New Primitives for Interlaced Memory Policies in Markov Regenerative Stochastic Petri Nets

Andrea Bobbio[1] Antonio Puliafito[2] Miklós Telek[3]

[1] Dipartimento di Informatica, Università di Torino
10149 Torino, Italy; e-mail:bobbio@di.unito.it

[2] Istituto di Informatica, Università di Catania
Viale A. Doria 6, 95025 Catania, Italy; e-mail:ap@iit.unict.it

[3] Department of Telecommunications, Technical University of Budapest
1521 Budapest, Hungary; e-mail:telek@hit.bme.hu

Abstract

The non-Markovian Stochastic Petri Net (SPN) *models appeared so far in the literature, are based on the assumption that the underlying marking process can be specified by assigning an individual memory policy to each timed transition. The present paper proposes to introduce interlaced, or state dependent, memory policies, where the memory of a transition can be modified by the occurrence of some condition on the net.*

Adhering to the spirit of the graphical language of the PN, *we introduce new primitives in the form of suitable arcs connecting places to transitions, and whose effect is to modify the memory policy of the transition.*

Through a number of simple examples, we show how the new primitives increase the modeling power of non-Markovian SPN *by allowing firing mechanisms which were not possible in the traditional models.*

Numerical results on the steady state behavior of a dependable processor system with two kinds of interruptions are also presented.

Key words: Non-Markovian Stochastic Petri Nets, firing mechanisms, Markov regenerative processes, memory resetting arc.

1 Introduction

The semantics of *PN* with generally distributed firing times has been extensively considered in [1]. In the above reference, in order to completely define the behavior of the marking process, each timed transition was assigned an individual memory policy specifying how the firing of the transition was dependent on its past history. The memory policy proposed in [1] was an attribute attached to each individual transition so that the memory of the overall marking process resulted from the superposition of the memories of the individual transitions.

Based on the concepts defined in [1], Ajmone and Chiola [2] developed the *Deterministic and Stochastic PN (DSPN)* model, where in each marking, a single transition is allowed to have associated a determinsitic firing time with enabling memory policy. Choi et al. [6] have derived the transient solution of the same model in terms of a Markov regenerative process, and have subsequently extended the *DSPN* model by accommodating at most a single transition with generally distributed firing time [7] in each marking. They have called this class of models *Markov Regenerative Stochastic PN (MRSPN)*. Further elaborations of *SPN* models with non exponential distributions but restricted to enabling memory policies, only, have been presented in [9, 8, 12, 11, 17, 18].

Bobbio and Telek have enlarged the class of *MRSPN* by introducing the concept of marking processes with non-overlapping memories. In this new framework, they have accommodated into the model age memory policies [5, 19, 20] and preemptive repeat identical policies [3, 21].

The aim of this paper is to remove the restriction that the memory policy is an attribute attached to each single transition, by definig a new formalism for specifying interlaced or state dependent memory policies. The need for interlaced memory policies arises when the occurrence of an event (represented by a set of markings in the *PN*) may modify the memory of a transition in a way that is not possible to represent by the existing memory policies and *PN* primitives.

Ciardo and Lindemann [9] and Ciardo et al. [8] have attempted to introduce a formalism for the mutual dependence of the memory of a given transition on the firing of another one. This formalism is vaguely defined and does not seem to be practically implementable in a *PN*-based model. Adhering to the spirit of the graphical language of the *PN*, the present paper proposes to specify the interlaced memory policies by defining new primitives. The new primitives are in the form of suitable arcs connecting a source place to a destination transition, and whose effect is to inhibit the destination transition when the source place is marked but at the same time modifying the memory of the destination transition and consequently its memory policy.

In particular, two kinds of arcs are introduced: the *memory resetting arc (mra)* that resets the whole memory of the destination transition when the source place is marked and the *age resetting arc (ara)* that resets only the age variable associated to the destination transition. This extension of Petri Net primitives results in a wider class of Petri net models with respect to those considered in [1] and in [21].

Some examples are reported in order to show the meaning of the new primitives and how their use generalizes the memory mechanisms so far introduced. We believe that the discussed approach provides a well established and sound formalism also for the implementation of simulation tools.

The rest of the paper is organized as follows. In Section 2, the individual memory model considered in the previous literature is briefly recalled, and in Section 3 the new primitive arcs are defined. Section 4 is devoted to illustrate several examples of application of the resetting arcs. Section 5 provides a formal approach to the analyis of non-Markovian Petri nets with resetting arcs based on the state space partitioning of the subordinanted process. Numerical results on the steady state behavior of a dependable processor system with two kinds of interruptions are presented in Section 6.

2 The individual memory model

A marked Petri Net is a tuple $PN = (P, T, I, O, H, M_0)$, where: P is the set of places, T the set of transitions, I, O and H are the input, the output and the inhibitor functions, respectively, and M_0 is the initial marking. The reachability set $\mathcal{R}(M_0)$ is the set of all the markings that can be generated from the initial marking M_0. The marking process $\mathcal{M}(t)$ denotes the marking occupied by the *PN* at time t.

We define a non-Markovian *SPN* as a stochastically timed *PN* in which the time evolution of the marking process can be more general than a *Continuous Time Markov Chain (CTMC)*. In the spirit of many modeling formalisms [15], in which the complexity of the solution must be hidden to the modeler, a complete set of specifications must be given at the *PN* level, in order to univocally define the underlying marking process. Therefore, the way in which the future evolution of the marking process depends on its past history needs to be specified at the *PN* level.

The most consistent way to introduce memory into a *SPN* is provided in [1]. Each timed transition t_g is assigned a random firing time γ_g with a general distribution $G_g(t)$ with support on $[0\,,\,\infty)$. A clock, associated to each individual transition, counts the time in which the transition has been enabled. An *age variable* a_g associated to the timed transition t_g keeps track of the clock count. A timed transition fires as soon as the memory variable a_g reaches the value of the firing time γ_g. A very similar formulation, in the simulation setting, has been discussed by Haas and Shedler [13, 14].

In the original view [1], two main firing policies were introduced:

- *enabling memory* if the age variable a_g is reset each time the corresponding transition t_g is disabled or fires;
- *age memory* if the age variable a_g is reset only when the corresponding transition t_g fires.

We define the *activity period* of a transition t_g as the interval of time during which the corresponding age variable a_g is different from 0. In [1], the firing time was implicitly assumed to be resampled at the beginning of any activity period of the transition.

However, in a more general view, the random firing time γ_g of a transition t_g can be sampled in a time instant antecedent to the beginning of an activity period. To keep track of the resampling condition of the random firing time associated to a timed transition, we assign to each timed transition t_g a binary indicator variable ι_g that is equal to 1 when the firing time is sampled and equal to 0 when the firing time is not sampled. We refer to ι_g as the *resampling indicator variable*. When a transition enters an activity period, if the resampling indicator variable ι_g is zero, the firing time is resampled and ι_g is switched to 1; whereas, if ι_g is already equal to 1, the firing time is not resampled. We can, therefore, define the *resampling period* of a transition as the time interval during which the indicator variable ι_g is equal to 1, i.e. the firing time of the transition maintains its constant value without any intermediate resampling.

The *resampling period* forms a further element of memory. Hence, in general, the memory of a transition t_g is captured by the tuple $(a_g,\ \iota_g)$. At any time epoch t, transition t_g has memory (its firing process depends on the past) if either a_g or ι_g are different from zero. Adopting the previous formalization of the memory concept, the following individual memory policies have been introduced in the past. A timed transition t_g can be:

- *Preemptive resume (prs)*:
  If the associated clock counts the time according to an age memory policy and the firing time is resampled when the transition becomes active. More formally, both the age variable a_g and the resampling indicator ι_g are reset only when t_g fires.
- *Preemptive repeat different (prd)*:
  If the associated clock counts the time according to an enabling memory policy and the firing time is resampled when the transition becomes active. More formally, both the age variable a_g and the resampling indicator ι_g are reset each time t_g is disabled or fires.
- *Preemptive repeat identical (pri)*:
  If the associated clock counts the time according to an enabling memory policy but the firing time is resampled only when the transition fires. More formally, the age variable a_g is reset each time t_g is disabled or fires but the resampling indicator ι_g is reset only when t_g fires.

In the described individual memory models, a *prs* transition cannot be disabled and restarted before firing, and a *pri* transition cannot be resampled before firing. The next section introduces new primitives aimed at removing these restrictions.

3 Interlaced memory model

The interlaced memory policy is represented by means of suitable arcs connecting a source place with a destination transition, and called *resetting arcs*. These new *PN* primitives are a semantical extension of the primitive *inhibitor arc*, in the sense that, when the source place is marked, the destination transition is inhibited (disabled) and its memory is partially, or totally, reset.

The resetting arc responds to the motivation of modeling situations in which an external event, represented by a given set of markings, can reset the memory of a transition.

Since the memory of a transition is composed by two attributes $(a_g,\ \iota_g)$, two kinds of resetting arcs are introduced: the *age resetting arc (ara)* which inhibits the destination transition and resets its *age* only, and the *memory resetting arc (mra)* which inhibits the destination transition and resets both its *age variable* and its *resampling indicator variable*. A third case of resetting is theoretically possible and could be obtained by resetting the *resampling indicator* only. But this third case is of no pratical meaning and will not be further considered.

3.1 The memory and age resetting arc

We propose, as a graphical symbol for the *mra*, an arc from a place p_{source} to a transiton t_{dest} with a filled small circle on the destination transition t_{dest}. The construct of the *mra* is illustrated in Figure 1a). When a token arrives in p_{source} (for instance by firing t_r), both the age variable a_{dest} and the resampling indicator variable ι_{dest} associated to t_{dest} are reset, independently of their previous value. Hence, in the next marking in which t_{dest} is enabled again, ι_{dest} is switched to 1 so that the firing time γ_{dest} is resampled and a_{dest} restarts counting from 0.

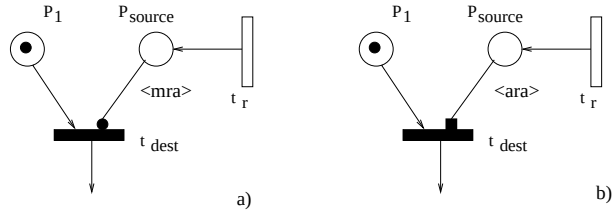

Figure 1: The Memory Resetting Arc

The concept of *ara* is similar to the concept of *mra*, but the *ara* resets only the age variable a_{dest} while it has no effect on the resampling indicator variable ι_{dest}. We propose, as a graphical symbol for the *ara*, an arc from a source place p_{source} to a destination transiton t_{dest} with a filled small square on the destination transition.

The construct of the *ara* is illustrated in Figure 1b). When a token arrives in p_{source} (for instance by firing t_r), the age variable associated to t_{dest} is reset, but ι_{dest} remains set to 1, so that the corresponding firing time is maintained. Hence, in the next marking in which t_{dest} is enabled again, a_{dest} restarts counting from 0 but the same firing requirement should be accomplished before firing.

3.2 Combining resetting arcs with memory policies

The interaction and the effect of the resetting arcs on the individual memory models defined in Section 2, are illustrated by means of Figure 2, when the resetting arc from p_{source} to t_{dest} is *mra* (as in the figure)

or *ara*, and when t_{dest} is assigned a *prd* or a *prs* or a *pri* memory policy.

When p_{source} is empty, the semantics of the model follows the individual memory policy. When p_{source} becomes marked (by firing t_r) the semantics depends on the nature of the resetting arc and on the memory policy of t_{dest}. For the sake of clarity, and with reference to Figure 2, we briefly examine all the possible combinations of memory policies and resetting arcs, when a token arrives in p_{source}.

- The memory policy of t_{dest} is *prd* and the resetting arc is:

  ara or *mra* - The *ara* or *mra* acts as an ordinary inhibitor arc since, according to the *prd* memory policy, both a_{dest} and ι_{dest} are reset as the transition is disabled.

- The memory policy of t_{dest} is *prs* and the resetting arc is:

  ara - Only the age variable a_{dest} is reset so that the firing time is not resampled. This combination of policies can be considered as a semantical extension of the individual *pri* policy.

  mra - Both a_{dest} and ι_{dest} are reset; the consequence is that when t_{dest} becomes enabled again (and no tokens are in p_{source}), the age variable is restarted from zero and the firing time resampled from the same distribution.

- The memory policy of t_{dest} is *pri* and the resetting arc is:

  ara - The *ara* acts as an ordinary inhibitor arc since both a_{dest} and ι_{dest} are reset as the transition is disabled.

  mra - The complete memory is reset; in the new enabling, the firing time is resampled from the same distribution.

4 Examples of interlaced memory policies

In order to illustrate the meaning of the new defined primitives, we develop two simple examples. The first one considers the execution of a job in a multiuser dependable systems, while the second one the transmission of a message in a shared medium with a hard deadline.

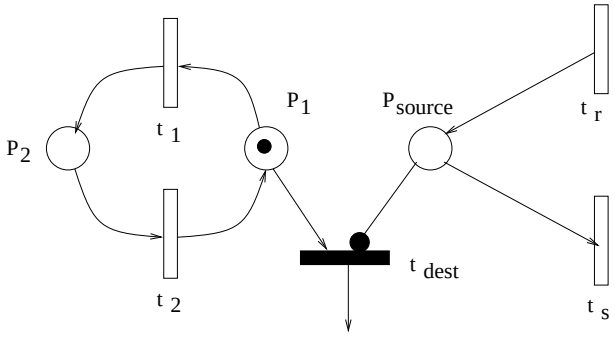

Figure 2: The Age Resetting Arc

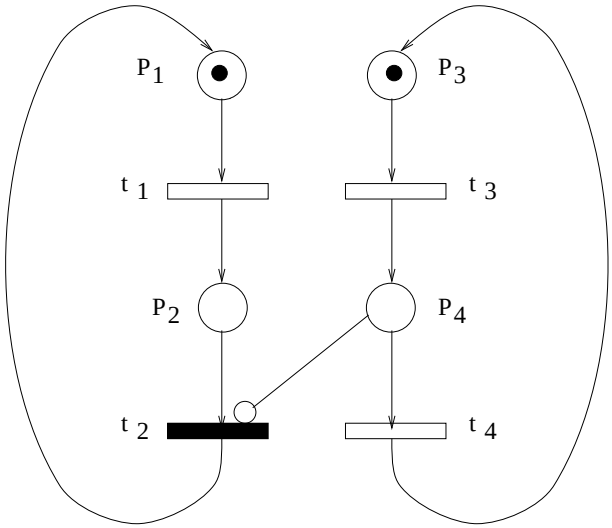

Figure 3: Petri net model of the processor system

4.1 Example 1: Job execution in a processor system

Consider a processor system with two terminals A and B, submitting jobs for execution. Jobs from terminal A require a generally distributed processing time, while jobs from terminal B experience an exponentially distributed processing time. Further, terminal B generates higher priority jobs, which preempt jobs coming from terminal A. Place p_1 in Figure 3 represents the terminal A in the thinking phase and transition t_1 models the exponentially distributed submission time. Place p_2 indicates that job A is being executed, and transition t_2 represents the random execution time. In a similar fashion, transition t_3 models the generation of higher priority jobs from terminal B (place p_3). A token in place p_4 represents a type_B job being processed. Transition t_4 is the processing time of jobs submitted by terminal B. Since type_B jobs have higher priority over type_A jobs, we intro-

duce an inhibitor arc from place p_4 to transition t_2 which interrupts the execution of any type_A job until the execution of the type_B job is completed. Transitions t_1, t_3 and t_4 are assumed to be exponentially distributed, while the service time modeled by transition t_2 is assumed to be generally distributed. The service policy of type_A jobs is assumed of preemptive resume type, which means that its execution is resumed from the point of interruption. This behavior is modeled by associating t_2 a *prs* memory policy. Note that the preemption policy assumed for t_4 is completely ininfluent for the overall behavior of the system as t_4 will always complete its activity once enabled.

4.1.1 Job execution in a dependable processor system

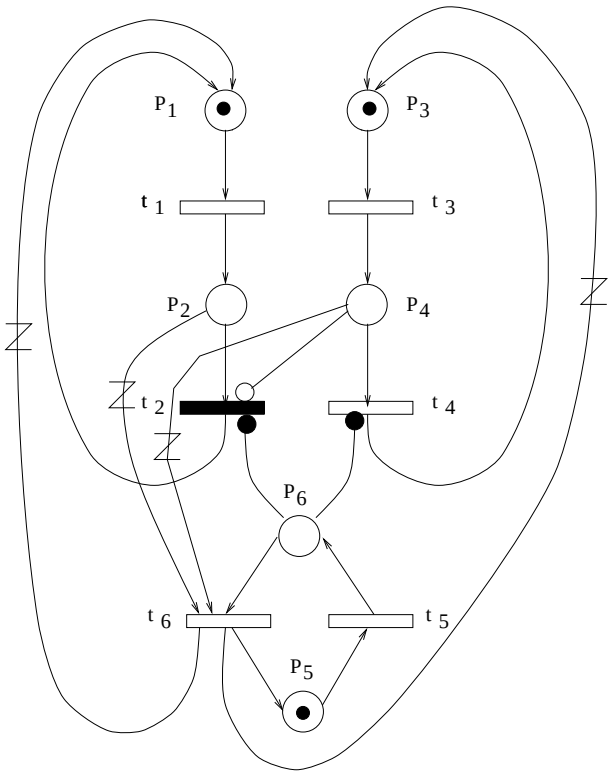

Figure 4: Petri net model of the dependable processor system

In this section, we extend the previous example by including the possibility that the processor can experience failures and repairs (Figure 4). Places p_1, p_2, p_3, p_4 and transitions t_1, t_2, t_3, t_4 play the same role. Place p_5 represents the processor up, while place p_6 represents the processor down. Transition t_5 models the exponentially distributed failure time, while transition t_6 models the exponentially distributed repair time. We assume that, when the processor fails, the job under execution (either A or B) is lost and the system restarts from the job submitting phase as the processor is up again. The next job after processor failure will have a different processing requirement completely independent from the one before the failure. The *PN* model has to enforce the memory reset of transitions t_2 and t_4 in case of processor failure, because, otherwise, the last part of the uncompleted job would be completed after the processor recovers. This is why we introduce two *mra* arcs from p_6 to t_2 and t_4 which reset both the corresponding age and inidicator resampling variables. The variable multiplicity arcs (indicated by a 'Z' on the arcs) from places p_2 and p_4 to places p_1 and p_3 allow to bring the system to the original starting state. Note that if in Figure 4 we substitute the *mra* arcs with *ara* arcs, a repeat identical type interruption is implemented.

4.1.2 Job execution with different interruptions in a dependable processor system

In this example, we extend the previous model to include two different failure modes of the processor (Figure 5). With probability c the failed processor remembers which job was under execution and the same job (i.e. a job with an identical requirement) is re-executed upon repair (*pri*-like interruption), while with probability *1-c* a new job is re-executed (*prd*-like interruption).

A token in place p_5 indicates that the processor is *up*. A failure can occur with an exponentially distributed failure time modeled by transition t_5. Immediate transitions t_8 and t_9 model the two different failure modes. With probability c a *pri*-like interruption occurs: two *ara* arcs connect the source place p_6 to the destination transitions t_2 and t_4 so that, when p_6 is marked, their age variables are reset while keeping the same value for the firing requirement. With probability $1 - c$ a *prd*-like interruption occurs: two *mra* arcs connect place p_7 to transitions t_2 and t_4 so that, when p_7 is marked, both their age variables and resampling indicator variables are reset. Transitions t_6 and t_7 model the processor repair time which is assumed to be exponentially distributed.

4.2 Example 2: Message transmission through a transmission line

Ciardo et al. [8] proposed the following example. A real time system is sending messages according to a given distribution. Each message must be transmitted inside a hard deadline, otherwise the whole message must be repeated. The *PN* model and associated

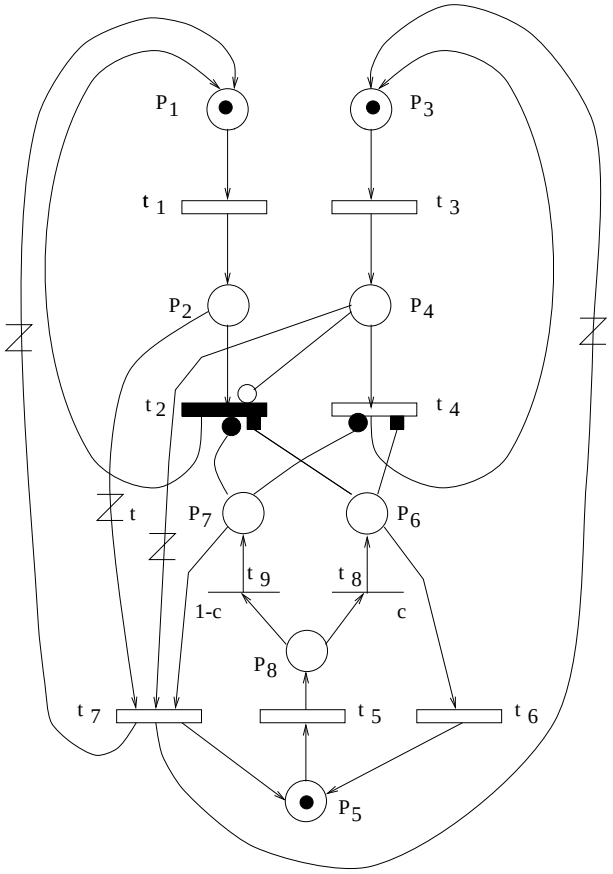

Figure 5: Petri net model of the dependable processor system with 2 kinds of interrupts

reachability graph are reported in Figure 6.

Place p_0 represents a station sending messages after a generally distributed preparation time modeled by t_0. p_1 is the message ready to be transmitted; t_1 is the deterministic hard deadline. p_2 is the message waiting for accessing the medium and t_2 is the exponentially distributed time to acquire the medium. p_3 models the message in the transmission phase, and t_3 is the transmission time. In [8], t_3 was deterministic, thus hiding the choice between a *prd* or *pri* behavior [3]. We remove this restriction by allowing t_3 to have assigned a generally distributed firing time.

If the timeout elapses before the message is transmitted (transition t_1 fires before t_3), the transmission is interrupted and the whole message needs to be resubmitted after a preparation time. The marking dependent arcs from p_2 and p_3 to t_1 have the function of removing the token from p_2 or p_3 to p_0 when t_1 fires.

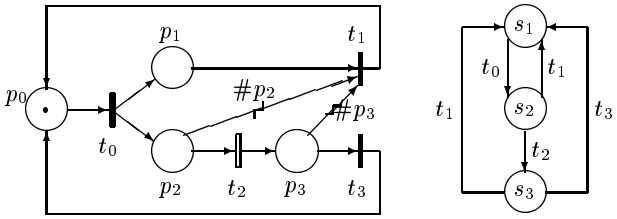

Figure 6: Petri net model of the transmission line

4.2.1 Message transmission through a shared transmission line

Suppose now that the trasmission medium is shared by other resources. We model the medium as a two state *PN* (see Figure 7) where place p_w indicates that the medium is available for transmitting the message under consideration, and place p_n indicates that the medium is not available. The inhibitor arc from p_n to t_3 prevents the transmission of the tagged message when the line is not available. The low level protocol is designed to work on a conservative way, i.e. the interrupted message is resumed from the point it was preempted by the firing of t_w. This physical behavior is modeled by assigning to t_3 an age memory policy.

In the case of a timeout the memory of t_3 has to be reset. For this reason we include in the model a resetting arc from p_0 to t_3. Each time the station is requested to send a message (token in p_0) all the memory in the system is cleared.

If the resetting arc is *mra* (as in Figure 7), also the firing requirement is reset and a new message taken from the same distribution is transmitted. If the resetting arc is of type *ara*, only the age associated to the stopped message is reset and a new message with identical requirement is retransmitted.

5 MRSPN with resetting arcs

Let the *memory cycle* of a transition t_g be the time interval in which either the corresponding age variable a_g is not 0 or the resampling indicator variable ι_g is equal to 1. Extending the definition in [5], we consider *MRSPN* in which the memory cycles of the different transitions do not overlap. Hence, any regeneration period in the marking process $\mathcal{M}(t)$ is dominated by a single transition.

The restriction of the marking process $\mathcal{M}(t)$ during the memory cycle of the dominant transition is called the subordinated process. According to the theory of *Markov Regenerative Processes (MRGP)* [10, 16], the

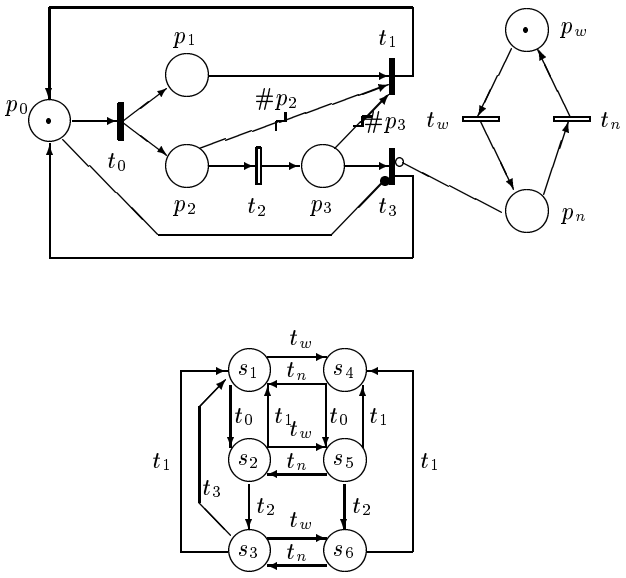

Figure 7: Petri net model of the shared transmission line

analysis of the whole process can be obtained from the analysis of the subordinated processes considered in isolation.

Hence, the behavior of a generic subordinated process dominated by a transition t_g, is considered in details. Extending and adapting a methodology already presented in [21], the state space of the subordinated process can be partitioned into the subsets E, D1, D2 and C, as in Figure 8.

- In the states of subset E (enabling) t_g is enabled; its age variable (a_g) is continuously increasing and $\iota_g = 1$.
- In the states of subset D1 (disabling) t_g is disabled, but a_g is unchanged ($a_g > 0$) and $\iota_g = 1$.
- In the states of subset D2 (disabling) t_g is disabled, the age variable a_g is reset to 0 and $\iota_g = 1$.
- When a state of the subset C is reached the regeneration period is over. Thus, a_g is reset and ι_g is set to 0.

These subsets derive directly from a careful analysis of the evolution of the subordinated process and, depending on the structure of the model, might also be empty. The classification provided in Figure 8, however, includes all the cases that can arise by combining

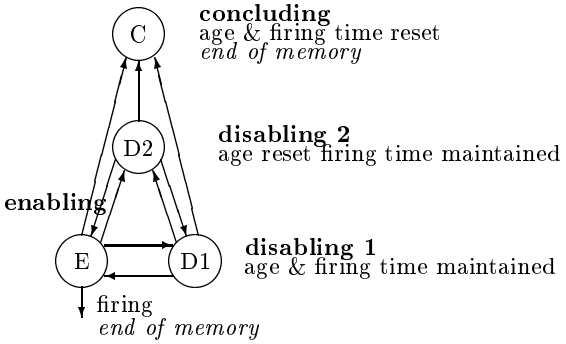

Figure 8: Structure of the state space of the stochastic process underlying the memory period of t_g with *mra* and/or *ara*

the non-Markovian *PN* already analyzed in the literature, with the new primitives proposed in this paper. It is worthnoting to observe, that once a dominant transition is enabled, the end of the regeneration period can occurs as a consequence of two events only: *i)* the firing of the dominant transition (from a state in subset E),or, *ii)* the passage into one state of subset C.

According to Section 2, it is possible to identify the causes that determine the passage from one subset to another one among those defined in Figure 8.

- E→D1
  - t_g with age memory becomes disabled
- E→D2
  - t_g with *pri* enabling memory becomes disabled
  - t_g with age memory becomes disabled by an *ara*
- E→C
  - t_g with *prd* enabling memory becomes disabled
  - t_g with *pri* enabling memory becomes disabled by a *mra*
  - t_g with age memory becomes disabled by an *ara*
- D1→D2
  - (the disabled) t_g with age memory is reset by a *ara*
- D1→C
  - (the disabled) t_g with age memory is reset by a *mra*

transition	state space
prd	no transition to D1 and D2
pri	no transition to D1 and C
prs	no transition to D2 and C
pri & mra	no transition to D1
prs & ara	no transition to C
prs & mra	no transition to D2
prs & mra & ara	any subset is reachable

Table 1: Allowed transitions between the subsets of states

- D2→C
  - (the disabled) t_g with *pri* enabling memory is reset by a *mra*
  - (the disabled, and *ara* reset) t_g with age memory is reset by a *mra*

Once these subsets have been identified, for each subordinated process, the probability measures of the marking process can be obtained from the evaluation of the first passage time probabilities among the non-empty subsets of Figure 8. The evaluation of the first passage time probabilities follows the same pattern already described in [21].

To better clarify the previous concepts, and to simplify the identification of these subsets, Table 1 shows the allowed connections of the different subsets for any possible comibination of preemption policy with *mra* and *ara* arcs.

6 Case study

In this section, we provide a fully developed numerical study for the example of the processor system with two different failure modes considered in Subsection 4.1.2 (see Figure 5). Details on the analytical solution method can be found in [4].

Table 2, reports the reachability set for the *MRSPN* of Figure 5, together with useful information for the analysis of the model. The state space is made of 12 tangible states. The first column provides a state numbering starting from the initial marking, while column 2 gives the marking. The third column indicates if in the given state only exponentially distributed transitions are enabled (*exp*) or if a generally distributed transition is enabled (∗). The states marked with an ∗ represent regeneration states whose regeneration period allows internal state transitions. Column 4 indicates the states that can be reached inside a regeneration period, i.e. the states that can be reached during the subordinated process starting from the given marking (note that for all the *exp* states, the subordinated process is composed by a single marking).

In order to completely specify the process, the information about which state initiates the subsequent regeneration period is needed, and the kind of event that determines the starting of the subsequent regeneration period. This information is provided in column 5, by two attributes *state(event)*. For the states indicated as *exp* the attribute *state* indicates the number of the states that are immediately reachable and the attribute *event* indicates the transition whose firing leads the process into the corresponding *state*. For the states marked with an (∗), the *state* indicates the number of the states that can initiate the subsequent regeneration period, and the *event* indicates the reason which forces the process to enter a new regeneration period. Considering state 2 (0110100), it can be noticed that the corresponding subordinated process is dominated by transition t_2 with generally distributed firing time.

The subsequent regeration period can start from states 1, 10 or 12, and the entering in one of these states is due to some event that makes transition t_2 to loose its memory. In particular, as indicated in Table 2, state 1 is reached when transition t_2 fires (from a state in the subset E). States 10 and 12 form the subset C, and are reached as a consequence of a failure of the processor (token in place p_7) which reset the memory of t_2 by means of a *mra*.

Figure 9 depicts the reachability graph of the subordinated process starting from state 2. Similar graphs can be drawn for all the states that can initiate a regeneration period. We concentrate our analysis on the subordinated process depicted in Figure 9, since it contains states in all the partitions classified in Figure 8. State 2(0110100) belongs to subset E. In this state the age variable a_2 of the dominant transition is greater than zero and increasing, and $\iota_2 = 1$. Three different states can be immediately reached from state 2. State 4(0101100), that belongs to subset $D1$, and is reached by the the firing of transition t_3. State 6(0110010) that belongs to subset $D2$, and is reached by the firing of transitions $t_5 + t_8$, in sequence. State 10(0110001) that belongs to subset C, and is reached by the firing of transitions $t_5 + t_9$, in sequence.

The steady state probabilities of the marking process $\mathcal{M}(t)$ of the *PN* of Figure 5, can be obtained by solving all the possible subordinated processes as the one reported in Figure 9.

No.	Marking	Type	Sub. Proc.	Subsequent reg. states
1	1010100	exp	1	2(t_1), 3(t_3), 5($c\lambda$), 9 ($(1-c)\lambda$)
2	0110100	*	2,4,6,8	1($firing$), 10($conc$), 12($conc$)
3	1001100	*	3,4,7,8	1($firing$), 2($firing$), 11($conc$), 12($conc$)
4	0101100	*	4,8	1($firing$), 12($conc$)
5	1010010	exp	5	1(t_6), 6(t_1), 7(t_3)
6	0110010	exp	6	2(t_6), 8(t_3)
7	1001010	exp	7	3(t_6), 8(t_3)
8	0101010	exp	8	4(t_6)
9	1010001	exp	9	1(t_7), 10(t_1), 11(t_3)
10	0110001	exp	10	1(t_7), 12(t_3)
11	1001001	exp	11	1(t_7), 12(t_3)
12	0101001	exp	12	1(t_7)

Table 2: The state space of the underlying MRGP

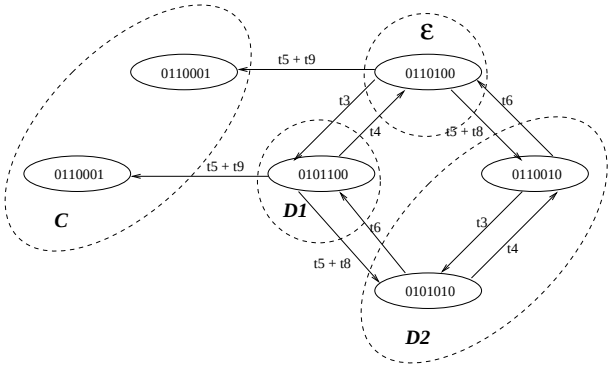

Figure 9: Subordinated process starting from state 2(0110100)

We have concentrated our analysis on the evaluation of a performance index which provides the loss probability for type_A and type_B jobs. The explanation which follows focusses on type_A jobs, but a similar approach can be followed for type_B jobs. The throughput ($Th1$) of transition t_1 is equal to the number of type_A jobs entering the system per unit time in steady state. However, these jobs will not necessarily be completed due to the possible preemption and restart caused by a processor failure with *prd* preemption. Indeed, each time the firing sequence $t_5 + t_9$ occurs, a token reaches p_7 and, by the effect of the *mra* pointing to t_2, the job in execution is stopped and discarded. If we look at the throughput ($Th2$) of transition $t_5 * (1-c)$ conditioned on the presence of a token in p_2 we get the number of type_A jobs discarded per unit time. The steady state loss probability of type_A jobs can thus be evaluated by:

$$P_{loss_A} = \frac{Th2}{Th1 + Th2}.$$

Figure 10 plots the loss probability of type_A and type_B jobs when the switching parameter c ranges from 0 to 1. The greatest loss arises when $c = 0$, which means that each processor fault results in discarding the job. A loss equal to zero occurs if $c = 1$ because, in this case, even if the processor fails the job is not lost, but reprocessed again from the beginning. The upper (lower) curve refers to P_{loss} of type_A (type_B) job.

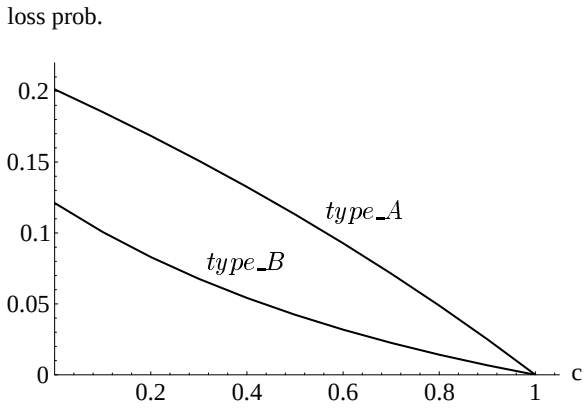

Figure 10: Loss probabilities of type_A and type_B jobs versus c

7 Conclusions

In this paper, practical examples are discussed whose stochastic behavior can not be captured by the traditional *PN* models. New primitives have been introduced which allow to deal with interlaced memory policies. The importance of extending the semantics of individual memory policy is highlighted through the examples described in the paper. Numerical results on the steady state behavior of a dependable processor system with two kinds of interruptions are presented.

Acknowledgements

This work has been partially supported by Italian CNR under Grant No. 96.01939.CT12 and Hungarian OTKA under Grant No. T-16637.

References

[1] 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*, SE-15:832–846, 1989.

[2] M. Ajmone Marsan and G. Chiola. On Petri nets with deterministic and exponentially distributed firing times. In *LNCS*, volume 266, pages 132–145. Springer Verlag, 1987.

[3] A. Bobbio, V.G. Kulkarni, A. Puliafito, M. Telek, and K. Trivedi. Preemptive repeat identical transitions in Markov Regenerative Stochastic Petri Nets. In *6-th International Conference on Petri Nets and Performance Models - PNPM95*, pages 113–122. IEEE Computer Society, 1995.

[4] A. Bobbio, A. Puliafito, and M. Telek. Analysis of non-Markovian Petri nets with resetting arcs. Technical report, Department of Telecommunications - Technical University of Budapest, June 1996.

[5] A. Bobbio and M. Telek. Markov regenerative SPN with non-overlapping activity cycles. In *International Computer Performance and Dependability Symposium - IPDS95*, pages 124–133. IEEE Computer Society Press, 1995.

[6] Hoon Choi, V.G. Kulkarni, and K. Trivedi. Transient analysis of deterministic and stochastic Petri nets. In *Proceedings of the 14-th International Conference on Application and Theory of Petri Nets*, Chicago, June 1993.

[7] Hoon Choi, V.G. Kulkarni, and K. Trivedi. Markov regenerative stochastic Petri nets. *Performance Evaluation*, 20:337–357, 1994.

[8] G. Ciardo, R. German, and C. Lindemann. A characterization of the stochastic process underlying a stochastic Petri net. *IEEE Transactions on Software Engineering*, 20:506–515, 1994.

[9] G. Ciardo and C. Lindemann. Analysis of deterministic and stochastic Petri nets. In *Proceedings International Workshop on Petri Nets and Performance Models - PNPM93*, pages 160–169. IEEE Computer Society, 1993.

[10] E. Cinlar. *Introduction to Stochastic Processes.* Prentice-Hall, Englewood Cliffs, 1975.

[11] R. German. New results for the analysis of deterministic and stochastic Petri nets. In *International Computer Performance and Dependability Symposium - IPDS95*, pages 114–123. IEEE CS Press, 1995.

[12] R. German and C. Lindemann. Analysis of stochastic Petri nets by the method of supplementary variables. *Performance Evaluation*, 20:317–335, 1994.

[13] P.J. Haas and G.S. Shedler. Regenerative stochastic Petri nets. *Performance Evaluation*, 6:189–204, 1986.

[14] P.J. Haas and G.S. Shedler. Stochastic Petri nets with timed and immediate transitions. *Stochastic Models*, 5:563–600, 1989.

[15] B.R. Haverkort and K. Trivedi. Specification techniques for Markov Reward Models. *Discrete Event Dynamic Systems: Theory and Applications*, 3:219–247, 1993.

[16] V.G. Kulkarni. *Modeling and Analysis of Stochastic Systems.* Chapman Hall, 1995.

[17] C. Lindemann. An improved numerical algorithm for calculating steady-state solutions of deterministic and stochastic Petri net models. *Performance Evaluation*, 18:75–95, 1993.

[18] C. Lindemann. DSPNexpress: a software package for the efficient solution of deterministic and stochastic Petri nets. *Performance Evaluation*, 22:3–21, 1995.

[19] M. Telek and A. Bobbio. Markov regenerative stochastic Petri nets with age type general transitions. In G. De Michelis and M. Diaz, editors, *Application and Theory of Petri Nets, LNCS*, volume 935, pages 471–489. Springer Verlag, 1995.

[20] M. Telek, A. Bobbio, L. Jereb, A. Puliafito, and K. Trivedi. Steady state analysis of Markov regenerative SPN with age memory policy. In H. Beilner and F. Bause, editors, *8-th Int Conf Modeling Techniques and Tools for Computer Performance Evaluation, LNCS*, volume 977, pages 165–179. Springer Verlag, 1995.

[21] M. Telek, A. Bobbio, and A. Puliafito. Steady state solution of MRSPN with mixed preemption policies. In *International Computer Performance and Dependability Symposium - IPDS96*, pages 106–115. IEEE Computer Society Press, 1996.