Proceedings of the 1989 Winter Simulation Conference
E.A. MacNair, K.J. Musselman, P. Heidelberger (eds.)

STOCHASTIC PETRI NET SIMULATION

Gianfranco Balbo
Giovanni Chiola
Dipartimento di Informatica
Universita` di Torino
Corso Svizzera, 185
10149 Torino, Italy

ABSTRACT

Petri nets are a formal modelling tool very well suited to the description of distributed and concurrent systems which exhibit synchronization and contention for shared resources. Adding random temporal specifications to Petri nets, stochastic Petri nets are obtained from which event driven simulators can be automatically constructed. This paper discusses the double role that stochastic Petri net simulation plays as an aid in the debugging of the correctness of the model and as a vehicle for the evaluation of the efficiency of the system. A modelling tool that has been developed for these purposes is illustrated with an emphasis on the modelling environment it provides and on its internal software architecture.

1. INTRODUCTION

Petri nets [Pet81, Rei85] are gaining a widespread acceptance as a formalism that is well suited for the description and the qualitative analysis of complex systems. Distributed and concurrent systems which exhibit synchronization and contention for shared resources are conveniently studied with the help of Petri net models (see [Ajm87] for extensive references).

High level Petri nets [Gen81, Jen81], that are generalizations of ordinary Petri nets, have both the power and the flexibility for representing real complex systems. Timed Petri nets derive from adding temporal specifications to (some of) the elements of Petri net based models. Timed Petri nets are useful for the quantitative (performance) analysis of real systems.

Stochastic Petri Nets (SPN) [Mol82, Flo85] and their generalizations (GSPN) [Ajm84, Ajm87a] are timed Petri nets in which the time specifications are introduced by associating exponentially distributed firing times with timed transitions. Methods exist for the automatic translations of SPN (as we will call in this paper both stochastic and generalized stochastic Petri nets) into their underlying stochastic processes. SPN can thus be analyzed with classical techniques for studying the performance characteristics of systems. SPN have the representation power that allows the precise description of the phenomena that are important during the design stage, but the complexity of many distributed systems often yields the construction of models that are very difficult to analyze. In these cases simulation represents the method of choice for deriving the desired performance measures.

The simulation of complex SPN models plays the double role of an aid in assessing their correctness, and of a vehicle for the evaluation of their efficiency. Stepping through the state space of a model complements the results of a structural analysis of the net to verify its correctness. The animation of the evolution of the state of a model during the early part of a simulation experiment helps in discovering specific functioning aspects and patterns that may outline desired or undesired features of the model, and that are difficult to capture with usual performance indexes that aggregate results of measurements performed during the simulation experiment. Finally, estimates obtained from the simulation of an extensively tested and well understood model provide reliable indications about the performance of the system under study.

In this paper, a software tool, called *GreatSPN* is described that has been developed for the analysis of SPN, and that is characterized by a *graphical editor* for the construction and the manipulation of SPN models, a *structural analyzer* for the computation of basic (structural) properties of these models, a *builder* for the automatic construction of an efficient event-driven Montecarlo simulation program (called *simulation engine*) and a *controller* that monitors the execution of simulation experiments providing the possibility of an on-line displaying of the model state evolution directly on the SPN graphical representation. *GreatSPN* integrates visual and interaction features in a general environment where the same model can be used in different studies concerning its validation and its numerical evaluation. In *GreatSPN* the computation of the structural properties of the net are exploited to construct an optimized event driven simulator.

1.1. Related Work

Simulation is often used in the Petri net field as a method for validating models whose complexity is such that their correctness cannot be established using formal methods based on the computation of their structural properties [Mar81]. In this context the word simulation however means a walk through the state space of the net without taking into account any timing specification. The evolution of the net is thus driven by the precedence constraints among transitions dictated by the structure of the net and no scheduling of events occurs. For this type of analysis, distributed simulation is becoming prevalent. Processes are associated with the nodes of the net, and assigned to dedicated processors. Processes are then let to compete to determine the movement of tokens and conflicts are solved by the scheduler of the underlying distributed operating system (e.g., [Tau87, But89]). None of the problems that are crucial in distributed discrete event simulation [Cha81, Jef87] are thus considered in these works.

The use of Petri net based models for performance evaluation is relatively new, and thus little experience has been accumulated on the topic of timed Petri net simulation. Pioneering work was done by Noe and Nutt [Noe73], who identified the possibility of deriving simulation models directly from the description of systems made using the E-net formalism. No indications were however provided by these authors on how to implement such a translation. The advent of powerful graphical workstations has made the development of software packages for the construction and the analysis of timed Petri net models a feasible task. It is thus increasingly important devising efficient techniques for the simulation of timed Petri nets that tend to be extremely complex as it becomes relatively easy to construct large models. Indeed, testing in a naive manner which transition is enabled in a given marking to perform a step in the evolution of the net can be extremely expensive in such large models. Some work on the choice of a proper data structure for the representation of the net has been reported in [Col86] where the simulation approach is still an interpretation of the net.

1.2. Visual and Interactive Simulation

The specification of a model using a net based formalism has the advantage of an explicit representation of the relations among the different nodes of the net (interconnections) and of the activities that are currently taking place (tokens or markers). Moreover the dynamics of the model is completely represented by the movement of such tokens. Advanced graphics allows to exploit these advantages recognizing their importance in the description of a model and in the understanding of its behaviour.

Visual simulation is a method that exploits the graphical description of the model to show the dynamic evolution that takes place during its simulation directly on the graphical representation. Static graphical facilities are used to describe the topology of the model, while dynamic graphical facilities are used to visualize the movement of the tokens, to provide snapshots on the state of the model, and to represent the evolution of computed (or measured) statistics. Visual simulation is very useful during the early stage of the construction of a model since it represents a powerful tool for its debugging. Moreover, the animation of a correct simulation model may provide important insights on the behaviour of the actual system, especially because of the possibility of highligthing particular evolution patterns that may be crucial for the good (or bad) behaviour of the system, but difficult to deduce from average performance indexes obtained with standard simulation methods.

Interactive simulation is a natural complement of visual simulation as it allows the user to inspect the state of the model when certain conditions are met. This feature can be obtained by defining the conditions directly on the graphical representation of the model (e.g. defining a target distribution of tokens on the net), and then stopping the animation when such conditions are met. When the simulation stops, the performance measures collected up to that moment, can be inspected to understand the behaviour of the model. Modification of certain parts of the model can also be introduced to correct errors.

Several software packages exist that provide both visual and interactive simulation. Most of these modelling tools assume a queueing network formalism for the specification of the model. PAW [Mel85] is among the first packages that provided these simulation features.

Few are instead the software packages that provide visual and interactive simulation for Petri net based models. The most interesting one is OLYMPUS [Nut89] which is a distributed modelling system where several users can simultaneously construct and simulate timed Petri net models. A client-server approach is used to allow a concurrent construction, editing, and simulation of the same model. This feature is included to allow a cooperative effort to take place for the modelling of large systems.

2. MODELLING

GreatSPN [Chi87] provides a very powerful and versatile interface based on the capabilities of high-power graphical workstations. *GreatSPN* is a window-based, object-oriented system, in which elements typical of Petri net models (places, transitions, and arcs) are manipulated under the assistance of basic syntactical rules that prevent the construction of (syntactically) incorrect models. The model drawn using this interface is internally translated into several files that are however completely transparent to the user. The graphical representation is thus the object to which all the commands provided by the interface apply. Having specified the model, it can be saved into a data-base from which it can be retrieved for any subsequent use.

The simulation of a SPN model is performed in *GreatSPN* by first constructing the model on the screen of a graphical workstation using the facilities provided by the graphical interface. Once that the simulation model has been specified, a debugging phase must take place to insure that it is first syntactically, and then semantically correct. Finally, the actual simulation is performed in which measurements on the behaviour of the model are collected and statistically analyzed to obtain estimates for the performance indexes that are relevant for the study. All these three phases are extensively supported by *GreatSPN* that provides an integrated environment for performing complex simulation experiments.

2.1. Graphical facilities

Constructing SPN models consists of drawing places and transitions connected by arcs. This is made possible by *GreatSPN* that provides a *drawing board* where graph nodes selected from a menu of basic objects can be deposited. To facilitate the drawing of (graphically) well organized models, a grid can be superimposed on the drawing board so that new elements are positioned on its corners (see Figure 1). Arcs that can be of input, output, and inhibition type connect nodes. They are drawn selecting first the origin node, and subsequently the destination node. Arcs behave as rubber-bands and can look like segmented lines through the use of intermediate points. Objects deposited on the drawing board can be repositioned by dragging them around with usual point-and-select methods. Arcs stick to the origin and destination nodes so that moving nodes affects connecting arcs as well. To make the model look nicer, segmented arcs can be turned into continuous (curved) lines by means of a *spline* option that smooths their angles (see Figure 2).

Groups of elements (subnets) can be graphically manipulated as a single object using a multiple selection option which then allows to move, rotate, and duplicate all the elements of a group. These operations on global objects preserve the relative positions of the component elements. Selected groups of elements can also be removed from the drawing with a single operation. To protect from destructive actions, a clipboard facility is provided, and a relatively short history of commands (and of their effects) is maintained so that a recovery can occur during the manipulation of a drawing.

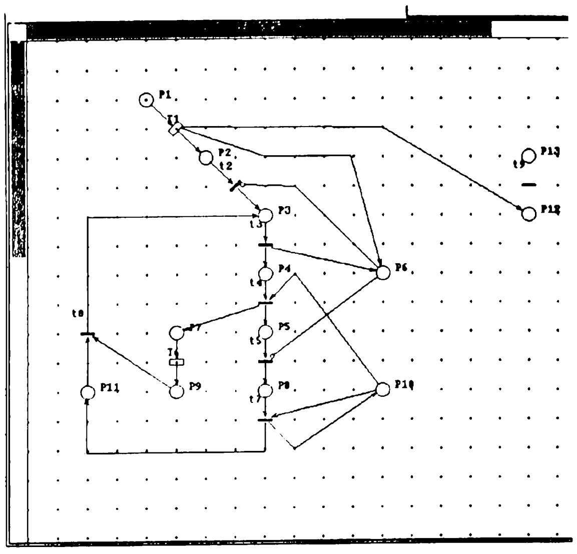


Figure 1: Drawing board with grid.

When the models are too complex, a possibility exists for their specification using *layers* that partition the nets into separate parts. Layers are well suited when the nets can be decomposed in portions that weakly interacts with

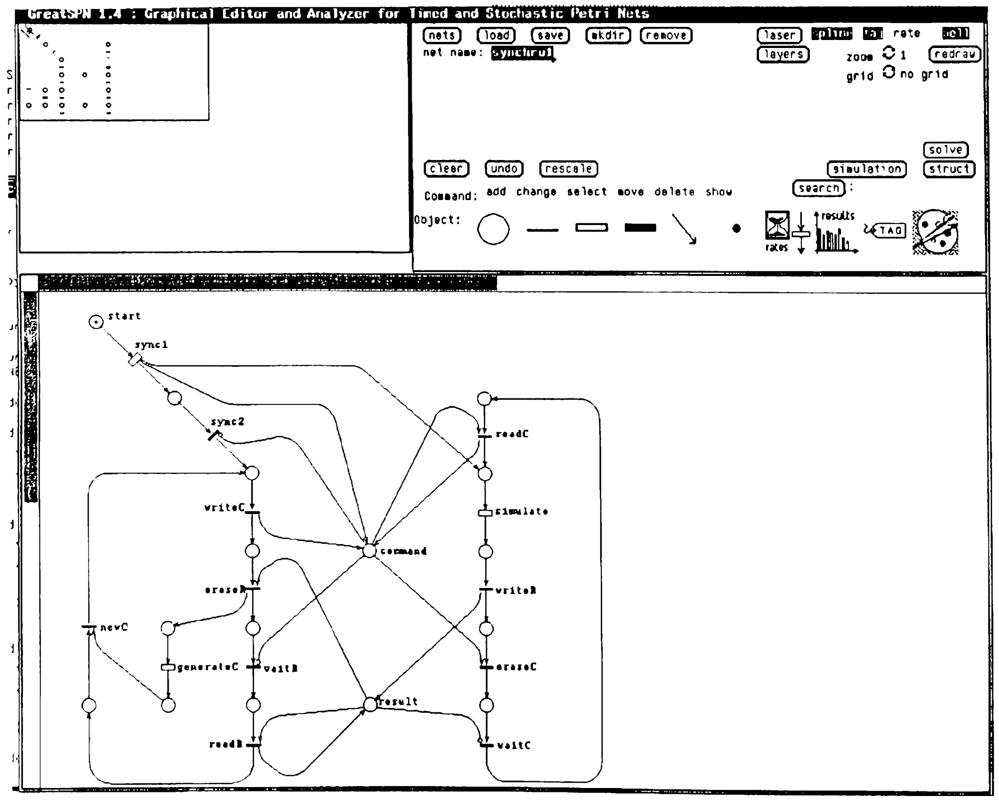


Figure 2: *GreatSPN* graphical interface.

each other especially from a graphical point of view. This means that portions of the nets exists that share only few places and/or transitions. Each individual layer is relatively simple to understand. The whole net is obtained by superimposing the layers onto each-other.

An important aspect of SPN models is the specification of the initial marking of the net that is relevant both as the starting point for the future simulation experiment, and as the specification that provides additional meaning to certain elements of the net and thus identifies the set of possible states of the model. The initial marking is obtained editing a place element parameter to specify the number of tokens that is initially contained in each place. A default value of zero tokens is introduced to make the specification of the initial marking quicker. Places that are defined with a non-zero number of tokens, are filled with as many black dots as the number of tokens dictates.

The specification of the model is completed with the definition of the parameters and of the performance indexes relevant for the study. All the transitions of the net possess default parameter values that can be modified with simple editing actions. When indications exist that the model will be used for several studies in which the effect of the variation of certain parameters will be investigated, these parameters can be specified as variables, using a simple context-free language. The same language is also used to define performance indexes that are functions of basic quantities such as probability distributions of tokens in places.

When the drawing of the model is completed, the model can be saved into a data-base that is actually a directory (that can be organized in subdirectories) containing all the files concerning a given model. Figure 3 shows the menus used to manage these directories.

2.2. Model Validation

SPN models of real systems tend to become extremely complex so that little can be said about their correctness by just looking at their picture. The theoretical foundation of the Petri net formalism assumes a crucial role during this phase as some structural properties of the model can be computed using algorithms whose complexity depends on the "size" of the model and does not depend on the size of its state space. The basic results that can be computed with

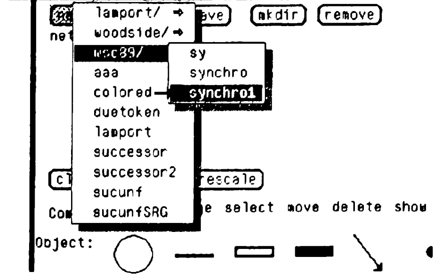


Figure 3: Menus for accessing the models' data-base.

these algorithms are the invariants of the net. Place and transition invariants are defined as the integer non-negative respectively left and right eigenvectors associated with the null eigenvalue of the incidence matrix of the Petri net [Mar81]. They can be used to decide whether the places of the net are bounded (e.g., the number of tokens in the net cannot grow unlimited) and whether the net has the possibility of returning to its initial state. Figure 4 shows a place invariant for the net that describes thei communication protocol used by *GreatSPN* to access shared files (see Section 3.3). Algorithms exist for testing whether a mutual exclusion relation exists among transitions, that is to decide whether it is possible or not for these transitions to be simultaneously enabled. A more elaborate use of these same results can provide information on the existence of deadlocks in the net. No algorithm is yet known for the automatic derivation of this last type of results, except for restricted subclasses of nets such as bounded, free-choice Petri nets.

The structural properties of a SPN represent a means for the static validation of the model. No initial marking is explicitly considered in their derivation. Obviously, not all the nets have a dynamic behaviour that is completely determined by their structure. When these more general models are under study, a dynamic validation can be obtained by playing the so-called *token game* that corresponds to an interpretation of the net starting from its initial marking. Enabled transitions can be made to fire moving the tokens to new positions that represent the new state reached by the net. When several transitions are simultaneously enabled, an arbitrary (non-deterministic) choice can be made to define which transition to fire next.

During a first qualitative analysis, the decision on which transition to fire next is made without considering any timing specifications. What is obtained is the *untimed* token game in which all the possible evolutions of the net can be explored. This type of study is supported by the

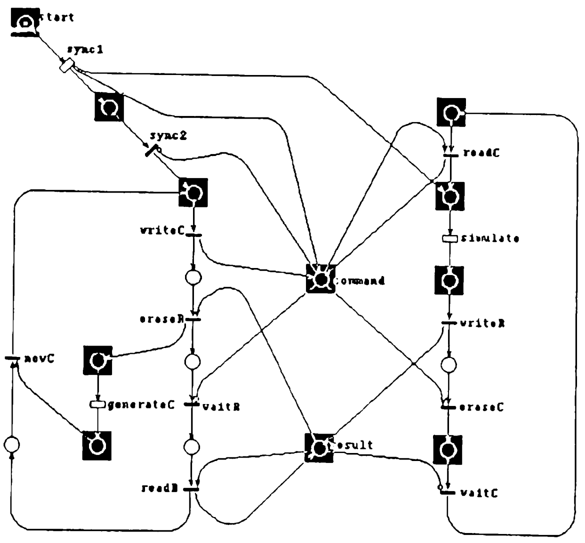


Figure 4: Display of a place invariant.

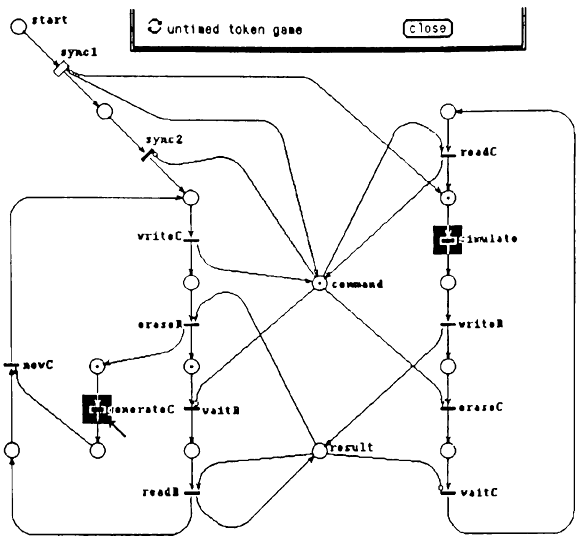


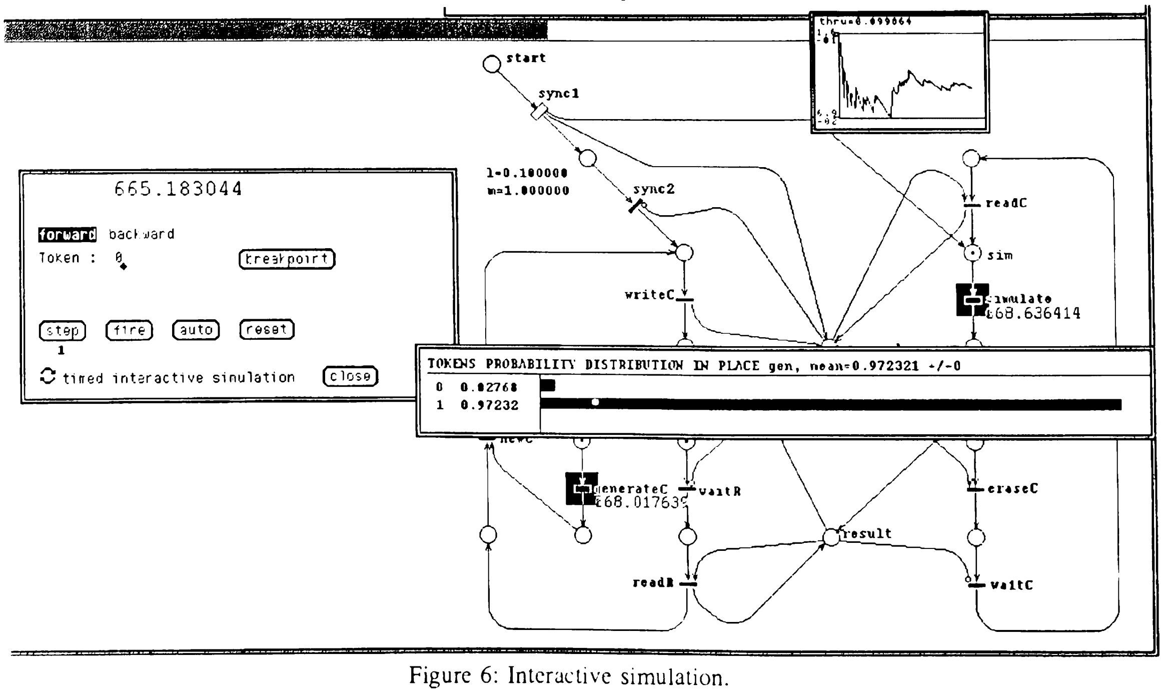
Figure 5: Untimed token game.

graphical interface that, given a certain marking, highlights all the enabled transitions among which the user chooses that to be fired next. Figure 5 shows the state that enables transitions "generateC" and "simulate". The actual selection is performed with a simple mouse action. The effect of firing the selected transition is shown by the interface that updates the distribution of tokens on the net (the net reaches a new marking) and highlights all the transitions that are enabled in the new marking. The updating of a marking as consequence of the firing of a transition can be made more evident by displaying the movement of tokens along the input and output arcs of the transition. Errors that may appear in the behaviour of the net while stepping through the state space in this manner, can immediately be corrected by properly modifying the net.

2.3. Model Simulation

A more precise validation of the model takes into account the actual firing times to decide which transition to fire next when severals are simultaneously enabled. When the firing time distributions have infinite support [Ajm89], the non-deterministic choice of the untimed token game is replaced by a probabilistic choice that derives from the computation of the distribution of the minimum firing time among simultaneously enabled timed transitions. When instead the firing time distributions have limited support, the behaviour of the timed net may be quite different from that of the untimed one, as certain firing sequences may become impossible.

To perform this validation task the construction of the net simulator is needed and the animation of the net becomes extremely important. The simulator is generated according to the standard techniques that are used in Montecarlo event driven simulators [Nay66]. Once that the simulator is obtained, the user has the possibility of playing a *timed token game* running the simulator in *interactive-mode* and interrupting its execution after the handling of each event. Also in this case the interface highlights the transitions that are enabled in a given marking, but the choice of the transition to fire next is determined by the event list of the simulator. Again the tokens can be shown to move along the arcs of the net and, after having reached the new marking the user has the possibility of inspecting the values of the statistics collected up to that point (see Figure 6 for an example). To investigate the effect of



```
665.183044
Token : 0
TOKENS PROBABILITY DISTRIBUTION IN PLACE gen, mean=0.972321 +/-0
0  0.02768
1  0.97232
```

Figure 6: Interactive simulation.

particular choices, the user has the possibility of forcing the simulator to fire a transition different from that prescribed by the event list. This action is reflected on the data structure of the simulator to avoid entering into an inconsistent state of the simulator. An option offered to the user is also that of backtracking the last portion of the execution to obtain a representation of the state evolution in reverse-time (undoing) exploiting the information contained in buffers. This feature helps in better understanding the behaviour of the net through a careful inspection of the path followed by the simulator to enter a specific state. The necessity of modifying the net to recover from possible behaviour errors is in this case more difficult to handle as changes in the structure of the net require to reconstruct the simulator (recompiling the net).

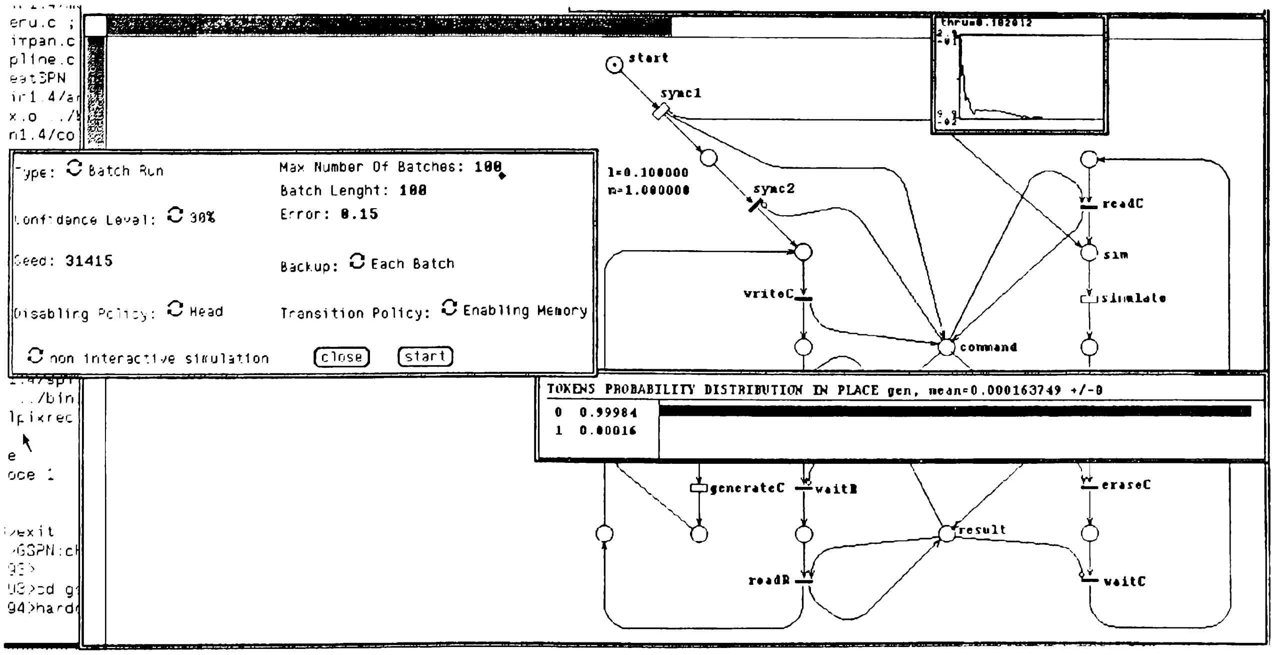
Stepping through the entire state space of a SPN with this approach can be extremely time consuming. A more efficient manner of controlling the (early) execution of the simulator is that of performing a batch of steps before stopping the simulation and allowing the interaction of the user. These batches of steps can be of constant or of variable length. In both cases, we can consider the simulation as controlled by breakpoints set by the user. Interesting breakpoints are those defined in terms of the markings visited by the net. In this way the timed token game enters the environment of the *automatic interactive simulation*.

When the breakpoint is set in terms of precision of the measured results the effect is the implementation of a sequential stopping rule [Lav77]. In particular, for what concerns the statistical analysis of the simulation output, the independent replication and the regenerative methods are implemented. The choice of which of the two methods to adopt is left to the user only if the conditions for the regenerative method are met, and if a preliminary pilot run shows that the regenerative points appears in the simulation with sufficient frequency.

This last mode of automatic simulation is actually a *batch-mode* in which, given the specification of the performance indexes that are relevant for the study and the accuracy that is desired, the simulation is run to completion and only final results are made available to the user in the form of interval estimates and histograms. In batch-mode periodical checkpoints are performed that are useful to allow the continuation of previously interrupted simulation experiments as well as to recover from unexpected terminations. When a performance index is of the single value type (e.g., resource utilization), a time diagram is constructed and periodically updated to obtain the index representation as a function of time. When a performance index is of the probability distribution type (e.g. distribution of the number of clients waiting for service), such a distribution is represented with a histogram that is periodically updated. Figure 7 provides an example of the performance indexes that are periodically displayed by the interface.

3. IMPLEMENTATION ASPECTS

The architecture of the simulator comprises several separate modules that are responsible for the construction and the animation of the SPN model, for its validation, for its simulation, and for the processing of measured data as well as for the control of the simulation experiment. All these modules can run independently of each other (possibly on different machines connected by a local area network) and interact by sharing files.



```
Type: Batch Run                     Max Number Of Batches: 100
                                    Batch Lenght: 100
Confidance Level: 30%               Error: 0.15

Seed: 31415                         Backup: Each Batch

Disabling Policy: Head              Transition Policy: Enabling Memory

non interactive simulation          close    start
```

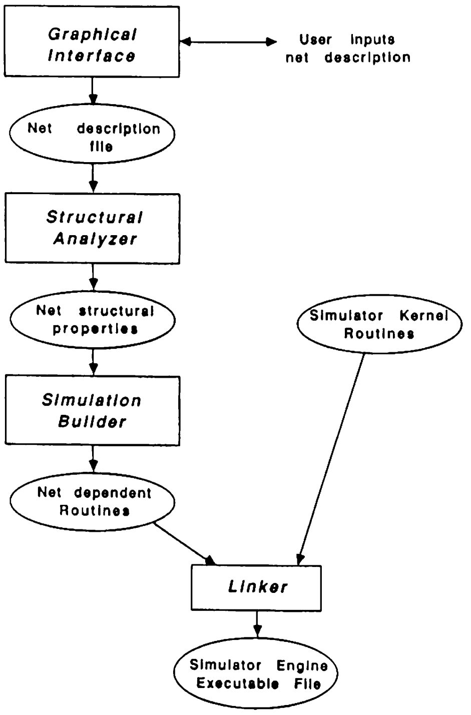
```
TOKENS PROBABILITY DISTRIBUTION IN PLACE gen, mean=0.000163749 +/-0
0  0.99984
1  0.00016
```

```
thru=0.102012
l=0.100000
m=1.000000
```

Figure 7: Batch-mode simulation results.

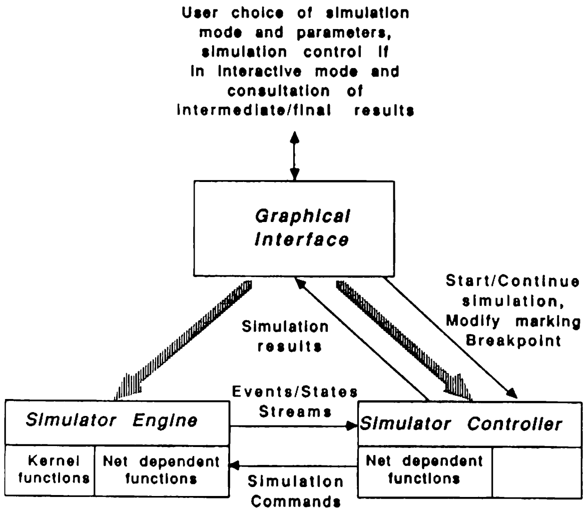
3.1. The Software Architectures

The simulation procedure is logically divided in three phases: the definition of a simulation model, its validation, and the actual simulation experiment. In order to support these three phases in a flexible manner, the simulation environment offers two main operating modes, both fully supported by the same graphical interface: a model management phase, and a model simulation phase. Two software architectures that connect different modules to the same graphical interface support these operation modes.



Graphical Interface
User inputs net description
Net description file
Structural Analyzer
Net structural properties
Simulator Kernel Routines
Simulation Builder
Net dependent Routines
Linker
Simulator Engine Executable File

Figure 8 : Modelling Environment Software Architecture



User choice of simulation mode and parameters, simulation control if in interactive mode and consultation of intermediate/final results
Graphical Interface
Simulation results
Start/Continue simulation, Modify marking Breakpoint
Events/States Streams
Simulator Engine
Simulator Controller
Kernel functions
Net dependent functions
Simulation Commands
Net dependent functions

Figure 9 : Simulation Environment Software Architecture

The construction and formal validation of a SPN model is performed within a modelling environment that is characterized by an architecture based on the interaction of two modules: the graphical interface and the structure analyser (see Figure 8). The graphical interface is used during this phase as the tool for drawing and specifying the SPN model (possibly exploiting a data base of previously defined models), as well as to display in graphical form invariant and other structural properties of the net that can be used for a formal validation of the model. When the user is satisfied with the SPN description, he can run a third module, the simulator builder, that reads the description of the model and its structural characteristics, and produces a set of optimized net dependent procedures that, linked together with a standard simulation kernel, generate the actual simulation program.

Once the specification of the SPN model to be simulated is completed, the functional validation and the actual simulation experiment are performed within a simulation environment characterized by an architecture based on the interaction of the graphical interface, the simulator controller, and the simulation engine (see Figure 9). During this phase, the graphical interface plays a new role characterized by two aspects. An input aspect is represented by the use of the interface to drive the simulation experiment through the specification of the simulation mode, of its parameters, of its termination conditions, and possibly of the definition of breakpoints. An output aspect is represented by the use of the graphical interface as a media for displaying the results of the simulation as soon as they are collected. The output of results can be performed both in terms of information that evolves during the simulation and in terms of performance indexes that can be interactively represented on the interface by showing at the end of the simulation the results of measurements performed during the simulation. The second module that belongs to the simulation architecture is obviously the simulation engine that manages the event list according to a combination of net-dependent and net-independent rules specified during the construction of the simulation model. The third module is the simulator controller that oversees the execution of the simulation engine providing a communication link between the graphical interface and the simulation engine itself. During the execution of a simulation experiment the possibility exists for running these different modules on different machines. To make the software tool simple and portable, communication among modules has been implemented through shared files.

3.2. The Graphical Interface

The advantage of using a net based formalism for the study of complex distributed systems is the descriptive power of these methods and their capability of capturing in an efficient manner the important features of the organization of these systems. In order to take the maximum possible advantage from the use of such techniques, it is important to use a graphical interface built for the construction, editing and management of net models. *GreatSPN* is implemented on top of the *SunView* graphical system running on SUN workstations. The implementation of this software component consists of more than 18,000 lines of C code (about 500 Kbytes of source), organized in 65 source files, each one ranging from 50 to 800 lines.

The construction of this graphical interface started in 1986, as a graphical editor for a package for the numerical solution of GSPN performance models based on Markov chain techniques [Chi85]. Subsequently, structural analysis [Chi87a], optimization [Chi88], and interactive simulation facilities [Bal89] have been added to the initial design. The good modularity of the programs and the object-oriented approach in the definition of the data structures allowed a smooth evolution of the tool and the inclusion of the new features with reasonable programming effort. Now the availability of these different modelling and evaluation techniques in a uniform, user-friendly environment probably constitutes the most interesting characteristics of *GreatSPN*.

Besides providing all the graphical primitives that are usually present in user-friendly graphical interfaces (resizing, moving, and replicating of individual as well as collective objects, selection of groups of objects for model construction, composition of elementary submodules, ...), *GreatSPN* provides also a way of representing the state of a SPN model by drawing black marks in the places of the net. The explicit representation of the state of the model is exploited by the interface to show the state evolution, by updating the token distribution as the result of the occurrence of an event. A graphical feature that makes this state updating to take place using a certain amount of time is the key element for obtaining the *animation* of a model following the state sequence dictated either by a prerecorded trace or by a real-time interpretation of the net. The traces corresponding to the state evolution of a model are read by the graphical interface from files produced by the simulator modules. The *interactive simulation* facility is implemented by running the simulator modules under the control of the graphical interface. In the latter operation mode, the interface is responsible for running the simulator and for providing it with the proper commands and parameters; a bidirectional communication is established always by means of shared files. The synchronization among the different processes (that can run on different machines) is achieved by a simple access protocol to the shared files.

3.3. The File Sharing Protocol

The communication between the graphical interface and the controller of the simulator is implemented by sharing two files, according to the simple synchronization protocol depicted by the Petri net in Figure 10. The model represents both the graphical interface and the simulator controller activities as shown by the areas surrounded by dotted lines. The two shared files are modelled by the two places "command" and "result" in the middle of the picture.

The interface starts the communication running the simulator processes with dummy parameters (arc from transition "sync1" to the input place of transition "simulate"), and preparing the corresponding dummy command file (arc from "sync1" to place "command"). Then the simulator

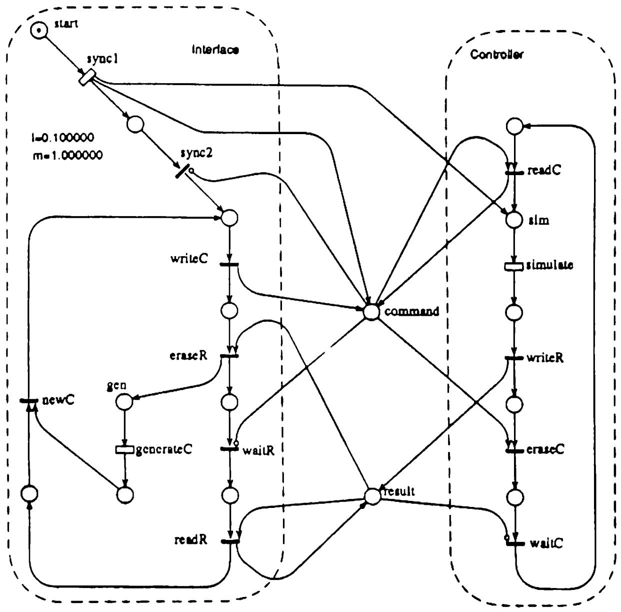


Figure 10 : Shared file synchronization protocol

executes the dummy command (firing of transition "simulate"), produces a result file (firing of "writeR"), and signals the availability of the results by deleting the command file (firing of "eraseC", which removes the token from place "command"). At this point transition "sync2" becomes enabled, and after its firing the protocol reaches its normal (synchronized) operation condition.

In steady-state, the graphical interface writes a simulation request in the command file (by firing "writeC"), and then deletes the previous result file (by firing "eraseR") to signal the availability of the new command file to the simulator. At this point the interface can use the previous results to implement the animation, and construct the next simulation command (which can actually correspond to a sequence of simulation steps) based on the interaction with the user. Concurrently with the user and interface activities, the simulator realizes the presence of the new request (firing of "waitC") and reads it (firing of "readC"). Then the proper simulation step is implemented, and at the end (firing of "simulate") the results are written in the shared file (transition "writeR"). The availability of the result file is signalled by deleting the command file (transition "eraseC"). When place "command" becomes empty, transition "waitR" can fire and then the interface reads the result file by firing "readR". When both "readR" and "generateC" have been fired, the next command is ready to be issued by the interface, and this is modelled by transition "newC".

3.4. The Structural Analyser

This component of the package is implemented by a set of independent programs that compute different characteristics of a Petri net model starting from its topological definition. The results of this step are used both by the modeller to check for some formal properties, and by the simulator builder to optimize the automatically generated programs. The first step is the computation of the place invariants. A program is called that implements a very efficient version of the algorithm proposed by Martinez and Silva [Mar81], and stores the results on a file. This information is used both to check conservation properties, and to devise optimal encodings for the marking of the net.

Using this partial results and the topology of the SPN, some relations between transitions of the model are computed, such as the mutual exclusion, the conflict and the causal connection [Chi87a], that are useful both to verify some expected properties of the model, and to optimize the management of the event list of a simulation program.

The theoretical complexity of the algorithms for the computation of the structural properties depends only on the number of places, transitions, and arcs of the SPN model, and not on the size of the state space. In practice, the execution time for these analysis programs is order of magnitudes lower than that of either a simulation experiment or a numerical analysis based on Markovian techniques. The optimization step based on the knowledge of these structural properties can thus be very cost effective.

3.5. The Simulator Builder and Kernel

Simulation of Petri nets is one of the most natural ways of exploiting their representation power. Discrete event simulation of Petri nets is a topic that has received little attention from Petri net researchers as it must translate the dynamic behaviour of a highly parallel model into a sequential representation characterized by the scheduling of the events that actually determine the evolution of the net. Discrete event simulation of Petri nets is however a very important topic if these models have to be used for performance evaluation purposes. The main problem with the simulation of stochastic Petri nets is the time-efficient identification of the transitions that are either enabled or disabled by the occurrence of an event (transition firing).

Managing event occurrences is indeed difficult because of the generality of the structure (interconnection pattern) of Petri net models. *GreatSPN* manages this intrinsic complexity by constructing functions whose evaluation helps in quickly identifying the updating that has to be performed on the set of currently enabled transitions, due to the occurrence of an event [Chi88]. Since these functions are net-dependent, the simulator builder starts from the description of the net produced by the graphical interface to construct a set of procedures, written in the C programming language. The structural properties of the net computed by the structural analyser are used to optimize the coding of these functions. Once this net-dependent code is produced, it is compiled and linked together with the simulation kernel to obtain the simulator *engine* that is the actual executable module used to perform the simulation of the model.

The simulator kernel included in *GreatSPN* provides the basic functions (a sort of run-time environment) for the construction of a classical [Nay66] event driven simulator of the net. The kernel of the *GreatSPN* simulator comprises the procedures for the management of an event list augmented with lists of affected transitions associated with each event-notice. The coding of some of these functions is net-dependent and is thus performed during the first stage of the simulator construction when the graphical structure of the net is actually compiled.

The basic feature characterizing the kernel is the capability of performing a step of an event driven simulation of the net, and of writing on the standard output the event that happened, the new value of the simulation clock, and the new state that has been reached. Indeed, both interactive-simulation and batch-simulation modes are obtained by combining calls to the kernel to reach particular simulation states or to perform sequences of simulation steps. This elementary action of managing an event involves the selection of a new event from the event-list, the updating of the model state, the scheduling of new events, and the descheduling of other events. The selection from the event list of the next event to occur is performed in a standard manner. The recording of the state modification induced by the event that is being processed requires the use of net-dependent functions that update the information concerning the enabling conditions of the transitions. The scheduling of new events is performed in an efficient manner exploiting the net-dependent functions mentioned before. The descheduling of events is an important aspect of event managing in Petri nets, as the completion of an activity associated with a transition that belongs to a set of conflicting ones causes the interruption of the other. This feature is peculiar of Petri net models and requires a proper handling of the interrupted activities as specified by the model [Ajm89].

To make the interactive simulation environment possible, the kernel supports also the possibility of setting breakpoints, forcing different choices (e.g., forcing the occurrence of extremely unlikely events), saving the simulation state on files, stopping and resuming simulation runs, and undoing simulation steps.

In case of natural regenerative simulation, an initial pilot simulation run is performed in which the states reached by the simulator are recorded with the aim of automatically detecting recurrent memoryless states, among which the regeneration state is automatically chosen. In this case also the efficiency of the representation of the markings of the net becomes critical, both in terms of space occupied by the data structure and of time needed to insert and search for an already encountered state. The encoding of the marking of the SPN is thus also optimized by generating efficient data structures and management procedures based on the knowledge of the place invariants of the net [Chi88]. These net-dependent data structures and procedures are used not only in the simulator modules, but also in the numerical solution algorithms based on Markov techniques that are available in *GreatSPN*.

3.6. The Simulator Controller

A simulation experiment consists of the measurement of user defined performance indexes performed on the sample path derived from the execution of the model driven by the occurrence of events. The simulator controller of *GreatSPN* has been devised as a measurement tool that, fed by the output of the simulation engine, processes the data and accumulates the desired performance index estimates. Striving for generality, the simulator controller is composed of a set of net-independent procedures that perform the statistical analysis of the output produced by the simulation engine. Point estimates and confidence intervals are computed with standard techniques. A net-dependent decoding function is also linked to the controller, to allow the communication of the states stream to the graphical interface. Using information provided by the graphical interface the simulator controller knows the technique (regenerative or independent replication method) that has to be used to obtain a sample of (approximatively) independent and identically distributed instances. A checkpoint is performed by the simulator controller to provide the possibility of resuming simulation experiments for obtaining better quality results as well as for recovering from possible system crashes. Checkpoint data are stored in an appropriate file.

The communication between the simulation engine and the controller is implemented with two Unix primitives, a shell-level "pipe" from the engine to the controller, on which the sequence of states and the events that produced them flow, and a "socket" that is used by the controller to transmit commands to the engine. This choice has been dictated both by efficiency and simplicity considerations, since a large communication throughput is required between these two models. The communication between the controller and the graphical interface is obtained by sharing two files, according to the simple synchronization protocol described in Section 3.3.

4. CONCLUSIONS

In this paper we have discussed the possibility of using SPN models to study the performance characteristics of modern computer/communication systems. When dealing with real cases, it often happens that the SPN models tend to become very complex. In these cases SPN models can be used as formal specifications of systems and automatically translated into detailed simulation programs in order to estimate performance results.

The construction and the analysis of complex SPN models can only be done with the help of powerful automatic tools such as the user-friendly software package *GreatSPN* whose modelling facilities and software architecture have been illustrated in this paper.

The integration within the same interface of the graphical facilities for the model construction, of the structural analysis algorithms for model validation, and of the control panels for performing the visual and interactive simulation, emphasizes the importance of including in a simulation experiment both validation and evaluation aspects. Moreover, the availability of Markovian techniques within the same modelling environment increases the opportunity of (quantitatively) validating the simulation results on scaled down models by direct comparison with exact results.

The structural results are useful for a preliminary assessment of the correctness of the model and for optimizing the code of the event driven simulator. The animation of the model, performed only when desired by the user, appears to be a very powerful tool that complements the structural results for the debugging and tuning of the models used for the performance analysis. The architecture of the simulation environment, organized with several cooperating modules turns out to be very efficient as it can easily exploit the computation power available in local area networks connecting several homogeneous workstations.

REFERENCES

[Ajm84] M. Ajmone Marsan, G. Balbo, and G. Conte, "A Class of Generalized Stochastic Petri Nets for the Performance Analysis of Multiprocessor Systems," *ACM Transactions on Computer Systems* 2(1)(May 1984).

[Ajm87] M. Ajmone Marsan, G. Balbo, G. Chiola, and G. Conte, "Applicability of Stochastic Petri Nets to Performance Modeling," in *proc.* 2nd *Int. Workshop Applied Mathematics & Performance/reliability Models of Computer/Communication Systems*, ed. G. Iazeolla, P.J. Courtois, O.J. Boxma,North-Holland, Roma, Italy (May 1987).

[Ajm87a] M. Ajmone Marsan, G. Balbo, G. Chiola, and G. Conte, "Generalized Stochastic Petri Nets Revisited: Random Switches and Priorities," pp. 44-53 in *Proc. Int. Workshop on Petri Nets and Performance Models*, IEEE-CS Press, Madison, WI, USA (August 1987).

[Ajm89] M. Ajmone Marsan, G. Balbo, A. Bobbio, G. Chiola, G. Conte, and A. Cumani, "The Effect of Execution Policies on the Semantics and Analysis of Stochastic Petri Nets," *IEEE Transactions on Software Engineering*, (July 1989).

[Bal89] G. Balbo, G. Chiola, and G. Franceschinis, "Stochastic Petri Net Simulation for the Evaluation of Flexible Manufacturing Systems," in *Proc. 1989 European Simulation Multiconference*, SCS, Roma, Italy (June 1989).

[But89] B. Butler, R. Esser, and R Mattmann, "A Distributed Simulator for High Order Petri Nets," pp. 22-34 in *Proc.* 10th *Int. Conference on Application and Theory of Petri Nets*, , Bonn, Germany (June 1989).

[Cha81] K.M. Chandy and J. Mishra, "Asynchronous Distributed Simulation via a Sequence of Parallel Computations," *Communications of the ACM* 24(4) pp. 198-206 (April 1981).

[Chi85] G. Chiola, "A Software Package for the Analysis of Generalized Stochastic Petri Net Models," in *proc. Int. Workshop on Timed Petri Nets*, IEEE-CS Press, Torino, Italy (July 1985).

[Chi87] G. Chiola, "A Graphical Petri Net Tool for Performance Analysis," in *proc.* 3rd *Int. Workshop on Modeling Techniques and Performance Evaluation*, AFCET, Paris, France (March 1987).

[Chi87a] G. Chiola, "Structural Analysis for Generalized Stochastic Petri Nets: Some Results and Prospects," pp. 317-332 in *proc. 8th European Workshop on Application and Theory of Petri Nets*, Univ. Zaragoza, Zaragoza, Spain (June 1987).

[Chi88] G. Chiola, "Compiling Techniques for the Analysis of Stochastic Petri Nets," in *proc. 4th Int. Conf. Modeling Techniques and Tools for Computer Performance Evaluation*, AFCET, Palma de Mallorca, Spain (September 1988).

[Col86] J.M. Colom, M. Silva, and J.L. Villarroel, "On Software Implementation of Petri Nets and Colored Petri Nets Using High-Level Concurrent Languages," pp. 207-241 in *Proc. 7th European Workshop on Application and Theory of Petri Nets*, Sheffield Polyt., Oxford, England (June 1986).

[Flo85] G. Florin and S. Natkin, "Les Reseaux de Petri Stochastiques," *Technique et Science Informatiques* 4(1)(February 1985).

[Gen81] H.J. Genrich and K. Lautenbach, "System Modelling with High-Level Petri Nets," *Theoretical Computer Science* **13** pp. 109-136 (1981).

[Jef87] D.J. Jefferson et.al., "Distributed Simulation and the Time Warp Operating System," pp. 77-93 in *Proc. 11th ACM Symposium on Operating System Principles*, ACM, Austin, TX, USA (November 1987).

[Jen81] K. Jensen, "Coloured Petri Nets and the Invariant Method," *Theoretical Computer Science* **14** pp. 317-336 (1981).

[Lav77] S.S. Lavenberg and C.H. Sauer, "Sequential Stopping Rules for the Regenerative Method of Simulation," *IBM Journal of Res. Develop.* **21** pp. 545-558 (1977).

[Mar81] J. Martinez and M. Silva, "A Simple and Fast Algorithm to Obtain All Invariants of a Generalized Petri Net," in *Proc. 2nd European Workshop on Application and Theory of Petri Nets*, Springer Verlag, Bad Honnef, West Germany (September 1981).

[Mel85] B. Melamed and R.J.T. Morris, "Visual Simulation: The Performance Analysis Workstation," *Computer* **18**(8) pp. 87-94 (August 1985).

[Mol82] M.K. Molloy, "Performance Analysis using Stochastic Petri Nets," *IEEE Transaction on Computers* **C-31**(9) pp. 913-917 (September 1982).

[Nay66] T.H. Naylor, J.L. Balintfy, D.S. Burdick, and K. Chou, *Computer Simulation Techniques*, Wiley, New York, NY (1966).

[Noe73] J.D. Noe and G.J. Nutt, "Macro E-nets Representation of Parallel Systems," *IEEE Transactions on Computers* **C-31**(9) pp. 718-727 (August 1973).

[Nut89] G.J. Nutt, "A Flexible Distributed Simulation System," pp. 210-225 in *proc. 10th Int. Conference on Application and Theory of Petri Nets*, G.M.D., Bonn, Germany (June 1989).

[Pet81] J.L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Englewood Cliffs, NJ (1981).

[Rei85] W. Reisig, *Petri Nets: an Introduction*, Springer Verlag (1985).

[Tau87] D. Taubner, "On the Implementation of Petri Nets," pp. 471-488 in *proc. 8th European Workshop on Application and Theory of Petri Nets*, Univ. Zaragoza, Zaragoza, Spain (June 1987).

AUTHORS' BIOGRAPHY

GIANFRANCO BALBO received his Doctor degree in Physics from the University of Torino, Italy, in 1970, and his M.S. and Ph.D. degrees in Computer Science from Purdue University, West Lafayette, Indiana, in 1975 and 1979, respectively. Since 1978 he has been with the Computer Science Department of the University of Torino where he is now a Full Professor in charge of two introductory courses on Operating System Theory and Performance Evaluation of Computer Systems. His research interests are in the area of performance evaluation of computer systems, queueing network models, stochastic Petri net models, and queueing theory. He is co-author with S.C. Bruell of the book "Computational Algorithms for Closed Queueing Networks", and with M. Ajmone Marsan and G. Conte of the book "Performance Models of Multiprocessor Systems". He has been member of the Program Committees of many international conferences. He is a member of the Association for Computing Machinery.

Prof. Gianfranco Balbo
Dipartimento di Informatica, Universita` di Torino
corso Svizzera 185, 10149 Torino, Italy
phone: (+39)-11-7712002

GIOVANNI CHIOLA received his Doctor in Electronics Engineering degree from the Politecnico di Torino in 1983. In March 1986 he joined the Faculty of the Computer Science Department of the University of Torino as a Researcher. He developed a computer package for the analysis and simulation of stochastic Petri nets, and (co)-authored about 25 papers on performance modelling and Petri nets. He is a member of the IEEE Computer Society, the Association for Computer Machinery, and SIGMETRICS.

Dr. Giovanni Chiola
Dipartimento di Informatica, Universita` di Torino
corso Svizzera 185, 10149 Torino, Italy
phone: (+39)-11-7712002