>
<thead>
<tr>
<th></th>
<th>WS1</th>
<th>WS2</th>
<th>WS3</th>
<th>WS4</th>
<th>LAN</th>
<th>FS1</th>
<th>FS2</th>
<th>FS3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nb of customer</td>
<td>14</td>
<td>10</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mean serv time</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>.45</td>
<td>5.2</td>
<td>5.2</td>
<td>5.2</td>
</tr>
<tr>
<td>nb of server</td>
<td>inf</td>
<td>inf</td>
<td>inf</td>
<td>inf</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Scheduling</td>
<td>PS</td>
<td>FS</td>
<td>PS</td>
<td>PS</td>
<td>FIFO</td>
<td>PS</td>
<td>PS</td>
<td>PS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>DISK1</th>
<th>DISK2</th>
<th>DISK3</th>
<th>DISK4</th>
<th>DISK5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nb of customer</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mean serv time</td>
<td>4.7</td>
<td>4.7</td>
<td>4.7</td>
<td>4.7</td>
<td>4.7</td>
</tr>
<tr>
<td>nb of server</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Scheduling</td>
<td>FIFO</td>
<td>FIFO</td>
<td>FIFO</td>
<td>FIFO</td>
<td>FIFO</td>
</tr>
</tbody>
</table>

Table II - Model parameters

The routing parameters are entered directly in the spreadsheet (Figure 5.8). All the parameters listed above can be found in that Figure. The spreadsheet uses some hidden links to make it easier to change the probability to access a particular file server. For example in row 0 column P, the probability P1 FS1 is the probability for a customer in WS1 to access the file server 1, and P3 FS1 is the probability for a customer in WS3 to access the file server 1. These probabilities are linked to the model parameters. If one of these probabilities is changed, the probabilities in the model are changed accordingly.

Many parameters have been entered automatically in the spreadsheet by the "Fill" command. All the zeros in the probability matrix and in the init parameters, all
the "fifo" scheduling and all the single server type parameters have been entered by the "Fill" command.

The service time parameters have been entered only once in row 5 then they have been copied (copy region command) for the other classes.

5.2.2-The baseline analysis of the model

In this section, the model in Figure 5.8 is solved and results are presented.

Figure 5.9 shows the results as given by the spreadsheet. These results can be treated by the spreadsheet or manually to get the utilization (Figure 5.10) and the system response time (Figure 5.11) for each workstation center.

Figure 5.10 shows clearly that no device is used at its full capacity. Only the file server FS1 has an utilization over 50%. The response time graphic demonstrates that queuing delays are a small portion of the overall response time and therefore congestion at common servers is not a problem.

As a last "loaded model" study, 25 customers (i.e. 25 workstations) are placed in each workstation center WS1, WS2 and WS3. The model is then solved for utilization and response time. Figures 5.12 and 5.13 graph the utilization and the response time respectively for the last model modification. One can remark that
<table>
<thead>
<tr>
<th>Case</th>
<th>ALP Class</th>
<th>LAW</th>
<th>FS1</th>
<th>FS2</th>
<th>FS3</th>
<th>D1801</th>
<th>D1802</th>
<th>D1862</th>
<th>D1803</th>
<th>D1804</th>
<th>D1805</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>ALL CLASS</td>
<td>0.090</td>
<td>0.204</td>
<td>0.120</td>
<td>0.215</td>
<td>0.243</td>
<td>0.270</td>
<td>0.253</td>
<td>0.270</td>
<td>0.270</td>
<td>0.270</td>
</tr>
<tr>
<td>Case 2</td>
<td>ALL CLASS</td>
<td>0.072</td>
<td>3.241</td>
<td>6.040</td>
<td>6.460</td>
<td>6.742</td>
<td>7.762</td>
<td>5.121</td>
<td>5.273</td>
<td>5.273</td>
<td>5.273</td>
</tr>
<tr>
<td>Case 3</td>
<td>ALL CLASS</td>
<td>0.060</td>
<td>0.216</td>
<td>0.120</td>
<td>0.270</td>
<td>0.301</td>
<td>0.351</td>
<td>0.220</td>
<td>0.346</td>
<td>0.346</td>
<td>0.346</td>
</tr>
<tr>
<td>Case 4</td>
<td>ALL CLASS</td>
<td>0.078</td>
<td>6.101</td>
<td>6.640</td>
<td>6.880</td>
<td>6.914</td>
<td>6.914</td>
<td>5.132</td>
<td>5.296</td>
<td>5.296</td>
<td>5.296</td>
</tr>
<tr>
<td>Case 5</td>
<td>ALL CLASS</td>
<td>0.079</td>
<td>0.204</td>
<td>0.120</td>
<td>0.270</td>
<td>0.291</td>
<td>0.291</td>
<td>0.102</td>
<td>0.119</td>
<td>0.119</td>
<td>0.119</td>
</tr>
<tr>
<td>Case 6</td>
<td>ALL CLASS</td>
<td>0.087</td>
<td>7.600</td>
<td>6.100</td>
<td>6.200</td>
<td>5.120</td>
<td>5.120</td>
<td>5.145</td>
<td>5.207</td>
<td>5.207</td>
<td>5.207</td>
</tr>
</tbody>
</table>

Figure 5.9 - Network solution in QUEST
Figure 5.10 - The device utilizations

Figure 5.11 - The system response time
the file servers are the most utilized devices (bottlenecks) and also that queuing delays are becoming important. The increase of queuing delays produces an important system performance degradation (50% of the response time is queuing delays).

This section shown how QUEST can be used to conduct a substantial performance study. Only a subset of QUEST features were used during the analysis:

- the "multiple run" lists were used when many solutions were needed to determine a graph.
- Commands like "block copy" or "fill" were used to save a lot of typing while building or transforming a model.
- the links were used to facilitate the investigation of the model. By linking many cells within the model to a particular cell, it is possible to change the probability routing by changing a unique cell.

5.3- Example of iteration for an approximation model

This section introduces an iterative model [HT1982] drawn from the literature. The model is used primarily to show how the QUEST iterative procedure can be apply to a specific example. Other QUEST features used in this section are cell linking, the "fill" command, the "block copy" command.

The reference considers computer performance models
Figure 5.12 - The loaded model utilization

Figure 5.13 - The loaded model response time
with certain types of parallel processing systems. A primary task circulates for ever in the system, and it can spawn a secondary tasks at some point during its execution. Except for queuing effects, the tasks execute independently of one another, until the secondary task terminates. The performance of the system is analyzed using queuing network models. However, because of the parallelism, the models do not have an analytical solution. An iterative solution method is developed.

5.3.1- Model description

We assume that the system under consideration consists of 5 active resources (CPU and disks). The workload consists of a set of statistically identical jobs where each job consists of a primary task and zero or more statistically identical secondary tasks. The secondary tasks are spawned by the primary task sometime during its execution. Because of concurrency within a job, the queuing network model of the system does not belong to the class of product-form networks.

We describe an iterative technique for solving a sequence of product-form queuing networks so that upon convergence, the solution of the product-form network closely approximates the solution to the system.

The queuing network model of the approximate sys-
tem has two chains, one closed and the other open. The closed chain models the behavior of primary tasks and hence the population of the closed chain is fixed and equal to \( n \). The open chain models the behavior of secondary tasks.

The standard central server model is pictured in Figure 5.14. The server 1 represents the CPU and the servers 2, 3, 4, 5 represent I/O devices. In the model, there are a fixed number, \( n \), of primary tasks. We assume that a primary task has a CPU processing service time of .01 second. Whenever a primary task finishes this processing, a secondary task is spawned with a probability of \( f \). If the secondary task is spawned, the primary task returns to the CPU for processing. The secondary task moves to the I/O devices with a probability \( p_{12} \) for device \( i \). The secondary task then returns to the CPU with a probability of \( p_2 \) or exits the system with probability \( 1 - p_2 \).

The model has the interpretation that a primary task can issue I/O requests which are handled by the secondary task and continue processing without waiting for those I/O requests to complete (overlapped I/O).

The approximation model (Figure 5.15) uses an extra node, node 0. When the primary task leaves the CPU it is routed to node 0 with a probability \( f \) and to the I/O devices with probability \( (1-f)p_{11} \). The throughput of
Figure 5.14 - The central server model

Figure 5.15 - The modified central server model
primary tasks at node 0 equals the rate at which secondary tasks are spawned. The secondary tasks are modeled by an open chain with all the arrivals routed to the I/O devices. The Poisson arrival rate of the open chain is set so that it equals the throughput of the closed chain of primary tasks at node 0. Note that, for any fixed arrival rate, this approximation model has a solution.

The model, as described, is entered in the spreadsheet (Figure 5.16).

We remark that the model has 6 service centers and one source used to generate the secondary tasks. The initial number of primary tasks is entered in cell H0. A link is used to set the model parameter C4 equal to the value entered in H0. The probability \( f \) for a primary task to spawn a secondary task is entered in cell H1. This cell is linked to many cells in the model. The linking permits to change the routing automatically when H1 is changed. The probability for a secondary task to leave the system is 1-\( p_2 \) where \( p_2 \) is entered in cell H2. The cell H2 is linked to the probability matrix which will change automatically when \( p_2 \) changes.

The cell I12 is used to save the source service time calculated from the previous iteration step. The cell H10 is used to transform the service time kept in I12 into a throughput value which can be compared with
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>type</td>
<td>chv</td>
<td>ser.1</td>
<td>ser.1</td>
<td>ser.1</td>
<td>ser.1</td>
<td>ser.1</td>
<td>n=10000</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>second</td>
<td>fifo</td>
<td>fifo</td>
<td>fifo</td>
<td>fifo</td>
<td>fifo</td>
<td>fifo</td>
<td>fifo</td>
<td>f=0.250</td>
</tr>
<tr>
<td>2</td>
<td>PRIMARY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>p2=0.000</td>
</tr>
<tr>
<td>3</td>
<td>prio</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>init</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>10.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>5</td>
<td>serv-1</td>
<td>0.033</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>6</td>
<td>serv-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>rate_serv</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>SOURCE</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>12.000</td>
<td>12.000</td>
<td>12.000</td>
<td>12.000</td>
<td>12.000</td>
</tr>
<tr>
<td>9</td>
<td>NET-0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>10</td>
<td>CPU</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>11</td>
<td>DIS-1</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>12</td>
<td>DIS-2</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>13</td>
<td>DIS-3</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>14</td>
<td>DIS-4</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>15</td>
<td>SECONDARY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>prio</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>init</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>18</td>
<td>serv-1</td>
<td>10000.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>19</td>
<td>serv-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>rate_serv</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>SOURCE</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>22</td>
<td>NET-0</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>23</td>
<td>CPU</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>24</td>
<td>DIS-1</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>25</td>
<td>DIS-2</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>26</td>
<td>DIS-3</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>27</td>
<td>DIS-4</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>28</td>
<td>case 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>receive</td>
<td>PRIMARY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>SOURCE</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>31</td>
<td>SECONDARY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>ALL-CLASS</td>
<td>0.089</td>
<td>0.010</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>33</td>
<td>accept</td>
<td>PRIMARY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>SOURCE</td>
<td>0.124</td>
<td>0.001</td>
<td>0.077</td>
<td>0.077</td>
<td>0.077</td>
<td>0.077</td>
<td>0.077</td>
<td>0.077</td>
</tr>
<tr>
<td>35</td>
<td>SECONDARY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>ALL-CLASS</td>
<td>1.000</td>
<td>0.762</td>
<td>0.001</td>
<td>0.762</td>
<td>0.762</td>
<td>0.762</td>
<td>0.762</td>
<td>0.762</td>
</tr>
<tr>
<td>37</td>
<td>receive</td>
<td>PRIMARY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>SOURCE</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>39</td>
<td>all-put</td>
<td>PRIMARY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>SOURCE</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>41</td>
<td>ALL-CLASS</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>

**Figure 5.16 - QUEST ready for iteration**

120
the new calculated throughput from the present iteration step. The comparison is accomplished in cell I10 (@fabs(h10-d39)<0.01?0:1). The first part of the equation @fabs(h10-d39) takes the absolute value of the difference between the calculated throughput from the previous iteration step (H10) and the calculated value from the present iteration step (D39). The second part <0.01?0:1 sets the cell value to zero if the absolute value is less than 0.01. This means that if the throughput difference between two iteration steps is less than 0.01 then the iteration is stopped by setting the stopping condition (I10) to zero.

The source service time (A5) is linked to the primary tasks throughput in station STAT_0 by the formula 1/D39. This link means that the primary task throughput in STAT_0 is equal to the throughput of secondary tasks generated by the source.

D39 is set to an initial value of 30. Then the iteration procedure is ready to begin. We start the iteration by entering in cell A8 the formula @ite(I10,10). The following steps are performed when the iteration is started:

- the spreadsheet updates all the "@val()" function.
- the model is solved and the result values are placed back in the spreadsheet.
- the spreadsheet reevaluates the cells. This updates
the source service time and makes the sheet ready for the next iteration.

- if the stopping condition (I10) is zero, the iteration is stopped. Else go to the first step.

Figure 5.17 shows the spreadsheet after the iteration. The number in cell I8 represents the number of iteration performed. We remark that the stopping condition (I10) is now zero. The 5 iterations needed to solve the model took only 60 seconds on a SUN workstation.

Note that the iteration procedure can be used in many situations (to optimize throughputs, to balance utilizations in a system, ...) and that the stopping condition can be programmed within the spreadsheet to meet very special conditions.
Figure 5.17 - QUEST after iteration

123
3. Conclusion

In this section, we compare QUEST and QNAP2 and we verify that the major goals of the project have been reached.

Three questions are stated for the comparison: what was retained from QNAP2, what was improved and what was lost.

6.1-What was retained from QNAP2:

We saw through chapters 4 and 5 that most of the QNAP2 features are available within QUEST including:

- all the station types (server, source, ...)
- all the scheduling disciplines (fifo, ps, ...)
- the capacity to use priority
- all the service time distributions (exp, hexp, ...);
- all the solvers (MVA, simulation, ...)
- all the results (Utilization, response time, ...)
- multiple run facility
- iteration procedure
- special QNAP2 report program can be automatically inserted in the program generated by QUEST
- special QNAP2 service time programs can be inserted automatically in the program generated by QUEST
6.2- What was improved:

We already mentioned that many new features and new ways of using QNAP2 features were introduced including:
- automatic code generation (no programming, no debugging of QNAP2 code)
- keeping the spreadsheet power (cell linking, block operations, ...)
- bottleneck analysis
- linear equation solver
- linear programming solver
- probability to visit ratio transformation
- visit ratio to probability transformation (QNAP2 needs the probability routing to solve a model. Now visit ratios can be specified and transformed into probabilities that can be used by QNAP2)
- menu driven commands to build a model.

6.3- What was lost

Going from a powerful language like the QNAP2 language to a more limited syntax like the spreadsheet cell, we can be assured that a part of the language versatility is lost.

The versatility permits the user to define very special models. For example a complicated iterative algorithm can set a different service time or scheduling discipline for the model based on the results obtained
from the previous iteration.

6.4- Major goals

Three major goals were set at the start of this work. The first goal was to provide queuing network performance analyst with a powerful interface which uses a 2 dimensional layout to describe stations and classes.

QUEST uses a 2 dimensional layout and gather together all the parameters relative to a class or to a station. It makes changes to a model quicker by providing many features like cell linking or block operations.

QUEST gives access to the majority of QNAP2 features. This provides the user with many solvers (MVA, simulation) and with many types of parameter definition (eg: source or semaphore stations) necessary to model a large number of systems. Chapter 5 gives 3 examples of systems that can be modeled within QUEST. Other types of models using shared resources or customers generation can be entered in QUEST as well.

Considering the facts that many features are provided to solve and define models and that the 2 dimensional layout gives a good model readability and makes modifications easier, we can say that the first goal has been reached.
The second goal was to reduce the debugging and the programming time involved in the model solution.

There is no programming required to solve a model in QUEST. So we have eliminated that part of the model solution from the user by automatically generating the QNAP2 code. Further more, the code is bug free and do not necessitate any debugging.

The time necessary to validate (debug) the model has been reduced because of the features like cell linking that makes modifications quicker. So considering that the programming and the code debugging has been eliminated and that the time required to modify a model has been reduced, we can say that this goal has been reached.

The third goal was to make the interface portable. To achieve this, we used during the coding the standard C functions and the standard curses library functions.

QUEST currently runs on SUN workstations (series 3, 4, and sparc) from the standard terminal or in sunview. It runs on VAX computers from different terminal types (eg: VT100, VT52). It runs remotely from a PC using a modem and a communication software that emulates terminals like VT100, VT52, ANSI, TN3270. QUEST runs on HP workstations from the standard terminal or from X windows. In the HP case, QNAP2 is located on a remote IBM mainframe. The files generated by QUEST are
transferred on the mainframe, the model is solved, and
the results are transferred back within QUEST. All this
is completely transparent to the user.

Considering that QUEST code uses standard C and
curses functions and that QUEST runs already in many
environments, we can say that this goal has been
reached.

6.5- Further work:

QUEST as it is now, is an excellent performance
analysis tool that is used to enter, modify and solve
queuing network models.

It would be very interesting to add some new fea-
tures that would make QUEST a more versatile tool. Those
features could include:

- a mouse interface
- a report generation module that could generate cus-
tom made reports from the spreadsheet cells.
- a graphic generator module that could use cells
  values to generate meaningful graphics.
- importing measurement data automatically from sys-
tem measurement tools (under investigation).
- reducing measurement data within the spreadsheet to
  get model parameters.
References


END
120491
FIN