The markovian solver of QNAP 2 and applications

D. Potier, M. Veran

INRIA

CENTRE DE ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.(3) 954 90 20

Rapports Techniques

N° 49

THE MARKOVIAN SOLVER OF QNAP2 AND APPLICATIONS

Dominique POTIER
Michel VÉRAN

Mars 1985

The Markovian Solver of QNAP2 and Applications

Dominique Potier and Michel Veran

INRIA

THE MARKOVIAN SOLVER OF QNAP2 AND APPLICATIONS

Dominique Potier

INRIA
Le Chesnay, France

Michel Véran

Bull Sems
Eychirolles, France

Résumé

L'analyse des performances d'architectures de systèmes comportant des mécanismes complexes tels que synchronisation, blocage, priorité, n'est pas réalisable simplement avec les résultats classiques de la théorie des réseaux de file d'attente. Les méthodes disponibles ne sont qu'approchées et, le plus souvent, la qualité des résultats est mal controlée. Pour ce type de problèmes l'approche Markovienne est potentiellement très puissante, mais pratiquement utilisable que si l'on dispose d'outils adaptés: il faut en effet que la génération du domaine d'état du modèle considéré et le calcul de l'état stationnaire soit réalisé de faàon automatique et efficace. L'analyseur markovien du logiciel de modélisation QNAP2 réalise ces fonctions pour une classe très large de modèles. Les conditions d'application et les principes de conception de cet analyseur sont présentés dans l'article. Plusieurs exemples illustrent son utilisation pour l'analyse de modèles complexes. Des extensions sont proposées en conclusion.

Abstract

Several important computer architectures features such as synchronization, blocking, priorities are difficult to analyse within the framework of product-form networks. Available methods are approximate and, in most cases, the quality of the results cannot be assessed. Markovian analysis has a strong potential for this class of problems, but its practical application requires that specific tools are available: state-space generation and steady-state computation must be performed in an automatic and efficient way. The Markovian solver of the modelling package QNAP2 performs these functions for a large class of models. The design principles and the application stipulations of this solver are presented in the paper. Several examples illustrate its application to the analysis of complex models. Extensions are discussed in the conclusions.

Contents

The Markovian Solver of QNAP2 and Applications

1/ INTRODUCTION

Product-form theorems provide a high-level analysis framework for solving queueing network models of computer architectures [Baskett 75, Lavenberg 83, Lazowska 84]. They summarize in a compact way general properties of these models, thereby allowing the analysis of a large class of systems without having to deal with the intricacies of their underlying stochastic processes. They also offer the theoretical basis for numerous developments in the area of approximate methods for analyzing non product-form networks [Chandy 78, Jacobson 83]. The practical application of the product-form theorems and their extensions is made possible through efficient computational algorithms embodied into flexible modelling packages with user oriented interfaces [Potier 1982, Potier 1985].

However, there are several important computer architectures features which are difficult to represent and analyse within this framework. Typical examples are priority mechanisms, process synchronizations and blocking phenomena. Several proposals have been made to take into account those aspects by various transformations of product-form queueing networks [Agrawal 83] which lead to heuristic approximate resolution methods. The main disadvantages of this approach are the difficulty to assess the accuracy of the results obtained (it may be very dependent on model parameters) and the level of expertise required to perform the adequate transformations.

In this situation it may be preferable to turn back to the underlying stochastic processes rather than to deal with more or less questionable approximations. The standard approach is then to consider the Markovian process associated with the model, build its complete state space and its transition matrix and solve numerically the matrix in order to obtain the steady-state solution of the model. It should be emphasized that a very large class of models can be considered within this framework: non-exponential distributions may be taken care of by the methods of stages and the features mentioned above (priority, synchronizations and blocking) preserve the Markovian nature of the stochastic processes. In practice, this approach has not been applied widely because of the lack of adequate and efficient tools. The practical use of Markovian analysis requires that the state space and the transition matrix are generated automatically from a high-level description of the model, and that powerfull numerical techniques are available to compute the steady-state solution.

The automatic generation of the state space was first discussed in [Wallace 72] and more recently in [Paul 85]. Stewart developed in 1976 a Markov chain analyser, MARCA [Stewart 76], and discussed in [Stewart 78] the numerical techniques. The work of Stewart served as a starting point for the development of the Markovian solver of the queueing network analysis package QNAP [Merle 78] which was afterwards extended into QNAP2 [Veran 84]. The growing interest in Markovian solvers concretized recently in several new proposals in the area of queueing network models [Mueller 85] and stochastic Petri nets [Natkin 80, Marsan 85, Dugan 84].

In this paper we describe the Markovian analyser of the modelling package QNAP2 and show how it can be used in a large variety of situations to perform the exact analysis of complex queueing systems including non-standard features. The

paper is organized as follows. In section 2 we give a brief overview of QNAP2. Section 3 is devoted to the presentation of the Markovian solver: state-space generation and steady-state computation. Various examples are presented in section 4. Extensions and conclusions are discussed in section 5.

2/ THE MODELLING PAKAGE QNAP2

QNAP2 is a portable modelling environment providing specific facilities for building, handling and solving queueing networks models. QNAP2 is comprized of:

- a collection of resolution algorithms, including discrete event simulation, exact and approximate mathematical methods,
- a common user interface for model description, analysis control and result presentation.

The highlights of QNAP2 are briefly reviewed below. For more details, see [Veran 84, Potier 84]

2.1/ Modelling framework

The modelling framework supported by QNAP2 is the representation of a system as a network of stations through which circulate customers (a station being defined as one or several servers serving a single input queue). The stations represent the physical or logical processing facilities of the system to be modelled, and the customers represent the processes being executed and competing for these facilities.

In QNAP2 this classical modelling framework is strictly enforced: a QNAP2 model includes only stations built according to the pattern defined above. All the extensions of generalized queueing networks are specified within the service description associated with the stations and not through specific nodes as it is the case in other modelling packages.

2.2/ The solvers

QNAP2 includes a large set of analysis techniques developed for standard or generalized queueing network models and widely validated: analytical solvers (convolution algorithms, mean value analysis and approximate methods for product-form networks and extensions), discrete event simulation with run length control facilities, and a Markovian solver. Within the set of analytical solvers of QNAP2 (exact or approximate) a dispatching algorithm selects the most appropriate solver in term of efficiency and numerical accuracy, according to the modelling features used in the model.

The three types of solvers are accessed from the same language interface in order to achieve a maximum independence between the description of a model and its analysis. In this way, the technical intricacies specific to each solver are hidden from the user. The activation of the solvers is done within the

algorithmic language level of QNAP2. This feature allows flexible combination of the different solvers for true hybrid and hierarchical modelling.

2.3/ The interface language

The QNAP2 language is a high level, object-oriented, interactive language. It includes two levels of specification:

- a command language consisting of a limited set of parametrized commands; these commands correspond to the main functionalities of QNAP2:
  - + declaration of the objects and variables manipulated in a model,
  - + specification of the service stations,
  - + analysis control and interactive dialogue specification.
- an algorithmic language, derived from PASCAL and SIMULA which allows:
  - + the extension and definition of object types,
  - + the specification of services involving complex mechanisms,
  - + the activation of the solvers,
  - + the definition of specific interfaces tailored to the user's need.

The following example illustrates some basic features of the two language levels of QNAP2. Other examples are given in section 5.

Example:

```
/DECLARE/ QUEUE a, b(10);                    & declaration of queues
          INTEGER nc, nq = 5;                & declaration of integer variables

/STATION/ NAME = a;                                    & associated queue
          SCHED = PS;                                  & scheduling discipline
          SERVICE = CST(0.5);                          & service specification
          RATE = 0.8;                                  & service rate
          INIT = nc;                                   & initial customers
          TRANSIT = b(1 STEP 1 UNTIL nq), (1 REPEAT nq);& transition rules

/STATION/ NAME = b(1 STEP 1 UNTIL nq);
          SERVICE = EXP(1.0);
          TRANSIT = a;
```

```
/CONTROL/ OPTION = NRESULT;                    & output option

/EXEC/ FOR nc:= 5 STEP 5 UNTIL 25 DO           & loop on number of customers
       BEGIN
       NETWORK(a, b(1 STEP 1 UNTIL nq));       & selection of the analysed network
       SOLVE;                                  & call to the analytical solvers
       PRINT("througput(",nc,") =", THROUGHPUT(a));
                                               & edition of throughput of queue a
       END;
throughput(        5.) =   1.495
throughput(       10.) =   1.597
throughput(       15.) =   1.600
throughput(       20.) =   1.600
throughput(       25.) =   1.600
```

3/ THE MARKOVIAN SOLVER

The Markovian solver transforms a model described with the language of QNAP2 into a first order Markovian process and then computes the steady-state solution of the process. The state variable considered takes into account the full configuration of each queue of the network: class of the customer occupying the i-th location of the queue and current stage of service for this customer.

The following operations are performed by the solver:

- verification that the assumptions are met and definition of the internal state representation;
- generation of all the states of the model and their transition probabilities;
- computation of the steady-state probabilities and derivation of the standard performance criteria.

We outline in the next sections the principles of these operations.

3.1/ Application stipulations

3.1.1/ Station types and scheduling disciplines

The model must be a closed network (with no source nor exit queue) with one or several classes of customers. Any combination of the following station types and scheduling disciplines is allowed:

- station types:

  + simple server,
  + multiple server,
  + infinite server,
  + simple queue (without server).

- scheduling disciplines:

  + FIFO or LIFO, with or without priorities among classes of customers, with or without preemption,
  + PS (processor sharing).

3.1.2/ Services

In QNAP2 the service performed by a server may be comprized of work demands and operations on objects (queues and customers) and variables. These demands and operations are specified by a statement of the algorithmic language associated with the parameter SERVICE of the command STATION. This statement may be a simple statement or a compound statement.

The work demands are expressed in work units (e.g. number of instructions to be executed, number of bytes being transferred). The probability distributions of these demands are described by means of appropriate procedures (e.g. CST, EXP) of the algorithmic language. The service rate of the considered station (as specified by the parameter RATE) determines the effective service time. If this rate is equal to 1 (1 work unit per time unit) then the service time is equal to the work demand expressed in service units (1 is the default value of the RATE parameter).

The Markovian solver allows service description containing one and only one demand procedure, followed or preceded by statements defining other operations. The following distributions of demands may be used:

- EXP(m): exponential (m = mean),
- HEXP(m,c2): hyper-exponential (m= mean, c2= squared coefficient of variation),
- ERLANG(m,k): Erlang (m = mean, and k=steps (c2=1/k)),
- CST (m): constant m, approximated by an ERLANG(m,5),
- COX (t): Coxian law with coefficients in table t.

The parameters (mean, variance,..) of the work demand distributions may depend on the instantaneous number of customers in any station (globally or for each class). In this case the parameters of the work demand should be defined in the service description before the call to the work demand procedure (the instantaneous number of customers may be accessed by the CUSTNB function).

Example:

```
/DECLARE/ QUEUE a;

/STATION/ NAME = a;
   SERVICE = IF CUSTNB (a) < 3
                THEN EXP (2)
                ELSE ERLANG (2,3);
```

If the number of customers in a is less than 3, the service is distributed according to an exponential distribution, otherwise it is distributed according to an Erlang distribution.

The Markovian solver allows the following operations on objects to be specified in a service description:

- move operations

  a move operation, specified by the procedure MOVE(q1, q2), moves the first customer of the simple queue q1 into the queue q2 (the procedure is ineffective if the queue q1 is empty);

- transit operation

  a transit operation, specified by the procedure TRANSIT(q [,c1]), forces the transition of the current customer to the queue q with the class c1.

Any operation on variables may appear in a service description.

3.1.3/ Transition rules

- static transition rules:

  the standard probabilistic transition mechanism applies in the normal way to specify transitions between couples (queue, class); this mechanism is specified by the parameter TRANSITION of the command station;

- dynamic transition rules:

  dynamic routing rules may be specified as shown in the previous section with the use of the procedures TRANSIT and MOVE; the execution of these procedures may be conditioned on the current state of the network.

  Example

```
/DECLARE/ QUEUE a, b ,c; CLASS x, y;

/STATION/ NAME = a;
   SERVICE(x) =
        BEGIN
        CST (1.5);
        IF CUSTNB(b, x) + CUSTNB(c, x) = 3
           THEN TRANSIT(a);
        END;

   TRANSIT(x) = b, 1, c, 1;
```

If the total number of customers of class x in b and c equals 3, the customer completing its service in a is sent back to a otherwise it is forwarded to b or c with equal probabilities.

- instantaneous transitions:

in many situations it is desirable to initiate instantaneous transitions once the network has reached a given state. A typical example is the handling of blocking situations: a customer blocked in a simple queue must be moved into a station as soon as the number of customers in this station is below a given threshold. Instantaneous transitions are specified by a specific statement associated with the parameter TEST of the command CONTROL. This statement may contain calls to the MOVE procedure causing the movement of a customer from a simple queue to any queue of the network. It is executed whenever a normal transition has ocurred, i.e. a service completion and a transition from one queue to another; if its execution results in a transition of a customer it is executed again, and so on until no instantaneous transition is possible.

Example

```
/DECLARE/ QUEUE a, b, w;

/STATION/ NAME = a;
   SERVICE = EXP (2);
   TRANSIT = w;

/STATION/ NAME = b;
   SERVICE = HEXP(4, 16);
   TRANSIT = a;

/CONTROL/ TEST = IF CUSTNB(b) < 3 THEN MOVE (w, b);
```

As long as station b contains 3 or more customers the customers coming from a wait in the simple queue w. As soon as a free place exists in b a customer of w is forwarded to b.

3.1.4/ Service rates

Service rates dependent on the state of the network may be specified explicitly in the service description as shown before. The general form of the RATE parameter may also be used to specified local dependencies.

3.2/ State-representation

Let NQ and NC be respectively the number of queues and the number of customers of the model studied. A state of the model is representated by a sequence of NS = NQ + NC status words defined as follows. The i-th status word contains information about the state of the i-th queue and associated station; the other NC status words contain information about the occupied locations of the queues of the model (there are as many occupied locations as customers). The last NC status words are ordered starting from the last occupied location of the first queue to the first occupied location of the last queue.

Each status word consists of four fields: STAGE, PREV, NEXT and CE or CLASS. For the status word of a queue, the fields contain the following information:

- STAGE: void;
- PREV: pointer to the status word describing the first occupied location of the queue;
- NEXT: pointer to the status word describing the last occupied location of the queue;
- NP: total number of customers in the queue.

For the status word of a given occupied location, these fields contain the following information:

- STAGE: if the location is the first of a queue STAGE contains the current stage of the customer in this location (i.e. the stage in the Coxian representation of the work demand distribution); otherwise STAGE contains the last performed stage (case of a preempted customer occupying this location);
- PREV: pointer to the status word describing the previous occupied location of the same queue, or number of the queue if the given location is the first one;
- NEXT: pointer to the status word describing the next occupied location of the same queue, or number of the queue if the given location is the last one;
- CLASS: class of the customer in this location.

This representation is simplified in the case of a network with only one class of customers. Then only the NQ status words of the queues are required.

In order to limit memory requirements, a numerical coding scheme is used which transforms the contents of a status word into an integer. Thus, each state of a queueing network with NQ queues and NC customers will require NQ memory words if the network is single class, NQ+NC words if the network is multi-class.

3.3/ State-space and transition matrix generation

The process by which the states and the transitions probabilities of a model are obtained is akin to running a discrete event simulation of this model where an event would be defined as the completion of an elementary exponential stage. Thus starting from the initial state of the network the behavior of the customers is simulated: customers execute in the servers and are moved from stations to stations according to the specified transition rules. Once a customer is allocated a server it executes the service description, i.e. the service description is dynamically evaluated as would be the case in standard simulation. A new state is recorded as soon as a customer has completed an exponential stage and cannot proceed further in the network (either because another stage remains to be done or because it is send in a queue and no server is available). If instantaneous transitions have been specified, they are applied repeatedly to this new state until no further transition occur. The obtained state is then compared to the list of existing states and added to the list if it is new one. In the course of this process the transition probabilities are obtained from the parameter of the corresponding exponential stage and from the routing probabilities (in the case where the state transition involves the probabilistic transition of a customer from one queue to another). A standard sparse-matrix compaction scheme is used to store the transition matrix.

In order to ensure that all possible states are considered the following rules are used. Given a current state, all the possible transitions from this state are first considered and analysed. The obtained new states are recorded. When this step is completed, the current state is marked and one of the newly obtained states becomes the current state. The generation process terminates when all the states in the list have been marked and no new state is obtained.

A trace facility is available which lists the complete description of all the states of the model, the transition probabilities and the steady-state probabilities.

3.4/ Numerical solution methods

The numerical solution method implemented in the Markovian solver of QNAP2 is based on the conclusions of a comparative study between several algorithms for Markovian systems [Cachard 81]. The following methods were considered: minimized iterations method [Arnoldi 51, Saad 80], simultaneous iterations method [Stewart 76], inverse iteration method and direct solution method. The method of Arnoldi was found to be the most efficient and robust method for the test cases considered, especially for large state-spaces (several thousand states). It is to be noted that the same algorithm is also used to compute the traffic intensities when the analytical solvers are activated.

4/ EXAMPLES

In this section we present several examples of queueing network models exhibiting various features. Two categories of queueing network models will be considered:

- extended product-form like closed queueing networks:

  this category includes closed queueing network models satisfying the basic product-form assumptions with any combination of the following features:

  + general service time distributions (Erlang, exponential, hyperexponential, Cox);

  + various scheduling disciplines (FIFO, LIFO, priority, preemption, PS);

- general closed queuing networks:

  this category includes closed networks with state dependent routing rules, non-local state dependent service rates, limited queue length...

4.1/ Extended product-form networks

Throughout this section we will use as illustration a simple queueing network consisting of two stations st1 and st2, and three classes c11, c12 and c13. Station st1 is IS (Infinite Server) with mean service times equal to 1.0 for all classes and will remain so in all the examples. Different configurations of station st2 will be considered (service time distributions, scheduling policies, number of server, dependent service rates). The mean service times of customers in class c11 (respectively c12 and c13) is 0.5 (respectively 1.0 and 1.5). Each class contains n customers. The basic QNAP2 description of this model is as follows:

```
/DECLARE/ QUEUE st1, st2;
          CLASS c11, c12, c13;
          INTEGER n = 2;

/STATION/ NAME = st1;
          TYPE = INFINITE;
          SERVICE = EXP(1.0);
          TRANSIT = st2;
          INIT = n;

/STATION/ NAME = st2;
          SERVICE(c11) = EXP(0.5);
          SERVICE(c12) = EXP(1.0);
          SERVICE(c13) = EXP(1.5);
          TRANSIT = st1;

/CONTROL/ CLASS = st2;
```

The FIFO scheduling and single server assumptions are implicit in the definition of station st2. The analysis yields:

```
/EXEC/ MARKOV;



***  MARKOVIAN ANALYSIS ***


TOTAL NUMBER OF STATES =       271
NUMBER OF NON-ZERO ELEMENTS IN MATRIX =       597

     9 ITERATIONS
**************************************************************************
*  NAME    *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THRUPUT *
**************************************************************************
*          *          *          *          *          *          *
* st1      * 1.000    *0.0000E+00* 1.009    * 1.000    * 1.009    *
*          *          *          *          *          *          *
* st2      * .9905    * .9993    * 4.991    * 4.947    * 1.009    *
*(c11      )* .5000    * .1733    * 1.653    * 4.770    * .3466    *
*(c12      )* 1.000    * .3348    * 1.665    * 4.973    * .3348    *
*(c13      )* 1.500    * .4911    * 1.673    * 5.109    * .3274    *
*          *          *          *          *          *          *
**************************************************************************
      MEMORY USED:      64938 WORDS OF 4 BYTES
       ( 24.98  % OF TOTAL MEMORY)
```

We now modify the distributions assumptions by setting:

```
/STATION/ NAME = st2;
          SERVICE(cl1) = ERLANG(0.5, 2);
          SERVICE(cl2) = EXP(1.0);
          SERVICE(cl3) = HEXP(1.5, 3.0);
/EXEC/ MARKOV;
```

```
***  MARKOVIAN ANALYSIS ***


TOTAL NUMBER OF STATES =        451
NUMBER OF NON-ZERO ELEMENTS IN MATRIX =       1081

    13 ITERATIONS
*************************************************************************
*  NAME    *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THROUPUT *
*************************************************************************
*          *          *          *          *          *          *
* st1      * 1.000    *0.0000E+00* 1.005    * 1.000    * 1.005    *
*          *          *          *          *          *          *
* st2      * .9945    * .9991    * 4.995    * 4.973    * 1.005    *
*(cl1      )* .5000    * .1710    * 1.658    * 4.848    * .3420    *
*(cl2      )* 1.000    * .3315    * 1.668    * 5.033    * .3315    *
*(cl3      )* 1.500    * .4966    * 1.669    * 5.041    * .3310    *
*          *          *          *          *          *          *
*************************************************************************
      MEMORY USED:       71335 WORDS OF 4 BYTES
       ( 27.44  % OF TOTAL MEMORY)
```

The increased number of states is due to the Erlang and hyperexponential distributions. For each non-exponential class an additional state is considered at station st2 representing the current stage of the service in the Coxian formulation of the non-exponential distribution.

In the next analysis PS scheduling is assumed at station st2:

```
/STATION/ NAME = st2;
          SCHED = PS;

/EXEC/ MARKOV;


***  MARKOVIAN ANALYSIS ***


TOTAL NUMBER OF STATES =       147
NUMBER OF NON-ZERO ELEMENTS IN MATRIX =       987

    13 ITERATIONS
*************************************************************************
*  NAME    *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THRUPUT *
*************************************************************************
*          *          *          *          *          *          *
* st1      * 1.000    *0.0000E+00* 1.162    * 1.000    * 1.162    *
*          *          *          *          *          *          *
* st2      * .8597    * .9993    * 4.838    * 4.162    * 1.162    *
*(c11     )* .5000    * .2863    * 1.427    * 2.493    * .5726    *
*(c12     )* 1.000    * .3435    * 1.657    * 4.823    * .3435    *
*(c13     )* 1.500    * .3695    * 1.754    * 7.119    * .2463    *
*          *          *          *          *          *          *
*************************************************************************
      MEMORY USED:      62925 WORDS OF 4 BYTES
       ( 24.20  % OF TOTAL MEMORY)


Note also that identical results are obtained using the solver SOLVE:


/EXEC/ SOLVE;


- MEAN VALUE ANALYSIS ("MVA") -
*************************************************************************
*  NAME    *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THRUPUT *
*************************************************************************
*          *          *          *          *          *          *
* st1      * 1.000    *0.0000E+00* 1.162    * 1.000    * 1.162    *
*          *          *          *          *          *          *
* st2      * .8597    * .9993    * 4.838    * 4.162    * 1.162    *
*(c11     )* .5000    * .2863    * 1.427    * 2.493    * .5726    *
*(c12     )* 1.000    * .3435    * 1.657    * 4.823    * .3435    *
*(c13     )* 1.500    * .3695    * 1.754    * 7.119    * .2463    *
*          *          *          *          *          *          *
*************************************************************************
      MEMORY USED:       3483 WORDS OF 4 BYTES
       (  1.34  % OF TOTAL MEMORY)
```

Consider now the LIFO, preemptive-resume policy at station st2. As predicted by product-form theorems, results identical to the PS configuration are obtained:

```
/STATION/ NAME = st2;
          SCHED = LIFO, PREEMPT;

/EXEC/ MARKOV;


***  MARKOVIAN ANALYSIS ***


TOTAL NUMBER OF STATES =       2923
NUMBER OF NON-ZERO ELEMENTS IN MATRIX =       5612

    70 ITERATIONS
**************************************************************************
*  NAME    *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THROUPUT *
**************************************************************************
*          *          *          *          *          *          *
* st1      * 1.000    *0.0000E+00* 1.162    * 1.000    * 1.162    *
*          *          *          *          *          *          *
* st2      * .8602    * .9992    * 4.838    * 4.165    * 1.162    *
*(c11      )* .5000    * .2858    * 1.428    * 2.500    * .5715    *
*(c12      )* 1.000    * .3436    * 1.656    * 4.821    * .3436    *
*(c13      )* 1.500    * .3699    * 1.753    * 7.110    * .2466    *
*          *          *          *          *          *          *
**************************************************************************
      MEMORY USED:     151909 WORDS OF 4 BYTES
       ( 58.43  % OF TOTAL MEMORY)
```

We can as well analyse the performance of the non preemptive LIFO policy:

```
/STATION/ NAME = st2;
          SCHED = LIFO;

/EXEC/ MARKOV;


***  MARKOVIAN ANALYSIS ***


TOTAL NUMBER OF STATES =       451
NUMBER OF NON-ZERO ELEMENTS IN MATRIX =      1081

    40 ITERATIONS
*************************************************************************
*  NAME    *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THRUPUT *
*************************************************************************
*          *          *          *          *          *          *
* st1      * 1.000    *0.0000E+00* .9733    * 1.000    * .9733    *
*          *          *          *          *          *          *
* st2      * 1.026    * .9991    * 5.027    * 5.164    * .9733    *
*(c11     )* .5000    * .1452    * 1.710    * 5.889    * .2903    *
*(c12     )* 1.000    * .3412    * 1.659    * 4.861    * .3412    *
*(c13     )* 1.500    * .5127    * 1.658    * 4.851    * .3418    *
*          *          *          *          *          *          *
*************************************************************************
      MEMORY USED:      71651 WORDS OF 4 BYTES
       ( 27.56  % OF TOTAL MEMORY)
```

In the same fashion, preemptive and non-preemptive priority scheduling disciplines can be analysed:

```
/STATION/ NAME = st2;
          SCHED = PRIOR;
          PRIOR(c11) = 1;
          PRIOR(c12) = 2;
          PRIOR(c13) = 3;
```

```
/EXEC/ MARKOV;


***  MARKOVIAN ANALYSIS ***


TOTAL NUMBER OF STATES =         91
NUMBER OF NON-ZERO ELEMENTS IN MATRIX =        339

     2 ITERATIONS
*************************************************************************
*  NAME     *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THRUPUT *
*************************************************************************
*           *          *          *          *          *          *
* st1       * 1.000    *0.0000E+00* .7807    * 1.000    * .7807    *
*           *          *          *          *          *          *
* st2       * 1.280    * .9993    * 5.219    * 6.685    * .7807    *
*(c11       )* .5000    *0.2346E-01* 1.953    * 41.63    *0.4692E-01*
*(c12       )* 1.000    * .2498    * 1.750    * 7.007    * .2498    *
*(c13       )* 1.500    * .7261    * 1.516    * 3.132    * .4840    *
*           *          *          *          *          *          *
*************************************************************************
      MEMORY USED:       59812 WORDS OF 4 BYTES
       ( 23.00  % OF TOTAL MEMORY)
```

```
/STATION/ NAME = st2;
          SCHED = PRIOR, PREEMPT;

/EXEC/ MARKOV;


***  MARKOVIAN ANALYSIS ***


TOTAL NUMBER OF STATES =         75
NUMBER OF NON-ZERO ELEMENTS IN MATRIX =        299

    34 ITERATIONS
**************************************************************************
*  NAME    *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THRUPUT *
**************************************************************************
*          *          *          *          *          *          *
* st1      * 1.000    *0.0000E+00* .7230    * 1.000    * .7230    *
*          *          *          *          *          *          *
* st2      * 1.382    * .9991    * 5.277    * 7.299    * .7230    *
*(cl1     )* .5000    *0.1007E-01* 1.980    * 98.34    *0.2013E-01*
*(cl2     )* 1.000    * .1303    * 1.870    * 14.34    * .1303    *
*(cl3     )* 1.500    * .8587    * 1.428    * 2.494    * .5725    *
*          *          *          *          *          *          *
**************************************************************************
      MEMORY USED:      59338 WORDS OF 4 BYTES
       ( 22.82  % OF TOTAL MEMORY)
```

4.2/ General queueing networks

For general queueing networks involving non-standard mechanisms, the specification of these mechanisms will be done at the algorithmic language level within the service descriptions. Explicit customer transitions can be defined at this level using the procedures MOVE and TRANSIT. Also the parameters of these procedures and of the work demand procedures can be defined at the same level before calling the procedures.

Instantaneous transitions, i.e. transitions without delay associated with some states of the model are specified in a specific parameter of the command /CONTROL/, the parameter TEST. The value of TEST is a statement of the algorithmic language which is executed whenever a new state of the model is reached. In this statement the movement of a customer from one queue to another can be specified by the procedure MOVE.

These elements are illustrated by various example in the next sections.

4.2.1/ General routing rules

General routing rules can be specified using the explicit transition procedure TRANSIT and analysed by the Markovian solver. As an example we consider a network consisting of three stations st1, st2 and st3. The station st1 is IS and customers leaving st1 go to the shortest queue (st2 or st3). If the queue length are equal, then the customers are routed at random to st2 or st3. For sake of simplicity, exponential services are assumed:

```
/DECLARE/ QUEUE st1, st2, st3; INTEGER n = 10;

/STATION/ NAME = st1;
          TYPE = INFINITE;
          SERVICE = BEGIN
                    EXP(1.0);
                    IF CUSTNB(st2) > CUSTNB(st3) THEN TRANSIT(st3);
                    IF CUSTNB(st3) > CUSTNB(st2) THEN TRANSIT(st2);
                    END;
          TRANSIT = st2, 1, st3, 1;
          INIT = n;
```

The predefined probabilistic routing mechanism will be used whenever the explicit transition procedure TRANSIT has not been activated (i.e. in case of equal numbers of customers in st2 and st3).

Note that the conditions defining the explicit transitions could include any expression involving the numbers of customers in the stations of the network (and numbers of customers for each class, if several classes have been defined)

```
/STATION/ NAME = st2, st3;
          SERVICE = EXP(0.5);
          TRANSIT = st1;
```

```
/EXEC/ MARKOV;


***  MARKOVIAN ANALYSIS ***


TOTAL NUMBER OF STATES =         36
NUMBER OF NON-ZERO ELEMENTS IN MATRIX =        270

     2 ITERATIONS
**************************************************************************
*  NAME    *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THRUPUT *
**************************************************************************
*          *          *          *          *          *          *
* st1      * 1.000    *0.0000E+00* 3.920    * 1.000    * 3.920    *
*          *          *          *          *          *          *
* st2      * .5000    * .9801    * 3.040    * 1.551    * 1.960    *
*          *          *          *          *          *          *
* st3      * .5000    * .9801    * 3.040    * 1.551    * 1.960    *
*          *          *          *          *          *          *
**************************************************************************
      MEMORY USED:       76446 WORDS OF 4 BYTES
       ( 29.40  % OF TOTAL MEMORY)
```

Symetrical results are obtained since the two stations st2 and st3 are identical.

These results can be compared to those obtained with a probabilistic routing from station st1:

```
/STATION/ NAME = st1; SERVICE = EXP(1.0);

/EXEC/ MARKOV;


***  MARKOVIAN ANALYSIS ***


TOTAL NUMBER OF STATES =         66
NUMBER OF NON-ZERO ELEMENTS IN MATRIX =        550

     4 ITERATIONS
**************************************************************************
*  NAME    *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THRUPUT *
**************************************************************************
*          *          *          *          *          *          *
* st1      * 1.000    *0.0000E+00* 3.430    * 1.000    * 3.430    *
*          *          *          *          *          *          *
* st2      * .5000    * .8576    * 3.285    * 1.915    * 1.715    *
*          *          *          *          *          *          *
* st3      * .5000    * .8576    * 3.285    * 1.915    * 1.715    *
*          *          *          *          *          *          *
**************************************************************************
      MEMORY USED:      78089 WORDS OF 4 BYTES
       ( 30.03  % OF TOTAL MEMORY)
```

In this case the solver SOLVE could be applied as well.

4.2.2/ General state dependent service rates

General state dependent service rates are specified in the service description of the station. It is enough in the service description to give the explicit expressions of the parameters of the work demand procedure and then call this procedure.

As an example, we consider a model consisting of an IS station, st0, and of three FIFO exponential stations, st1, st2 and st3. The service rate of st1 depends on the total number of customers in the three stations st1, st2 and st3:

```
/DECLARE/ QUEUE st0, st1, st2, st3;
          INTEGER npop, n = 10;

          PROCEDURE pop;
              BEGIN
              npop:=  CUSTNB(st1)+ CUSTNB(st2)+ CUSTNB(st3);
              END;
```

```
/STATION/ NAME = st0; TYPE = INFINITE; SERVICE = EXP(1.0); TRANSIT = st1;

/STATION/ NAME = st1;
          SERVICE = BEGIN
                    pop;
                    EXP(npop);
                    END;
          TRANSIT = st2, 1, st3, 1;
          INIT = n;

/STATION/ NAME = st2, st3; SERVICE = EXP(0.5); TRANSIT = st0;
/EXEC/ MARKOV;


***  MARKOVIAN ANALYSIS ***


TOTAL NUMBER OF STATES =       286
NUMBER OF NON-ZERO ELEMENTS IN MATRIX =      1595

     1 ITERATIONS
*************************************************************************
*  NAME    *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THROUPUT *
*************************************************************************
*          *          *          *          *          *          *
* st0      * 1.000    *0.0000E+00* .1011    * 1.000    * .1011    *
*          *          *          *          *          *          *
* st1      * 9.887    * 1.000    * 9.847    * 97.36    * .1011    *
*          *          *          *          *          *          *
* st2      * .5000    *0.2528E-01*0.2594E-01* .5130    *0.5057E-01*
*          *          *          *          *          *          *
* st3      * .5000    *0.2528E-01*0.2594E-01* .5130    *0.5057E-01*
*          *          *          *          *          *          *
*************************************************************************
      MEMORY USED:      90618 WORDS OF 4 BYTES
       ( 34.85  % OF TOTAL MEMORY)
```

4.2.3/ Networks with limited capacities

Networks with limitations on the capacity of a station or the capacity of a subnetwork can be analysed with the Markovian solver. Consider the previous model where the maximum number of customers in the subnetwork (st1, st2, st3) is limited to m. First we assume that a customer leaving st0 is fedback to st0 if the maximum capacity is reached:

```
/DECLARE/ INTEGER m = 2;

/STATION/ NAME = st0;
          SERVICE = BEGIN
                    EXP(1.0);
                    pop;
                    IF npop >= m THEN TRANSIT(st0);
                    END;
/EXEC/ MARKOV;


***  MARKOVIAN ANALYSIS ***


TOTAL NUMBER OF STATES =       286
NUMBER OF NON-ZERO ELEMENTS IN MATRIX =      1595

    14 ITERATIONS
**************************************************************************
*  NAME    *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THRUPUT *
**************************************************************************
*          *          *          *          *          *          *
* st0      * 1.000    *0.0000E+00* 8.057    * 1.000    * 8.057    *
*          *          *          *          *          *          *
* st1      * 1.919    * .9501    * 1.672    * 3.377    * .4951    *
*          *          *          *          *          *          *
* st2      * .5000    * .1238    * .1355    * .5475    * .2475    *
*          *          *          *          *          *          *
* st3      * .5000    * .1238    * .1355    * .5475    * .2475    *
*          *          *          *          *          *          *
**************************************************************************
      MEMORY USED:      90649 WORDS OF 4 BYTES
       ( 34.86  % OF TOTAL MEMORY)
```

We can also assume that the customer leaving st0 goes into a simple queue, wait, where it waits until it can enter the subnetwork. The departure from the wait queue will be specified in a test statement by the procedure MOVE. This procedure will be activated as soon as the network enters a state such that the total number of customers in the subnetwork is smaller than m.

The procedure MOVE(q1, q2) moves the first customer of the queue q1 to the queue q2. If q1 is empty MOVE is ineffective.

```
/DECLARE/ QUEUE wait;

/STATION/ NAME = st0;
          SERVICE = EXP(1.0);
          TRANSIT = wait;

/CONTROL/ TEST = BEGIN
                 pop;
                 IF npop < m THEN MOVE(wait, st1);
                 END;
/EXEC/ MARKOV;


***  MARKOVIAN ANALYSIS ***


TOTAL NUMBER OF STATES =       964
NUMBER OF NON-ZERO ELEMENTS IN MATRIX =      4673

     1 ITERATIONS
*************************************************************************
*  NAME    *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THRUPUT *
*************************************************************************
*          *          *          *          *          *          *
* st0      * 1.000    *0.0000E+00* .4819    * 1.000    * .4819    *
*          *          *          *          *          *          *
* st1      * 2.000    * .9639    * 1.735    * 3.600    * .4819    *
*          *          *          *          *          *          *
* st2      * .5000    * .1205    * .1325    * .5500    * .2410    *
*          *          *          *          *          *          *
* st3      * .5000    * .1205    * .1325    * .5500    * .2410    *
*          *          *          *          *          *          *
* wait     * 2.075    * 1.000    * 7.518    * 15.60    * .4819    *
*          *          *          *          *          *          *
*************************************************************************
      MEMORY USED:     121837 WORDS OF 4 BYTES
       ( 46.86  % OF TOTAL MEMORY)
```

4.2.4/ Networks with parallel customers

The various facilities presented in the previous sections make it possible to specify and analyse complex synchronization schemes. In the next example we consider a queueing network model with parallel customers synchronized by fork-join operations. An approximate solution of this model is presented in [Heidelberger 83].

The model consists of a central server network with m+1 stations, st0, st(i), i=1,..m, n primary customers and n synchronization queues. With each primary customer is associated a class. Before entering the network, a primary customer splits into two or more parallel secondary customers. A primary customer is the parent of its secondary customers and the latter are said to be siblings. The secondary customers of a given parent have the class of their parent and are synchronized in the following way. They enter the network at the same instant of time. Then they execute concurrently and, except for queueing effect, independently of one another. When a secondary customer leaves the network it goes to the synchronization queue associated with its parent. There it waits until all of its siblings have completed. When this condition is met, all the parallel customers are synchronized: they are sent again in the network and the process is repeated.

We assume that the central server is PS with different service times for each class and that the other servers are FIFO exponential with mean service times x(i), i=1,...m. Transition probabilities from the central server to the other servers depend on the class of the secondary customers.

This model is specified in QNAP2 by first declaring the queues of the model and then defining a new object type, primary, which represents primary customers. The object type primary is built with reference to the predefined type CLASS. The attributes of the object type primary define the characteristics of a primary customer: the vector of transition probabilities, trans, for its secondary customers, their mean service time at station st0, x0, its number of secondary customers, ns, and the synchronization queue, synch, where they join.

```
/DECLARE/ INTEGER m = 2, n = 2, i;
          QUEUE st0, st(m);
          REAL x(m) = (0.5 REPEAT m);

     CLASS OBJECT primary;
           REAL trans(m), x0;
           INTEGER ns;
           QUEUE synch;
          END;
```

Then n primary customers are created in an array parent(n) and the stations of the central server network are specified:

```
                primary parent(n); REF primary r_prim;

/STATION/ NAME = st0;
          SCHED = PS;
          SERVICE = EXP(x0);
          TRANSIT = (st, synch), (trans, 1.0);
          INIT = ns;

/STATION/ NAME = st;
          SERVICE = EXP(1.0);
          TRANS = st0;
```

The join operation is described in a test statement. Whenever a new state is entered this statement is executed. The statement loops on all the declared primary customers and checks for each of them whether all of its secondary customers have joined in its synch queue. If the condition is met for a primary customer, then all its secondary customers are moved from their synch queue to the central server station st0.

```
/CONTROL/ TEST = FOR r_prim:= ALL primary DO
                   IF CUSTNB(r_prim.synch, r_prim) = r_prim.ns THEN
                    FOR i:= 1 STEP 1 UNTIL r_prim.ns DO MOVE(r_prim.synch, st0);

          CLASS = ALL QUEUE;
```

```
/EXEC/ BEGIN
       FOR r_prim:= ALL primary DC
         BEGIN
         r_prim.ns:= 2;
         r_prim.x0:= 0.5;
         r_prim.trans:= (1.0 REPEAT m);
         END;

       MARKOV;
       END;


***  MARKOVIAN ANALYSIS ***


TOTAL NUMBER OF STATES =        134
NUMBER OF NON-ZERO ELEMENTS IN MATRIX =        514

     3 ITERATIONS
*************************************************************************
*  NAME     *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THRUPUT *
*************************************************************************
*           *          *          *          *          *          *
* st0       * .5000    * .7005    * 1.333    * .9515    * 1.401    *
*(paren  1)* .5000    * .3503    * .6665    * .9515    * .7005    *
*(paren  2)* .5000    * .3503    * .6665    * .9515    * .7005    *
*           *          *          *          *          *          *
* st      1 * 1.000    * .4670    * .6760    * 1.447    * .4670    *
*(paren  1)* 1.000    * .2335    * .3380    * 1.447    * .2335    *
*(paren  2)* 1.000    * .2335    * .3380    * 1.447    * .2335    *
*           *          *          *          *          *          *
* st      2 * 1.000    * .4670    * .6760    * 1.447    * .4670    *
*(paren  1)* 1.000    * .2335    * .3380    * 1.447    * .2335    *
*(paren  2)* 1.000    * .2335    * .3380    * 1.447    * .2335    *
*           *          *          *          *          *          *
*prima  1. *          *          *          *          *          *
* synch     * 2.815    * .6575    * .6575    * 2.815    * .2335    *
*(paren  1)* 2.815    * .6575    * .6575    * 2.815    * .2335    *
*           *          *          *          *          *          *
*prima  2. *          *          *          *          *          *
* synch     * 2.815    * .6575    * .6575    * 2.815    * .2335    *
*(paren  2)* 2.815    * .6575    * .6575    * 2.815    * .2335    *
*           *          *          *          *          *          *
*************************************************************************
      MEMORY USED:      65778 WORDS OF 4 BYTES
       ( 25.30  % OF TOTAL MEMORY)
```

The behavior of the system when the secondary customers are not synchronized is simply obtained by specifying that secondary customers

leaving the network are directly fedback into station st0. The analysis is restricted to the set of stations (st0, st(i) i=1,...m) because the simple queues synch are no longer needed.

```
/STATION/ NAME = st0;
          TRANSIT = (st, st0), (trans, 1.0);

/EXEC/ BEGIN
       NETWORK( st0, st);
       SOLVE;
       END;
```

```
- MEAN VALUE ANALYSIS ("MVA") -
*************************************************************************
*  NAME     *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THRUPUT *
*************************************************************************
*           *          *          *          *          *          *
* st0       * .5000    * .8309    * 2.030    * 1.221    * 1.662    *
*(paren  1)* .5000    * .4154    * 1.015    * 1.221    * .8309    *
*(paren  2)* .5000    * .4154    * 1.015    * 1.221    * .8309    *
*           *          *          *          *          *          *
* st      1 * 1.000    * .5539    * .9852    * 1.779    * .5539    *
*(paren  1)* 1.000    * .2770    * .4926    * 1.779    * .2770    *
*(paren  2)* 1.000    * .2770    * .4926    * 1.779    * .2770    *
*           *          *          *          *          *          *
* st      2 * 1.000    * .5539    * .9852    * 1.779    * .5539    *
*(paren  1)* 1.000    * .2770    * .4926    * 1.779    * .2770    *
*(paren  2)* 1.000    * .2770    * .4926    * 1.779    * .2770    *
*           *          *          *          *          *          *
*************************************************************************
      MEMORY USED:       4204 WORDS OF 4 BYTES
       (  1.62 % OF TOTAL MEMORY)
```

As expected, the throughput of the system is increased when the secondary customers are no longer synchronized.

4.2.5/ Producer-consumer synchronization problem

We consider here the standard producer-consumer problem. A limited number, n_buffer, of buffers is available. There is a set of n_prod identical producers which produce units of information which are then stored in the buffers. If all the buffers are full, the producer which produced the unit is blocked and waits until one buffer becomes available. There is also a set of n_cons consumers which process the units of informations present in the buffers and then retrieve them. If no unit is ready to be processed, the consumer is blocked until one become available. We assume that the

times to produce (respectively to process) one unit are exponential i.i.d random variables with mean t_proc (respectively t_cons).

In the QNAP2 specification of this problem we represent the set of producers and consumers by two IS stations, producer and consumer, with initial numbers of customers n_prod and n_cons. Blocked producers (consumers) are sent in a simple queue, wait_p (wait_c), from where they will be removed as soon they can resume their activity. A buffer belongs to one of the following states: empty (no unit stored), ready (one unit stored waiting to be processed) and active (one unit stored being processed). A queue is associated with each of these states, empty, ready and active, and contains the number of buffers in the corresponding state.

In the initial state of the system there are n_buffer empty buffers in the queue empty; n_prod active producers in the queue producer; n_cons blocked consumers in the queue wait_c;

```
/DECLARE/ QUEUE producer, consumer, wait_p, wait_c, empty, ready, active;
          INTEGER n_buffer, n_prod, n_cons;
          REAL t_prod, t_cons;

/STATION/ NAME = empty;
          INIT = n_buffer;

/STATION/ NAME = wait_c;
          INIT = n_cons;
/STATION/ NAME = producer;
          TYPE =INFINITE;
          INIT = n_prod;
          SERVICE = BEGIN
                    EXP(t_prod);
                    IF CUSTNB(empty) > 0 THEN BEGIN
                                              MOVE(empty, ready);
                                              TRANSIT(producer);
                                              END
                                         ELSE TRANSIT(wait_p);
                    END;
/STATION/ NAME = consumer;
          TYPE = INFINITE;
          SERVICE = BEGIN
                    EXP(t_cons);
                    MOVE(active, empty);
                    IF CUSTNB(ready) > 0 THEN BEGIN
                                              MOVE(ready, active);
                                              TRANSIT(consumer);
                                              END
                                         ELSE TRANSIT(wait_c);
                    END;
```

The removal of blocked producers or consumers from queues wait_p and wait_c is specified in a test statement:

```
/CONTROL/ TEST = BEGIN
                 IF (CUSTNB(wait_p) > 0) AND (CUSTNB(empty) > 0)
                     THEN BEGIN
                          MOVE(empty, ready);
                          MOVE(wait_p, producer);
                          END;
                 IF (CUSTNB(wait_c) > 0) AND (CUSTNB(ready) > 0)
                     THEN BEGIN
                          MOVE(ready, active);
                          MOVE(wait_c, consumer);
                          END;
                 END;
```

```
/EXEC/ BEGIN
       n_prod:= 5; n_cons:= 5;
       t_prod:= 0.5; t_cons:= 1.0;
       n_buffer:= 8;
       MARKOV;
       END;


***  MARKOVIAN ANALYSIS ***


TOTAL NUMBER OF STATES =         14
NUMBER OF NON-ZERO ELEMENTS IN MATRIX =        110

     3 ITERATIONS
*************************************************************************
*  NAME     *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THRUPUT *
*************************************************************************
*           *          *           *          *          *          *
* producer  * .5000     *0.0000E+00* 2.495     * .5000     * 4.991     *
*           *          *           *          *          *          *
* consumer  * 1.000     *0.0000E+00* 4.991     * 1.000     * 4.991     *
*           *          *           *          *          *          *
* wait_p    * .2000     * .8712     * 2.505     * .5750     * 4.356     *
*           *          *           *          *          *          *
* wait_c    * .1000     *0.6311E-02*0.9255E-02* .1467     *0.6311E-01*
*           *          *           *          *          *          *
* empty     *0.1271E-01*0.6345E-01* .1180     *0.2364E-01* 4.991     *
*           *          *           *          *          *          *
* ready     * .1975     * .9855     * 2.891     * .5793     * 4.991     *
*           *          *           *          *          *          *
* active    * .2004     * 1.000     * 4.991     * 1.000     * 4.991     *
*           *          *           *          *          *          *
*************************************************************************
      MEMORY USED:      110237 WORDS OF 4 BYTES
       ( 42.40  % OF TOTAL MEMORY)
```

5/ CONCLUDING REMARKS AND FUTURE WORK

We have presented in this paper the usefulness of Markovian techniques in analysing complex queueing systems provided that adequate tools are available. With this respect, the value of the Markovian solver of QNAP2 has been demonstrated in several examples. The potential of this solver of becoming a still more powerful tool is important. As a continuation of the work presented here several extensions are planned. They concern the following aspects:

- state variables: as indicated in the presentation the state variables take only into account the full configuration of the queues of the

network. In several situations, as in the last example of the previous section, it would be desirable to include boolean or integer variables into the state variable so as to represent conditions or counters. This will be done through a new data type, STATE, so that variables declared with this type will be added to the internal state representation.

- explicit synchronizations: in the present version the explicit synchronization mechanisms (semaphores, resources and flags) available in simulation are not handled by the Markovian solver. They have to be represented in a somewhat indirect fashion using the procedures MOVE and TRANSIT and the concept of instantaneous transitions. In some situations this may lead to intricate descriptions. An implementation of semaphores (handled by P and V procedures) and flags (handled by WAIT, SET and RESET procedures) is currently under investigation. Thus, for example, the bloking of a server can be described an analysed easily.

- numerical algorithms: obviously, alternative numerical solution techniques have to be considered and the existing techniques should be improved. However, it is our opinion that significant progress in the size of the models studied can only be obtained by using super-computers and developing vectorized or parallel algorithms for state generation and numerical solution as well. Improvements of two or more orders of magnitude in model complexity are to be expected. QNAP2 is currently being ported on a CRAY-1S machine and comparative performance figures will be available soon.

REFERENCES

[Agrawal 83]
S. C. Agrawal, "Metamodeling: A Study of Approximations in Queueing Networks", Ph.D. Dissertation, Dept. of Computer Science, Purdue University, IN. (1983).

[Arnoldi 51]
W.E. Arnoldi, "The principle of minimized iterations in the solution of the matrix eigenvalue problem", Quart. Appl. Math., 9, 17-29 (1951).

[Baskett 75]
F. Baskett, K.M. Chandy, P.R. Muntz and F.G. Palacios, "Open, closed and mixed networks of queues with different classes of customers", J. ACM 22, 2, 248-260 (1975).

[Cachard 81]
M. Cachard and M. Veran, "Comparaison de logiciels

numeriques pour l'analyse de reseaux de files d'attente dans QNAP", Rapport de Recherche IMAG, Genoble, France (1981).

[Chandy 78]

K.M. Chandy and C.F. Sauer, "Approximate Methods for Analysing Queueing Networks", Computing Surveys, 10, 3, 281-317 (1978).

[Dugan 84]

J.B. Dugan, K.S. Trivedi, R.M. Geist and V.F. Nicola, "Extended Stochastic Petri Nets: Applications and Analysis" Performance '84, North-Holland (1984).

[Heidelberger 83]

P. Heidelberger and K.S. Trivedi, "Analytic Models for Programs with Internal Concurrency", IEEE Transactions on Computers, C-32, 73-82 (1983).

[Jacobson 84]

P.A. Jacobson, "Approximate Solution Techniques for Queueing Networks with Simultaneous Resource Possession", Technical Report 84-03-02, Dept. of Comp. Sc., Univ. of Washington (1984).

[Lavenberg 83]

S.S. Lavenberg (ed.), "Computer Performance Modelling Handbook", Academic Press (1983).

[Lazowska 84]

E.D. Lazowska, J. Zahorjan, G.S. Graham and K.C. Sevcik, "Quantitative System Performance", Prentice-Hall (1984).

[Marsan 85]

M.A. Marsan, G. Balbo, G. Ciardo and G. Conte, "A Software Tool for the Automatic Analysis of Generalized Stochastic Petri Net Models" in Modelling Techniques and Tools for Performance Analysis, North-Holland (1985).

[Merle 78]

D. Merle, D. Potier and M. Veran, "A Tool for Computer Performance Analysis", Proc. Int. Conf. on Performance of Computer Installations, North-Holland, 195-213 (1978).

[Mueller 85]

B. Mueller, "NUMAS: a Tool for the Numerical Modelling of Computer Systems" in Modelling Techniques and Tools for Performance Analysis, North-Holland (1985).

[Natkin 80]

S. Natkin, "Reseaux de Petri Stochastiques", These de

Docteur-Ingenieur, CNAM Paris (1980).

[Paul 85]

D.W. Paul, "An Approach toward a Universal Specification Language for Discrete Stochastic Systems" in Modelling Techniques and Tools for Performance Analysis, North-Holland (1985).

[Potier 82]

D. Potier, "Performance modelling tools", Proceedings of the International Symposium on Applied Mathematics and Information Science, Kyoto University, Kyoto, 4.1-4.8 (1982).

[Potier 84]

D. Potier, "New User's Introduction to QNAP2" Rapport Technique INRIA, No 40 (1984)

[Potier 85]

D. Potier (ed.), "Modelling Techniques and Tools for Performance Analysis" North-Holland (1985).

[Saad 80]

Y. Saad, "Variation on Arnoldi Method for Computing Eigen-elements of Large Unsymmetric Matrices", Linear Algebra and Applications, 34, 269-295 (1980).

[Stewart 76]

W.J. Stewart, "MARCA: Markov Chain Analyser", Rapport de Recherche IRISA, No 45, Univ. de Rennes (1976).

[Stewart 78]

W.J. Stewart, "A Comparison of Numerical Techniques in Markov Modelling", C. ACM 21, 2, 144-152 (1978).

[Veran 84]

M. Veran and D. Potier, "QNAP2: a Portable Environment for Queueing Systems Modelling", Rapport de Recherche INRIA, No 314 (1984).

[Wallace 72]

V. Wallace, "Towards an Algebraic Theory of Markovian Networks", Proc. Symp. on Computer Communication Networks and Teletraffics", Prooklyn, 397-408 (1972).

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique