A SOFTWARE PACKAGE FOR THE ANALYSIS OF GENERALIZED STOCHASTIC PETRI NET MODELS

Giovanni Chiola

Dipartimento di Elettronica
Politecnico di Torino, Italy

ABSTRACT

The paper describes a user-friendly, modular, portable software package for the construction and solution of generalized stochastic Petri net (GSPN) models. Two are the main components of the package: an interactive editor that simplifies the construction and the modification of a GSPN model, and a set of solution programs allowing the steady state and the transient analysis of the model. The editor capabilities are briefly outlined, and a list of the commands available to the user is presented. The most relevant characteristics of the solution programs are described, and an example of the use of the package in the analysis of a multiprocessor GSPN model is given.

1. INTRODUCTION

Generalized Stochastic Petri Nets (GSPN) [1] were developed as a tool for the specification and performance evaluation of computer systems at the Electronics Department of Politecnico di Torino and at the Computer Science Department of Universita' di Torino, in the frame of the Computer Science Program of the Italian National Research Council, MUMICRO project. The development of GSPN was stimulated by the results described in the Ph.D. thesis of M.K. Molloy [10], in which Petri nets with stochastic exponentially distributed transition firing times (SPN) were defined.

In order to improve the effectiveness of SPN, a new class of transitions was introduced in GSPN; these transitions fire in zero time, and are therefore named immediate transitions. The firing of immediate transitions takes place with priority over timed transitions. A solution algorithm that exploits the reduction of the size of the Reachability Set due to the presence of immediate transitions was described in [1]. Several computer programs were developed to implement the steady state numerical solution of GSPN models, leading to a first software package for their analysis [5]. However the primary purpose of these first programs was the acquisition of insight into the memory and CPU time requirements of the solution algorithms as function of the size of "significant" GSPN models, and little attention was paid to the portability and flexibility of the tool as well as to the model definition facilities.

Experience was subsequently gained on the modeling power of GSPNs in different environments and with different goals of the analysis (communication protocol analysis [11], study of the performance degradation due to software synchronization [2], fault tolerance and reliability analysis [3], proof of the existence of product-form solution of subsystems [4]). It was thus decided to develop a new software tool whose aim is to be user friendly, portable on different machines, easily upgraded, and both time and space efficient. To achieve the goal of portability we chose to adopt the language Pascal, using standard features whenever possible. The package is highly modular, in order to be able to easily adapt to different uses, and to be easily upgraded as soon as new theoretical results provide new analysis algorithms. Indeed we were subsequently able to include in the package a new algorithm for the analysis of a class of models containing some deterministic timed transitions [7]. The GSPN analysis programs focus on exact analytical solution techniques only, but approximations can be easily introduced either in the solution algorithms or at the Net level; the latter method can usually be implemented by simply avoiding the definition of too much behaviour details. Montecarlo simulation can be usefull in the performance estimation of analytically untractable models and a simulation program has also been embedded in the package.

The package is available to Institutions for research purposes through the author. Any comment, improvement or suggestion will be welcome.

2. PACKAGE ORGANIZATION

Many logically separate steps can be identified in the use of GSPN models. Figure 1 depicts a GSPN representing a typical modeling process. Among the four timed transitions representing the steps of the modeling activity, the one labeled "solve" (i.e. the Reachability Graph construction and the solution of the Markov process), requires a computer package but no human analyst action; on the other hand, the transition labeled "check" (the results) requires some analyst work but can hardly be assisted by a computer; finally, both the construction and the update of the GSPN model involve human activity but can be highly simplified by the availability of a suitable computer tool.

Therefore we recognized the need for two main software utilities to make effective the use of GSPN models:

(1) an interactive tool to help the user to define and update the description of all the relevant model specifications, i.e. the GSPN itself, togheter with its parameters and the performance indices to be computed;

(2) an automated tool to implement the solution, i.e. the derivation of a Markov or semiMarkov chain from the graphical model and the computation of either its equilibrium or its transient performance figures.

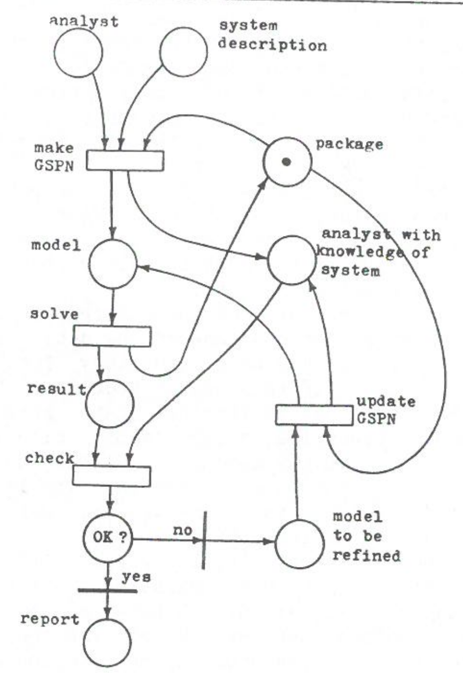

Figure 1: A typical modeling process using GSPNs.

2.1. Model definition

In principle the definition of a GSPN model can be performed in many different ways. For example, the user can write the net definition (in some language) on a file to be used as input by the analysis program. Alternatively, a first section of the analysis program itself can interact with the user and ask for all the parameters needed to specify the model. The second choice is certainly much more user friendly, and it can be easily exploited also by inexperienced users. On the other hand, either for the definition of large nets or for repeated analysis of similar models, the first solution may be preferable.

We tried to achieve the advantages of both approches by storing the model description into a file and using a semantics guided editor to create and update it. In this way the user only deals with the GSPN entities (places, transitions, etc.) and the editor provides both a tutorial guide for inexperienced users, and a powerful and flexible command set for adding, deleting and changing any entity of a previously defined net. Indeed, very often the definition of a correct model requires debugging and/or refining the initial GSPN description [6], just as in the case of software development.

In order to simplify the sequential analysis of topologically identical GSPNs with different firing rates and/or initial marking, we introduced the concept of parameters. Two types of net parametrization were considered: one for the initial marking, and the other for transition firing rates. Marking and rate type parameters may be defined in the description of the GSPN, and then referenced by name during the definition of the initial marking and of transition firing rates.

The modeling of systems with GSPNs often requires the definition of marking dependent transition firing rates. A simple mechanism of "enabling dependence" is automatically provided; a transition may be defined to be either single server or multiple server. In the former case the transition firing rate is assumed to be marking independent, while in the latter case the firing rate defined by the user is multiplied by the minimum between the number of "servers" modeled by the transition and the maximum number of consecutive possible firings of the transition starting from the current marking. The definition of more complex marking dependencies is accomplished by writing a definition statement of the form outlined in Table 1. The logical conditions are sequentially tested until either a true one or the "ever" clause is found, then the corresponding real expression is evaluated. A real expression is any algebraic expression containing parameters, numbers and markings.

Furthermore, in the construction of a GSPN model it is also possible to define the performance indices that must be automatically computed in the model analysis step. This can be obtained using result definition statements similar to the above described marking dependent rate definition statements, in which the user can define and use the probability and/or the expected value of given marking conditions.

2.2. GSPN Analysis

In the present version of the software package the analysis of the GSPN model is subdivided in many logical steps, each one separately implemented with a program. The programs communicate with each other through the extensive use of intermediate result files that are automatically managed by the modules.

An effort is made to keep all the information relevant to the user into the net description files generated by the guided editor, so that the whole solution phase can be run in batch mode. In this way the editor represents the only user-package interface.

TABLE 1 : Structure of the definition statement for marking dependent transition rates.

```
when <logical_cond_1_on_marking> : <real_expr_1>;
  .
  .
  .
when <logical_cond_N_on_marking> : <real_expr_N>;

ever  <real_expression_N+1> ;
```

At present three versions of the package exist, running on a VAX 11/780 under VMS operating system, on a VAX 11/780 under Berkeley UNIX 4.1 operating system, and on a SUN workstation under Berkeley UNIX 4.2 operating system. In the VAX VMS version the programs are executed under the control of a command (DCL) procedure, which provides a minimal user interface (through a very simple dialogue) to perform the standard steady state or transient numerical solution of GSPN models. Similar command procedures can be easily and quickly written to accomplish more sophisticated objectives (e.g. multiple runs with varying parameters or Reachability Graph analysis) by just running the modules in the right order (and possibly writing some more programs which elaborate the informations stored in the intermediate files as desired). The UNIX versions rely on the "make" system utility to keep the result files consistent with the net description files through the running of the proper package modules. The execution of the make program is performed by the user through a Cshell "alias" definition, so that the same user interface of the VMS version is provided (i.e. command "GED netname" to edit and command "GSO netname" to solve a GSPN).

3. THE EDITOR

GED is an interactive Semantics-oriented editor for GSPN description files. Each net description is uniquely identified with a name, consisting of an alphabetic character possibly followed by up to eight alphanumeric characters. The program automatically handles two files (netname.NET and netname.DEF) containing respectively the whole net description and the possible definition of complex marking-dependent transition rates and probabilities as well as the expression of the desired output results.

Marking dependent firing rates and output result definitions are implemented using a simple context-free formal language allowing the definition of logical expressions on GSPN markings and the evaluation of algebraic expressions. These specifications are then automatically translated into Pascal external functions, compiled and linked with the proper solution programs. In this way the higher efficiency of compiled functions over interpreted functions is achieved, while maintaining the control of the semantic editor on the whole model definition. Unfortunately, although almost all Pascal compilers provide facilities for the linkage of separately compiled procedures, no standard exists for this purpose, so that the pattern files used by the translation programs to prepare the external Pascal procedures have probably to be modified in order to run the package under other operating systems.

The program does not handle multiple versions of the same net, i.e when a net description is updated, the previous version is lost; if the user wishes to save several versions of the same net, he should do it explicitly by giving different net names. Since also file handling is not standard for Pascal language, the part of the program implementing the net name management (about twenty code lines in each program) is strictly machine-dependent.

3.1. Main Features

The editor has two major operating modes, namely the Input Mode for the creation of a new net description, and the Command Mode for the correction and update of previously defined GSPNs. In Command Mode two editor parameters control the level of "friendlyness" of the user interface: when the Verbose Mode is set, the editor issues a detailed menu instead of a brief promt for any required input; the Autoshow Mode is used to automatically perform a "show" command of the current value of the GSPN objects before and after any update affecting them.

At the beginning of the editing session, the user is asked for the name of the net. The program then looks into the user directory for the file netname.NET ; if this search succeeds, the file is opened, read into an internal data structure and closed. Otherwise a new (empty) file is created and closed, and the message CREATING A NEW GSPN appears on the terminal screen. The last netname given by the user is assumed as the default value for the rest of the editing session.

The editor may also be explicitly reinitialized on a (previously existing) GSPN through the command "e" (edit); this feature can be exploited to edit several nets without exiting the editor and to restart from the previous net version when some erroneous command has seriously damaged the current net description.

Indeed, all updates made to the net during the editing session become executive only after a "w" (write) command is executed. Both the e and w commands require the specification of the name of the net to be edited or written.

When the editor is called on a new net, it enters the Input Mode to allow the user to define an initial net description. After initialization (or if the editor has been called on a previously existing net), the editor enters the command mode. A very simple regular grammar is used for the command recognition. Each object of the net (place, timed transition, immediate transition, etc.) can be referred to by name or by ordinal number.

A symbol table is used to perform semantic checks during all editing phases; to this purpose the object names must be unique and objects must be defined before being referenced.

There are two classes of editor commands:
- to control the net editing

q	quit the editing
e	edit a net
w	write a net
wq	write a net and quit the editing

(the latter should be the normal way of terminating an editing session)
- to update the current net description

a	add net entities;
c	change net entities;
d	delete net entities;
s	show net entities;

the commands of the second class can be applied to one or more objects or just to some object attribute.

Commands can either be issued complete in all their fields or set-up by subsequent refinement steps. A command becomes executive only when it is recognized to be defined in all its allowed fields or when the Exclamation terminator "!" is given. On the other hand, if either the command is not fully refined or the Semicolon terminator ";" is given, the command is considered as an intermediate step of a multilevel command. Exploiting this facility it is possible to taylor sequences of commands having some minimal common denominator.

The terminal input is always under the control of a Monitor that provides a Software Interrupt handling. The monitor commands are implemented as Software Interrupts transparent to the normal behaviour of the editor. They can be classified into three groups:
- setup commands
- help requests
- control commands.

Using the setup commands it is possible to set and reset the "Autoshow editing mode", i.e. the automatic printing of the current value of any object (or object attribute) before and after any update, and to set and reset the "Verbose editing mode", i.e. to tell the editor whether the user desires to read either a full menu at each editing step or just some prompt. By default these editor facilities are initially set.

The help request commands are of three kinds: the "$" command displays some general purpose help messages, the ">" command shows the current state of the command stack, and the "?" command provides the menu during the command refinement phase. Note that a "?" command is automatically performed before any command refinement request if the verbose editing mode is set.

Using control commands it is possible, for example, to skip (command "<") and to abort ("\") the current editor command. A command skipped or aborted before completion has no effect on the current net description (i.e. commands are actually executed only after the last required input is terminated with a RETURN).

4. THE SOLUTION MODULES

In the current version of the package the following programs are implemented:

GRG	construction of the Reachability Graph;
GMD	translation of the definition of marking dependent firing rates;
GMT	construction of the Markov chain Transition Matrix;
GGE	steady state solution with a Gauss Elimination algorithm;
GGS	steady state solution with a Gauss Seidel algorithm;
GTR	transient solution with a matrix exponentiation algorithm;
GRE	translation of the definitions of the desired result statistics;
GST	computation of the result statistics;
GPR	preparation of a standard output file report;
EMT	construction of the semiMarkov Transition Matrices for nets containing deterministic timed transitions;
EIM	construction of Embedded Markov chain for nets containing deterministic timed transitions;
EST	steady state solution with a Gauss Seidel algorithm for nets containing deterministic timed transitions;
SIM	Montecarlo simulation.

In Figure 2 the relationship among modules is depicted using a non marked Petri-net-like graph: each module is represented with a transition, and each place represents a file containing some piece of information about the GSPN model; an input arc of a transition means that the module reads the information contained in the file represented by the place, while an output arc of a transition means that the corresponding module modifies the information contained in the file. All programs are run automatically under the control of a command procedure, which takes care of the management of all intermediate files, so that the user has only to issue the command: "GSO netname" in order to perform a GSPN analysis.

The Reachability Graph expansion is performed by program GRG, which reads the net description (file NET) and produces three result files (TRS, VRS and RGR), containing respectively the Tangible and Vanishing Markings of the GSPN Reachability

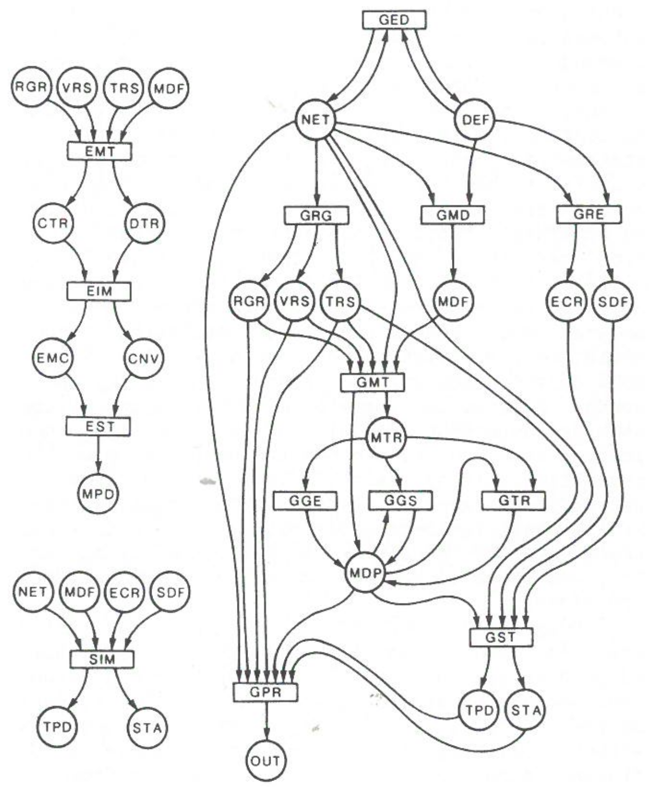

Figure 2
Relationship among the modules of the package.

Set, and the symbolic marking transition description, i.e. the label of each enabled transition for every marking, togheter with the label of the next marking determined by their firing. The algorithm begins by putting the initial marking into the Reachability Set, then all enabled transitions in newly found markings are fired. Timed transition firing proceeds using a breadth-first policy, while immediate transition paths are followed depth-first until a tangible marking is reached. The ordering of the markings in the Reachability Set resulting from this firing policy is exploited by the following modules of the package to implement a very efficient reduction of the Vanishing Markings in the case that no zero-time loop is present [1]. In the program data structure, markings are lexicographically ordered by means of a balanced binary tree in order to improve the efficiency of the search procedures. In the VAX VMS version of the program the careful choice of the data structure allows the analysis of GSPN models with up to 16385 markings using about 1 Mbyte of memory. The analysis of models with fairly large Reachability Set takes about ten millisecond of CPU time per Marking on a VAX 11/780.

In order to obtain the Markov chain state transition matrix, the following procedure is automatically implemented (through DCL or Cshell commands). First, program GMD is run to convert the marking dependent transition firing rate definitions. It uses a recursive descendent algorithm to perform the syntactic analysis of the definition statements, and, using a pattern description file, translates them into a sequence of if-then-else Pascal statements (one for each logical condition on markings), putting them into an external function definition module. Then the system Pascal compiler is invoked on the resulting file. Since both syntax and semantic analysis are performed by program GMD, the only kind of compilation errors could be a conflict between the definition of some GSPN object names and the predefined Pascal keywords. Program GMT togheter with the object of the marking dependent definition function is then loaded and run to perform the computation of the transition firing rates. This module uses a depth-first algorithm to follow the immediate transition tangible-to-tangible paths [1], and keeps trace of the resulting probabilities of already followed paths to achieve a higher computational efficiency.

Programs GGE and GGS are implemented using standard sparse matrix computation algorithms, adapted to the solution of a set of linearly dependent equations augmented with the probability normalization to 1. The Gauss elimination proceeds without pivoting to preserve the sparse band-diagonal structure of the matrix resulting from the breadth-first firing of the Reachability Graph. Indeed, row or column permutation would determine a large fill-in during the elimination phase, thus making impractical the analysis of large matrices. Despite this algorithm simplification we observed a surprisingly good numerical stability; this is probably due to the fact that the diagonal elements used as pivot, actually are (by definition) partial pivots, since they are made equal to the sum of all the other elements in the row.

The transient solution of the Markov chain is accomplished by program GTR, using a matrix exponentiation algorithm. The major problem in this case is a "good" choice of the time integration step: large integration steps, depending on the matrix eigenvalues, can result in large round-off errors and poor numerical stability; on the other hand, too small steps may involve unnecessary row by column matrix products. In our program an initial estimate of the optimal integration step is made according to the maximum transition rate found in the matrix; then the step is dynamically adjusted in order to keep it as large as possible, without incurring in too large round-off errors during vector additions.

In some particular case, the restriction on the memoryless probability distribution of timed transition can be dropped, and an Embedded Markov Chain (EMC) technique (similar to that used in [9] to study the M/D/1 queueing system) can be used to exactly compute the steady state solution for nets containing deterministic transitions. In the cases of exclusive and competitive [8] transitions a constant firing delay can be used freely. In the case of concurrent transitions, all markings must enable at most one deterministic transition. Under the above restrictions, a discrete-time Markov chain can be defined at the instants of deterministic transition firings. When either no deterministic transition or an exclusive deterministic transition is enabled in a given marking M_i, this marking trivially maps into a state S_i of the EMC. In the case of markings enabling several competitive transitions, among the deterministic ones only that with the shortest firing time can fire, in race with the possibly enabled exponential transitions; the GSPN marking maps onto a state of the EMC, and both the transition probabilities to other states and the mean sojourn time in the state can be computed. When a deterministic transition t_d is enabled concurrently to exponential transitions, the next state of the EMC is sampled only at the instant of firing of t_d, i.e. the EMC state evolution does not account for the marking changes due to the firings of exponential transitions during the enabling interval τ_d of t_d, but "delays" those possible state changes to the instant of firing of the deterministic transition. The state transition probability of the EMC is computed using the Chapman-Kolmogorov equation as:

$$\exp(\tau_d\, Q')\, D(t_d)$$

where Q' is the infinitesimal generator of the exponentially distributed part of the process, and $D(t_d)$ is the state change matrix corresponding to the firing of transition t_d. From the steady state solution of the EMC, it is possible to compute the steady state probability distribution of the GSPN, by weighting the state probabilities with the mean sojourn times, and converting the probabilities of states enabling concurrent deterministic transitions with the conversion matrix:

$$C(t_d) = \int_0^{\tau_d} \theta \exp(\theta\, Q')\, d\theta$$

which accounts for the correspondence between marking sojourn times in the GSPN and state sojourn times in the EMC. Both the EMC transition probability matrix and the probability conversion matrix can be efficiently computed using a truncated Taylor series expansion of matrix exponential.

The GSPN Montecarlo simulation program uses a "Natural Regeneration" mechanism [12] to provide point estimates of the average number of tokens in each place together with their confidence intervals, as well as to collect the user defined statistical results. Machine independent congruential pseudorandom sequence generators are used to implement stochastic transition timings. This program provides a bypass of the other solution modules, and can be effectively exploited to obtain results in the cases of analytically untractable models.

5. EXAMPLE

As an example of the use of our package consider the GSPN depicted in Figure 3. This net models the contention for physical resources in a multiple bus multiprocessor system; its semantics and behaviour are thoroughly described in [1]. Table 2 reports the listing of the GSPN description provided by a "show" editor command issued after the net definition. Table 3 contains the listing of the results of the steady state analysis of the GSPN as provided by program GPR.

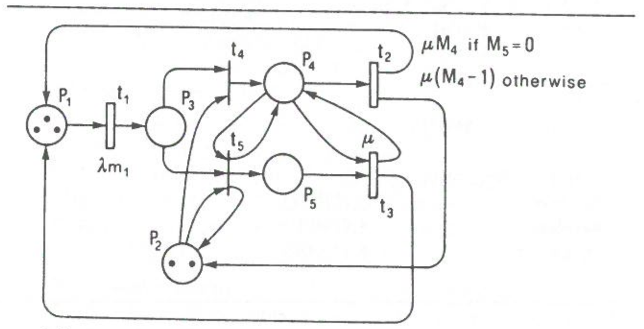

Figure 3: GSPN model of a 2-bus 3-processor 4-memory multiple bus multiprocessor system.

TABLE 2 : GSPN description.

```
***   START OF GSPN EDITING   ***

The prompt character ">" means that some user input is expected.
At any time you can obtain help by issuing the "$" command.
All commands have to be terminated by hitting the RETURN key.

Comment on this GSPN :

This is a GSPN model of a 2-bus P-processor M-memory
Multiple Bus Multiprocessor System.

 ...hit RETURN for more >

You can use the following commands :
a  : add object(s);
c  : change object(s);
d  : delete object(s);
q  : quit the editing session;
q! : quit without writing the GSPN;
e" or eNEWNET : reinitialize the editing on previous or NEWNET GSPN;
s  : show object(s);
 w" or wNEWNET   : write the previous or NEWNET GSPN description;
wq" or wqNEWNET : write-quit the previous or NEWNET GSPN description.

Objects are places, transitions, parameters, groups of immediate trans.

Hit "??" if you wish to see the following refinement menu.

command > s!

Executing command : show


MARK PARAMETER   1  Name: P   Value:    3

MARK PARAMETER   2  Name: M   Value:    4

REAL PARAMETER   1  Name: lambda   Value:  1.00000000000000e-01

REAL PARAMETER   2  Name: mu       Value:  1.00000000000000e+00


PLACE   1   Name: Active     Initial Marking: P
Input Bag :                    EndAccess      EndM1withQue
Output Bag:                    Memrequest

PLACE   2   Name: IdleBus     Initial Marking:   2 tokens
Input Bag :                    EndAccess      queueup
Output Bag:                    access      queueup

PLACE   3   Name: BusQueue    Initial Marking: ...empty...
Input Bag :                    Memrequest
Output Bag:                    access      queueup

PLACE   4   Name: Accessing    Initial Marking: ...empty...
Input Bag :                    EndM1withQue      access      queueup
Output Bag:                    EndAccess      EndM1withQue  queueup

PLACE   5   Name: M1Queue    Initial Marking: ...empty...
Input Bag :                    queueup
Output Bag:                    EndM1withQue


TIMED TRANSITION   1  Name: Memrequest
Enabling-Dependence: Infinite-Server   Rate : lambda
Input Bag :                    Active
Output Bag:                    BusQueue

TIMED TRANSITION   2  Name: EndAccess
Enabling-Dependence: Single-Server   Rate : ...mark-dep...
 when #M1Queue = 0 : mu * #Accessing ; ever mu*(#Accessing -1);
Input Bag :                    Accessing
Output Bag:                    Active      IdleBus

TIMED TRANSITION   3  Name: EndM1withQue
Enabling-Dependence: Single-Server   Rate : mu
Input Bag :                    Accessing      M1Queue
Output Bag:                    Active      Accessing


GROUP   1   Name: seizeBus   is :

IMMEDIATE TRANSITION   4  Name: access   Group:seizeBus
Enabling-Dependence: Single-Server   Probability: ...mark-dep...
 ever (M-1)/M ;
Input Bag :                    IdleBus      BusQueue
Output Bag:                    Accessing

IMMEDIATE TRANSITION   5  Name: queueup   Group:seizeBus
Enabling-Dependence: Single-Server   Probability: ...mark-dep...
```

```
  ever 1/M;
Input Bag :                     IdleBus    BusQueue   Accessing
Output Bag:                     IdleBus    Accessing  M1Queue

RESULT DEFINITIONS :
ProcPower  : E( #Active );
MeanQueue  : E( #BusQueue ) + E( #M1Queue );
Throughput : mu P( #Accessing > 0) + mu P( #Accessing > 1);

 ...hit RETURN for more >
```

Table 3 : GSPN Results.

```
Comment on this GSPN :

This is a GSPN model of a 2-bus P-processor M-memory
Multiple Bus Multiprocessor System.


The Reachability Set contains  9 Tangible and  5 Vanishing Markings.

Do you wish to print the Reachability Graph ?

Hit "n" for no;
    "t" for Tangible Markings of the GSPN Reachability Set;
    "m" for both Tangible and Vanishing Markings of Reachability Set;
    "a" for both Reachability Set and whole Reachability Graph;
    "r" for  the reduced GSPN Tangible Reachability Graph;
    "i" for reduced Tangible Reachability Graph of the Inverse GSPN.
> r

GSPN Tangible Reachability Graph :

Mark #   1 :    3   2   0   0   0
  to M   2 path :   3*t  1 ->     t  4

Mark #   2 :    2   1   0   1   0
  to M   4 path :   2*t  1 ->     t  4
  to M   3 path :   2*t  1 ->     t  5
  to M   1 path :     t  2

Mark #   3 :    1   1   0   1   1
  to M   7 path :     t  1 ->     t  4
  to M   6 path :     t  1 ->     t  5
  to M   5 path :     t  2
  to M   2 path :     t  3

Mark #   4 :    1   0   0   2   0
  to M   8 path :     t  1
  to M   2 path :     t  2

Mark #   5 :    2   2   0   0   1
  to M   3 path :   2*t  1 ->     t  4

Mark #   6 :    0   1   0   1   2
  to M   9 path :     t  2
  to M   3 path :     t  3

Mark #   7 :    0   0   0   2   1
  to M   3 path :     t  2
  to M   4 path :     t  3

Mark #   8 :    0   0   1   2   0
  to M   4 path :     t  2 ->     t  4
  to M   3 path :     t  2 ->     t  5

Mark #   9 :    1   2   0   0   2
  to M   6 path :     t  1 ->     t  4


Do you wish to print the steady-state probabilities ?
The result accuracy is 5.0e-06
Hit "n" for no;
    "a" for all Tangible Markings probabilities;
    "t" for the token probability  distribution in each place;
    "m" for only mean number of token in each place.
> t

Steady-state Probability Distributions of tokens in places:

# token   p  1  name: Active              p  2  name: IdleBus
  0          0.001540                        0.018056
  1          0.027994                        0.235431
  2          0.223954                        0.746512
  3          0.746512
average =    2.715439                        1.728456

# token   p  3  name: BusQueue            p  4  name: Accessing
  0          0.999160                        0.746512
  1          0.000840                        0.235431
  2                                          0.018056
  3
average =    0.000840                        0.271544

# token   p  5  name: M1Queue
  0          0.988102
  1          0.011618
  2          0.000280
  3
average =    0.012177

Additional Statistics :
ProcPower        =        2.71543881
MeanQueue        =        0.01301731
Throughput       =        0.27154388
```

6. FUTURE EXTENSIONS

Although the software package in its present version already represents a very useful tool for the definition and for the analysis of GSPN models, several extensions could further expand its capabilities. Two main additions to the package are being considered: a graphic interface and an algorithm for the hierarchical construction and analysis of GSPN models.

The availability of a graphic interface would complete the interactive GSPN editor by allowing the GSPN topology to be defined and represented in graphical form, and to be stored in a form that is suitable to successively retrieve both the graphical representation and the list of the connections. The graphical definition of the GSPN topology would greatly reduce the number of errors, since the user has only to reproduce on the screen the desired GSPN graph, and hence has complete and immediate control on the model while it is being constructed. The other parameters of the model can then be input in alphanumeric form simply by answering the editor questions about the characteristics of each place and transition. A drawback of the graphic interface is its very limited portability. At present the implementation on a SUN workstation is being

studied.

A tool for the hierarchical definition of GSPN models appears to be very useful, especially when dealing with large nets. Indeed, in many cases a large GSPN is constructed by repeating several times in the model identical (or similar) subnets. It would then be very convenient to define the subnet once and then use it as a building block when assembling the entire GSPN. In the model solution phase, a hierarchical procedure can also be exploited by separately generating the subnet Reachability Graphs and the Reachability Graph of the GSPN model in which each subnet is represented by either a place or a transition. This reduces the complexity of the computation necessary to obtain the resulting complete Reachability Graph, while still allowing an exact model solution. Moreover, the existence of subnets can also be used to obtain with little computational effort approximate solutions of the model.

REFERENCES

[1] M. Ajmone Marsan, G. Balbo and G. Conte "A Class of Generalized Stochastic Petri Nets for the Performance Analysis of Multiprocessor Systems", ACM TOCS, Vol. 2, n.2, (May 1984).

[2] M. Ajmone Marsan, G. Chiola and G. Conte "Performance Models of Task Synchronization in Computer Systems", 1st International Conference on Computers and Applications, Beijing, China, (June 1984).

[3] M. Ajmone Marsan, A. Bobbio, G. Conte and A. Cumani "Performance Analysis of Degradable Multiprocessor Systems using Generalized Stochastic Petri Nets", IEEE Computer Society, Distributed Processing T C Newsletters, 6, SI-1 (1984).

[4] M. Ajmone Marsan, G. Balbo, G. Chiola and S. Donatelli "On the Product Form Solution of a Class of Multiple Bus Multiprocessor System Models", International Workshop on Modeling and Performance Evaluation of Parallel Systems, Grenoble, France, (December 1984).

[5] M. Ajmone Marsan, G. Balbo, G. Ciardo and G. Conte "A Software Tool for the Automatic Analysis of Generalized Stochastic Petri Net Models", 1st International Conference on Modeling Techniques and Tools for Performance Analysis, Paris, France, (May 1984).

[6] M. Ajmone Marsan and G. Chiola "Construction of Generalized Stochastic Petri Net Models of Bus Oriented Multiprocessor Systems by Stepwise Refinements: a case study", 2nd International Conference on Modeling Techniques and Tools for Performance Analysis, Sophia Antipolis, France, (June 1985).

[7] M. Ajmone Marsan and G. Chiola "Modeling Discrete Event Systems with Stochastic Petri Nets", ISCAS 85, Kyoto, Japan, (June 1985).

[8] J.B. Dugan, K.S. Trivedi, R.M. Geist and V.F. Nicola "Extended Stochastic Petri Nets: Applications and Analysis", Performance 84, Paris, France, (December 1984).

[9] L. Kleinrock "Queueing Systems; vol.1 : Theory", John Wiley, (1975).

[10] M.K. Molloy "On the Integration of Delay and Throughput Measures in Distributed Processing Models", Ph.D. Thesis, UCLA, 1981.

[11] M.K. Molloy "Performance Analysis Using Stochastic Petri Nets", IEEE Transactions on Computers, Vol. C-31, pp.913-917, (September 1982).

[12] S.S. Lavenberg "Statistical Analysis of Simulation Outputs", IBM Res. Report (1980).