Comparison of three SPN Packages
GreatSPN1.6, DSPNexpress1.2, SPNP3.0

Isabel Rojas Mujica *

May 20, 1994

Abstract

For the appropiate analysis and simulation of various types of Petri nets, modelling real systems or applications, the assistance of computer packages or tools is indispensable. This paper provides an brief introduction to the use of Stochastic Petri Nets packages, specifically GreatSPN1.6, DSPNexpress1.2 and SPNPv3.Initially a review of the main concepts employed in the analysis of Stochastic Timed Petri Nets is given. It is expected that the reader will have a general knowledge of Petri Nets and that this would act as a revision and as the introduction to rather more complex concepts. The idea of this section is to supply a theoretical background to the facilities offered by the packages that are analysed in this paper. For each package we describe its main features, what it offers and how its implemented. As a guide for the selection of which package is more appropiate for the readers particular needs a comparison section between the packages is offered. The information supplied for each package reviewed, has come from both the experience gained using the packages and from the existing documentation for each one.

Contents

*The author is supported by a grant from the Venezuelan Government Research Council (CONICIT).

1 Stochastic Petri Nets Concepts

A *Petri net* (PN) ([25], [23]) can be seen as a directed bipartite graph whose set of nodes V can be partitioned into two disjoint sets, *places* (P) and *transitions* (T), $V = P \cup T$. Arcs can only go from places to transitions (*input arcs, I*) or from transitions to places (*output arcs, O*). Multiple arcs from places to transitions or viceversa are allowed, specified by placing the multiplicity of the arc a $(m(a)\epsilon N^{+})$ beside it. There is also a third set of arcs, called *inhibiting arcs, (H)*, which connect places with transitions. In our example (Fig. 1)

$$
\begin{aligned}
&P = \{p_1, p_2, p_3, p_4\},\\
&T = \{t_1, t_2\},\\
&I = \{(p_1, t_1), (p_2, t_2)\},\\
&O = \{(t_1, p_3), (t_2, p_4)\},\\
&H = \{(p_2, t_1)\}\\
&m(a) = 1,\ \forall a \in (I \cup O \cup H) - \{(p_2, t_2)\}\\
&\text{and } m((p_2, t_2)) = 2.
\end{aligned}
$$

Each place can have a number of tokens associated with it. A function M: P$\rightarrow N^{+}$ is said to be a *marking*, assigning for each place a certain number of tokens. Each marking can be considered as representing a state of the PN. M_o is the initial marking of a net, representing the initial number of tokens that is associated with each place in the net. A marking can also be represented as a multiset of places, where the number of occurrences of the place in the multiset corresponds to the number of tokens in that place. In the example shown in Fig. 1,

$$M_o(p_1) = 1,\ M_o(p_2) = 2,\ M_o(p_3) = 0 \text{ and } M_o(p_4) = 0$$

and if represented as a multiset,

$$M_o = p_1 + 2p_2$$

A transition t_k is said to be *enabled* if each of its input places has at least as many tokens as the multiplicity of the input arc from the place to the transition t_k and if each inhibiting place p_h of the transition has less tokens than the multiplicity of the inhibiting arc (p_h, t_k). Enabled transitions can fire, leading to a change in the marking or marking function. This corresponds to the elimination of as many tokens as the multiplicity of the input arc from each place, and placing as many tokens as the multiplicity of the output arc in each of its output places. If for every inhibitent arc to a transition the place of origin of the arc has at least as many tokens as the multiplicity of the arc, then the transition is said to be inhibiting and cannot fire until the condition stated is withdrawn.

A firing sequence is a sequence of transition firings. A marking M_j is said to be *reachable* from a marking M_i if it is obtained by a firing sequence from M_i. The *Reachability Set* (RS) R(M_o) of a PN is the set of all markings reachable from M_o, including M_o. The *Reachability Graph* (RG) G(M_o) of a PN is the labeled directed graph whose vertices are the elements of R(M_o), and whose arcs (M_i, M_j) are labeled k indicating that marking M_j can be reached from marking M_i by firing transition t_k (arcs represent a firing sequence of length 1). It is assumed that only one transition firing can lead directly from one marking to another.

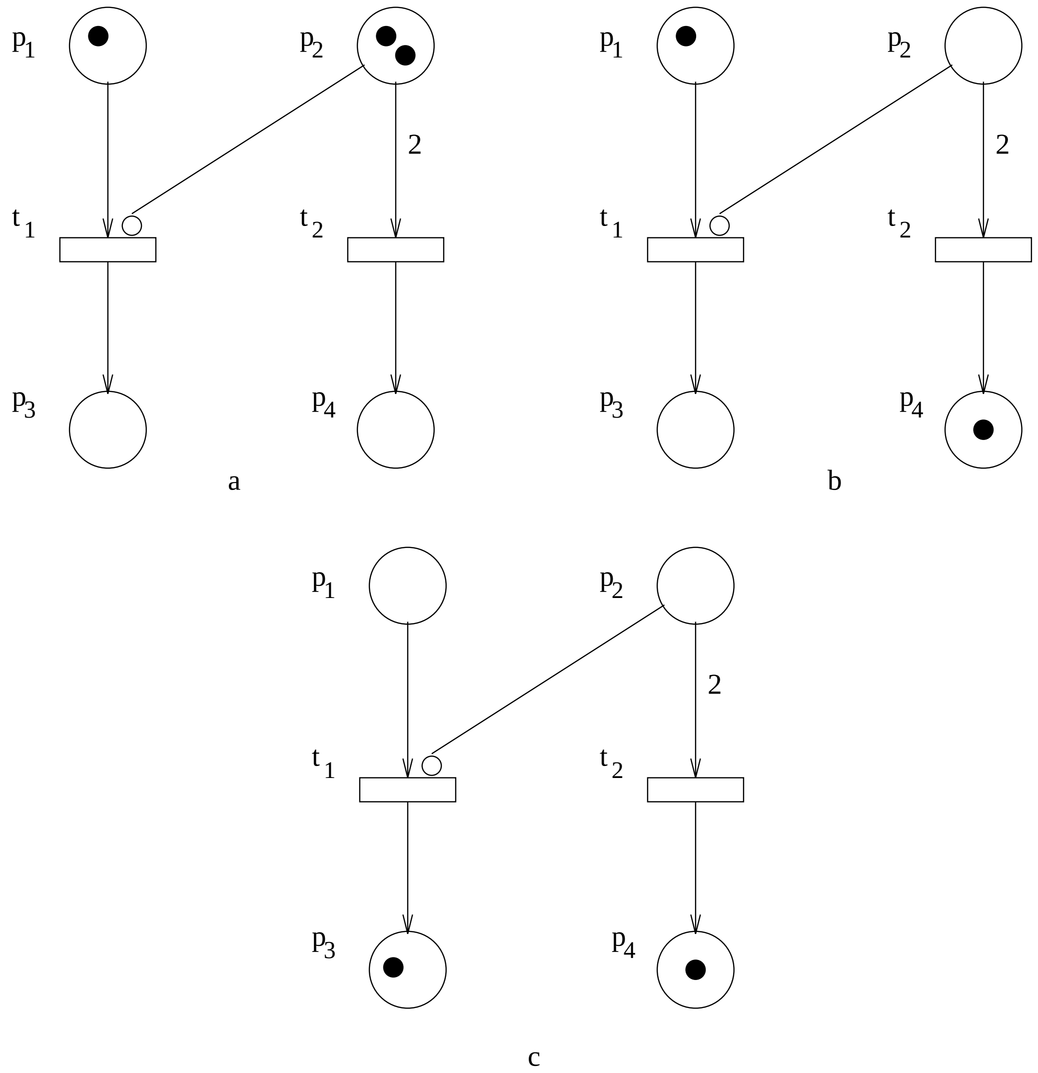


Fig. 1 A Petri net. a) is the initial state of the net, b) the state after the firing of transition t_2 and c) is the net after the firing sequence $< t_2, t_1 >$

In our example shown in Fig. 1, $R(M_o) = \{M_o, M_1, M_2\}$ where $M_1 = p_1 + p_4$ and $M_2 = p_3 + p_4$. M_1 is reachable from M_o through the firing sequence $< t_2 >$ and M_2 is reachable from M_o through the firing sequence $< t_2, t_1 >$. When t_2 fires it disactivates the inhibition over t_1 converting it into an enabled transition.

In *Timed Petri Nets* (TPN) a firing delay is associated with each transition. It specifies the amount of time that must elapse before the transition can fire once it is enabled and has been chosen to fire. The *Stochastic Petri Net* (SPN) model is obtained from the TPN model by associating a probability distribution function to the firing time of each transition. Most authors associate exponentially distributed firing delays with PN transitions, because of the memoryless property of the distribution which is useful for the solution of the embedded Markov Chain formed by the states (markings) of the PN. The parameters of an exponential or general distribution are said to be *marking dependent* if they can differ according to each marking.

When several timed transitions are simultaneously enabled in a given marking it is frequently assumed that the one with the lowest firing delay will fire first.

The solution of a Continuous Time Markov Chain (CTMC) model consists of the computation of the probability mass function over the state space S either at any arbitrary time instant t or in equilibrium conditions. When an equilibrium or steady-state probability mass function (pmf) exists, and is independent of the initial state, the CTMC is said to be *ergodic* [1]. We can stablish a isomorphism between the RG of the SPN and its embedded CTMC by the association of the nodes in the reachability graph with the states of the embedded Markov Chain.

A sufficient condition for a SPN to be *ergodic* is that the initial marking M_o is reachable from any $M_i \in R(M_o)$. If the SPN is ergodic so is the isomorphic CTMC. The solution of this CTMC provides the steady-state probability distribution on the markings of the SPN. From the steady-state distribution it is possible to obtain quantitative estimates of the behaviour of the SPN. The limitations of the SPN are that the graphical representation of systems becomes rapidly difficult when system size and complexity increase. Moreover, the number of states of the associated CTMC grows very fast with the dimensions of the net.

Generalized SPNs (GSPNs) [1] try to mitigate the problem of explosion of the number of states of the CTMC by defining two types of transitions: *timed transitions* and *immediate transitions*; these are referred to as exponentially distributed transitions and deterministic with value 0 transitions, respectively. A set of transitions is said to be *conflicting* if they are simultaneously enabled. Immediate transitions are usually considered to have firing priority over timed transitions. In the case of simultaneous enabling of several immediate transitions, a probability function must specify the probability of firing of each immediate conflicting transition.

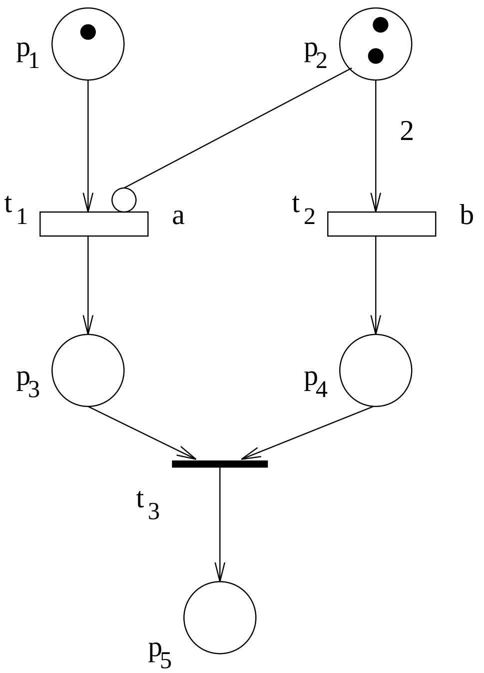


Figure 1: **Fig. 2** A Generalized Stochastic Petri Net (GSPN)

A marking is said to be *tangible* if it enables no immediate transition, otherwise it is called *vanishing*. A marking which does not enable any transition is *absorbing*, hence it is tangible by definition.

In the GSPN of Fig. 2, the thick horizontal lines represent immediate transitions and the boxes represent exponentially distributed timed transitions. The marking $M_1 = p_1 + p_4$ is a tangible marking whereas the marking $M_2 = p_3 + p_4$ is vanishing.

From now on $O_j(t_k)$, $I_j(t_k)$ and $H_j(t_k)$ will represent the multiplicity of the output, input and inhibiting arcs in between a place p_j and a transition t_k, respectively; and $O(t_k)$, $I(t_k)$ and $H(t_k)$ will represent bags of output, input and inhibiting places of a transtion t_k, respectively.

The incidence matrix C of a PN has entries $c_{jk} = O_j(t_k) - I_j(t_k)$. Any vector $\mathbf{i}$, that is an integer solution of the matrix equation $C^T\mathbf{i}=0$ is a place invariant of the PN. All the place invariants of a PN can be obtained as a combination of a finite set of generators, called *minimal-support* place invariants (P-invariants). The scalar product between a P-invariant and any marking $M \in R(M_o)$ yields a constant called the token count of the invariant. The linear equation resulting from this scalar product will be indicated as a marking invariant (M-invariant)[11]. The concept of *transitions invariants* (T-invariants) is dual to P-invariants [19]. A PN is said to be covered by P-invariants(T-invariants) if all its places (transitions) are included in the set of P-invariants(T-invariants).

Structural analysis, or the study of invariants, attempts to isolate from a set subnets with special properties. Place invariants can be used for static deadlock detection as pointed out by [22].There are also important applications of transition invariants in modeling logic programs comprising Horn clauses [15]. The same algorithm can be used to compute both place and transition invariants [19].

Consider a set $D \subseteq P$ in a PN. Let

$D^{*} = \{t|t$ is an output transition of a place $p \in D\}$

${}^{*}D = \{t|t$ is an input transition of a place $p \in D\}$.

If a set D satifies ${}^{*}D \subseteq D^{*}$ we say that the PN is deadlocked. If a set D satisfies $D^{*} \subseteq {}^{*}D$, it is called a *trap*[13]. In Fig. 3 we give examples of traps and deadlocks (taken from [25]).

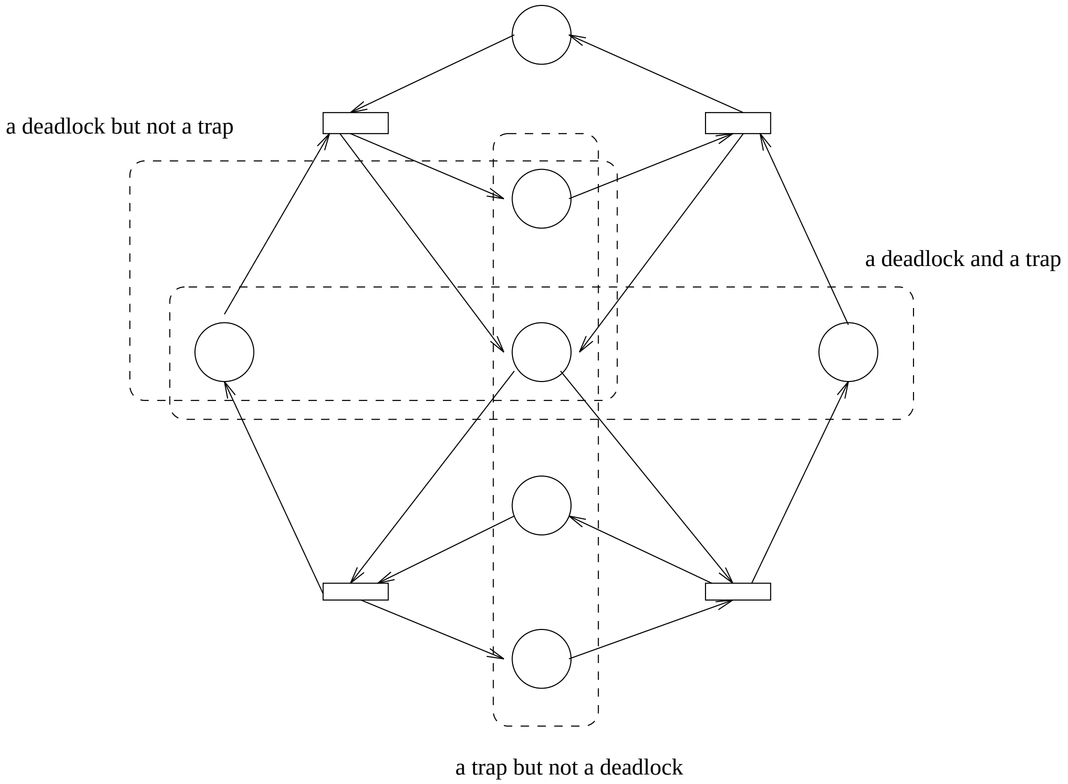
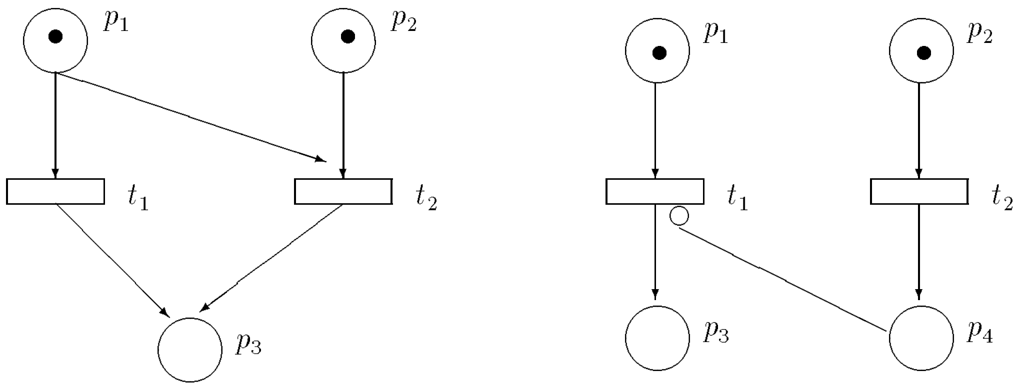


Fig. 3 Examples of Deadlocks and Traps

When the firing of a transition t_m disables another previously enabled transition t_l, we say that t_l is in conflict with t_m (Fig. 4). A transition t_l is said to be in *structural conflict* with t_m,denoted $t_l SC t_m$, iff

$$I(t_l) \cap (I(t_m) - O(t_m)) \neq \emptyset \vee$$

$$H(t_l) \cap (O(t_m) - I(t_m)) \neq \emptyset$$



a b

Fig. 4 Structural Conflict examples

In case (a) of Fig. 4 both transitions are initially enabled. If transition t_2 fires first it will take one token from p_1 and one from p_2 and place a token in p_3, disabling transition t_1. If t_1 fires first it will take a token from p_1 and place one in p_3, disabling transition t_2. In this case both $t_1 SC t_2$ and $t_2 SC t_1$. In case (b) of Fig. 4 both transitions are initially enabled. If t_2 fires first it will place a token in p_3 which would inhibit t_1 because there is an inhibiting arc from p_4 to t_1. In the case that t_1 fires first, t_2 will remain enabled after the firing, thus $t_1 SC t_2$ and not$\{t_2 SC t_1\}$.

Two transitions t_l and t_m are *mutually exclusive*with respect to an initil marking if they cannot be enabled together in any reachable marking (Fig. 5).

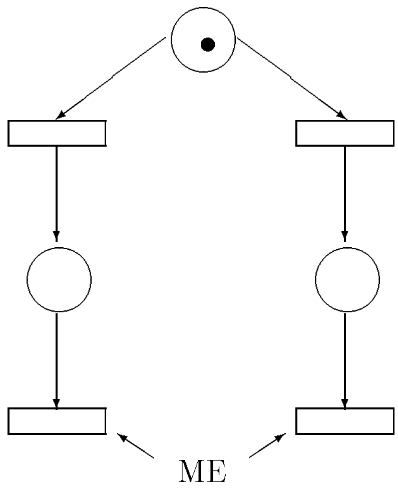


Fig. 5 Example of two Mutually Exclusive transitions

Two transitions t_l and t_m are said to be in *Symmetrically Structural Conflict*,

denoted $t_l SSC t_m$, iff

$[t_l SC t_m \vee t_m SC t_l] \wedge not(t_l ME t_m)$

The *extended conflict set* (ECS) of a transition is defined as:

$ECS(t) = \{t_i | t_i SCC^* t\}$

where SSC^* is the equivalence relation generated by the transitive and reflexive closure of the SSC, that can be used to partition immediate transitions into classes of possible conflicting sets.

In a general Petri Net the actual resolution of a conflict (i.e. deciding which among the conflicting transitions is to be fired next) may depend on the firing of sequences of transitions that are not in conflict with each other. This problem is known as *confusion*, and appears when the firing of a transition t_l, that is not in the ECS of a given transition t_k, enables a third transition t_m that instead belongs to the ECS of t_k ($t_m \in ECS(t_k)$). This implies that, in a *confusion free net*, the resolution of conflicts is completely determined within a single ECS, while in the case of confusion, the specification at the level of ECS might not be sufficient to determine the behaviour of a net.

An immediate transition t_l is said to be *Causally Connected* (CC) to t_m with respect to a third transition t_k (Fig. 6), denoted by $t_l CC_{t_k} t_m$, iff

$$[(O(t_l) - I(t_l)) \cap I_{-k}(t_m) \neq \emptyset \vee$$
$$(I(t_l) - O(t_l)) \cap H_{-k}(t_m) \neq \emptyset] \ \wedge \ not[t_l MEt_m]$$

where $I_{-k}(t_m) = I(t_m) - I(t_k)$, and $H_{-k}(t_m)$ is defined by:

$$\begin{cases} H(t_m) & if \forall p \in P : h_p(t_m) > 0 \wedge h_p(t_k) = 0 \\ H(t_k) - H(t_m) & if \forall p \in P : h_p(t_k) > h_p(t_m) > 0 \\ \emptyset & otherwise \end{cases}$$

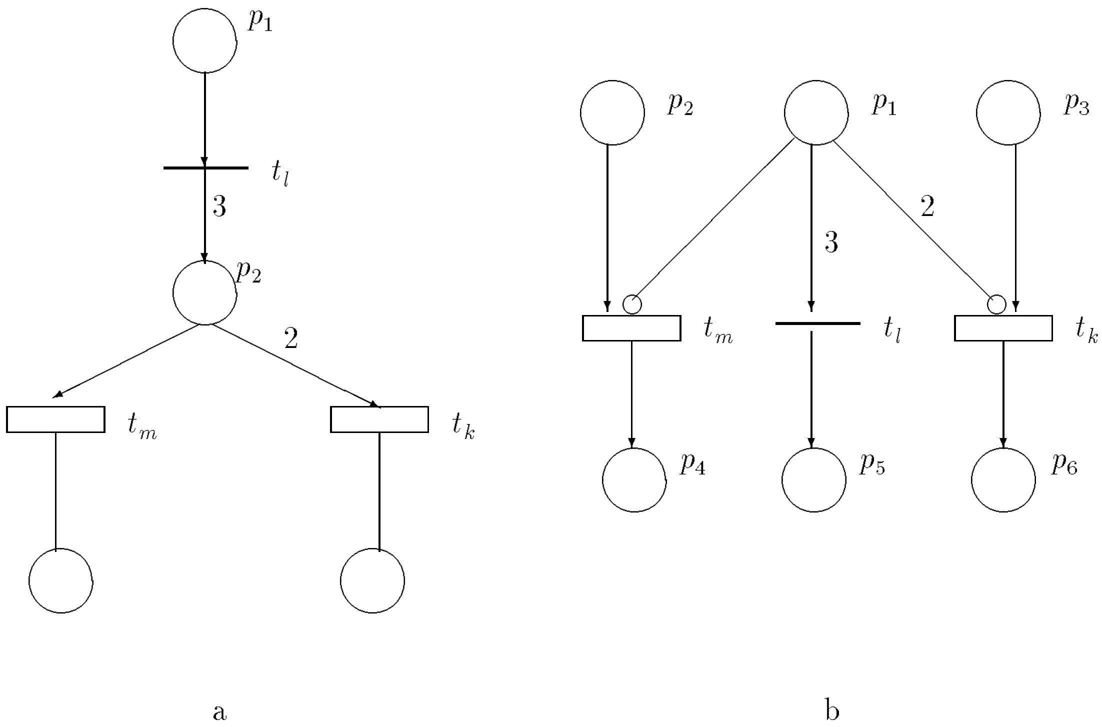


Fig. 6 Example of a Causual Connection

In case (a) of Fig. 6, after the immediate transition t_l fires inserting the corresponding tokens in its output place, if transition t_k fires, there will still be tokens placed by the firing of t_l in the input place of t_m that enable transition t_m, thus $t_l CC_{t_k} t_m$. In other cases it may only contribute to the enabling of the transition. In case (b) of Fig. 6 when t_l fires it will extract the corresponding number of tokens from its input place, which is not also an output place for t_l. As $h_{p_2}(t_k) > h_{p_2}(t_m)$ (i.e. $2 > 1$) and $(H(t_k) - H(t_m)) \cap (I(t_l) - O(t_l)) \neq \emptyset$, the firing of t_l will contribute to the enabling and possible firing of t_m, thus $t_l CC_{t_k} t_m$.

The transitions causally connected with a given transition form a Causally Connected Set (CCS). This set contains all the transitions that directly or indirectly could be enabled by transition t_m [11].

A sufficient condition for a PN to be *structurally confusion free* is:

$$\forall t_i \in T \ \forall t_j \in ECS(t_i) \wedge j \neq i \ CCS_{t_i}(t_j) = \emptyset$$

which means that a transition belonging to the same conflict set of another transition t can be causally connected only with transitions that are mutually exclusive of t itself. [11]

A PN is *k-bounded* for some integer $k \geq 1$ if for every state M in the reachability set, M(p_i)$\geq k$ for any place p_i $\in$P, i.e., no place in the Petri net will ever receive more than k tokens. A PN is *bounded* if it is *k-bounded* for some k.

A Petri net is a *free choice Petri net* (FCPN) if for every transition $t_j \in T$ and place $p_i \in I_j$ either (1) t_j is the only output transition of p_i or (2) p_i is the only input place of t_j. A Petri net is *persistent* if for every state M in the reachability set and any transitions t_i and t_j $(i \neq j)$, if t_i and t_j are enabled at M, then the sequence $t_i t_j$ is fireable at M. A Petri net is *conflict free* if each place $p_i \in P$ satifies either (1) there is at most one arc out of p_i or (2) $\forall t_j \ p_i \in I_j$ iff $p_i \in Oj$. Note that all conflict free nets are persistent but the converse is not true [13].

A Priority Petri Net (PPN) or a Petri Net with priorities is a Petri Net with a non-negative function $\prod(.)$ defined over the set of transitions which associates each transition with a non-negative number that represents its priority level [7]. The priority structure defined on transitions can be used to partition the reachability set of a Petri Net according to the priority level of a transition. From the priorities angle, GSPNs can be seen as priority Petri Nets, where immediate transitions have priority 1 and exponentially timed transitions have priority 0 .

The Tangible Reachability Set (TRS) of a PPN is the set of all tangible markings reachable from M_o, including $M0$ only if it enables a tangible marking. The Tangible Reachability Graph (TRG) of a PN is the labeled directed graph whose vertices are the elements of the TRS, and whose arcs (M_i, M_j) are labeled k indicating that marking M_j can be reached from marking M_i by firing a sequence where the only non-immediate transition is transition t_k.

A transition t_j is *live* at state M if for any fireable sequence σ at M there is a $\tau\in T^*$ such that $\sigma\tau t_j$ is fireable at M, where T^* is the set of all possible firing sequences including the null sequence where no transition fires. A Petri net is live if every transition of the net is live at the initial state [13].

Liveness is one of the most classic problems in the qualitative analysis of Petri net models. This problem is solvable at the structural level only for the class of FCPN, for which the Commoner's property holds:

> A marked free-choice net is live and bounded if and only if any deadlock contains at least one trap, which is not empty in the initial marking.

Where a marked FCPN, is a FCPN for which an initial marking has being defined. For larger classes of nets, only necessary or sufficient conditions can be determined at the structural level. For example,

> A bounded Petri net is live only if it is covered by T-invariants.

The actual determination of the liveness property is in general feasible for bounded nets only by looking at the structure of the Reachability Graph.

The computation of the actual liveness degree for each transition [4] is automatically performed after the computation of the Tangible Reachability Graph. Moreover, the structure of the TRG is checked in order to determine the maximal strongly connected components.

If only one maximal strongly connected component is found, and this component contains more than one tangible marking, then the model has a "home space" which is composed by the markings belonging to the strongly connected component. If the initial marking belongs to the home space then the TRG is strongly connected and each marking is reachable from all other markings.

If more than one maximal strongly connected component is found, then the model has several "livelocks" or "deadlocks", and no home state. A deadlock is a particular case of a strongly connected component of the TRG comprised of a single marking in which no transition is enabled.

A livelock is a subset of markings in which there is always at least one transition enabled, but such that other livelocks or deadlocks are not reachable.

If the initial marking is neither a home state nor a deadlock, then the model exhibits a transient behaviour, during which it reaches markings that might never be produced again.

In the case where the initial marking is found not to be a home state, the information concerning the liveness degree of transitions is split in three components:

1. the maximum enabling degree found in the transient states;
2. the maximum enabling degree among all livelocks;
3. the minimum among all livelocks and deadlocks of the maximum enabling degree within each livelock or deadlock (and this is called the liveness bound, in the sense that it is the maximum enabling degree that is guaranteed to be reachable in the long run of the model).

A place p is *implicit* in a net system iff its elimination preserves the firing sequence of the net system; in other words, p is never the unique place that prevents the firing of a transition [9].

2 GreatSPN1.6

GreatSPN (Graphical Editor Analyzer for Timed and Stochastic Petri Nets) was developed at the University of Torino, by Giovanni Chiola and collaborators. It is based on the notion of Generalized Stochastic Petri Nets (GSPNs), which was developed as a tool for the specification and performance evaluation of computers [18].

The primary purpose of the first version of GreatSPN was to experiment with new modelling tools and gain insight into the memory and CPU requirements of the solution algorithms as functions of the size of the GSPN models. This implied that little attention was devoted to either the portability and flexibility of the programs, or to the model definition facilities.

The first version of *GreatSPN* [6] included the algorithms for the generation of the underlying Markov Chain of a GSPN and for its steady-state and transient solutions. It also offered a new algorithm for the analysis of a class of models in which the timings for the transitions could be either exponentially distributed or deterministic (DSPN), and an algorithm for the analysis of new class of deterministically timed Petri Nets (DTPN).

The early GSPN (as well as DSPN and DTPN) analysis programs used to focus on exact numerical solution techniques. In *GreatSPN 1.0* a Monte Carlo simulation program with confidence level estimation was also introduced for two main reasons: first, to provide a tool for performance evaluation in the general case of Timed Petri Nets (TPN) that are not analytically solvable (i.e. nets with mixed random and deterministic timing that do not enjoy the DSPN applicability condition [2]), and second, to provide a tool for the validation of compact (possibly approximate) models when numerical solutions cannot be implemented due to the size of the Reachability Graph (number of states of the Markov chain).

At this point *GreatSPN* started to become an interesting and useful support for performance modelling. The Torino group started to look more carefully at the traditional techniques and algorithms used in classical Petri net theory for the study of qualitative structural and behavioural properties. A major improvement in the validation capabilities of the package was achieved with the implementation of Martinez and Silva's algorithm for the computation of Place and Transition invariants [20]. That allows an easy check of structurally necessary or sufficient conditions for boundedness and ergodicity before the exaustive enumeration of the state space. Some features such as the numerical solution of DSPNs and DTPNs were inefficiently implemented and have been subsequently removed since version 1.5 of the package.

In *GreatSPN1.6*, algorithms and techniques have been implemented for the reduction of immediate transitions, and the Tangible Reachability Graph is directly produced [7], further reducing the space and time requirements of this phase with respect to the technique proposed in [5].

Another new approach incorporated since version 1.5 of the package is that based on Linear Programming for the verification of some structural properties and the fast evaluation of throughput bounds. This approach has been developed in co-operation with the Universidad de Zaragoza, Spain.

In order to allow the visualization of the structural, behavioural, and performance results obtained by the analysis modules, *GreatSPN1.6* offers information on:

- Deadlocks, Traps, and implicit places
- Causal Connection and Liveness bounds
- Transition throughputs and probability distribution of the number of tokens in each place.

GreatSPN1.6 includes the validation of behavioural properties based on the same Tangible Reachability Graph that is used for the construction of the Embedded Markov Chain. The Tangible Reachability Graph of a priority net (i.e. a net in which transitions are assigned firing priorities) is obtained according to the definition given in [7], and qualitative analysis based on it are performed. Immediate transitions are divided in non-connected independent subnets, and inside each subnet, they are arbitrarily assigned a unique priority level as a function of the ECS they belong to. With this approach, the interleaving in the firing of immediate transitions belonging to different ECSs is avoided, since only one among the many possible paths is selected. In [7] it is proved that this technique does not affect the correctness of the TRG derived.

The Embedded Markov Chain underlying a GSPN model is constructed starting from the TRG of the underlying PN, and labelling the arcs with the correspon ding transition firing probabilities. The average mean sojourn time in each tangible marking is also computed by looking at the firing rates of the enabled transitions, and used to reconstruct the state probabilities of the continuous-time stochastic process by renormalization of the state probability distribution of the EMC.

The Steady-state numerical solution is obtained using an iterative Gauss-Seidel to solve th EMC. The numerical computation of the transient marking probability distribution at a given time with respect to the start time is obtained by the use of a matrix exponentiation algorithm based on a truncated Taylor's series expansion.

A number of default performance indices, as well as user defined ones, are calculated automatically when the steady-state or transient solutions are computed. The default performance indices are the probability distribution of the number of tokens in each place, and the throughput of each transition.

GreatSPN1.6 offers the facilities for the simulation of a Petri Net, both of the basic *untimed* model and of the *timed* Petri net.

- For the simulation of the basic untimed model the user can determine interactively the direction of each firing (backwards or foward). For each marking the enabled transitions will be highlighted and it is the user who selects the transition (out off the enabled ones) that will fire. We must remember that in the basic untimed model all the transitions have equal priority. The speed in which the tokens will move is determined by the field *tokens moves*, where the lower the number the faster the token movement. The markings of transitions can be change during the simulation process.

- For the timed Petri net simulation *GreatSPN1.6* supports the definition of diverse distributions for the determination of the delay of a transition.These are: the Normalised Cox distribution, the Hyperexponential distribution, the Erlang distribution and Linear and Discrete distributions. Timing semantics may also be specified: age/enabling memory for conflicting transitions and disabling and reanabling policy for multiple server transitions

  Default specifications are "enabling memory" with "random" preemptiion and reanabling policies.

  Example: *gd erlang 2 3.14* . Erlang distribution with two stages and mean time 3.14. With the default timing semantics.

  There are two basic ways in which a timed Petri Net simulation can be performed (the combination of these is also supported). In all cases the timed elapsed (according to the delay of the transitions, not real time) since the begining of the simulation is shown. This time is also used to determine the delay of the non-deterministic timed transitions in each marking, along with other random variables.

  - As in the untimed model the user determines the firing sequence, with the exception that in this case immediate transitions are considered to have a higher priority than timed transitions.
  - A step lenght is introduced,indicating the amount of firings that should be executed until the next interaction with the user. For example a step of length 10, represents the firing of 10 transitions (considering both immediate and timed ones). For a marking or step where no immediate transition is enabled and there is more than one timed transition enabled, the one to fire will be that with the smallest delay.

  The *automatic* option where the user introduces an amount of time, and the simulation would be performed until the amount of time elapsed equals or exceeds the specified by the user, is not yet available but should soon be incorporated.

In the case of transitions that can fire in parallel, although they will be sequentially fired, in the calculus of the elapsed timed they are treated as firing in parallel.

Basic Coloured nets are also supported under *GreatSPN1.6*. The user can define a Colour Set, a Colouring Function and/or a Coloured Marking. There is a defined BNF syntax for the definition of these elements.

Coloured sets can be defined in four different ways: by a element list, by automatic enumaration, by union or intersection of sets or by the Cartesian product of sets.

A Petri net can be represented in different layers (not to be confussed with hierarchy). This allows a creater flexibility in the size of the net, its modularity and the clarity of its representation.

The standard method for introducing a Petri Net model in *GreatSPN1.6* for its analysis is by a graphical interface offered by the package under Open Windows 3.0. The user "draws" the Petri Net model incorporating the places, immediate and stochastic timed transitions, arcs and the initial markings for each place. An alternative method would be to creat the files describing the net (.net and .def), for the package to read them. Once a file has been loaded into the graphical interface, the analysis methods described above can be applided through the graphical interface and its pop-up menus. The results of each analysis would be placed in detail in a respective output file and a summary of the results would be presented on the screen.

When needed, the structure of the TRG can be decoded and printed in ASCII form from outside the graphical interface.

All modules of *GreatSPN1.6* are written in the C programming language. It runs on Sun workstations under csh with the SunOS 4.1.3, using Open Windows 3.0 for its graphical interface. SPARC stations with at least 16MB RAM are recommended (although the tool could work slowly on 8MB machines). A complete porting under X11R5 is currently under study.

The package also includes printing facilities. Postscript files can be produced ready to be included into LaTeXdocuments by means of the "special" command.

Research is going on for the incorporation of new features in *GreatSPN*, such as: high-level (coloured) nets, hierarchical modelling, structural reduction of nets with interaction with queueing network models, among others.

3 DSPNexpress1.2

The development of this package was motivated by the lack of powerful software packages for the numerical solution of deterministic and stochastic Petri Nets (DSPNs) and the complexity requirements imposed by evaluating memory consistency models for multicomputer systems.

DSPNexpress [16] is based on version 1.4 of *GreatSPN*. I offers facilities for the analysis of the structural properties of a DSPN such as the calculus of the P-invariants, ECS and token movement simulation facilities.

It offers a graphical interface based on X11. To check or validate the structural properties of a DSPN, it calculates the P-invariants, ECS and offers a token movement simulation facility.

Compared to *GreatSPN1.6*, the software architecture of *DSPNexpress* is particulary tailored to the numerical evaluation of DSPNs.

DSPNexpress contains an efficient numerical algorithm for calculating the state transition probabilities of the embedded Markov Chain of a DSPN and the corresponding conversion factors. A similar algorithm is employed for calculating transient solutions of a GSPN. These numerical algorithms are based on the randomization technique (see subsection Randomization Technique) improved by a stable calculation of Poisson probabilities.

The DSPN solution module of *DSPNexpress* treats each connected component of a Markov chain as being subordinated to a deterministic transition of a DSPN, separately, for calculating the corresponding transition probabilities of the embedded Markov Chain and the conversion factors. This leads to a considerable reduction of the computational effort and memory requirements of the DSPN solution algorithm. Moreover, it allows multiple instances of the appropiate procedure to be invoked which may run in parallel on a cluster of workstations.

Using an extension of the DSPN solution process, DSPNexpress incorporates a numerical solution approach for dealing with marking dependent delays of deterministic transitions. The basic idea is to scale the transition by the delay specified for the corresponding marking. Thus, the general approach for dealing with marking-dependent firing delays has been tailored to the deterministic case. This feature constitutes an extension of the modeling power of DSPNs which is useful for representing load-dependent deterministic service times at a resource.

The organization of *DSPNexpress* exploits the property that each GSPN can be considered as a DSPN without deterministic transitions. As a consequence, a unified solution process for both DSPN and GSPN models is provided by *DSPNexpress*.

In *DSPNexpress*, sparse implementation of the direct Gaussian elimination and the iterative adaptive accelerated succesive overrelaxation method have been adopted. Depending on the properties of the transition probability matrix of the embedded Markov Chain the appropiate numerical method for solving the linear system of its global balance equations is chosen. If an iterative method is selected, the convergence will be monitored. In a case of bad convergence, the algorithm retries the calculation of the state probability vector by the stable Gaussian elimination method.

The reachability graph construction is performed for confusion-free DSPNs by a software component which is based on the method proposed in [11]. The coding structure of the search tree is organised as proposed by [5].The Markov chains subordinated to the deterministic transitions are derived during the reachability graph construction.

During the generation of the reachability graph of a DSPN, the Markov chains defined by exponential transitions competitively or concurrently enabled with deterministic solution, are derived. These Markov chains are subsequently called subordinated Markov chains (MC). For each connected component of such a subordinated MC, the transient state probabilities and the mean sojourn times in their states are efficiently calculated by the numerical algorithm introduced in [17].

The interaction between software modules of DSPNexpress is performed mostly by interprocess communication by means of sockets. Allowing parallel transient evaluation of the subordinated MC on different machines.

The module in charge of the solution of the subsequent MC, first determines the transition rates of markings in which only exponential transitions are enabled. Additionally, for each deterministic transition in the DSPN the transition matrices of the connected components of its subordinated MC are derived.

The numerical solution method of DSPN requires that in no marking are two or more transitions concurrently enabled, i.e. the states of subordinated the MC build disjoint sets, allowing parallelism in the solution.

The code of DSPN is a combination of C and Fortran 77 modules, but mostly C. DSPNexpress includes support for the incorporation of marking dependent firing delays on deterministic transitions. Since the memoryless property does not hold for deterministic distributions, additional semantics have to be considered to properly define the association of marking-dependent firing delays to deterministic transitions. In cases where the marking condition which specifies the marking-dependent delay of a deterministic transition does not remain constant during its enabling interval, an execution policy of its firing must be specified.

3.1 Randomization Technique

Several numerical techniques for computing time-dependent state probabilities of a continuous-time Markov chain with finite state space and sparse generator matrix have been evaluated by Reibman and Triverdi [24]. In particular, the randomization technique [12] and two linear multi-step methods for numerically solving the Chapman-Kolmogorov differential equation have been considered. All theses methods have implementations which exploit the sparsity of the generator matrix Q. Their study has shown that in case of nonstiff and moderately stiff problems the randomization approach is more efficient than general solution techniques for differential equations. The Randomization Technique consists of the following: From the generator matrix Q of dimension N (The number of states in the chain) of a continuous-time Markov chain and a scalar q (as definied by Wallace and Rosenberg in [26]) an aperiodic matrix A is obtained, which represents the generator matrix of the subordinated discrete Markov chain.

The computation of the transient probability vector of the continuous-time Markov chain is reduced to the computation of the transient probability vector of the discrete-time Markov chain with probability matrix A and appropriate Poisson probabilities. The probability vector of the discrete-time Markov chain can be effeciently computed by recursive vector-matrix multiplications.

The probability mass function of the Poisson distribution thins when the parameter of the function grows, so its computation may get affected by round-off errors. In this sense the randomization technique is enhanced by a stable calculation of the Poisson probabilities by establishing a lower and higher bound of the generating series.

The randomization approach is suitable for calculating other transient quantities such as transient probabilities at multiples instants of time and cumulative measures.

3.2 Numerical solution for the DSPNs

Sampling the stochastic behaviour of the DSPN only at appropriately selected instants of time, defines a discrete-time stochastic point process which possesses the Markovian property. Where only exponential transitions are enabled the stochastic behaviour of the DSPN is sampled at the instant of its firing. If a deterministic transition is competitively enabled with some exponential transitions, the stochastic behaviour is sampled when either the deterministic or the exponential transition fires. If a deterministic transition is concurrently enabled with some exponential transition, the stochastic behaviour is sampled at the instant of time of firing the deterministic transition.

As a consequence, a marking change in the DSPN due to firing of an exponential transition concurrently enabled with a deterministic one is not represented in the embedded Markov chain by a corresponding state change. Thus, the embedded Markov chain typically contains fewer states than the number of tangible markings of the DSPN. This continuous-time Markov chain is subsequently refered to as the subordinated Markov chain of a deterministic transition.

In the first step of the DSPN solution process the Reachability Graph is generated, following the algorithm proposed in [11].

The second step is the calculation of the transition probability matrix P of the embedded Markov chain and the conversion matrix C. For markings which enable a deterministic transition T_k competitively or concurrently with some exponential transitions the corresponding entries of these matrices are derived by calculating time-dependent quantities of the subordinated Markov chain. Transient state probabilities of the Markov chain subordinated to the deterministic transition T_k determine the corresponding transtion probabilities of the embedded Markov chain.

Conversion factors are employed to derive the steady-state probability vector of the DSPN of dimension N from the steady-state probability vector of the embedded Markov chain of dimension N'.

4 SPNP Version 3.0

SPNP (Stochastic Petri Net Package) Version3.0 was developed at Duke Universityby Kishor Trivedi together with his former students Gianfranco Ciardo and Jogesh Muppala.

SPNP can run on a wide range of UNIX platforms such as VAX, Sun 3 and 4, Convex, Gould and under the VMS system on VAX.

When in a SPN transitions of different types are simultaneously enabled, the decision of which one would fire first is based on priority values that must be introduced.

Complementary to the concept of inhibiting arc, an enabling/disabling function is added, in order to allow complex constructions. To each transition a priority must be assigned, to be able to solve the firing problem in the case of conflicting transitions. To avoid theoretical difficulties, timed and immediate transitions cannot have the same priority. If the enabling function evaluates to zero in a certain marking then the corresponding transition would be disabled.

An interesting feature allowed in the *SPNP* model, which differs from the other packages, is the notion of marking dependent arc multiplicity. This possibility was defined to be able to model, in a compact way, behaviours that would otherwise require complex subnets [8]. A typical example is the case where all tokens from place p must be moved to place q when transition t fires. An input arc from p to t and an output arc from t to q, both with marking dependent multiplicity equal to the number of tokens in place p are enough to model this behaviour. Of course this could make the SPN harder to understand, and would consume more computation time for the solution of the model.

One of the main contributions of the development of *SPNP* is the implementation of algorithms for an efficient generation of the reachability graph of large SPNs and an automated sensitivity analysis of model parameters. Moreover, *SPNP* can process very general reward specifications which are useful for integrated and dependable evaluation of complex systems. The package also contains an efficient software module for transient analysis of GSPNs which is based on the randomization technique with left truncation. Nevertheless *SPNP* does not contain a numerical solution algorithm for DSPNs.

The description of the PN to be analysed must be described in a CSPL (C-based Stochastic Petri Net Language) file [8], which is a C file specifying the structure of the SPN and the desired outputs, by means of predefined functions. The CSPL file has then to be compiled, linked to other files constituting the package and run.

Any legal C construct can be used, as needed. The user is allowed to define his or her own variables and functions and use them anywhere in the CSPL file.

A CSPL file must specify the following functions:

parameters: where desired output files, method of resolution, precision, and other program features are specified. In this section additional user's parameter may be specified. They can be programed in such a way that they are introduced interactively, by the use of the function *input*.

net: where the structure of the net is introduced. Specifying the time rates of each timed transition, the probability of each immediate transition, this being the way to differentiate the type of transition; and for all conflicting transitions the firing priority. While solving the net, the program would report any incorrect definition or the existence of conflicting transitions whose firing priorities have not been specified. By default transitions have the lowest priority (0) and have no enabling function (or better, their enabling function is set to the constant 1).

When sensitivity analysis is needed, both the rate and its derivative need to be specified (no default type, rate or probability exists).

Marking dependent functions can be defined for the multiplicity of an arc or for the firing rate of a transitions.

assert(): This is called during reachability graph construction, to check the validity of each newly found marking. The user can add conditions to the normal check in order to specify systems of greater complexity. Nevertheless absence of legal markings or legal firing sequences cannot be detected by this check.

*ac*_init(): This is called before starting reachability graph construction. In it we can call the predefined function: pr_net_info(); to output data about the SPN.

In the function (or section) *ac*_reach we can call pr_rg_info() to obtain information on the reachability graph.

When either the steady-state analysis or steady-state sensitivity analysis is requested, *ac*_final is called after the solution of the CTMC has completed, supplying user-prerequested values, specified by:

pr_mc_info(); For information on the CTMC and its solution. Desired output measures must be requested in this function.

pr_std_average(); Outputs for each place the probability that it is not empty and its number of tokens. For each timed transition it outputs the probability that it is enabled and its average throughput.

pr_std_average_der(); Prints the derivatives of all the above standard measures. In this function we can include reward values for the transitions.

When transient analysis or transient sensitivity analysis is required, *ac*_final is called before the solution of the CTMC. For this type of analysis a time point needs to be specified.

4.1 Specialized output functions

The package was initially aimed at the steady-state solution of SPNs whose underlying CTMC is ergodic. There are a number of measures which could be considered "unusual". but closely related to the steady-state. In particular, they do not require the implementation of a new-solver; they can be computed either from the steady-state probabilities or by solving a slightly different (non-homogeneous) linear system.

These measures were defined and implemented to perform decomposition-iteration techniques allowing approximated solution of SPNs whose state space is too large to be studied directly.

4.2 Planned work

A simulation solution method will be available. General or empirical distributions will be allowed in conjunction with simulation.

The definition of a higher-level language, or specifically the availability of a graphical input tool, will improve user-friendliness.

Global check on the absence of legal markings or firing sequence. An alternative standard format containing the distributions, instead of the averages, is under preparation.

5 Comparison

In this section we try to summarise the features provided by each of the packages reviewed in this paper, at the same time as we compare them.

The features have been divided into several categories in other to facilitate the comparison of the packages.

To point out the common and different features of the packages we have chosen a simple example (Fig. 7). Printouts of the results offered by each package are presented in the appendix A for better understanding of the features discussed.

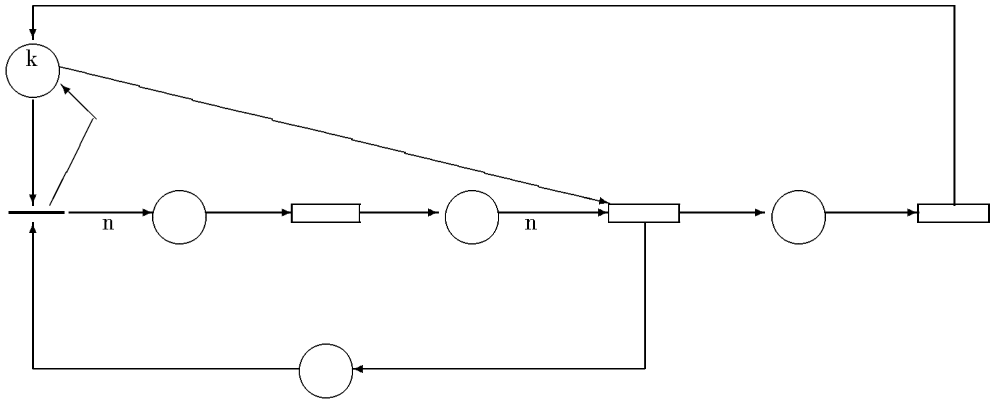


Fig.7.- Erlang queue Er/D/1/K, with $\lambda = 9$, $\tau = .1$, 5 phases and 3 buffers (K)

5.1 Structural Analysis

All three packages offer basic information on the structural composition of the net. Number of vanishing, tangible and absorbing markings (See Appendix A for the results offered by each of the packages for the example proposed).

A feature only offered by *SPNP3.0* is the preservation of vanishing markings in the construction of the reachability graph. It also permits the user to add conditions to the normal check of the net in order to detect illegal markings.

GreatSPN1.6 offers extensive features for structural analysis, such as: place and transition invariants, deadlocks and traps, implicit places, causal connection and structural conflict, mutual exclusion, ECS, structural boundness and unbounded transition sequences. The reachability graph construction is done for confusion-free nets (See Appendix A).

DSPNexpress only considers some features for the structural analysis (Place-invariants and ECS), also obtaining the reachability graph for confusion-free nets.

5.2 SPN extensions

This refers concepts additional to the basic SPN available in the packages. All three packages work with exponential and immediate transitions. *DSPNexpess* adds the possibility of incorporating deterministic timed transitions to the net.

SPNP3.0 allows the definition of marking dependent graph multiplicity. It also offers the definition of enabling functions, apart from the basic inhibiting arcs. For the sake of sensitivity analysis it supports the incorporation of reward values and functions.

DSPNexpress permits the definition of marking dependent firing times for deterministically timed transitions. It also allows the analysis of the net by the variation of a marking or a delay parameter in a certain range (see Fig. A.2.4 of Appendix A).

*GreatSPN*1.6 incorporates the definition of colouring sets and markings.

5.3 Numerical Steady State Analysis for GSPNs

SPNP3.0 offers the option of applying either an iterative near optimal SOR or a Gauss-Seidel solution method. On the other hand *GreatSPN1.6* directly applies an iterative Gauss-Seidel.

DSPNexpress offers the following options for the solution of the linear system of equations of the Steady state solution:

- automatic: an adaptive accelerated SOR with convergence monitoring, which in case of failure applies a sparse implementation of direct Gaussian Elimination.
- iterative: applies an adaptive accelerated SOR
- direct: applies a sparse implementation of a direct Gaussian Elimination

also offering the option of a sequential or parallel execution of the algorithms for the solution of the sub-Markov Chains.

5.4 Numerical transient analysis for GSPNs

In *SPNP3.0* randomization is enhanced by left-truncation and steady-state check, *GreatSPN1.6* uses an adaptive matrix exponentiation and in *DSPNexpress* randomization is enhanced by a stable calculation of Poisson Probabilities.

5.5 Stochastic simulation of SPNs with arbitrary timing

GreatSPN1.6 offers a Discrete event simulator, where several probability distributions are allowed for transition timings (including uniform, discrete, Cox, Erlang, etc), allowing timed interactive simulation for quantitative validation.

DSPNexpress also offers a basic Discrete event simulator .

In *SPNP3.0* this feature is not supplied, nevertheless it is being considered as an element to be incorporated in future versions of the software.

5.6 Model Composition or decomposition

Only *SPNP3.0* supports automated GSPN composition. The other packages do not offer anything in this sense, although the hierarchical analysis of the GSPNs is a feature under study by the group developing *GreatSPN1.6*.

5.7 Non-standard techniques

SPNP3.0 offers automated sensitivity analysis and methods for the calculation of the mean time to absorption.

5.8 Interaction between software modules

In respect of the way in which the different modules of the packages communicate *DSPNexpress* has the advantage of using a reliable interprocess communication by means of sockets. The modules of *SPNP3.0* and *GreatSPN1.6* communicate by the use of input/output files.

5.9 User interface

SPNP3.0 employs CSPL (C-based Stochastic Petri Net language), but at the moment a graphical user interface is under development.*GreatSPN1.6* offers a graphical interface running under Open Windows 3 , and *DSPNexpress* uses a graphical interface running under X11.

5.10 Portability

SPNP3.0 can run on a wide range of UNIX platforms such as VAX, Sun 3 and 4, Convex and Gould; and under VMS system on VAX.

DSPNexpress, can run on a compatible environment to a Sun^{TM} 4 workstation under SunOS4.1 and the window system X11.

GreatSPN1.6 Runs on SUN workstations using Open Windows 3.0. A complete porting under X11R5 is currently under study.

6 Intended and Possible Uses

The design and specification of a program allowing static analysis, and thus yielding information on possible deadlocks, mutual exclusion and resource utlization is highly desirable [10].

We need a formalism that is capable of representing both the characteristics of the architecture and the pecullarities of the program on a parallel computer in such a way that both validation and performance evaluation can be performed using basically the same model [3].

Petri nets, as well as process algebras, allow a precise description of the system due to their formal syntax, behavioural semantics, algebraic reasoning, deduction of properties and equational transformations preserving behaviour [21],[3] and [10].

The analysis of the performance of a program on a certain architecture, normally requires the use of two Petri nets, one representing the structure and characteristics of the architecture and the other representing the structure and needs the software application. The Petri net representing the architecture (hardware) characteristics is initially solved to produce the average processing times for the different types of existing tasks in the sofware specification, normally qualified by their resource requirements. The solution of the Petri net representing the hardware also yields information on the hardware behaviour in general as stated before.

In the Petri net representing the software application the transitions would represent the tasks or processes and the token flow indicates data and control flow among the tasks.

The fact of having two nets to represent the computational system retricts the flexibility in the analysis of the performance of a software application by varying the characteristics of the hardware, such as resource availability, policy of assignment, etc. In order to support this integration of the software and hardware specifications of the system, the Petri net representation should supply the ways of distinguishing the elements represented in the net. Transitions and places could be distinguished by tags but to distinguish the different types of tokens we would require a marking colouring facility.

In order to use Petri nets as a method for the specification and design of parallel programs, the package would have to support the fact of simultaneous firing of transitions, both of exponential and deterministic transitions. Given the complexity involved in trying to incorporate this feature by the use of numerical algorithms, maybe this requirement could be satisfied by the use of analytical approximations, as proposed in [10] and [14]. In addition possible conversion factors could be required in order to obtain better approximations.

Performance requirements must be effectively captured in parallel programs specifications. Specification methods should support functional and temporal specifications, suitably representing various aspects of parallel systems, such as software (control-flow, data flow, communication,synchronization, non-determinism, etc.) and hardware (resources like processing elements, memory modules, communication, media, etc.), by simple but expressive graphical means. The capability of investigating on various levels of abstraction (i.e. providing concepts of modularity and hierarchical decomposability) for (automated) analysis concerning performance, functional validity, correctness, expected behaviour, support for generating executable and analysable application prototypes and generation of high level language source code are other properties desired to be supported [10].

We would like to be able to study the different levels of Petri nets, in a hierarchical specification of a program. Trying to consider each net as a whole, by "supplying" it with the resources needed and then integrating the solution of this Petri net as a compound transition of a higher level Petri net.

7 Conclusions

To introduce the user to the Petri Net modeling tools reviewed in this paper an initial overview of the main Petri Nets concepts employed when using PNs as System Modeling tools for performance measurement was given.

The information that we have supplied for each of the packages reviewed, has come both from the experience gained by working with each package and from the existing documentation for each one.

None of the packages studied offer all the modeling features that we intend to apply in the Performance Oriented Specification and Design Technique for Parallel Programs. We should however keep in mind that this technique has not yet been completely formalised and that the developers of packages studied are still incorporating new modeling features to their packages.

References

[1] M. Ajmone. Stochastic petri nets: An elementary introduction. *Advances in Petri Nets 1989. Lecture Notes on Computer Science.Grzegorz Rozenberg (Ed.)*, pages 1–29, 1989.

[2] M. Ajmone and G. Chiola. On petri nets with deterministic and exponentially distributed firing times. *Advances in Petri Nets 1987. Lecture Notes in Computer Science. G. Rozenberg ed. Springer, Berlin*, 266:132–145, 1987.

[3] G. Balbo, S. Donatelli, and G. Franceschinis. Understanding parallel program behavior through petri net models. *Journal of Parallel and Distributed Computing 15*, 15:171–187, 1992.

[4] J. Campos, G. Chiola, J.M. Colom, and M. Silva. Tight polynomial bounds for steady-state performance of marked graphs. *Proceedings of the 3rd International Workshop on Petri Nets and Performance Models. Kyoyo, Japan*, December 1989.

[5] G. Chiola. Compiling techniques for the analysis of stochastic petri nets. *Proceedings of the 4th International Conference on Modeling Techniques and Tools for Computer Performance Evaluation. Palma de Mallorca, Spain*, September 1988.

[6] G. Chiola. Greatspn1.5 software architecture. *Proceedings of the 5th International Conference on Modeling Techniques and Tools for Performance Analysis. Torino, Italy*, pages 117–132, 1991.

[7] G. Chiola, G. Ajmone Marsan, M. amd Balbo, and G. Conte. Generalized stochastic Petri nets: A definition at the net level and its implications. *IEEE Transactions on Software Engineering*, 19(2):89–107, feb 1993.

[8] G. Ciardo, J. Muppala, and K.S. Trivedi. Manual for the spnp package version 3.0, May 1991.

[9] J.M. Colom and M. Silva. Improving the linearly characterization of p/t nets. *Advances in Petri Nets 1990. Lecture Notes in Computer Science. G. Rozenberg ed.. Springer, Berlin.*, 483:113–145, 1991.

[10] A. Ferscha. A petri net approach for performance oriented parallel program design. *Journal of Parallel and Distributed Computing*, 15:188–206, 1992.

[11] Balbo G., Chiola G., Franceschinis G., and Molinar G. On the efficient construction of the tangible reachability graph pof generalized stochastic petri nets. *Proceedings of the 2nd International Workshop on Petri Nets and Performance Models. Madison, U.S.A.*, pages 136–145, 1987.

[12] D. Gross and D.R. Miller. The randomization technique as a modeling tool an solution procedure for transient markov processes. *Oper. Res.*, 32:345–361, 1984.

[13] N. Jones, L. Landweber, and Y. E. Lien. Complexity of some problems in petri nets. *Theoretical Computer Science 4. North-Holland Publishing Company*, (4):227–299, 1977.

[14] A. Kapelnikov, R. Muntz, and M. Ercegovac. A modeling methodology for the analysis of concurrent systems and computations. *Journal of Parallel Computing*, 6:568–597, 1989.

[15] C. Lin, A. Chandhury, A. Whinston, and D. Marinescu. Logical inference on horn clauses in petri net models. Technical report, Purdue University (CSD-TR-91-031), 1991.

[16] C. Lindemann. Dspnexpress: A software package for the efficient solution of deterministic and stochastic petri nets. *Proceedings of the 6th International Conference on Modeling Techniques and Tools for Computer Performance Evaluation. Edinburgh, Scotland. Rob Pooley and Janes Hillston (Ed.)*, pages 15–30, 1992.

[17] C. Lindemann. An improved numerical algorithm for calculating the steady-state solutions of deterministic and stochastic petri net models. *Performance Evaluation 18. Elsevier Science Publishers B.V. (North-Holland)*, pages 79–95, 1993.

[18] Ajmone M., Balbo G., and Chiola G. A class of generalized stochastic petri nets for performance analysis of multiprocessor systems. *ACM Transactions on Computer Systems*, 2, May 1984.

[19] D. Marinescu, M. Beaven, and R. Stansifer. A parallel algorithm for computing invariants of petri net models. *Proceedings of the 4th International Workshop on Petri Nets and Performance Models. Melbourne, Australia*, pages 136–143, 1991.

[20] J. Martinez and M. Silva. A simple and fast algorithm to obtain all invariants of a generalized petri net. *Proceedings of the 2nd European Workshop on Application and Theory of Petri Nets. Springer Verlag, Bad Honnet, West Germany.*, September 1981.

[21] A. Moore. The specification and verified decomposition of system requirements using csp. *IEEE Transactions on Software Engineering*, 16(4):932–947, September 1990.

[22] T. Murata, B. Shenker, and S.M. Shatz. Detection of ada static deadlock using petri net invariants. *IEEE Transactions on Software Engineering*, 15, No 3:314–326, 1989.

[23] J.L. Peterson. *Petri Net Theory and The Modeling of Systems.* Englewood Cliffs,NJ: Prentice-Hall, 1981.

[24] A. Reibman and K. Trivedi. Numerical transient analysis of markov models. *Comput. Oper. Res.*, 15(1):19–36, 1988.

[25] W. Reisig. *Petri Nets. An Introduction.* Springer-Verlag, 1985.

[26] V. L. Wallace and R.S. Rosenberg. Markovian models and numerical analysis of computer systems behavior. *Proc. ACM-IEEE Comp. Soc. Spring Joint Comp. Conf.*, pages 141–148, 1966.

8 Appendix A

8.1 Some features of *GreatSPN1.6*

Fig. A.1.1.- General view of the package's interface

Fig. A.1.2.- Graphical representation of the places forming a deadlock

Fig. A.1.3.- Graphical representation of the T-invariants

Fig. A.1.4.- Untimed simulation

Fig. A.1.5.- Timed simulation

Fig. A.1.6.- Numerical analysis results for the Erlang queue example

Fig. A.1.1.- General view of the package's interface

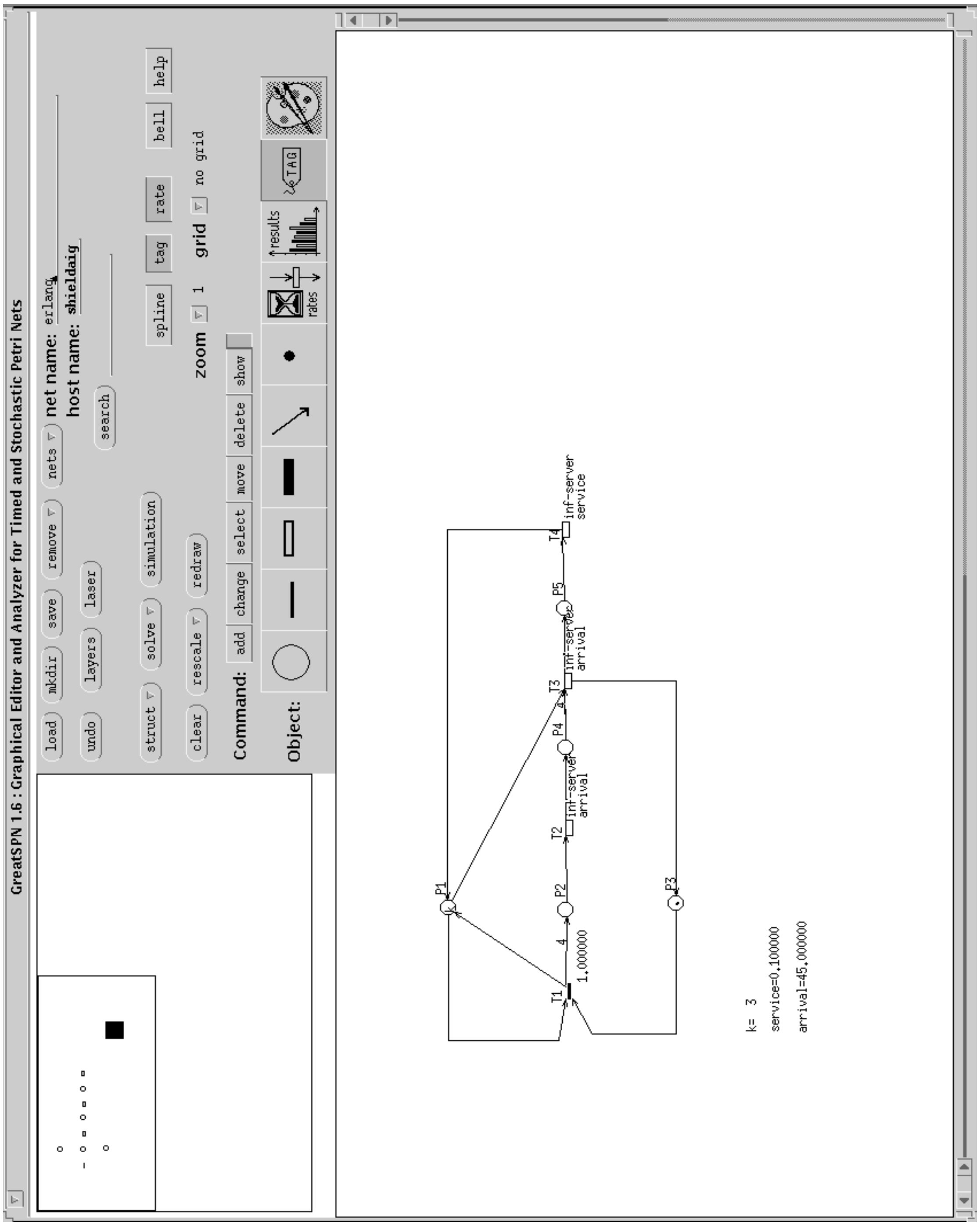


Fig. A.1.2.- Graphical representation of the places forming a deadlock

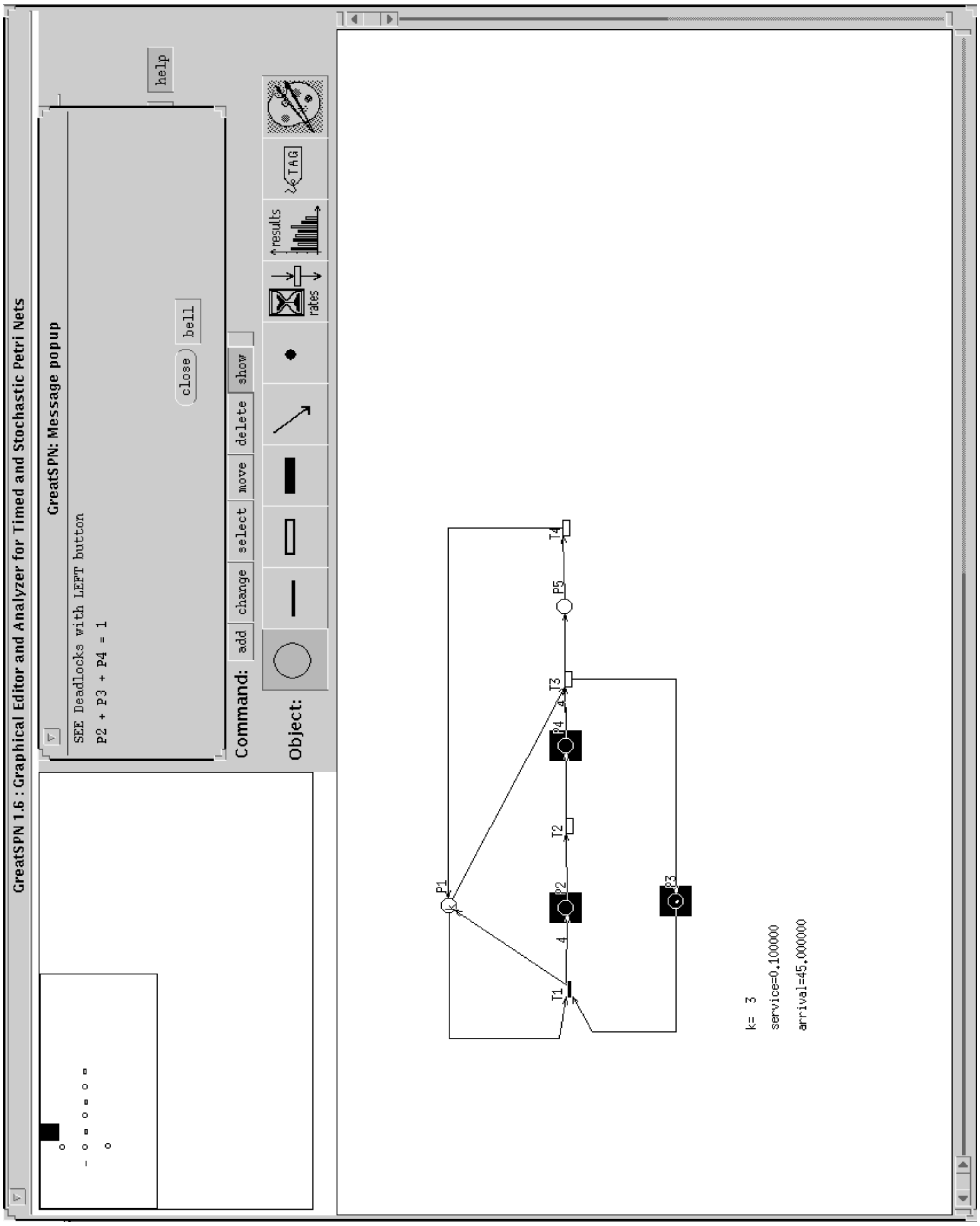


Fig. A.1.3.- Graphical representation of the T-invariants

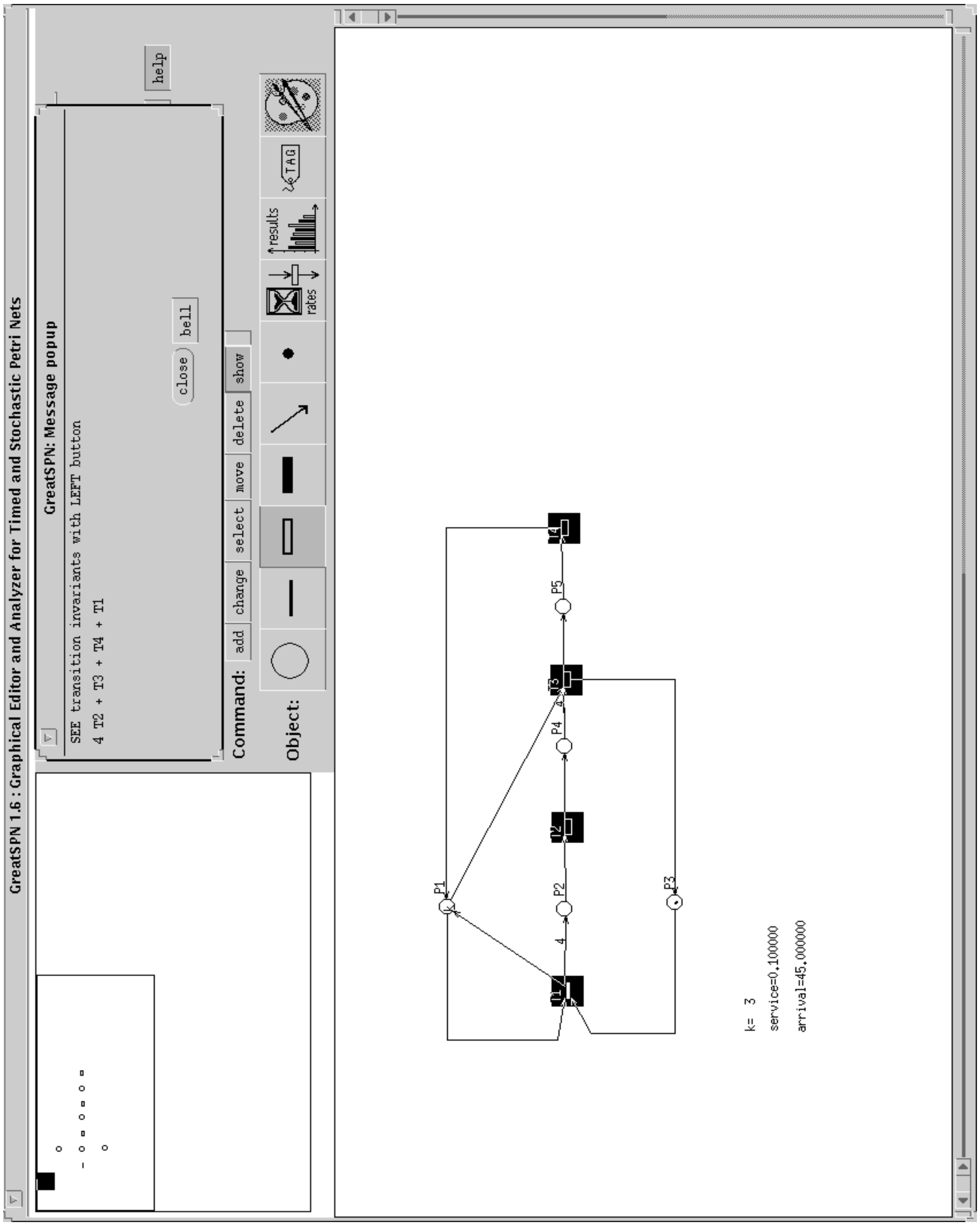


Fig. A.1.4.- Untimed simulation

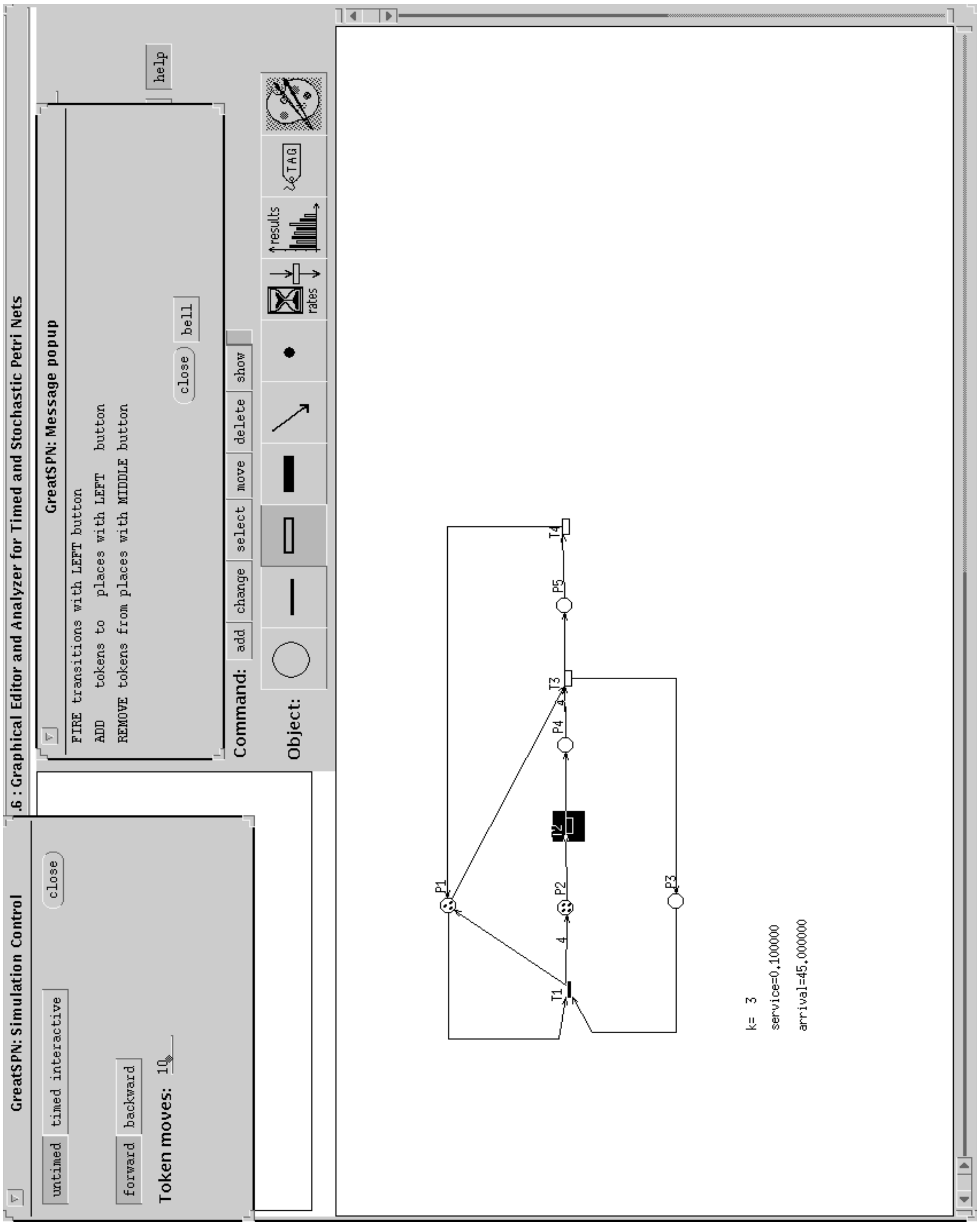


Fig. A.1.5.- Timed simulation

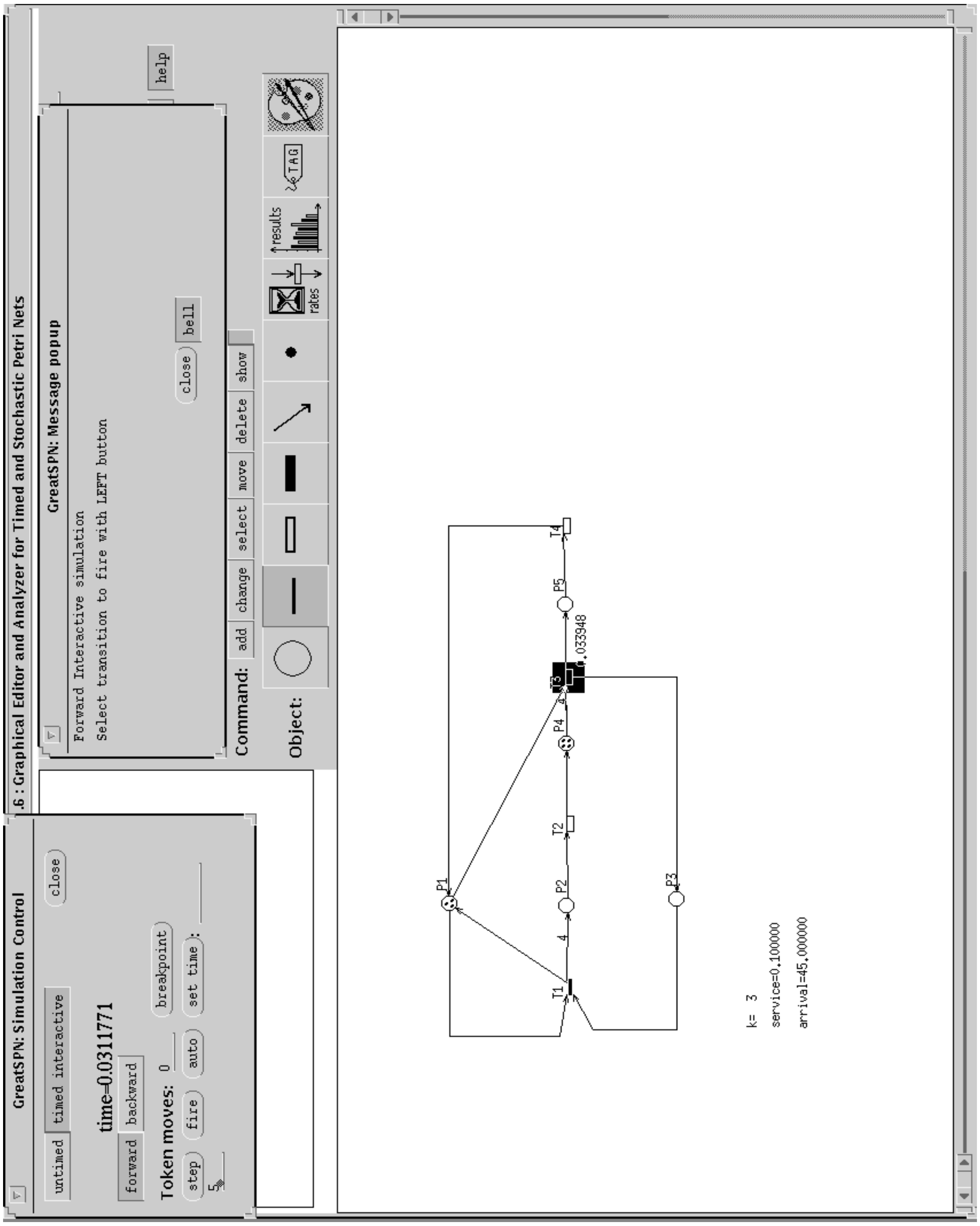
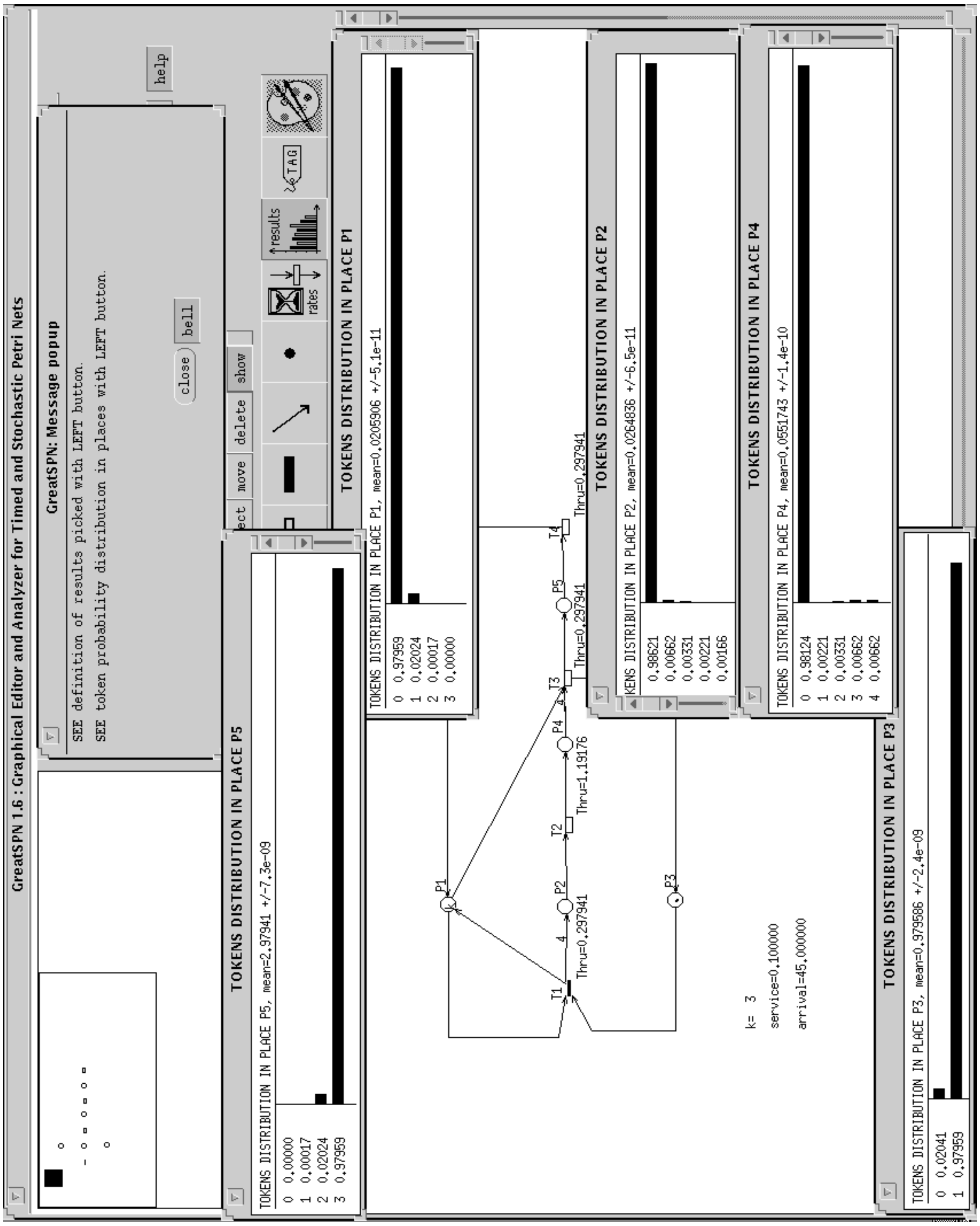


Fig. A.1.6.- Numerical analysis results for the Erlang queue example



8.2 Some features of *DSPNexpress1.2*

Fig. A.2.1.- General view of the package's interface

Fig. A.2.2.- Calculation of the P-Invariants

Fig. A.2.3.- Selection of the solution method for a DSPN

Fig. A.2.4.- Defining a experiment

Fig. A.2.5.- Transition firing simulation in *DSPNexpress1.2*

Fig. A.2.6.- Numerical analysis results for the Erlang queue example

Fig. A.2.7.- Log file for the Erlang queue example

Fig. A.2.1.- General view of the package's interface

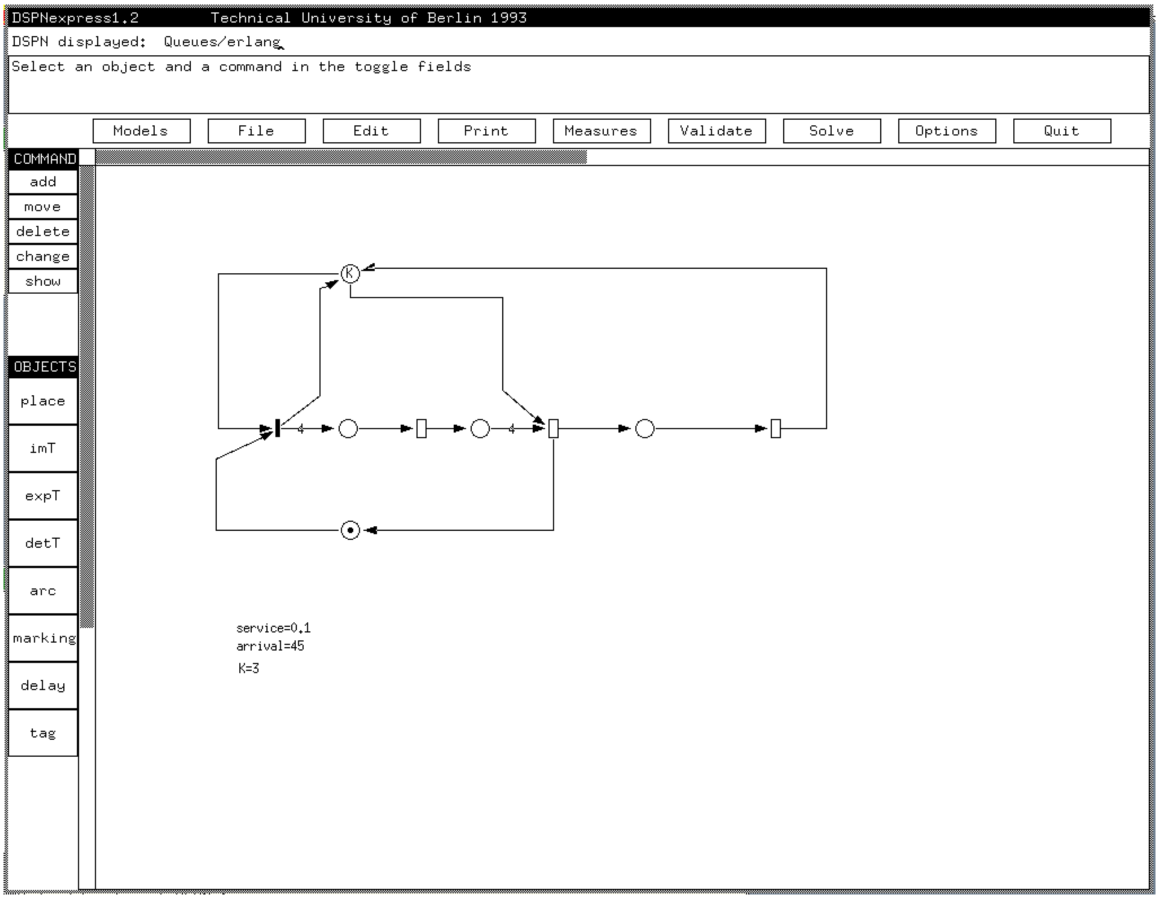


Fig. A.2.2.- Calculation of the P-Invariants

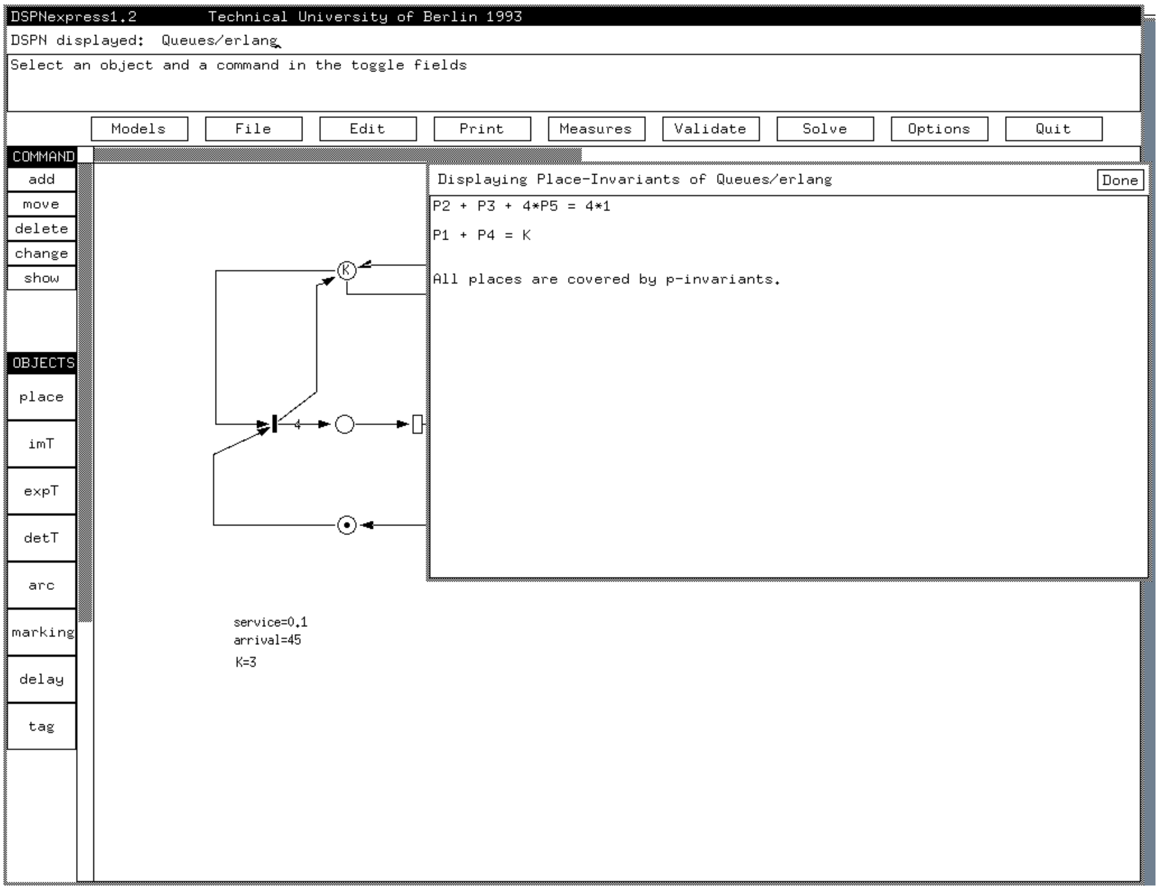


Fig. A.2.3.- Selection of the solution method for a DSPN

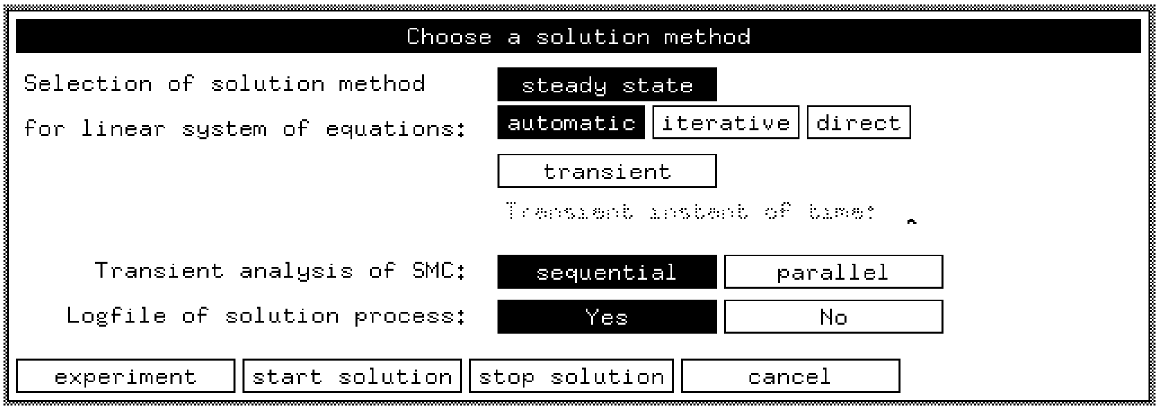


Fig. A.2.4.- Defining a experiment

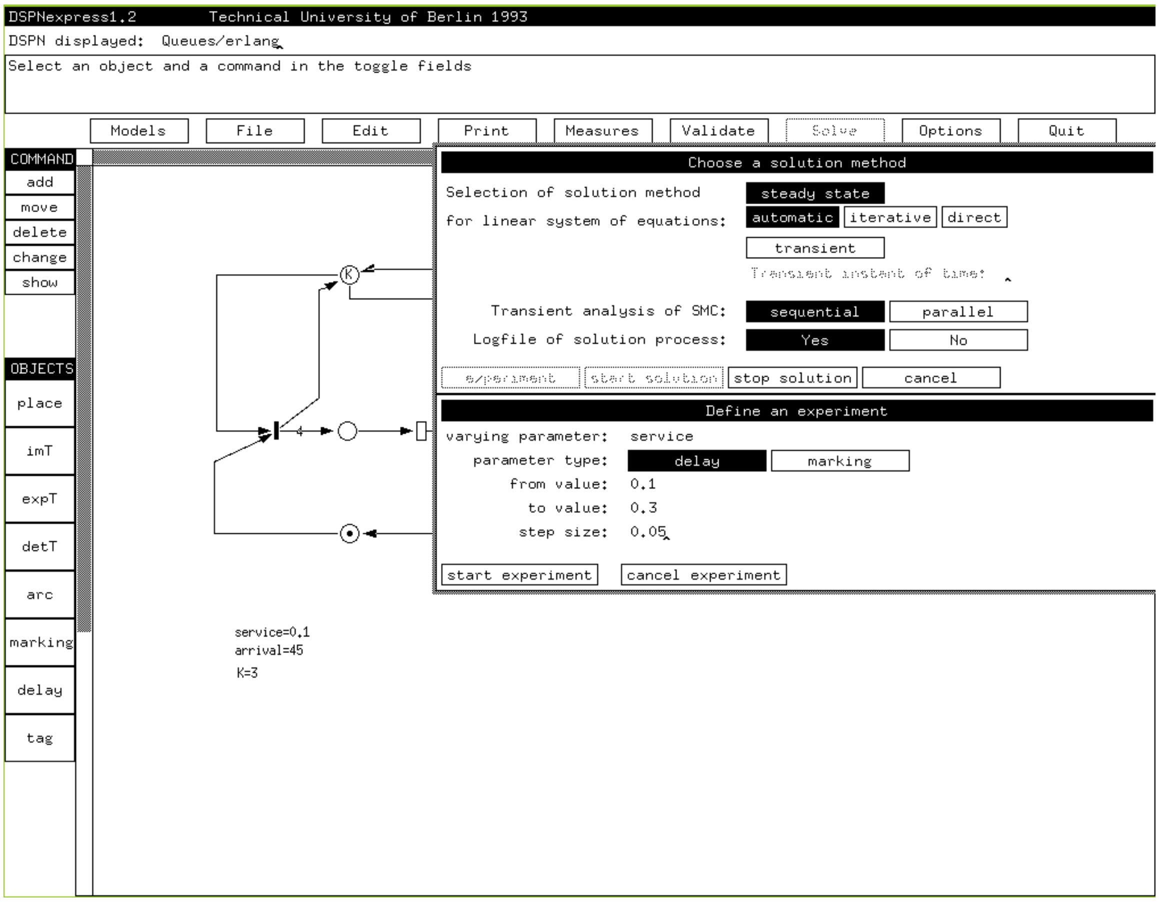


Fig. A.2.5.- Transition firing simulation in *DSPNexpress1.2*

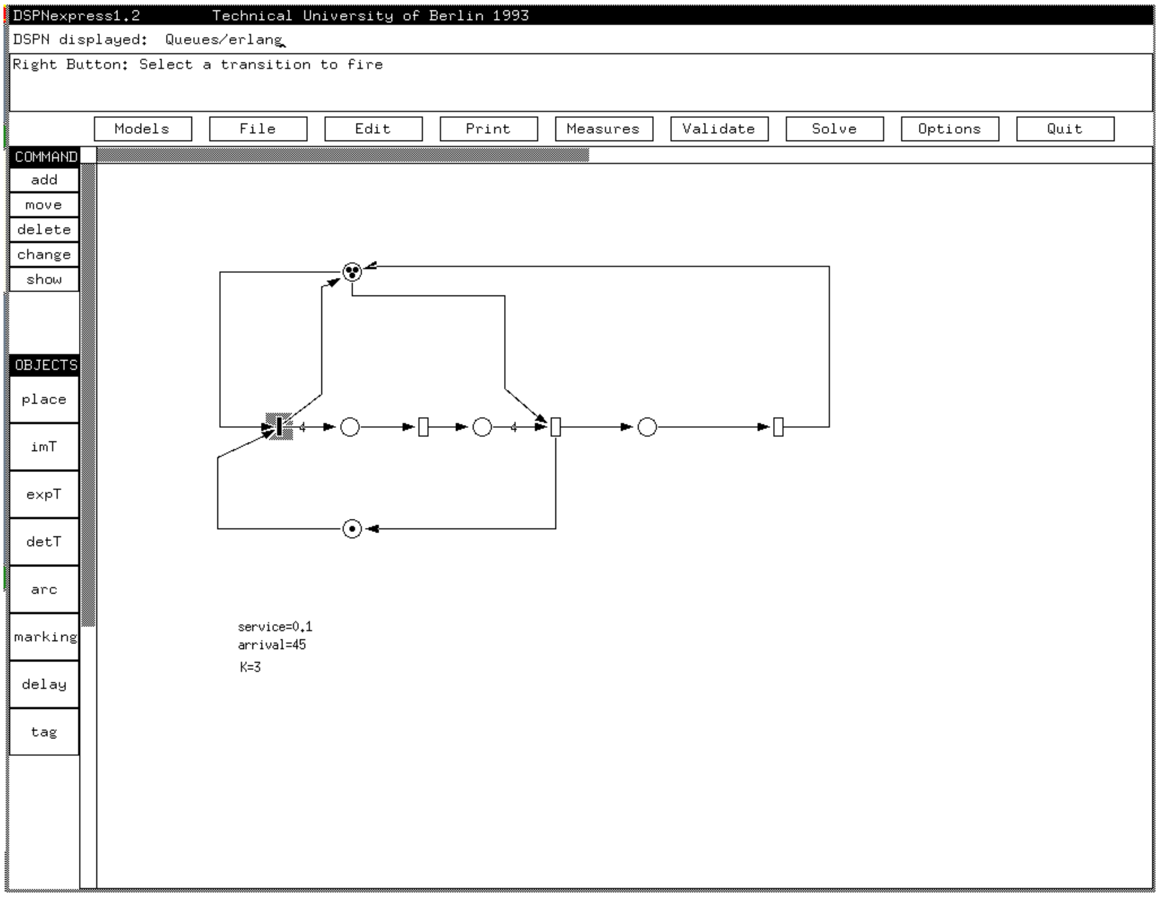


Fig. A.2.6.- Numerical analysis results for the Erlang queue example

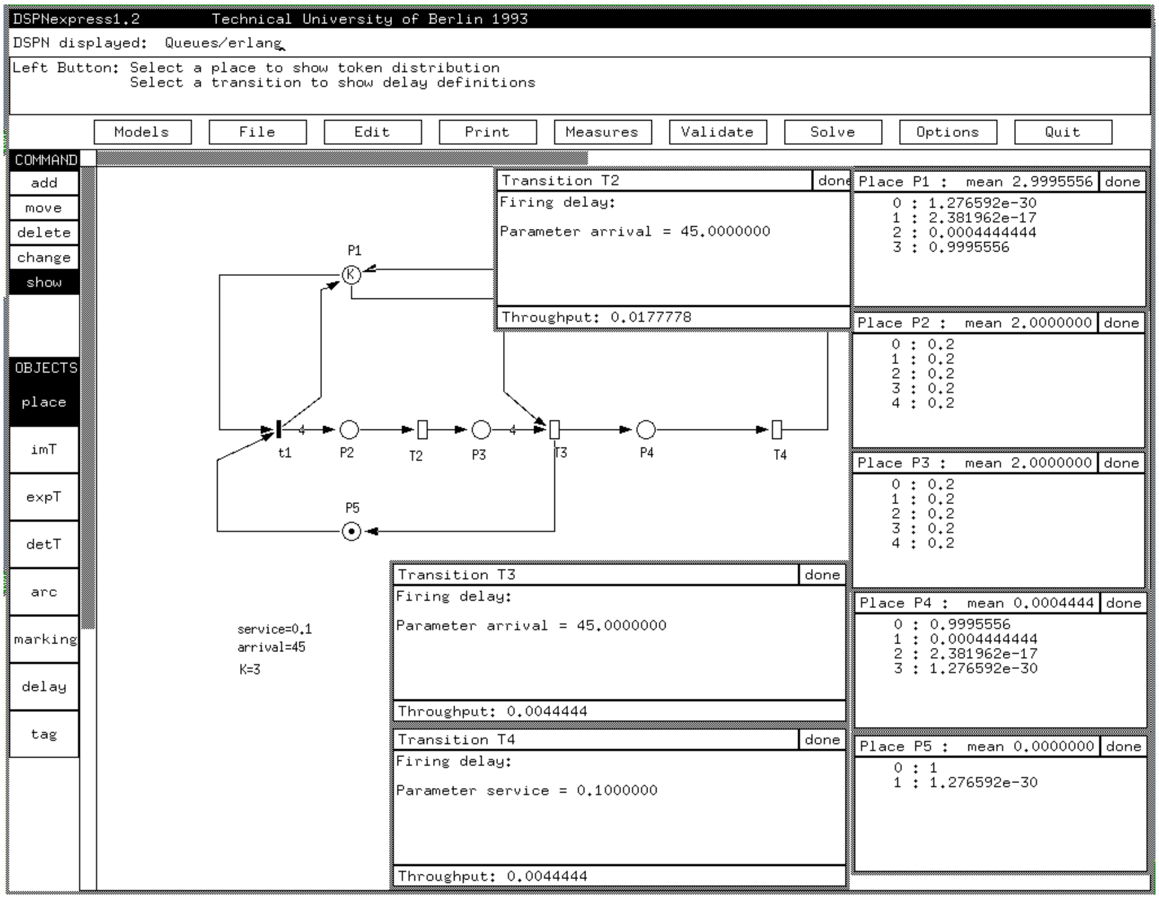


Fig. A.2.7.- Log file for the Erlang queue example

```
STRUCTURAL ANALYSIS

GENERATING THE REDUCED REACHABILITY GRAPH

The net contains 16 tangible markingsld.so: warning: /usr/lib/libc.so.1.7.3 has older
DDED MARKOV CHAIN OF THE DSPN ...

Starting all rand processes locally with max. 2 concurrent processes.

Processing states of set Mexp.

Processing states of set Mexp terminated.
delta = 10.0222  eta = 26
Elapsed CPU time: 0.11 sec.
Elapsed overall time: 00:00:01

Total CPU time: 0.11 sec.
Total overall time: 00:00:01


SOLVING THE LINEAR SYSTEM OF GLOBAL BALANCE EQUATIONS ...

Deriving sparse representation of transition probability matrix in column-wise format

Number of nonzeros :            26
Number of nonzeros per row :    6.10
Smallest entry :                2.22e-03

Trying Solution with iterative SOR ...
Iterative solver did not converge within 5000 iterations
for the pre-defined error tolerance of 1e-07

Retrying Solution with Gauss-Elimination ...
Number of off-diagonal elements of U: 15
Elapsed CPU time: 0.10 sec.
Elapsed overall time: 00:00:01
Required memory: 416 KB

DERIVING TOKEN PROBABILITY DISTRIBUTION IN EACH PLACE ...

SOLUTION OBTAINED.
```

8.3 Some features of *SPNP*

Fig. A.3.1.- The CSPL for the Erlang Er/D/1/K queue definition

```
 /* This is an example for a Erlang Er/D/1/K */

#include "user.h"

double tau=0.1,arrival;
int lambda=9,phases,buffers;

parameters() { phases=0;buffers=0;
                while ((phases<1) || (buffers<=1)) {
phases = input("Introduce number of phases");
        buffers = input("Introduce number of buffers");
if ((phases<=1) || (buffers<=1))
  printf("invalid parameters\n");}
iopt(IOP_METHOD,VAL_GASEI);
iopt(IOP_PR_FULL_MARK,VAL_YES);
iopt(IOP_PR_RSET,VAL_YES);
iopt(IOP_ITERATIONS,20000);
fopt(FOP_PRECISION,0.00000001);
iopt(IOP_PR_RGRAPH,VAL_YES);
iopt(IOP_PR_MC,VAL_YES);
iopt(IOP_PR_PROB,VAL_YES);
iopt(IOP_MC,VAL_CTMC);

}


net() {
place("p1");
init("p1",buffers);
place("p2");
place("p3");
place("p4");
place("p5");
init("p5",1);
trans("t1");
probval("t1",0.09);
trans("t2");
trans("t3");
trans("t4");
arrival=phases*lambda;
rateval("t2",arrival);
rateval("t3",arrival);
```

```
        rateval("t4",tau);
priority("t1",1);
miarc("t3","p3",phases-1); moarc("t1","p2",phases-1);
oarc("t2","p3");
iarc("t1","p1"); oarc("t3","p4");
iarc("t1","p5"); oarc("t3","p5");
iarc("t2","p2"); oarc("t4","p1");
iarc("t4","p4");
iarc("t3","p1"); oarc("t1","p1");
}


assert(){return(RES_NOERR);}
ac_init(){pr_net_info();}
ac_reach(){fprintf(stderr,"\The reachability graph has been generated\n\n");
   pr_rg_info();}
ac_final(){pr_mc_info();
   pr_std_average();}
```

Fig. A.3.2.- Log file for the Erlang Queue example

```
The reachability graph contains:
       16 tangible markings
        3 vanishing markings
       29 arcs
After elimination of redundant arcs:
# of remaining arcs:     29
ERROR/WARNING: transient initial marking. 0 markings reach it.
After the elimination of vanishing markings:
# of remaining arcs:     26
Solving the Markov chain...
...Markov chain solved
Reading the reachability graph info ...
End of execution.
```

Fig. A.3.3.- Numerical analysis results for the Erlang queue example

```
INPUT: Introduce number of phases = 5
INPUT: Introduce number of buffers = 3
NET:
==========================================================================
places: 5
immediate transitions: 1
timed transitions: 3
constant input arcs: 6
constant output arcs: 6
constant inhibitor arcs: 0
variable input arcs: 0
variable output arcs: 0
variable inhibitor arcs: 0
==========================================================================
RG:
==========================================================================
tangible markings: 16
vanishing markings: 3
marking-to-marking transitions: 29
==========================================================================
CTMC:
==========================================================================
states: 16
nonzero entries: 26
iterations: 6
precision: 1.13741e-10
==========================================================================
AVERAGE:
==========================================================================
    PLACE                          Pr[nonempty]          Av[tokens]
  0: p1                             1.111110682281e-02    1.118600274565e-02
  1: p2                             8.888885458146e-03    2.222221364512e-02
  2: p3                             8.888885458346e-03    2.222221364611e-02
  3: p4                             9.999996138712e-01    2.988813997254e+00
  4: p5                             9.888888931772e-01    9.888888931772e-01
    TRANSITION                     Pr[enabled]           Av[throughput]
  1: t2                             8.888885458146e-03    3.999998456166e-01
  2: t3                             2.222221364661e-03    9.999996140974e-02
  3: t4                             9.999996138712e-01    9.999996138712e-02
==========================================================================
```