QNAP 2:A portable environment for queueing systems modelling

M. Veran, D. Potier

INRIA

CENTRE DE ROCQUENCOURT

Rapports de Recherche

collection RP

N° 314

QNAP 2:
A PORTABLE ENVIRONMENT FOR QUEUEING SYSTEMS MODELLING

Michel VÉRAN
Dominique POTIER

Juin 1984

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

QNAP2 : A PORTABLE ENVIRONMENT FOR QUEUEING SYSTEMS MODELLING

Michel Véran

Bull Sems
Eychirolles, France

Dominique Potier

INRIA
Le Chesnay, France

RESUME

QNAP2 est un environnement de modélisation portable offrant des fonctions spécifiques pour la construction, la manipulation et la résolution de modèles à réseaux de files d'attente généralisés. Il comprend un ensemble d'algorithmes de calcul et un interface d'utilisateur unique pour la description des modèles, le contrôle de leur analyse et la mise en forme des résultats. L'article rappelle brièvement les aspects les plus significatifs de QNAP2 et présente les éléments principaux du langage et des méthodes de résolution de modèles. Plusieurs exemples sont discutés.

QNAP2: A Portable Environment for Queueing Systems Modelling

Michel Veran

Bull Sems
Eychirolles, France

Dominique Potier

INRIA
Le Chesnay, France

QNAP2 is a portable modelling environment providing specific facilities for building, handling and solving queueing network models. It is comprized of a collection of resolution algorithms and of a common user interface for model description, analysis control and result presentation. This paper briefly reviews the highlights of QNAP2 and presents the main features of the QNAP2 language and of its solvers. Several examples are given as illustrations.

1/ INTRODUCTION

Modelling and evaluation of computer systems performance have benefited from an important research effort in the recent years. This effort has brought forth major theoretical advances in methods and tools, as well as the elaboration of a general methodology for Performance Modelling of computer architectures and installations. This methodology is based on the representation of a computer system as a network of queues and servers. The value of this representation has been validated in numerous instances, and the underlying methodology is now regarded as operationnal. Furthermore, this modelling framework is also well suited to the represention of other classes of discrete flow systems such as communication systems or manufacturing facilities.

Research efforts in the field of modelling and evaluation have also resulted in a wide spectrum of efficient resolution techniques for queueing network models having complementary utilization domains (Lavenberg 83). These methods differ by their cost of operation and by the limitations they impose on the generality of the models being studied. Each method implements a trade-off between cost, accuracy and applicability. It is thus 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 to be analysed, and for the actual value of its parameters (Potier 82).

However, these techniques rely on different theoretical bases, and, practically, the application of several resolution methods on a given model is a long and cumbersome task. In the absence of specific aids, the non-specialist has to restrict his attention to one method (the one he knows and masters) and to discard the other approaches.

The objective of the QNAP project, launched in 1977 by INRIA and Bull, was to provide systems designers and users with a tool which facilitates the practical use of these various methods. This project resulted in 1978 in a first package, QNAP (Queueing Network Analysis Package) (Merle 78), which offered at this time a multi-method modelling tool with a high level interface language. This tool was released in France by SEMA, a subsidiary of METRA International, and installed in several research and university environnments.

The research and development effort was then continued in several directions: extensions of the interface language, implementation of new solvers, design of an internal interface, improved documentation and testing. This paper presents the resulting product, QNAP2.

The organization of the paper is as follows. In section 1 the highligths of QNAP2 are briefly presented and discussed. Section 2 introduces the modelling language. The modelling mechanisms supported by the package are summarized in section 3. Section 4 is devoted to the solvers and illustrative examples are given in section 5. The current usage of the package and its evolution are dicussed in the last section. Throughout the paper it is assumed that the reader is familiar with the basic concepts of queueing network modelling.

2/ QNAP2 HIGHLIGTHS

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.

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 is the case in other modelling packages.

As a consequence a customer process is described as follows in a QNAP2 model: for each station, the actions required to handle this particular customer (time delays, synchronisation operations, customer generation,...) must be specified as well as the corresponding routing rules. These actions will be executed by one of the servers of the station, according to the station scheduling policy (FIFO, priority, quantum,...).

Rather than providing a large set of predefined modelling mechanisms (memory allocation policies, polling, dynamic queue selection,...) which, experience shows, is seldom exactly what is needed to handle whatever specific problem is under investigation, QNAP2 offers a limited set of standard primitives and mechanisms and powerful language features allowing to build on demand any complex mechanism. Through procedure definitions and macro-processing facilities such user defined mechanisms are as easy to use as the predefined mechanisms.

2.2/ The QNAP2 solvers

The modelling framework of QNAP2 makes it possible to support a large set of analysis techniques developed for standard or generalized queueing network models.

Indeed, keeping in mind the need to support the continuing and productive research effort in the field of queueing network modelling, QNAP2 has been designed in such a way as to allow easy experimentation of new methods and their implementation at the cost of a limited programming effort. The addition of a new solver into QNAP2 is made possible through a unified internal interface as far as extended product-form like networks (with general service distribution and various scheduling disciplines) are concerned.

Presently QNAP2 provides the user with the most efficient and most widely validated techniques experimented with to date: 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 powerful Markovian solver of specific interest for the exact analysis of limited size models involving intricate mechanisms. 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 full independence between the description of a model and its analysis. In this way, the technical intricacies specific to each solver are made transparent to 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. More generally, using the algorithmic language level of QNAP2, new solvers built on top the existing solvers (for example heuristics based on the iterative solution of product-form subnetworks) can be specified and validated.

2.3/ QNAP2 usage

The QNAP2 system can be used in all the application fields involving discrete flow systems and resource contention problems: computer architectures, communication networks, manufacturing facilities,...

Within these fields of applications, four types of usage of QNAP2 can be identified:

- modelling workbench: for efficient and progressive problem analysis relying on the conciseness of the language and the spectrum of methods which are directly accessible;

- modelling environment: for the development of large and complex modelling studies: specific features enable the user to build large models involving repetitive or imbedded structures. Also, dedicated hybrid solution techniques can be defined for improved efficiency of the analysis and, if the model is to be used intensively, a tailored user interface can be added to the model for parameter input and result output (this interface is specified using the algorithmic language of QNAP2). Models or parts of a model can be saved and restored using the QNAP2 models library handling procedures.

- modelling testbench: for research in queueing network modelling: as indicated above, new solvers can be implemented and tested against existing solvers.

- education: for introducing basic modelling concepts and solution techniques.

2.4/ Portability

The QNAP2 system is coded in a subset of FORTRAN 77 and is thus fully portable. The interactions with the host operating system use the standard FORTRAN Input/Output interface. It can thus be used in an interactive mode on any time-sharing system.

2.5/ Support for cooperation.

QNAP2 is released free of charge to non-profit research or education institutions. It is expected that a wide usage of the package will enhance international scientific cooperation and provide a common basis for exchanging new algorithms and modelling experiences.

2.6/ QNAP2 industrial status

An important effort has been devoted to raise QNAP2 to the higher industrial standards in terms of quality:

- modularity and self-documentation of the FORTRAN source code;
- internal and external documentation;
- intensive testing; the testing of the solvers has been done with the following approaches:
  + automatic generation of test networks (topology, scheduling, distributions, populations);
  + cross-checking of the results through comparisons of different solvers;
  + functional tests with auto-validation of the correctness of the results.

3/ THE QNAP2 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; basic structure of the service stations; model analysis control and interactive dialogue specification.
- an algorithmic language, derived from PASCAL and SIMULA. This algorithmic language allows:
  + the specification of services involving complex mechanisms,
  + the activation of the solvers,
  + the definition of specific interfaces tailored to the user's need.

The main features of the two language levels of QNAP2 are presented in the next sections. The examples given in section 6 are used as illustrations

3.1/ The command language

A QNAP2 session consists of a sequence of commands. Each of these commands corresponds to a specific function:

- creation of variables or structured objects (command /DECLARE/);

- description of stations (command /STATION/). Each parameter of this command defines an attribute of the station to be described (type, scheduling, service, transition rules,...).

- analysis control (command /CONTROL/). The parameters of this command include the following aspects:

  + input/output control (for example, tracing options),
  + resolution control (for example, computation of marginal probabilities),
  + simulation control (for example, run length control options).

- execution of a sequence of algorithmic statements (command /EXEC/). This command can be used for:

  + the dynamic creation of parameterized models;
  + the acquisition and processing of input data;
  + the activation of the solvers (one solver or several solvers applied to different submodels);
  + the handling of the results for final edition, for filing (off-line processing) and for input to an other resolution step.

The command language supports two important features of QNAP2:

- interactivity: each command is interpreted sequentially. Thus, the user may at any step in a QNAP2 session input any command, for example for modifying the structure of the model being defined, for creating new variables or objects, or for editing additional results.

- structuration: the command language enforces the modelling framework of QNAP2. The command /STATION/ provides a general pattern for describing the topology and the modelling mechanisms involved in a model.

It is to be noted that the command language offers the basic structure for menu-driven interfaces as projected for the usage of QNAP2 on personal work-stations.

3.2/ The data types of QNAP2

Besides the usual scalar types (INTEGER, REAL, BOOLEAN, STRING), the QNAP2 language includes three main predefined object types: QUEUE, CLASS and CUSTOMER. These basic types may be enhanced with new attributes and new object types can be created over them (the properties and the attributes of the original object type are transfered to the object type thus created). Objects can be referenced either by static identifiers, or by dynamic variables.

A specific internal data structure is associated with each object of a given type. This data structure is handled at different levels of the system:

- the command language: for example the parameters of the command /STATION/ define various items of the data structure associated with an object of type QUEUE (scheduling, number of servers, service description,...).

- the algorithmic language: items of the data structure of an object may be accessed as variables considered as attributes of the object.

- the solvers make use of this data structure in order to check that the required assumptions are satisfied, retrieve and process the information needed for the analysis.

3.3/ The algorithmic language

The algorithmic language is close to PASCAL as far as expressions and statements are concerned. In addition, specific features have been introduced for handling lists of objects (selection of items which meet a given criterion, repetition of items, list of all objects of a given type,...).

In expressions, attribute variables used without explicit reference to an object are interpreted with respect to the context (for instance, in a command /STATION/, the queue associated with the station is part of the context of this station) . This facility is useful for simplifying the description of a station.

The algorithmic language includes a large set of system functions and procedures. These procedures may be used either in a service description to specify modelling mechanisms, or in an /EXEC/ command to specify input-output operations or solver activations.

4/ MODELLING MECHANISMS

4.1/ Overview

As indicated in sections 2, the modelling framework of QNAP2 is based on the concepts of stations and customers. A station is comprized of a single queue and it may belong to the following types:

- service stations (simple, multiple or infinite server);

- passive stations (resources and semaphores);

- source stations.

A customer entering a service station competes with the other customers for the allocation of a server, according to the station scheduling policy. When a server is allocated, it processes the service description of the station (thus, in QNAP2, the execution of a service may be seen as the activation of a process

associated with the customer). The execution of the service of a customer may be interrupted for various reasons (for example, preemption of the server by a customer having a higher priority). At service completion time, the customer is routed according to the transition rules specified in the station.

Resources and semaphores are queues with which one or several tokens are associated. These queues contain only passive customers which represent the requests for tokens made by active customers in the service stations.

Sources or generation stations work as infinite sources of customers.

The modelling mechanisms of QNAP2 are specified at the two language levels:

- the command language level provides the most usual modelling features; this level is in general sufficient for basic analytic modelling;

- the algorithmic language level provides access to a set of statements, procedures and functions for specifying any non-standard behavior.

4.2/ Scheduling policy

Predefined scheduling policies can be specified at the command language level (parameter SCHED of the command /STATION/): FIFO, LIFO, preemption, priorities, quantum.

Any other queueing disciplines or server allocation policy can be described using the algorithmic language (by direct manipulation of the customers in the queue), as illustrated in the example given in section 6.3.

4.3/ Service description

The service description of a station (parameter SERVICE of the command /STATION/) may involve any statements of the algorithmic language, including:

- work demand procedures: these demands (with constant or random distributions) are served according to the service rate of the station (parameter RATE);

- object creation procedures: a customer may in its service create one or several descendant customers;

- dynamic transition procedures: a customer may in its service force its own transition, or the transition of any other customer, into a given queue. This procedure is specifically useful for representing preemption mechanisms.

- synchronization procedures: these procedures allow operations on resources, semaphores or flags.

4.4/ Routing rules

The predefined routing mechanism is a probabilistic switch with fixed probabilities. The list of destination queues and classes with the corresponding probabilities is then specified at the command language level (parameter TRANSIT of the command /STATION/). This mechanism is activated at the service completion time of a customer to decide where this customer is to be routed.

In addition, any user defined routing policy can be specify within the service description by using explicit transition procedures (MOVE, TRANSIT).

Example : transition to the shortest queue

At service completion time, customers in station a are routed to stations b1 or b2, depending of the number of customers in b1 or b2 (wating or being serviced). If these numbers are equal, a customers are routed to b1 or b2 with equal probabilities.

```
/STATION/ NAME = a;
          SERVICE = BEGIN
                    EXP(10);
                    IF CUSTNB(b1) > CUSTNB(b2) THEN TRANSIT(b2);
                    IF CUSTNB(b1) < CUSTNB(b2) THEN TRANSIT(b1);
                    END;
          TRANSIT = b1,1,b2,1;
          ...
```

The routing mechanism (parameter TRANSIT = ...) will be used whenever the explicit transition procedure TRANSIT has not been executed in the service algorithm (i.e. in case of equal numbers of customers).

4.5/ Customers

The concept of customer can be considered at different modelling levels. A model may involve:

- either one simple homogeneous population,
- or several distinct populations, or classes of customers.

  The concept of class of customers is predefined in QNAP2. It is represented by the object type CLASS. The concept of class is used to handle independent populations of customers having different behaviors in term of population size, service rate, service description, routing rules and priorities.

- or individual customers with their own behavior.

Individual customers can be created using the predefined object type CUSTOMER, and manipulated at the algorithmic language level. The object type CUSTOMER can be enhanced by new attributes (constituting the customer's context). Customer priority levels may be assigned dynamically.

4.6/ Customer generation

Two predefined customer generation mechanisms are available at the command language level:

- static initialization of queues (parameter INIT of the command /STATION/);
- source stations (parameter TYPE of the command /STATION/). The SERVICE parameter is then interpreted as the interval of time between two customer generations.

Any other user defined customer generation scheme can be specified at the algorithmic language level using the customer creation function.

Example: simple source

```
/DECLARE/ QUEUE gen, cpu, terminal;
          CLASS batch, ts, apl;
          ...
/STATION/ NAME = gen;
          TYPE = SOURCE;
          SERVICE = CST(10);
          TRANSIT = cpu, batch, 0.5, terminal, ts, 0.1, terminal, apl;
          ...
```

The source gen generates a customer every 10 units of time and sends it to queue cpu (in class batch) with probability 0.5, to queue terminal (in class ts) with probability 0.1, and to queue terminal (in class apl) with probability 0.4 (complement to 1).

Example: bulk arrival process

Simulation of bulk arrivals is specified by using the customer creation function NEW(CUSTOMER). The following description represents the generation of 5 customers sent to queue a, separated by a random exponential delay.

```
/DECLARE/ QUEUE bulk, a;
          ...
/STATION/ NAME = bulk; INIT = 1;
          SERVICE = BEGIN
                    FOR i:=1 STEP 1 TO 5 DO TRANSIT(NEW(CUSTOMER),a);
                    EXP(10);
                    END;
          TRANSIT = bulk;
          ...
```

4.7/ Synchronization mechanisms

The algorithmic language level offers a set of specialized procedures, allowing operations on resource, semaphores and flags, which can be used to build tailored synchronization mechanisms. These mechanisms may be invoked as user defined procedures or macro-statements.

Operations on resources and semaphores are done by the procedures P and V (see the example of section 6.3). Flags are available as predefined objects and provide a simple synchronization mechanism involving no queueing. Flags are manipulated by the procedures SET, RESET and WAIT.

4.8/ Independence between model description and model analysis

The modelling framework of QNAP2 is designed so that the independence between the description of a model and its analysis is achieved. Provided that the required assumptions are satisfied, a given model description can be analysed with different resolution techniques.

However, because of the flexibility of the language and the diversity of the modelling mechanisms available, a given model can be specified in several conceptualy different ways. For instance a product-form queueing network will satisfy the requirements of the analytical solvers only if it is specified according to the basic modelling framework, i.e. customers competing for active servers. If an other scheme is followed, the system may not be able to recognize the queueing network structure of the model. The following example illustrates this point.

Example: machine repairman model

The first description gives the usual queueing network representation of the machine repairman model.

```
/DECLARE/ QUEUE machine, repairman;
          REAL z, s;
          INTEGER n;
/STATION/ NAME = machine; TYPE = INFINITE;
          INIT = n;
          SERVICE = EXP(z);
          TRANSIT = repairman;
 /STATION/ NAME = repairman;
          SERVICE = EXP(s);
          TRANSIT = machine;
```

The second description represents a transactional view of the same model. Customers loop in the station machine where they request the repairman considered as a passive resource. This description cannot be analysed by the analytical solvers of QNAP2.

```
/STATION/ NAME = machine; TYPE = INFINITE;
          INIT = n;
          SERVICE = BEGIN
                    EXP(Z);
                    P(repairman); EXP(s); V(repairman);
                    END;
          TRANSIT = machine;

/STATION/ NAME = repairman; TYPE = RESOURCE, MULTIPLE(m);
          ...
```

Thus, it should be pointed out that an efficient use of the package, especially as far as the anaalytical and markovian solvers are concerned, can be only achieved if the basic concepts of queueing network modelling are known by the user. Otherwise, the functions of QNAP2 would be restricted to those provided by general purpose simulation packages.

5/ QNAP2 SOLVERS

5.1/ Overview

In QNAP2, solving a model consists in computing or estimating the performance criteria which characterize the steady state of the model. Numerical solvers and an event driven simulator are available for this purpose.

QNAP2: a Portable Environment for Queueing Systems Modelling

The numerical solvers of QNAP2 include:

- several analytical solvers, exact and approximate;
- an exact Markovian solver;

The exact solvers produce the exact theorical values of the performance criteria characterizing the steady state of the model. The approximate solvers produce an estimate of the exact solution, but this approximation is generally sufficient in practical cases.

With some caution a simulation run can produce a good estimate of the steady state performance criteria of a model. Discrete event simulation can therefore be regarded as an approximate solver. Specific facilities enable the user to control the statistical quality of the point estimates produced.

Although the QNAP2 system is basically oriented towards steady-state analysis, transient analysis can be performed by using the discrete event simulator.

The solvers are activated by procedure calls (SOLVE, MARKOV, SIMUL) in the algorithmic sequence associated with an /EXEC/ command. The analysis may be restricted to a given sub-network in order to perform hierarchical modelling, and any sequence of analyses, combining different types of solvers, may be specified using the features of the algorithmic language.

When a solver is activated, the system first performs the evaluation of the expressions appearing in the following parameters of each station of the network:

- station type (parameter TYPE);
- probabilistic transitions (parameter TRANSIT);
- initial number of customers (parameter INIT);
- classes priority levels (parameter PRIOR);
- quantum values (parameter QUANTUM);
- service rates (parameter RATE).

Then, if an analytical solver is selected, the service description (parameter SERVICE) is evaluated in order to get the work demand distribution. If the Markovian solver or the simulator is selected, the statement associated with the service description is dynamically evaluated during the resolution process.

After the normal completion of a resolution procedure, the performance criteria characterizing the steady state of each queue of the model are produced in a standard report. The performance criteria contained in this report are:

- utilization factor (BUSY PCT);
- mean service time (SERVICE);
- mean number of customers (CUST NB);
- mean response time (RESPONSE);
- throughput (THRUPUT).

Example: standard report

The model is an open network with two stations and two classes of customers. The source s sends into queue a customers of class x and y with equal probabilities. Customers leaving station a are sent to b with the class x or are destroyed.

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

/STATION/ NAME = s; SERVICE =  EXP(10.);
          TRANSIT = a,x,1,a,y,1;TYPE = SOURCE;
/STATION/ NAME = a; SERVICE = EXP(1.);
          TRANSIT =  b,X,1,OUT,3;
/STATION/ NAME = b; SERVICE = EXP(3.);TRANSIT =  a;
```

The following controls request the computation of statistics per class for station a and queue length distribution of queue b.

```
/CONTROL/ CLASS = a; MARGINAL = b, 3;
/EXEC/    SOLVE;
```

```
- CONVOLUTION METHOD ("CONVOL") -
*************************************************************************
*  NAME    *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THRUPUT *
*************************************************************************
*          *          *          *          *          *          *
* s        * 10.00    * 1.000    * 1.000    * 10.00    * .1000    *
*          *          *          *          *          *          *
* a        * 1.000    *.1333     * .1538    * 1.154    * .1333    *
*(x       )* 1.000    *0.8333E-01*0.9615E-01* 1.154    *0.8333E-01*
*(y       )* 1.000    *0.5000E-01*0.5769E-01* 1.154    *0.5000E-01*
*          *          *          *          *          *          *
* b        * 3.000    *.1000     *.1111     * 3.333    *0.3333E-01*
*          *          *          *          *          *          *
*************************************************************************

*** MARGINAL PROBABILITIES: STATION b        ***

 GLOBAL         : CUSTNB=     0     1     2     3
                  PROB=   0.900 0.090 0.009 0.001
                  VARIANCE OF CUSTOMER NUMBER =        0.123
```

The edition of results according to specific user's requirements may be obtained using predefined result access functions and output procedures. Tracing and debugging options are also available.

5.2/ Analytical solvers

QNAP2: a Portable Environment for Queueing Systems Modelling

5.2.1/ Overview

QNAP2 includes a set of analytical solvers which yield exact or approximate steady-state solutions. They apply to product-form like networks with the following extensions:

- general service time distributions (Erlangian, hyperexponential, coxian distributions);
- various scheduling disciplines (FIFO, LIFO, priorities, preemption, processor sharing).

The analytical solvers presently implemented in QNAP2 apply to various combinations of these extensions. The complete list of the analytical solvers is given in section 5.2.3.

An internal interface is defined for this class of networks so that new solvers may be easily implemented in FORTRAN within the system. This interface includes a set of FORTRAN functions which provide a simple access to the internal data structure of the system (i.e traffic intensities, station types,...).

The ergodicity conditions are verified in QNAP2 taking into account the initial state of the network. The transient states , i.e. the pairs (station, class) which contain customers in the initial state of the network but which do not contain customers in the steady state, are flagged by a warning. If a non ergodic sub-chain is detected, the associated transitions are suppressed from the network and the analysis process continues.

5.2.2/ Dispatching mechanism

Because QNAP2 includes several analytical solvers, an automatic dispatching facility is built in, which searchs for the more appropriate solver depending on the characteristics of the model (Drix 82). Nevertheless, the user may request the invocation of any specific solver (controlled dispatching mode).

The dispatcher works on an ordered list of solvers built from a subset of the available solvers.

- automatic dispatching: this list is defined according to the following criteria:
  + accuracy of the results;
  + numerical stability;
  + efficiency;
- controlled dispatching: the list of solvers used by the dispatcher is specified by the user.

The dispatcher activates the first solver of the list matching the

assumptions of the model. If this solver fails (for instance because of a memory overflow or an incorrect numerical convergence) the next solver, if any, is activated and so on. If all the solvers fail an error condition is raised and signalled to the user.

5.2.3/ List of the analytical solvers

The following solvers are implemented in the current version of QNAP2:

CONVOL (convolution algorithms):

CONVOL is an exact solver, but numerically unstable for networks with large populations of customers. It implements the product-form theorems of Baskett, Chandy, Muntz and Palacios with convolution algorithms (Reiser 75, Merle 78).

MVA (mean value analysis):

MVA is an exact solver. It implements the mean value analysis approach (Reiser 80) which avoid the numerical unstabilities of the convolution algorithm. However, its conditions of application are more restrictive than those of CONVOL (closed networks with no dependent service rates).

MVANCA (mean value analysis and normalized convolution algorithms):

MVANCA is an exact solver. It implements an hybrid method based on the mean value analysis and the normalized convolution algorithms (Reiser 81). It is numerically stable and applies to closed mono-chain networks with dependent service rates.

HEURSNC (heuristic algorithm):

HEURSNC is an efficient approximate solver developped for closed networks with large populations of customers (Neuse 81).

PRIORPR (algorithm for preemptive priority scheduling):

PRIORPR is an approximate solver. It implements an algorithm for the solution of closed multiclass queueing networks with preemptive priority scheduling (Veran 83).

ITERATIV (iterative algorithm):

ITERATIV is an approximate solver which applies to closed mono-class networks with non-exponential FIFO servers (Marie 78).

DIFFU (diffusion algorithms):

DIFFU is an approximate solver (Gelenbe 77). It applies to open multi-class networks with non-exponential FIFO servers.

5.3/ The Markovian solver

The Markovian solver transforms a model described with the language of QNAP2 into a first order Markovian process with a finite number of states (Stewart 78). The transformation is based on an internal state representation involving a sufficient and minimal set of variables. Although this transformation could be done for most modelling mechanisms, it would result in such a high cost in term of memory requirements that its practical usage would be too limited (i.e. general synchronisation involving explicit P and V operations).

Thus, the implementation of the Markovian solver is the result of a trade-off between efficiency and flexibility. The models which can be processed include:

- all the extended product-form type networks (any combination of general service time distributions and scheduling policies);
- general dependent transitions and service rates. The dependencies may concern the current number of customers in any queue of the network (no other variable may be considered as a state variable). Thus, synchronizations can be expressed as state dependent transition rules.

The solver processes the model description and performs the following operations:

- 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 stationary state probabilities;
- derivation of the standard performance criteria.

The numerical technique used for computing the stationary state probabilities is derived from the Arnoldi Method (Saad 80). For current usage, networks involving up to a few thousands states may be analysed. An example of application is presented in section 6.2.

5.4/ Simulation

5.4.1/ Overview

Discrete event simulation may be used for the analysis of any model specified in the language of QNAP2. Therefore, the simulation and the language features of QNAP2 provide mechanisms comparable to those found in general purpose simulation languages (GPSS, SIMULA, SIMSCRIPT,...). Moreover, the simulator of QNAP2 includes several facilities for confidence interval estimation which are briefly described in the next sections.

5.4.2/ Estimation of confidence intervals

QNAP2 includes three methods for confidence interval estimation:

- the replication method (Lavenberg 77),
- the regeneration method (Iglehart 78),
- the spectral method (Heidelberger 80).

These methods rely on various assumptions described in the following three sub-sections and have different characteristics in term of computational complexity and memory requirement. As for other aspects of QNAP2, a default option is provided for current usage, and a complete tool set is made available for more specific user needs. The default confidence interval method is the spectral method.

5.4.2.1/ The replication method

Independent replications is the most straightforward method for obtaining confidence intervals. It consists in generating several sample paths of the model studied, so that these sample paths are statistically independent and identical. This is achieved by reseting the original initial state of the model at the beginning of each replication, and by using a different random number stream for each replication.

The replication method may be used for obtaining confidence intervals on the results of a transient analysis.

5.4.2.2/ The regeneration method

The basic principle is to split the simulation run into several successive intervals, so that the model behavior in these intervals are statistically equivalent and independent. Thus there are as many independent sub-simulations as there are intervals and the results within each simulation sub-run are considered as samples of independent identically distributed random variables; these variables will be used to estimate the confidence intervals and the final results.

Regeneration points are defined in a QNAP2 model by specifying a condition defining the regeneration states and by explicitly calling a predefined procedure (SAMPLE) if the condition is satisfied. The function of this procedure is the computation of partial statistics on the last regeneration interval and of global statistics and confidence intervals using the information collected from the beginning of the

simulation up to the current regeneration point. In this way partial results on the simulation are available during the simulation run (point estimates and confidence intervals) and can be used to define sequential stopping rules.

The more general way to specify a condition is to program it in a special process, called a test sequence, which can be activated with a user defined period of whenever an event is processed by the simulator (the condition is then continuouly tested). Another way to specify a regeneration state is to program the condition in a service description, provided that the condition is met only when this service is active. This is in general a more efficient approach because the condition will be evaluated less frequently than in the previous case.

Example: definition of a regeneration state

```
/DECLARE/ QUEUE gen, system;

/STATION/ NAME = gen; TYPE = SOURCE;
          SERVICE = EXP (1);
          TRANSIT = system;

/STATION/ NAME = system;
          SERVICE = HEXP (0.7, 5);
          TRANSIT = OUT;

/CONTROL/ TMAX = 15000;

          ACCURACY = system;
          ESTIM = REGENERATION;
          PERIOD = 0;
          TEST =   IF system.NB = 0 THEN SAMPLE;

/EXEC/    SIMUL;
```

```
***SIMULATION WITH REGENERATIVE METHOD ***
... TIME =    15000.00 , NB SAMPLES =   4427 , CONF. LEVEL = 0.95
*************************************************************************
*   NAME    *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  SERV NB *
*************************************************************************
*           *          *          *          *          *          *
* gen       * 1.018    * 1.000    * 1.000    * 1.018    *      14726*
*           *          *          *          *          *          *
* system    * .7132    * .6990    * 5.193    * 5.271    *      14698*
*   +/-     *0.2435E-01*0.2567E-01* .8906    * .8758    *          *
*           *          *          *          *          *          *
*************************************************************************
... END OF SIMULATION ...
```

In the case of approximate regeneration points, the statistical independance of the intervals must be thoroughly accessed before meaningfull interpretations of the confidence intervals are made. For this purpose the computation of autocorrelation functions can be requested for the basic measures used in the derivation of the standard performance criteria.

5.4.2.3/ The spectral method

The basic principle of the other methods is to build a set of independent and identically distributed samples and to apply standard statistical techniques for estimating confidence intervals. The spectral method is somewhat different in the sense that it does not rely on the independence assumption and applies to correlated samples provided that the identical distribution property is satisfied. The main advantage of the spectral method over the other methods is that the user needs not bother with the choice of specific parameters (number of replications, regeneration state,..): the method applies to the analysis of the stationary behaviour of most simulated model.

6/ EXAMPLES

The following examples have been selected to illustrate in a straigthforward manner the modelling features and the solvers of QNAP2. They should not be considered as realistic examples of the usage of QNAP2 as far as the size or the complexity of the presented models is concerned.

6.1/ Simple product-form networks

This example illustrates how the various analytical solvers of QNAP2 can be used depending on the assumptions of the model considered. The Markovian solver is used concurrently on the same model in order to check the accuracy of the results produced by the approximate analytical solvers.

The model consists of four stations (terminal, cpu, disk1 and disk2) with two classes of customers (c1 and c2). The services of the two classes of customers at stations terminal and cpu are distinct. In a first step it is assumed that the station terminal is infinite and that the station cpu is processor-sharing, so that the product-form assumptions are satisfied. The QNAP2 description of this simple model is as follows:

```
/DECLARE/ QUEUE terminal, cpu, disk1, disk2;
          CLASS c1, c2;
          INTEGER i;

/STATION/ NAME = terminal; TYPE = INFINITE;
          INIT(c1) = 2; INIT(c2) = 1;
          SERVICE(c1) = EXP(1000); SERVICE(c2) = EXP(2000);
          TRANSIT = cpu;

/STATION/ NAME = cpu; SCHED = PS;
          SERVICE(c1) = EXP(10);
          TRANSIT(c1) = disk1,10,disk2,15,terminal,1;
          SERVICE(c2) = EXP(20);
          TRANSIT(c2) = disk1,15,disk2,5,terminal1,1;

/STATION/ NAME = disk1;
          SERVICE = EXP(30); TRANSIT = cpu;

/STATION/ NAME = disk2;
          SERVICE = EXP(25); TRANSIT = cpu;
```

The model is analysed using the analytical solvers (SOLVE procedure) and then the Markovian solver (MARKOV procedure). Only the throughput at station terminal for both classes is edited. The calls to the solvers and the edition of the results is specified in a procedure which will be used for all the next steps of the example.

```
/CONTROL/ OPTION = NRESULT;
          CLASS = ALL QUEUE;

/DECLARE/
  PROCEDURE resolution;

      BEGIN
      PRINT(" ");
      PRINT("solver      c1          c2");
      SOLVE;
      PRINT("SOLVE :",MTHRUPUT(terminal, c1), MTHRUPUT(terminal, c2));
      MARKOV;
      PRINT("MARKOV:",MTHRUPUT(terminal, c1), MTHRUPUT(terminal, c2));
      END;
```

The analyses are now requested:

```
/EXEC/ resolution;

solver      c1           c2
SOLVE :  0.9080E-03  0.3015E-03
MARKOV:  0.9080E-03  0.3016E-03
```

The dispatching algorithm selects the Mean Value Analysis algorithm. As would be shown in the standard report the Markovian solver identifies 50 states for this model. It is verified that similar results are produced by the two solvers.

The second step consists in modifying the service distributions at the station cpu and checking that still similar results are obtained. However, the Markovian solver identifies in this case a greater number of states (92) because of the internal Coxian representation of the non-exponential distributions.

```
/STATION/ NAME = cpu; SCHED = PS;
          SERVICE(c1) = HEXP(10,4); SERVICE(c2) = ERLANG(20,2);

/EXEC/ resolution;

solver      c1           c2
SOLVE :  0.9080E-03  0.3015E-03
MARKOV:  0.9079E-03  0.3015E-03
```

We consider now a priority, preemptive scheduling at station cpu, assuming that the class c2 has the highest priority. The model is no longer a product form network. The dispatching algorithm selects then the approximate solver PRIORPR. The Markovian solver provides a reference for checking the accuracy of the approximation.

```
/STATION/ NAME = cpu;
          SERVICE(c1) = EXP(10); SERVICE(c2) = EXP(20);
          SCHED = PRIOR, PREEMPT;
          PRIOR(c1) = 1; PRIOR(c2) = 2;

/EXEC/ resolution;

solver      c1           c2
SOLVE :  0.8851E-03  0.3138E-03
MARKOV:  0.8712E-03  0.3192E-03
```

Coming back to the initial assumptions of step 1, the topology of the network is changed so that a customer visits successively each station. The mean service times are also changed so that the global requests of a customer remain unchanged.

```
/STATION/ NAME = cpu;
          SERVICE(c1) = EXP(26*10);
          SERVICE(c2) = EXP(21*20);
          TRANSIT(c1,c2) = disk1;
          SCHED = PS;

/STATION/ NAME = disk1;
          SCHED  =  PS;
          SERVICE(c1) = EXP(30*10); SERVICE(c2) = EXP(30*15);
          TRANSIT(c1,c2) = disk2;

/STATION/ NAME = disk2;
          SCHED  =  PS;
          SERVICE(c1) = EXP(15*25); SERVICE(c2) = EXP(5*25);
          TRANSIT(c1,c2) = terminal;

/EXEC/ resolution;

solver      c1            c2
SOLVE :  0.9080E-03   0.3015E-03
MARKOV:  0.9080E-03   0.3015E-03
```

The results are similar to those obtained in steps 1 and 2 due to the fact that the steady-state solution of a product network is independent of its topology.

This property is no longer verified for non product-form networks as shown in the next step. If priority preemptive scheduling is again assumed at station cpu as in step 3, the exact results are not similar to those obtained previously.

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

/EXEC/ resolution;

solver      c1            c2
SOLVE :  0.8885E-03   0.3137E-03
MARKOV:  0.8795E-03   0.3130E-03
```

6.2/ A Markovian model of a multi-processor architecture

This example shows the application of the Markovian solver for the analysis of a computer architecture involving few customers and complex synchronization schemes.

M. Veran and D. Potier

The model represents in a simplified way a multi-microprocessor system with shared memories. An asynchronous bus is the global communication support between the processors. Each processor is linked to its own memory through a local bus. A processor can access any non-local memory through the global bus.

In order to access a non local-memory a processor must first request the global bus. Then a bus delay, t_bus, is incurred before the memory access can be performed. If the memory is busy serving a local request, the non-local request waits until the completion of the local access. The global bus is busy until the non-local access is completed.

This system can be modeled as a queueing network with resource possession. It can be analysed using discrete event simulation, approximate analysis (Jacobson 82, Marsan 83), or Markovian analysis as presented here.

The following parameters are used to describe the behavior of a processor:

- t_proc: mean active processor time between two consecutive memory accesses (local or non-local);
- p_local: probability of a local memory access;
- t_local: mean memory service time for local accesses;
- t_global: mean memory service time for non-local accesses.

A non-local access is distributed with equal probabilities over the non-local memories.

The QNAP2 description of the model is parametrized by the number, n, of processors and memories. A user-defined attribute used as an index is added to the object types QUEUE and CLASS. With each processor proc(i) is associated a local memory mem(i) and the activity of processor proc(i) is represented by a customer of class c(i).

```
/DECLARE/ QUEUE INTEGER ip;  & index of a queue
          CLASS INTEGER ic;  & index of a class

          INTEGER n = 3;        & number of processors

          QUEUE proc(n), mem(n);
          QUEUE bus, wait_bus;

          CLASS c(n);
          INTEGER i;

          REAL t_proc,        & mean time between memory accesses
               t_local,       & mean memory access time (local)
               t_global,      & mean memory access time (global)
               p_local,       & proportion of local accesses
               t_bus;         & mean bus time (excluding memory access)
          BOOLEAN bus_free;
```

The description of the stations proc, mem and bus makes use of the queue and class attributes ip and ic. This allows a concise expression of the routing rules for any configuration of the model.

```
/STATION/ NAME = proc;
          INIT = IF ip = ic THEN 1 ELSE 0;
          SERVICE = EXP(t_proc);
          TRANSIT = mem(ic), p_local, wait_bus;

/STATION/ NAME = mem;
          SERVICE = IF ic = ip THEN EXP(t_local)
                               ELSE EXP(t_global);
          TRANSIT = proc(ic);

/STATION/ NAME = bus;
          SERVICE = EXP(t_bus);
          TRANSIT = mem WITH QUEUE<>mem(ic), 1.0 REPEAT n-1;
```

In the present version of QNAP2, the synchronisation procedures P and V can only be used with simulation. Thus, the synchronisations have to be specified by describing the transition rules. In the model, a customer requesting a non-local memory is sent to a simple FIFO queue, wait_bus. The bus is allocated to the first customer of this queue as soon as the bus is freed by the previous non-local request. This condition is expresssed in a test sequence, activated at each state transition of the model. The current state of the model is accessed through the predefined function CUSTNB, which return the current number of customers in a given queue. The condition "bus available" is tested by checking in all the memories that there is no non-local access waiting or being processed.

```
/CONTROL/
    TEST =BEGIN
          bus_free :=  TRUE;
          FOR i := 1 STEP 1 UNTIL n DO
             IF ( (CUSTNB(mem(i)) = 1) AND (CUSTNB(mem(i),c(i)) = 0) )
                                       OR  (CUSTNB(mem(i))>1)
             THEN bus_free := FALSE;

          IF bus_free THEN MOVE(wait_bus, bus);
          END;

          OPTION = NRESULT;
```

In the next /EXEC/ command the indices of the processors, memories and classes are initialized. Then the parameters of the models are set and the analysis is requested for several values of the local access probability. The edited performance criteria is the efficiency of the system defined as the ratio between the effective throughput of a processor (in terms of the number of memory accesses per unit of time) and the throughput which would be obtained if no bus nor memory contention occured.

```
/EXEC/ BEGIN
       FOR i := 1 STEP 1 UNTIL n DO
         BEGIN proc(i).ip := i; c(i).ic := i; mem(i).ip := i; END;

       t_proc := 10; t_local := 2; t_global := 3;
       t_bus := 4;

       PRINT(" ");
       FOR p_local :=  0.8, 0.4, 0.2 DO
            BEGIN
            MARKOV;

            PRINT("Percentage of local access = ",p_local,
                  "Efficiency = ",MTHRUPUT(proc(1))*(t_proc+p_local*t_local+
                                        (1-p_local)*(t_bus+t_global))  );

            END;

      END;
```

```
Percentage of local access =    .8000    Efficiency =    .9752
Percentage of local access =    .4000    Efficiency =    .8990
Percentage of local access =    .2000    Efficiency =    .8574
```

The example analysed here (3 processors) lead to a Markov chain with 274 states and 1051 non-zero elements in the transition matrix.

6.3/ A simulation model of a memory allocation mechanism

This model illustrates the use of the predefined synchronization mechanisms for defining complex customer management policies.

The model describes a transaction system processing different types of transactions. The system consists of a set of terminals and three service stations cpu, disk1 and disk2. A transaction is represented as a customer class (predefined object type CLASS) with user defined attributes. The attributes of a transaction are its mean number of accesses to each disk, its mean service time at the cpu, and the number of memory pages required. Users at terminals issue the different types of transactions according to fixed probabilities.

QNAP2: a Portable Environment for Queueing Systems Modelling

```
/DECLARE/
  CLASS REAL n_disk1, n_disk2,      & mean access number to disks 1 and 2
            t_cpu,                  & mean service time at cpu
            pr;                     & rate of requests to this transaction

  CLASS INTEGER demand;             & number of pages requested

        REAL t_term;                & mean think time at terminal
        INTEGER n_user;             & number of user terminals

        CLASS t1, t2, t3;
        REAL  p1, p2, p3;
```

In a first step it is assumed that the memory is divided into n_part equal partitions. A partition must be allocated to a transaction before it can be activated. In the model, the memory is represented as a resource station, memory, and each memory partition corresponds to one token. The allocation of a memory partition is requested in the station "request" by the procedure call P(memory). When a transaction is completed it leaves the system through the station "release" where the procedure call V(memory) is executed (the P and V operations could as well be specified in the station "terminal" without creating ad hoc stations).

```
QUEUE cpu, disk1, disk2, terminal;  & stations of the model
QUEUE request, release, memory;

REAL t_disk;                        & disk service time
INTEGER n_part, i;

/STATION/ NAME = terminal; TYPE = INFINITE;
        SERVICE = EXP(t_term);
        TRANSIT = request, t1, p1, request, t2, p2, request, t3, p3;
        INIT(t1) = n_user;

/STATION/ NAME = request; TYPE = INFINITE;
        SERVICE = P(memory);
        TRANSIT = cpu;

/STATION/ NAME = cpu;
        SERVICE = EXP(t_cpu);
        TRANSIT = release, 1, disk1, n_disk1, disk2, n_disk2;

/STATION/ NAME = disk1, disk2;
        SERVICE = EXP(t_disk);
        TRANSIT = cpu;

/STATION/ NAME = release;
        SERVICE = V(memory);
        TRANSIT = terminal;

/STATION/ NAME = memory; TYPE = RESOURCE, MULTIPLE(n_part);
```

The attributes of the transactions and the other parameters of the model are now specified. For sake of simplicity the values are given by assignement statements rather than by a user defined dialogue.

```
/EXEC/ BEGIN
      n_part:= 3;
      t_disk:= 40;
      n_user:= 20;
      t_term:= 5000;
      p1:= 0.3; p2:= 0.5; p3:= 0.2;

      t1.n_disk1:= 13; t1.n_disk2:= 5; t1.t_cpu:= 10; t1.demand:= 20;
      t2.n_disk1:= 7;  t2.n_disk2:= 3; t2.t_cpu:= 25; t2.demand:= 30;
      t3.n_disk1:= 0;  t3.n_disk2:= 45;t3.t_cpu:= 50; t3.demand:= 100;

      END;
```

The next statements specify various control options of the simulation run (maximum simulation time, estimation of confidence intervals, edition of final results). The confidence interval estimation method used is the spectral method.

```
/CONTROL/

TMAX = 20000;
OPTION = NRESULT;

ACCURACY = cpu, release;

EXIT = BEGIN
      PRINT(" ");
      PRINT("cpu utilization:", MBUSYPCT(cpu), "+/-", CBUSYPCT(cpu) );
      PRINT("thruput:        ", MTHRUPUT(release), "+/-", CTHRUPUT(release) );
      PRINT("multipr.  level:", MCUSTNB(cpu)+MCUSTNB(disk1)+MCUSTNB(disk2) );
       END;

/EXEC/ SIMUL;

cpu utilization:    .7524     +/-    .1258
thruput:          0.1150E-02+/-  0.4533E-03
multipr.  level:    2.774
```

In a second step it is assumed that the memory is divided into n_page pages. A transaction can be activated only if its required number of pages is allocated. The transactions competing for memory allocation are ordered according to their priority level. The priority level is periodicaly increased in order to avoid a starving phenomenon. The initial priority levels are assigned in order to favor the small transactions.

In the station request an incoming transaction gets its memory allocation if it is available and then proceeds to the cpu queue. Otherwise, it remains in the station request while increasing its priority level. Upon completion the transaction goes to the release station where it releases its pages and checks if the first transactions waiting for memory allocation in the request queue can be served. This mechanism is specified by using explicit operations on individual customers.

```
/DECLARE/
      REAL t_delta;
      INTEGER p_delta;

      REF CUSTOMER c, c0;

      INTEGER n_page;
/STATION/ NAME = request;
          SCHED = PRIOR;
        SERVICE = BEGIN
                  IF n_page>demand
                  THEN BEGIN
                       n_page:= n_page - demand;
                       TRANSIT(cpu);
                       END
                  ELSE BEGIN
                       PRIOR(100-demand);
                       WHILE TRUE DO
                          BEGIN
                          CST(t_delta);
                          PRIOR(CPRIOR+p_delta);
                          END;
                       END;
                  END;
/STATION/ NAME = release;
```

```
        SERVICE = BEGIN
                  n_page:= n_page + demand;
                  c:= request.FIRST;
                  WHILE c<>NIL DO
                     BEGIN
                     IF c.CCLASS.demand<=n_page
                      THEN BEGIN
                           n_page:= n_page - c.CCLASS.demand;
                           c0:= c; c:= c.NEXT;
                           TRANSIT(c0, cpu);
                           END
                       ELSE c:= NIL;
                      END;
                    END;
/EXEC/ BEGIN

      n_page:= 300;
      t_delta:= 500; p_delta:= 10;

      END;

/EXEC/ SIMUL;

cpu utilization:    .6505     +/-  0.9821E-01
thruput:          0.2500E-02+/-  0.7278E-03
multipr.  level:    3.725
```

6.4/ An approximate model of internal concurrency

The example presented in this section is a queueing network model for programs with internal concurrency. This example shows how QNAP2 can be used to build new solution techniques from existing solvers and thus analyse models whose complexity prevents the use of standard techniques. The model and its solution have been published in (Heidelberger 82).

6.4.1/ Presentation of the model

The system studied consists of a finite number, m, of processors and a finite number, n, of jobs. A job consists of a primary task and two or more secondary taks. Each primary task executes on a sequence of processors, this sequence of processors forming a Markov chain. There is a particular node, labeled 0, with 0 service time such that whenever a primary task enters this node it is split into two or more secondary tasks. A primary task is the parent of its secondary tasks and the latter are said to be siblings. The secondary tasks execute concurrently, except for queueing effects, independently of one another. Each secondary task execute a sequence of processors, the sequence of processors forming its own Markov chain. A secondary task is considered complete upon entering node 0. At node 0 the secondary task must wait until all of its siblings

have completed execution at wich time the primary task becomes active again and the process is repeated. Synchronization between secondary tasks is achieved by requiring all siblings to complete execution before the job can continue processing.

A solution to this model consists of iteratively solving a sequence of product-form networks so that upon convergence, the solution to the product-form network closely approximate the solution of the system. Each of these product-form networks will have m+1 service stations (the m processors and a fictitious station) and d+1 classes of customers if the number of secondary tasks spawned by one primary task is d.

Each secondary task is in one of the three macro-state: dormant, active, or waiting for the completion of its siblings. The primary task is in one of two states: active or waiting for all its secondary tasks to complete. In the active state a task consists of a sequence of visits to system resources and hence is easily represented by a chain of the queueing network. The waiting state of a task will be represented by a fictitious delay type server, i.e. an infinite server station. Similarly, the dormant state will be represented by the delay server.

For sake of simplicity we make the following assumptions. The network of processor is of the central server type. The queueing discipline at the central processor is PS (processor-sharing); the queueing discipline at the other processors if FIFO. All service times have exponential distributions. The tasks may have different mean service times at the central processor but they have identical service times at the other processors. All tasks start on the central processor and a task cannot split or complete after executing the central processor.

6.4.2/ QNAP2 description of the model

The description and analysis of this model with QNAP2 will illustrate the object management facilities of QNAP2 and the use of QNAP2 for building and coding new solution algorithms over existing solvers.

6.4.2.1/ Description of the processor queues

In order to represent the processors and the fictitious station we add two additional attributes to the object type QUEUE: a real variable t representing the mean service time of tasks and an integer variable index. Then an array of 21 queues which will be used to manipulate the processors is created (the maximum number of processors in the model will be limited to 20).

```
/DECLARE/ QUEUE REAL t; QUEUE INTEGER index;

QUEUE proc(0:20);
```

6.4.2.2/ Description of the tasks

A task is specified as an object type defined with reference to the predefined type CLASS. The attributes of a task are:

- two vectors, transit1 and transit2, of transition probabilities,
- its mean service time tx at the central processor,
- a reference, parent, to its parent task,
- two real variables s0 and s1 representing its mean delay at the fictitious station at two consecutive iterations of the iterative procedure,
- the number n of instances of this task,
- a string containing the name of the task.

The vector transit1 contains the transition probabilities of the task from the central processor (assumed processor number 1) to the processors 2, 3...,m. For a primary task the element transit2(i) contains the probability that the task splits after executing on the processor i (i > 1); for a secondary task the element transit2(i) contains the probability that the task completes after executing on the processor i (i > 1).

```
CLASS OBJECT task;
      INTEGER n;
      REAL transit1(2:20), transit2(2:20), tx, s0, s1;
      REF task parent;
      STRING name;
      END;

REF task r_task1, r_task2;
```

6.4.2.3/ Description of the stations processor

The service stations representing the processors and the fictitious station are now described. First, an INTEGER, m, representing the actual number of processors is declared and then the processors are described using the command /STATION/.

```
/DECLARE/ INTEGER m;

/STATION/ NAME = proc(2 STEP 1 UNTIL 20);
          SERVICE = EXP(t);
          TRANSIT = proc(0), transit2(index), proc(1);

/STATION/ NAME = proc(0);
          SERVICE = EXP(s0);
          TRANSIT = proc(1);
          TYPE = INFINITE;
          INIT = n;

/STATION/ NAME = proc(1);
          SERVICE = EXP(tx);
          TRANSIT = proc(2 STEP 1 UNTIL m),transit1(2 STEP 1 UNTIL m);
          SCHED = PS;
```

Note that the description of the processor stations is parametrized by m. The expressions appearing in the right hand side of the parameters of the /STATION/ command will be evaluated when a solver is activated (with the exception of the parameter NAME which is evaluated at compile time).

6.4.2.4/ Specification of a user defined procedure

The approximate method used requires the computation of the mean value of a random variable defined as the MAX of k independent exponential random variables. A specific procedure is defined to carry out this computation:

```
/DECLARE/ REAL x, z(50);

PROCEDURE emax ( k );
          INTEGER k, l; REAL b, c;
          BEGIN
```

for sake of brevity the complete description of the procedure emax is not given here. k is the number of independent random variables considered, z(k) is the mean of the k-th variable. The result is contained in the global variable x. l, b and c are local variables.

```
          END;
```

6.4.3/ Specification of an input dialogue

The following commands specify an input dialogue. First, working variables are declared and control parameters are set.

```
/DECLARE/ OBJECT answer; END;
          answer yes, no;

          LABEL def0, def1, def2, iter; INTEGER niter;

/CONTROL/ CLASS = ALL QUEUE; OPTION = NRESULT;
/CONTROL/ UNIT = GET(5);

/DECLARE/ INTEGER i, j; REAL epsilon =0.005;
```

Then an EXEC block specifies a dialogue for the input of the number of processors and the definion of the configuration of the system (characteristics of primary tasks, number and characteristics of secondary tasks for each type of primary tasks) and perform the analysis of the model:

```
/EXEC/ BEGIN

& definition of the number of processors

  def0: PRINT("number of processors ?");
        m:= GET(INTEGER);
        IF m > 20 THEN BEGIN
                       PRINT("number of processors limited to 20. Try again");
                       GOTO def0;
                       END;

& initialization of the processor queues index

        FOR i:=0 STEP 1 UNTIL m DO proc(i).index:= i;

& definition of primary tasks

        PRINT(" ");
   def1:PRINT("definition of a primary task (yes or no)");
        IF GET(answer) = yes
        THEN BEGIN
             r_task1:= NEW(task);
             PRINT(" ");
             PRINT("name of the primary task ?");
             r_task1.name:= GET(STRING);
             PRINT("number of such primary tasks");
             r_task1.n:= GET(INTEGER);
             PRINT("mean service time at central processor");
             r_task1.tx:= GET(REAL);
             PRINT("transition probabilities from processor 1 to 2,..",
             m);
             FOR i:= 2 STEP 1 UNTIL m DO r_task1.transit1(i):= GET(REAL);
             PRINT("transition probabilities from processors 2,..",
             m," to node 0 (split)");
             FOR i:= 2 STEP 1 UNTIL m DO r_task1.transit2(i):= GET(REAL);
```

```
& definition of secondary tasks

          PRINT(" ");
     def2:PRINT("definition of secondary task (yes or no)");
          IF GET(answer) = yes
          THEN BEGIN
               r_task2:= NEW(task);
               r_task2.parent:= r_task1;
               PRINT(" ");
               PRINT("name of the secondary task ?");
               r_task2.name:= GET(STRING);
               r_task2.n:= r_task1.n;
               PRINT("mean service time at central processor");
               r_task2.tx:= GET(REAL);
               PRINT("transition probabilities from processor 1 to 2,..",
               m);
               FOR i:= 2 STEP 1 UNTIL m DO
                   r_task2.transit1(i):= GET(REAL);
               PRINT("transition probabilities from processors 2, ...",
               m," to node 0 (join)");
               FOR i:= 2 STEP 1 UNTIL m DO
                   r_task2.transit2(i):= GET(REAL);
               GOTO def2;
               END;
          GOTO def1;
          END;

     PRINT(" ");
     PRINT("mean service times at processors 2,...",m);
     FOR i:= 2 STEP 1 UNTIL m DO proc(i).t:= GET(REAL);

     FOR r_task1:= ALL task DO r_task1.s0:= r_task1.tx;

& iterative solution

     niter:=1;

     NETWORK( proc( 0 STEP 1 UNTIL m) );

iter: SOLVE;

     FOR r_task1:= ALL task DO
         BEGIN
         x:= 0.;
         FOR i:= 1 STEP 1 UNTIL m DO
             x:= x+ MCUSTNB(proc(i), r_task1);
         r_task1.s0:= x/MTHRUPUT(proc(0), r_task1);
         END;
     FOR r_task1:= ALL task WITH parent = NIL DO
         BEGIN
         i:= 0;
         FOR r_task2 := ALL task DO IF r_task2.parent = r_task1 THEN
```

```
            BEGIN
            i:= i+ 1;
            z(i):= r_task2.s0;
            END;
        emax(i);
        FOR r_task2 := ALL task DO IF r_task2.parent = r_task1 THEN
            r_task2.s0:=x- r_task2.s0+ r_task1.s0;
        r_task1.s0:= x;
        END;
```

& convergence test

```
    i:= 0; x:= 0.;
    FOR r_task1:= ALL task DO
        BEGIN
        i:= i+1;
        x:= x+ (r_task1.s0- r_task1.s1)**2;
        r_task1.s1:= r_task1.s0;
        END;
    x:= (x**0.5)/i;
    IF x >= epsilon THEN BEGIN
                         niter:=niter+ 1;
                         GOTO iter;
                         END;
```

& results

```
    PRINT(" ");
    PRINT("task   thruput      delay    proc 1 busy pct");
    PRINT(" ");
    FOR r_task1:= ALL task DO
        PRINT(r_task1.name, MTHRUPUT(proc(1), r_task1),
              MSERVICE(proc(0), r_task1),
              MBUSYPCT(proc(1), r_task1));
```

& detailed results

```
    PRINT(" ");
    PRINT("detailed results requested (yes or no)");
    PRINT(" ");
    IF GET(answer) = yes THEN OUTPUT;
```

& definition of a new configuration

```
    PRINT(" ");
    PRINT("definition of a new workload (yes or no)");
    IF GET(answer) = yes
    THEN BEGIN
         FOR r_task1:= ALL task DO r_task1.n:= 0;
         GOTO def0;
         END;
    END;
```

6.4.4/ Analysis of a model

The input parameters are now entered according to the dialogue defined. The configuration is made of 5 processors. The workload consists of 5 identical primary tasks, each primary task splitting into two secondary tasks. The mean service of all tasks at the central processor is 0.01; The mean service time of all tasks at the other processors is 0.04; the probability that a primary task splits after executing on one of the processors 2, 3,..5 is 0.1; the probability that a secondary task completes is 0.1 for the first secondary task, 0.05 for the other one.

```
number of processors ?
5

definition of a primary task (yes or no)
yes

name of the primary task ?
"t1 "
number of such primary tasks
5
mean service time at central processor
0.01
transition probabilities from processor 1 to 2,..        5.
0.25 0.25 0.25 0.25
transition probabilities from processors 2,..        5. to node 0 (split)
0.1 0.1 0.1 0.1

definition of secondary task (yes or no)
yes

name of the secondary task ?
"t11"
mean service time at central processor
0.01
transition probabilities from processor 1 to 2,..        5.
0.25 0.25 0.25 0.25
transition probabilities from processors 2, ...        5. to node 0 (join)
0.1 0.1 0.1 0.1
```

```
definition of secondary task (yes or no)
yes

name of the secondary task ?
"t12"
mean service time at central processor
0.01
transition probabilities from processor 1 to 2,..        5.
0.25 0.25 0.25 0.25
transition probabilities from processors 2, ...        5. to node 0 (join)
0.05 0.05 0.05 0.05
definition of secondary task (yes or no)
no
definition of a primary task (yes or no)
no

mean service times at processors 2,...        5.
0.04 0.04 0.04 0.04

task   thruput      delay     proc 1 busy pct

t1    14.45         2.403        .1445
t11    14.45        2.403        .1445
t12    28.94        1.423        .2894
```

QNAP2: a Portable Environment for Queueing Systems Modelling

```
detailed results requested (yes or no)

yes
*************************************************************************
*  NAME    *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THRUPUT *
*************************************************************************
*          *          *          *          *          *          *
* proc   1 * 2.076    *0.0000E+00* 9.006    * 2.076    * 4.338    *
*(task   1)* 2.403    *0.0000E+00* 3.473    * 2.403    * 1.445    *
*(task   2)* 2.403    *0.0000E+00* 3.473    * 2.403    * 1.445    *
*(task   3)* 1.423    *0.0000E+00* 2.059    * 1.423    * 1.447    *
*          *          *          *          *          *          *
* proc   2 *0.1000E-01* .5785     * 1.199    *0.2073E-01* 57.85    *
*(task   1)*0.1000E-01* .1445     * .3054    *0.2113E-01* 14.45    *
*(task   2)*0.1000E-01* .1445     * .3054    *0.2113E-01* 14.45    *
*(task   3)*0.1000E-01* .2894     * .5881    *0.2032E-01* 28.94    *
*          *          *          *          *          *          *
* proc   3 *0.4000E-01* .5785     * 1.199    *0.8290E-01* 14.46    *
*(task   1)*0.4000E-01* .1445     * .3054    *0.8452E-01* 3.613    *
*(task   2)*0.4000E-01* .1445     * .3054    *0.8452E-01* 3.613    *
*(task   3)*0.4000E-01* .2894     * .5881    *0.8129E-01* 7.235    *
*          *          *          *          *          *          *
* proc   4 *0.4000E-01* .5785     * 1.199    *0.8290E-01* 14.46    *
*(task   1)*0.4000E-01* .1445     * .3054    *0.8452E-01* 3.613    *
*(task   2)*0.4000E-01* .1445     * .3054    *0.8452E-01* 3.613    *
*(task   3)*0.4000E-01* .2894     * .5881    *0.8129E-01* 7.235    *
*          *          *          *          *          *          *
* proc   5 *0.4000E-01* .5785     * 1.199    *0.8290E-01* 14.46    *
*(task   1)*0.4000E-01* .1445     * .3054    *0.8452E-01* 3.613    *
*(task   2)*0.4000E-01* .1445     * .3054    *0.8452E-01* 3.613    *
*(task   3)*0.4000E-01* .2894     * .5881    *0.8129E-01* 7.235    *
*          *          *          *          *          *          *
* proc   6 *0.4000E-01* .5785     * 1.199    *0.8290E-01* 14.46    *
*(task   1)*0.4000E-01* .1445     * .3054    *0.8452E-01* 3.613    *
*(task   2)*0.4000E-01* .1445     * .3054    *0.8452E-01* 3.613    *
*(task   3)*0.4000E-01* .2894     * .5881    *0.8129E-01* 7.235    *
*          *          *          *          *          *          *
*************************************************************************

definition of a new workload (yes or no)
no
```

7/ CONCLUSIONS

In a survey paper on the language development tools available on the UNIX System (Johnson 1980), S.C. Johnson from Bell Laboratories presents the following basic ideas on the concept of tool in computer science:

"The keys to useful tools are high-quality specialized knowledge and high-quality packaging. Without knowledge, tools are ponderous and unacceptably slow, offering little over hand coding. Without proper packaging, tools remain unused because they solve the wrong problem or are too difficult to apply to real situations.

In many times, theoretical insights and algorithms are essential to the success of a tool. Applying theory directly from textbooks, however, is likely to lead to problems... The tool builder's challenge is to make the theoretical insights apply to the 90 to 95 percent of the problems the model fits, without making the remaining 5 to 10 percent impossible to do at all."

Although formulated in a different context, these basic principles apply directly to the development of performance modelling tools as exemplified in the design of QNAP2 presented in this paper.

ACKNOWLEDGEMENT

The authors would like to acknowledge the major contributions of the following persons to the development of QNAP2: Marc Badel, Dominique Chandesris, Philippe Drix, Jean-Jacques Guillemaud, Raymond Marie, Didier Merle, Philippe de Rivet and William J. Stewart.

QNAP2: a Portable Environment for Queueing Systems Modelling

REFERENCES

(Drix 82)

P. Drix, "Algorithmes de Resolution de Reseaux de Files d'Attente Fermes BCMP et Heuristiques Associees", These d'Etat, Paris VI, (Nov. 1982).

(Gelenbe 77)

E. Gelenbe, G. Pujolle, "A Diffusion Model for Multiple Class Queueing Networks", Rapport de recherche IRIA 242, (1977).

(Heidelberger 80)

P. Heidelberger, P.D. Welch, "A Spectral Method of Confidence Interval Generation and Run Length Control in Simulation", CACM 24, 4, (April 1981).

(Heidelberger 82)

P. Heidelberger, K.S. Trivedi, "Queueing Network Models for Parallel Processing with Asynchronous Tasks", IEEEE Transactions on Computers C-31,(1982), pp. 1099-1108.

(Iglehart 78)

D.L. Iglehart, "The Regenerative Method for Simulation Analysis", in K.M. Chandy and R.T. Yeh, editors, Current Trends in Programming Methodology, Vol. III: Software Modelling and Its Impact on Performance, Prentice-Hall, (1978).

(Jacobson 82)

P.A. Jacobson, E.D. Lazowska, "Analysing Queueing Networks with Simultaneous Resource Possession", Comm. ACM 25, 2, (Feb. 1982), pp. 142-151.

(Jonhson 80)

S.C. Jonhson, "Language Development Tools on the UNIX System", Computer, 16-20.

(Lavenberg 77)

S.S. Lavenberg, C.H. Sauer, "Sequential Stopping Rules for the Regenerative Method of Simulation", IBM J. of Res. and Dev. 21, (Nov. 1977), pp. 545-558.

(Lavenberg 83)

S.S. Lavenberg (Editor), "Computer Performance Modelling Handbook", Academic Press, Inc., New-York, (1983).

(Marie 78)

R. Marie, "An Approximate Analytical Method for General Queueing Networks", IEEE Trans. on Software Engineering, Vol. SE-5, (May 1979), pp. 503-538.

(Marsan 83)

M. Ajmone Marsan, G. Balbo, C.Conte, F. Gregoretti, "Modelling Bus Contention and Memory Interference in a Multiprocesor System", IEEE Trans. on Comput., Vol. C-32, 1, (Jan. 1983), pp. 60-71,

(Merle 78)

D.Merle, D.Potier, M.Veran, "A Tool for Computer Performance Analysis", Performance of Computer Installations, Ferrari D. (editor), North-Holland (1978).

(Merle 79)

D.Merle, "Algorithmes de calcul des probabilites stationnaires de reseaux de files d'attente", Rapport de Recherche IRIA, No. 279, (1979).

(Neuse 81)

D. Neuse, K.M. Chandy, "SCAT: an Heuristic Algorithm for Queueing Networks of Computing Systems", ACM Sigmetrics 10, 1, (1981), pp. 59-79.

(Potier 82)

D. Potier, "Performance Modeling Tools", Proc. Int. Symp. on Applied Mathematics and Information Science, Kyoto University, (1982).

(Reiser 75)

M. Reiser, H. Kobayashi, "Recursive Algorithms for General Queueing Networks", IBM J. of Res. and. Dev., (May 1975), pp.283-294.

(Reiser 80)

M. Reiser, S. Lavenberg, "Mean-Value Analysis of Closed Multichain Queueing Networks", JACM 27, 2, (April 1980), pp. 313-322.

(Reiser 81)

M. Reiser, "Mean-Value Analysis and Convolution Method for Queue Dependent Servers in Closed Queueing Networks", Performance Evaluation Review 1, 1, (Jan. 1981).

QNAP2: a Portable Environment for Queueing Systems Modelling

(Saad 80)

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

(Sauer 81)

C.H. Sauer, "Approximate Solution of Queueing Networks with Simultaneous Resource Possession", IBM J. of Res. and Dev. 25, 6, (Nov. 1981), pp.894-903.

(Stewart 78)

W.J. Stewart, "A comparison of Numerical Techniques in Markov Modelling", CACM 21, 2, (Feb. 1978), pp. 144-152.

(Veran 83)

M. Veran, "Exact Analysis of a Priority Queue with Finite Source", Proceedings of the International Seminar on Modelling and Performance Evaluation Methodology, Paris (Jan. 1983). To appear in Lecture Notes in Control and Information Sciences, Vol. 60, Springer, (1984).

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique