Dependability Modeling Using Petri-Nets

Manish Malhotra
AT&T Bell Laboratories, Holmdel
Kishor S. Trivedi
Duke University, Durham

Key Words — **Combinatorial model type, dependability, fault-tree, generalized stochastic Petri net, Markov model, stochastic reward net**

Summary & Conclusions — **This paper describes a methodology to construct dependability models using generalized stochastic Petri nets (GSPN) and stochastic reward nets (SRN). Algorithms are provided to convert a fault tree (a commonly used combinatorial model type) model into equivalent GSPN and SRN models. In a fault-tree model, various kinds of distributions can be assigned to components such as defective failure-time distribution, non-defective failure-time distribution, or a failure probability. The paper describes subnet constructions for each of these different cases, and shows how to incorporate repair in these models.**

We consider the cases: 1) Each component has an independent repair facility. 2) Several components share a repair facility; such repair dependency cannot be modeled by combinatorial model types such as fault trees. We illustrate how such dependencies and various scheduling disciplines (for the repair queue) such as first-come first-served (FCFS), processor-sharing, preemptive priority with resume, and non-preemptive priority repair, can be modeled by GSPN & SRN. If the operational dependence of a system on its components is specified by means of a fault-tree and a repair dependence is described in some (other) form, then our methodology provides an automatic way to generate GSPN & SRN models of system dependability.

The subnet constructions allow us to compare SRN with GSPN as dependability model types. For the dependability models of repairable systems, the complexity (number of places and transitions) of GSPN models is appreciably higher than the complexity of equivalent SRN models. The state-space of the underlying continuous-time Markov chain (CTMC) remains the same, however. Thus SRN reduce the complexity of model specification at the net level, but the complexity of model solution remains the same. Since SRN include all the features of GSPN, the additional features of SRN such as reward rates, variable cardinality arcs, halting condition, and timed transition priorities, greatly simplify model construction & specification.

1. INTRODUCTION

Acronyms[1]

CTMC Continuous Time Markov Chain
FCFS First Come First Served
FTRE Fault Tree with Repeated Events
GSPN Generalized Stochastic Petri Net\\
SPN Stochastic Petri Net
SPNP Stochastic Petri Net Package
SRN Stochastic Reward Net.

Petri-net based models have been extensively used for performance & performability modeling to analyze computer & communication systems [1 - 7]. However, in the dependability modeling community, Petri-net based models have received considerably less attention [8 - 10]. This paper describes a methodology to construct dependability models using Petri nets. Among the Petri-net based model types, we consider GSPN [2] and SRN [11].

SRN extend the GSPN, *ie*, they include all the features of GSPN and many more, *eg*, guards (previously called *enabling functions*), timed transition priorities, variable cardinality arcs, halting condition, and reward rates. None of these extensions enhances the modeling power since every SRN model can be converted to a CTMC which are isomorphic to GSPN [2]; although SRN allow calculation of some reward-based measures which are not possible through GSPN [12]. Thus any system that can be modeled by a SRN can also be modeled by a GSPN. However, SRN and GSPN differ in the conciseness of model specification. SRN permit a much more concise description of system dependability than GSPN do. An aim of this paper is to bring out this distinction among these two different Petri-net model types. By converting dependability models specified as FTRE models to equivalent GSPN and SRN models, we illustrate how the features of SRN greatly simplify the model construction. A popular software tool for GSPN models is GreatSPN [13] and for SRN models is SPNP [12].

The model types used for dependability are in 2 categories: 1) combinatorial, and 2) state-space. Among the former are reliability block diagrams, fault trees, and reliability graphs. State-space model types include continuous-time Markov chains and Petri-net based models. Ref [14] compares various combinatorial model types based on their modeling power, and shows that FTRE is the most powerful combinatorial model type. The major handicap of combinatorial model types is that they cannot model certain kinds of dependencies, the most common one being the repair dependency among components where several components share a repair facility.

We first illustrate how an FTRE [15, 16] model can be converted to an equivalent GSPN or SRN model, and provide algorithms for these conversions. These algorithms can be easily modified to convert a reliability block diagram or reliability graph into GSPN or SRN models. The subnet constructions involved in these conversions are based on the kind of distribution assigned to each component of the system. For instance, in an FTRE model, it is common to assign failure probabilities (a distribution with mass at time zero and mass at infinity which sum to one) to each component, or assign a failure-time distribution in which a component can be faulty from the very start of system operation (mass at time zero). We illustrate subnet constructions for such commonly occurring cases.

[1]The singular & plural of an acronym are always spelled the same.

0018-9529/95/$4.00

We then show how repair dependencies such as shared repair persons between various components of a system, which cannot be modeled by FTRE models, can be modeled by Petri-net based models. Failed components queue up for repair if the repair facility is busy. The repair requests in the queue can be serviced according to some scheduling discipline such as FCFS, processor-sharing, non-preemptive priority, and preemptive resume priority. We provide GSPN & SRN subnet constructions for these 4 scheduling disciplines. These constructions provide insight into the differences between GSPN and SRN and reveal that certain features of SRN can be very useful in succinctly capturing failure-repair characteristics of systems. For a given system whose operational dependency on components is specified by an FTRE model (or any combinatorial model type), our methodology provides a direct way to construct a GSPN or a SRN model which incorporates repair dependencies among the components. For other applications of SRN to dependability modeling, refer to [17 - 19].

Section 2 briefly introduces Petri nets. Section 3 describes 2 examples that are used throughout this paper: 1) a simple *series-parallel* system, and 2) a more complex multiprocessor system. Sections 4 & 5 describe how an FTRE reliability model without repair can be converted into equivalent GSPN and SRN models respectively. Section 6 describes how to introduce repair (without any repair dependency) in GSPN and SRN models. Section 7 describes how to introduce repair (without dependency) into GSPN & SRN models. Section 8 describes how to incorporate repair (with dependency — shared repair facility among components) and different queuing disciplines in GSPN and SRN models.

Notation

x.up, x.dn a place where the presence of a token implies that component x is [up, down]

y.fl	a transition implying failure
sys.dn	a place where the presence of a token implies that the system is down
t	time instant
$p_i(t)$	Pr{component is in state i at time t}
π_i	steady-state probability of being in state i
r_i	reward rate in marking i
$\mathfrak{T}$	set of tangible markings
Z, $Z(t)$	[steady-state, instantaneous] reward rate, a r.v.
$\phi(t)$	continuous-time Markov chain
Ω	state-space of $\phi(t)$
$q_{i,j}$	transition rate from state i to state j, $i \neq j$
q	a number chosen such that $q \geq \max_i\{\lvert q_{i,i}\rvert\}$
$\boldsymbol{Q}$	infinitesimal generator matrix of $Z(t)$
$\boldsymbol{P}(t)$	state probability vector of $\phi(t)$ at time t
$\boldsymbol{P}(0)$	initial state probability vector of $\phi(t)$
$\boldsymbol{\Pi}$	steady-state probability vector of $\phi(t)$
M_i, P_i	[memory, processor] module i
N	interconnection network
$D_{i,j}$	disk j of processing subsystem i
X	time to failure of a component, a r.v.
λ	failure rate of the component
x	a node in an FTRE
c_i	input (child) i to a gate (AND or OR)
p	constant probability-mass at time ∞
RC[x]	reachability-count of node with label x
#tokens(x)	number of tokens in place x of a Petri net
C_i	component i
μ_i	repair rate of C_i
x_i	priority of transition t_i.

Other, standard notation is given in "Information for Readers & Authors" at the rear of each issue.

2. PETRI NETS

A Petri net [20] is a directed bipartite graph with 2 disjoint sets of nodes: *places* and *transitions*. In a graphical representation of a Petri net, places are represented by circles, and transitions are represented by bars. In a Petri net, there are a finite number of places and transitions. Nodes are connected by directed edges. A place is an *input* to a transition if there is an edge from the place to the transition. That edge is an *input arc*. A place is an *output* of a transition if there is an edge from the transition to the place. That edge is an *output arc*. An integral multiplicity can be associated with each input and output arc (default is 1).

A Petri net is *marked* if *tokens* are associated with the places. The dynamic behavior of the system is determined by the movement of tokens. The tokens move based upon the *firing* of transitions. A transition is *enabled* to fire if the number of tokens in each of its input places is at least equal to the multiplicity of the corresponding input arc from that place. When a transition fires, tokens are removed from each of its input places and deposited in each of its output places. The number of tokens removed from each of the input places of a firing transition is equal to the multiplicity of the corresponding input arc; the number of tokens deposited in its output places is equal to the multiplicity of the corresponding output arc. At any instant of time, more than one transition can be enabled but only one transition is allowed to fire. In a graphical representation of a Petri net, tokens are denoted by small dots or integers within a place. Multiplicity of arcs is denoted by putting a backslash on the arc and placing a positive integer with it. If multiplicity is not indicated, then default multiplicity of 1 is assumed.

A *marking* of a Petri net is the distribution of tokens in the set of places of the Petri net. Thus firing of a transition results in a new marking. Each marking defines a state of the system. If the number of tokens in the net is bounded, then there are a finite number of markings. A marking is *reachable* from an original marking if there is a sequence of transition firings starting from the original marking which results in that marking. The *reachability set* (*graph*) of a Petri net is the set of all markings that are reachable from the initial marking.

Petri nets have been extended for increased ease of use and enhanced modeling power. For instance, inhibitor arcs can be allowed that prevent firing of a transition when there is a token in any one of its inhibitor input places. Most importantly, firing times can be associated with transitions. When the distribution of firing times for all transitions is exponential, the net is a *stochastic Petri net* (SPN) [5]. Ajmone-Marsan *et al* [2] proposed *generalized stochastic Petri nets* (GSPN) which allow transitions to have 0 firing time (immediate transitions) or exponentially distributed firing times (timed transitions). SPN & GSPN are equivalent to CTMC. Extensions that allow non-exponential distributions are discussed in [21]. There are 2 types of markings of a GSPN: 1) *vanishing* (at least one immediate transition is enabled in that marking), and 2) *tangible* (otherwise).

Ciardo *et al* [12] introduced structural extensions to GSPN *eg*,

- enabling functions (guards) for transitions — transitions enabled based on some explicitly stated conditions and not just on distribution of tokens in input places,
- marking dependent arc multiplicities,
- timed transition priorities.

The resulting net with all these extensions can still be converted to a CTMC. However, there are tradeoffs with these extensions. Whereas these extensions make the task of modeling very simple and reduce the size of the net considerably, the complexity of understanding does increase. Ciardo et al [1] also introduced *stochastic reward nets* (SRN). SRN differ from GSPN in that reward rates (numerical values) are specifiable at the net level and translate into a reward rate being associated with each marking of the net. Use of reward rates reduces the size of the net because many aspects of a system that would have to be modeled explicitly by places & transitions in a GSPN can be expressed by arithmetic and Boolean expressions involving reward rates in an SRN.

Example 2-CPS

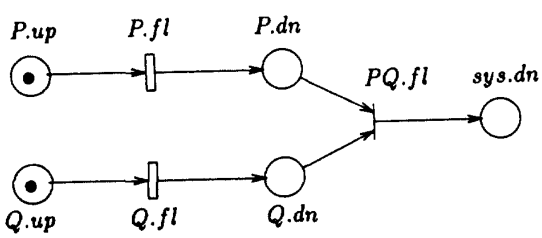


Figure 1. GSPN Model of a 2-Component *Parallel* System

There is a 2-component *parallel*[2] system. The components are labeled P & Q. Figure 1 is the GSPN reliability model. Firing of the transition Z.fl represents the failure of component W ($W=P,Q$). It results in removal of the token from the place W.up and its deposit in the place W.dn. A constant failure rate of component W is associated with this transition. Transition PQ.fl is an immediate transition which fires iff there is a token in place P.dn and a token in place Q.dn (both P & Q have failed). This results in a token being deposited in the place sys.dn, which implies failure of the system.

The system unreliability at time t is:

$$\Pr\{\text{there is a token in place sys.dn at time } t\}.$$

The analytic solution of this GSPN model requires conversion of the GSPN model to a Markov model and computation of the probability of being in the state, corresponding to the marking of the net in which there is one token in place sys.dn at time t.

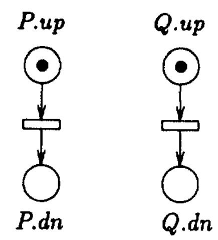


Figure 2. SRN Model of a 2-Component *Parallel* System

Figure 2 is a SRN reliability model of this system. The reward rate assignment to compute the reliability of the system is:

```
if (number of tokens in P.up == 1) or (number of tokens in Q.up == 1)
  then reward rate = 1 (system is operational)
  else reward rate = 0 (system is down)
endif
```

The mean value of the reward rate at time t gives the reliability of the system at time t. Thus SRN computes system dependability measures as reward-based measures. Compared with the GSPN model in figure 1, the SRN model in figure 2 does not need the 3 extra places (P.dn, Q.dn, sys.dn) nor an extra immediate transition (PQ.fl). The purpose of these added places & transitions is to check whether the system is failed in a given marking. The specification of reward rates obviates this explicit checking. ◄

Other measures can also be computed as reward-based metrics.

$$\mathrm{E}\{Z\} = \sum_{i \in \Im} r_i \cdot \pi_i,$$

(no time is spent in the vanishing markings).

$$\mathrm{E}\{Z(t)\} = \sum_{i \in \Im} r_i \cdot p_i(t).$$

Similarly, accumulated reward measures [1] can also be computed.

[2]The terms, *series & parallel* are used in their logic-diagram sense, irrespective of the schematic-diagram or physical-layout.

The π_i & $p_i(t)$ can be computed by solving the underlying CTMC obtained from the SRN.

Notation

$\{\phi(t),\ t \geq 0\}$ CTMC with state-space Ω
Ω $\{1,2, \dots ,n\}$.
$\boldsymbol{Q}$ $[q_{i,j}]$
$q_{i,i}$ $-\sum_{j=1, j\neq i}^{n} q_{i,j}$: infinitesimal generator matrix of the CTMC
q $\max_i |q_{i,i}|$
$\boldsymbol{P}(t)$ $[p_1(t), p_2(t), \dots , p_n(t)]$
$\boldsymbol{\Pi}$ $[\pi_1, \pi_2, \dots , \pi_n]$.

State i is the same as marking i.

$\boldsymbol{P}(t)$ is computed by solving a system of first order linear differential (Kolmogorov) equations:

$$\frac{d\boldsymbol{P}(t)}{dt} = \boldsymbol{P}(t) \times \boldsymbol{Q}; \qquad (1)$$

the initial condition is specified by $\boldsymbol{P}(0)$.

For numerical solution, see [22, 23].

$\boldsymbol{\Pi}$ is obtained by solving a system of linear equations:

$$\boldsymbol{\Pi} \cdot \boldsymbol{Q} = 0, \qquad (2a)$$

$$\sum_{i=1}^{n} \pi_i = 1. \qquad (2b)$$

For numerical solution, see [24].

3. EXAMPLES USED

These 2 examples are used throughout this paper.

3.1 A *Series-Parallel* System

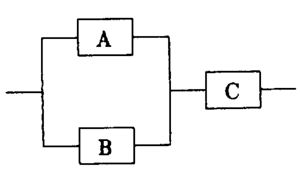


Figure 3. A 3-Component *Series-Parallel* System

A *series-parallel* system has 3 components *A, B, C,* as shown in figure 3. The system is operational as long as component *C* and at least one of components *A* or *B* are operational. Dependability of this system can be modeled by the FTRE model shown in figure 4. Any real fault-tolerant system typically consists of subsystems, many of which have *series* and *parallel* dependencies on basic components. The conclusions we draw based on this example are therefore generic and apply to any system with such dependencies.

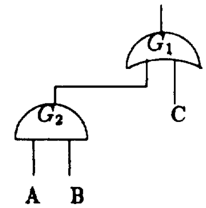


Figure 4. FTRE Model of the *Series-Parallel* System in Figure 3

Example 3.2 considers a more complex system whose subsystems have *series* and *parallel* dependencies, and it is easy to see how the conclusions from a simple model carry over to the models of complex systems. More complex dependencies [25] such as k-out-of-n dependence also occur in practice, but our purpose is served by illustrating with this simple *series-parallel* system.

3.2 A Fault-Tolerant Multiprocessor System

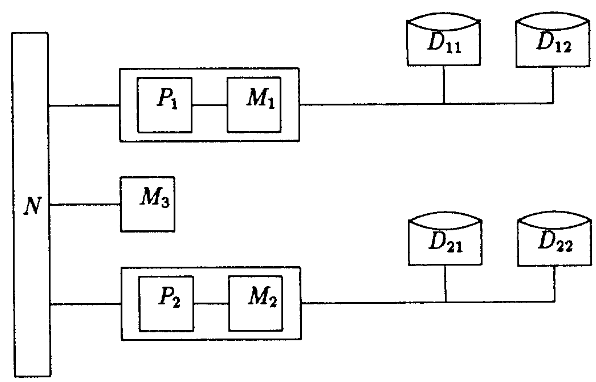


Figure 5. Fault-Tolerant Multiprocessor System

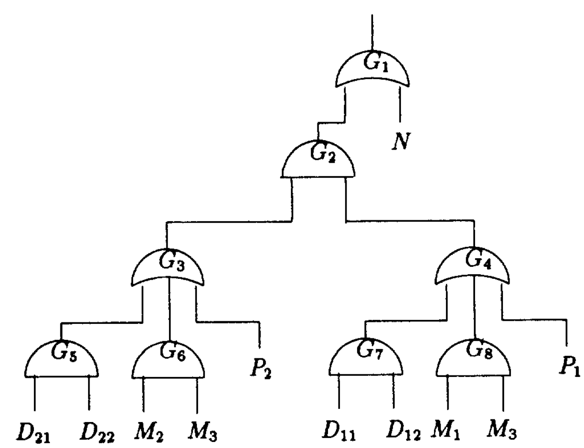


Figure 6. FTRE Model of the Multiprocessor System in Figure 5

Figure 5 shows the fault-tolerant multiprocessor system which consists of 2 processors P_i $(i=1,2)$ with a private memory M_i. A processor and a memory form a processing unit. Each processing unit is connected to a mirrored disk system. This forms a processing subsystem. Memory module M_3 is shared by P_1 & P_2. If M_1, M_2, or both M_1 & M_2 fail, then both the processing subsystems continue to function since P_1 & P_2 can both access M_3 if necessary. However, if M_1 & M_3 fail, then the processing subsystem of P_1 fails. Both the processing subsystems and M_3 are connected via an interconnection network N. The system is functional iff N is functional and 1 of the processing subsystems is functional. For a processing subsystem to be functional, the processor, memory module or shared memory module, and 1 of the 2 mirrored disks must be functional. For simplicity and sake of illustration, we restrict ourselves to this 2 processor system. This architecture is easily scaled to many processors. The reliability of this multiprocessor system is modeled by the FTRE in figure 6. M_3 is a repeated event in this model.

4. CONVERTING FTRE TO GSPN

This section illustrates how an FTRE model of a nonrepairable system can be converted to an equivalent GSPN model.

Assumption (unless otherwise stated)

1. The time-to-failure and time-to-repair distributions of any component are exponential. ◄

In principle, one could convert an FTRE to a Markov chain [26] and then convert the Markov chain into a GSPN model [14]. However, the GSPN model could be totally non-intuitive and appreciably less compact than the one obtained by careful design. Our direct conversion algorithm yields more-compact and more-intuitive GSPN models.

To generate the equivalent GSPN model, we must consider the input associated with the basic events in the FTRE model. This could be the time-to-failure distribution, failure probability, or instantaneous unavailability of the component. The time-to-failure distribution can be further classified as:

	mass at time 0	mass at time ∞
1. non-defective	no	no
2. defective	yes	no
3. defective	yes	yes
4. defective	no	yes.

Case 1 is discussed in detail. Cases 2 & 3 are discussed briefly since they can be handled in a similar fashion. Case 4 is not discussed because it does not commonly occur in practice.

4.1 Non-defective Failure-Time Distribution

$$\Pr\{X=0\} = 0,$$

$$\Pr\{X \le t\} = 1 - \exp(-\lambda \cdot t),$$

$$\Pr\{X < \infty\} = 1.$$

Assumption

2. Basic events as well as outputs of gates can be shared (repeated). ◄

Conversion Algorithm

0a. For each event: Count the number of times an event (basic or output of a gate) is repeated and set RC[x]. Then RC[x] is at least 1 for each x, unless there is some error in the specification of the FTRE.

0b. For each node x: set DONE[x] ← FALSE (this indicates that the subnet for this component/gate has not been generated).

/* These two steps, 0a & 0b, can be carried out in $O(n)$ time (n is the number of events in the FTRE). */

1. Use Algorithm FTRE_to_GSPN(x) in table 1.

/* This is a postorder traversal of the FTRE starting from the root. The equivalent GSPN model is obtained by calling this procedure; x points to the top gate of the FTRE. This algorithm is recursive (calls itself). */

TABLE 1
Conversion Algorithm for FTRE to GSPN

```
Algorithm FTRE_to_GSPN(x)
begin
if (x ≠ NIL) then
  Test True
   Case (x is a basic event) and (DONE[x] == FALSE):
    Construct the subnet in figure 7a and label each place;
    DONE[x] ← TRUE; EndCase
   Case (x is an AND gate):
    Let c_1,...,c_x be the inputs (children) of x
     foreach c_i, i ← 1,...,x do
      FTRE_to_GSPN(c_i)
    Construct the subnet in figure 7b
    DONE[x] ← TRUE; EndCase
   Case (x is an OR gate):
    Let c_1,...,c_x be the inputs (children) of x
     foreach c_i, i ← 1,...,x do
      FTRE_to_GSPN(c_i)
    Construct the subnet in figure 7c
    DONE[x] ← TRUE; EndCase
   EndTest
  EndIf
end
Construct inhibitor arcs from root.dn to all the timed transitions.
```

At the end, after the postorder traversal of the entire FTRE is completed, construct inhibitor arcs from the place root.dn (the dn place for the top gate in the FTRE) to all the timed transitions. This is done so that after the system fails, failure of operational components is disallowed to prevent generation of unnecessary markings. This reduces the number of states of the underlying Markov chain. Thus, both storage & time are saved since a smaller Markov chain needs to be generated &stored.

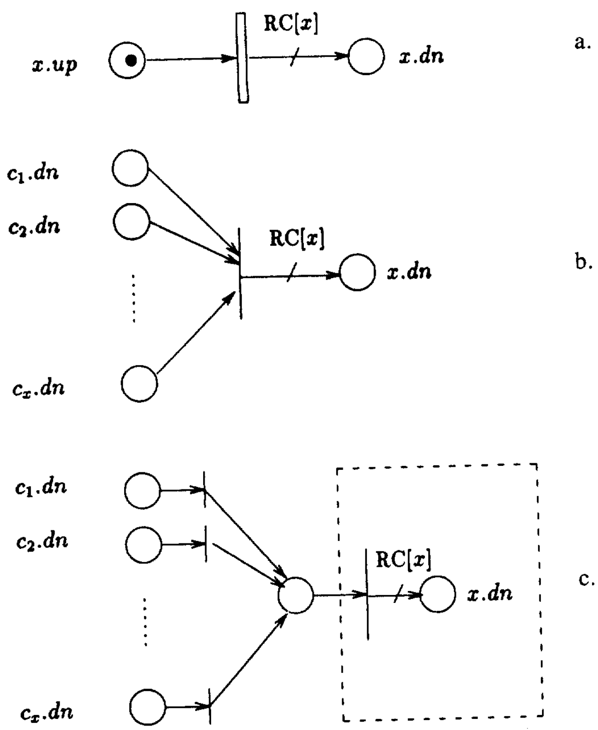


Figure 7. GSPN Subnets for Converting an FTRE Model to a GSPN Model

The complexity of the GSPN model can be expressed in terms of number of places & transitions. After this conversion algorithm, then:

$\#(\text{places}) \geq 2 \cdot \#(\text{components}) + \#(\text{gates})$
$\#(\text{timed transitions}) = \#(\text{components})$
$\#(\text{immediate transitions}) \geq \#(\text{AND gates}) + \Sigma_{g \in \text{OR gates}} \#(\text{inputs to gate g})$

The number of immediate transitions & places could be more than the sum on the r.h.s. if any of the **OR** gates in the FTRE are repeated since an extra place and immediate transition (see the dashed rectangular box in figure 7c) are needed in that case.

The GSPN models obtained from converting the FTRE models of the *series-parallel* system (figure 4) and the multiprocessor system (figure 6) are shown in figures 8 & 9 respectively. They show how the subnet constructions for the *series-parallel* dependence carry over from the simple *series-parallel* system to the more complex multiprocessor system.

3.2 Failure-Time Distribution with Mass at t=0

A defective distribution with probability $1-p$ at time 0 is assigned to each component.

$\Pr\{X=0\} = 1 - p,$

$\Pr\{X \leq t) = 1 - p + p \cdot (1 - \exp(-\lambda \cdot t)),$

$\Pr\{X < \infty) = 1.$

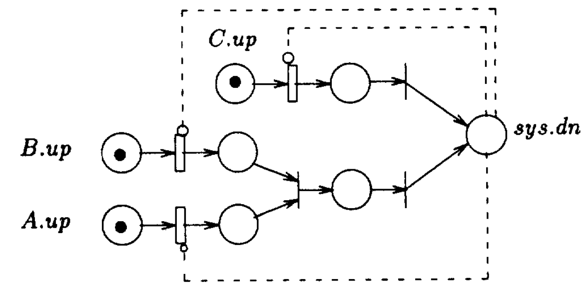


Figure 8. GSPN Model of the *Series-Parallel* System in Figure 4

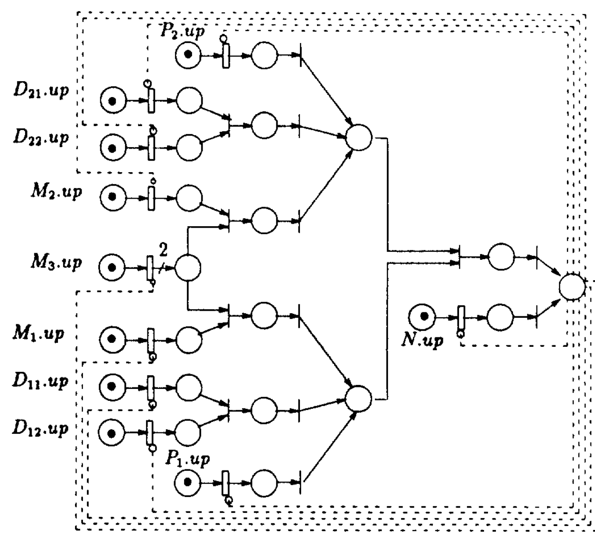


Figure 9. GSPN Model of the Multiprocessor System in Figure 6

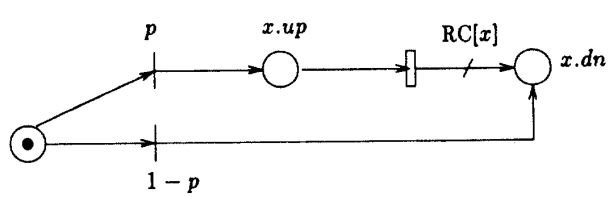


Figure 10. GSPN Subnet when Failure-Time Distribution Has Mass at $t=0$

A common example occurs when a component can be faulty in the beginning (time 0) with some probability. To model such a scenario, we need to modify figure 7a as shown in figure 10. Another way to look at this is to compute a new initial-state distribution. The initial distribution is:

With probability p, there is one token in place x.up at the start and with probability $1-p$, there is a token in place x.dn at the start. Figures 7b & 7c remain the same and so does the algorithm in table 1.

4.3 Failure-Time Distribution with Mass at $t = 0$ & ∞

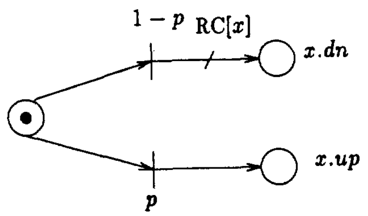


Figure 11. GSPN Subnet with Specified Failure-Probability of a Component

A constant failure probability $1-p$ is assigned to each component. Looking at it as a distribution of defectives implies that there is probability $1-p$ at time 0 and probability p at time ∞.

$$\Pr\{X = 0\} = 1 - p,$$

$$\Pr\{X = \infty\} = p.$$

An example occurs when a component is either faulty or fault-free (does not fail as time progresses) from the very start of system operation. To model this scenario, we need to modify figure 7a as shown in figure 11. Figures 7b & 7c remain the same and so does the algorithm in table 1.

4.4 Discussion

In all these cases (sections 4.1 - 4.3), the algorithm to construct the overall GSPN model remains the same. Only the subnet for each component changes depending upon the kind of distribution assigned to each component. By virtue of our constructions, this methodology extends to the case where a defective failure-time distribution is specified for some components while a non-defective failure-time distribution is specified for the others. The important thing is proper labeling of places x.dn and x.up in the subnet for a component x. Once these places have been generated for each component x, the construction of the rest of the net remains the same regardless of the kind of distribution assigned to each component.

5. CONVERTING FTRE TO SRN

This section illustrates how an FTRE reliability model (no repair) can be converted to an equivalent SRN model. In the process of doing so, we hope to distinguish between SRN and GSPN. As we did for GSPN in section 4, we discuss different cases based on the kind of failure-time distributions assigned to the basic events in the FTRE model.

5.1 Non-Defective Failure-time Distribution

The algorithm to convert an FTRE into an SRN is based on a similar postorder traversal of the FTRE as the algorithm (table 1) for converting an FTRE to a GSPN. The difference is in the actions taken when a node is encountered. Every time a gate is encountered, instead of constructing a subnet of immediate transitions and places, a reward rate function is constructed.

Unlike the conversion to GSPN, we do not need to perform the preprocessing step to count the number of times an event (basic or output of a gate) is repeated, because the value of RC[x] is not needed in this algorithm. For each node x, set DONE[x] ← FALSE. The remaining steps are carried out by a postorder traversal of the FTRE starting from the root. Every time a node (a basic event or a gate) is encountered, a specific action is performed. Table 2 shows the algorithm for the postorder tree-traversal

TABLE 2
Conversion Algorithm for FTRE to SRN

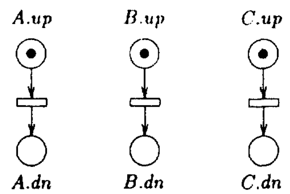
```
Algorithm FTRE_to_SRN(x)
begin
if (x ≠ NIL) then
  Test True
    Case (x is a basic event) and (DONE[x] == FALSE):
      Construct the subnet in figure 7a and label each place;
      bool(x) ← (#token(x.dn) == 1)
      DONE[x] ← TRUE; EndCase
    Case (x is an AND gate):
      Let c1,...,cx be the inputs (children) of x
      foreach ci, i ← 1,...,x do
        FTRE_to_SRN(ci)
      bool(x) ← bool(c1) ∧ bool(c2) ∧ ... ∧ bool(cx)
      DONE[x] ← TRUE; EndCase
    Case (x is an OR gate):
      Let c1,...,cx be the inputs (children) of x
      foreach ci, i ← 1,...,x do
        FTRE_to_SRN(ci)
      bool(x) ← bool(c1) ∨ bool(c2) ∨ ... ∨ bool(cx)

      DONE[x] ← TRUE; EndCase
    EndTest
  EndIf
end
if (bool(root) == 1)
  then disable all the transitions in the net.
  endif
```

The idea behind the halting condition is to prevent generation of unnecessary markings. Suppose that the system fails due to failure of some components, and is shut down. This shut-down implies that no more activity takes place in the system, *ie*, operational components can no longer fail. Thus, all the transitions within the system must be disabled. If the transitions are not disabled, then many more markings will be generated, each of which represent a system failure state. The halting-condition disables all the transitions after system failure and prevents generation of these unnecessary markings.

The SRN model is obtained by calling: **FTRE_to_SRN(root).** The reliability of the system is specified by the reward function:

if (bool(root) == 0)
  then $r=1$ (system is operational)
  else $r=0$ (system is failed)
  endif

The system reliability at time t is computed as the mean value of the reward rate r at time t.



Gate	Boolean Function
$bool(G_2)$	$(\#tokens(A.dn) == 1) \wedge (\#tokens(B.dn) == 1)$
$bool(G_1)$	$bool(G_2) \vee (\#tokens(C.dn) == 1)$

Halting Condition
if ($bool(G_1) == 1$) then disable all the transitions

Figure 12. SRN Model of the *Series-Parallel* System in Figure 4

The SRN models obtained from converting the FTRE models of the *series-parallel* system (figure 4) and the multiprocessor system (figure 6) are shown in figures 12 & 13 respectively. bool(G_1) is used to specify the system reliability in both the cases. Upon comparing these SRN models with the equivalent GSPN models in figures 8 & 9 we note:

- SRN models replace the mesh of immediate transitions and places in GSPN models by a reward rate assignment.
- The use of the halting condition in SRN avoids the need of inhibitor arcs to prevent generation of unnecessary markings. ◄

The complexity of the SRN model in figure 13 is:

#(places) = 2 · #(components)

#(timed transitions) = #(components)

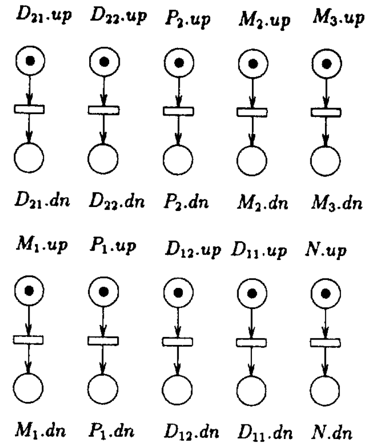
#(immediate transitions) = 0

5.2 Failure-Time Distribution with Mass at $t=0$

The SRN subnet for each component is the same as the GSPN subnet in figure 10.

5.3 Failure-Time Distribution with Mass at $t = 0$ & ∞

The SRN subnet for each component is the same as the GSPN subnet in figure 11.



Gate	Boolean Function
$bool(G_8)$	$(\#tokens(M_1.dn) == 1) \wedge (\#tokens(M_3.dn) == 1)$
$bool(G_7)$	$(\#tokens(D_{11}.dn) == 1) \wedge (\#tokens(D_{12}.dn) == 1)$
$bool(G_6)$	$(\#tokens(M_2.dn) == 1) \wedge (\#tokens(M_3.dn) == 1)$
$bool(G_5)$	$(\#tokens(D_{21}.dn) == 1) \wedge (\#tokens(D_{22}.dn) == 1)$
$bool(G_4)$	$bool(G_7) \vee bool(G_8) \vee (\#tokens(P_1.dn) == 1)$
$bool(G_3)$	$bool(G_5) \vee bool(G_6) \vee (\#tokens(P_2.dn) == 1)$
$bool(G_2)$	$bool(G_3) \wedge bool(G_4)$
$bool(G_1)$	$bool(G_2) \vee (\#tokens(N.dn) == 1)$

Halting Condition
if ($bool(G_1) == 1$) then disable all the transitions

Figure 13. SRN Model of the Multiprocessor System in Figure 6

5.4 Discussion

In these cases (sections 5.1 - 5.3) the SRN model consists simply of such subnets for each component, unlike GSPN models which need the mesh of immediate transitions, places, and inhibitor arcs.

6. MODELING REPAIR

(Without Repair-Dependencies)

In sections 3 - 5 the system was non-repairable. We now consider how to model repairable systems. In practice, the repair of a failed component consists of calling the repair person, removal of bugs, purchase of new component, replacement of faulty component, installing the new component, reconfiguring the new component, and testing the new component. For simplicity & illustration, we group all these steps together into a collective action called *repair*. Combinatorial-model types such as reliability block diagrams, FTRE, and reliability graphs, can model only the case where each component of the multiprocessor system has an s-independent repair person.

6.1 GSPN Models

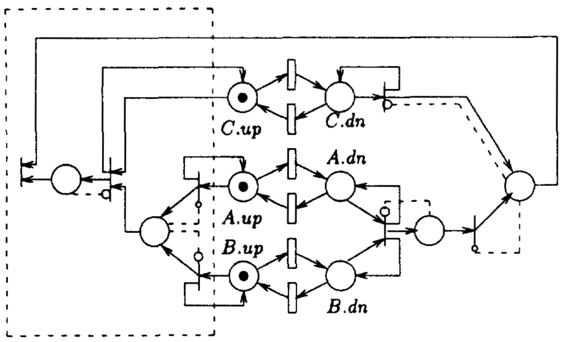


Figure 14. GSPN Model of the *Series-Parallel* System (in Figure 4) with Repair

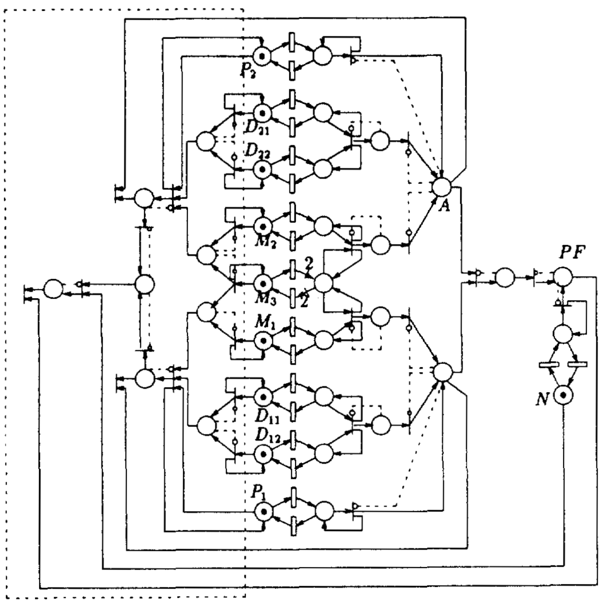


Figure 15. GSPN Model of the Multiprocessor System (in Figure 6) with Repair

The GSPN models of the *series-parallel* system and the multiprocessor system where each component has its *s*-independent repair facility are shown in figures 14 & 16 respectively. In these models, we have not shown the inhibitor arcs (to disable failure transitions after system failure) for sake of clarity, but they are present just as in the models in figures 8 & 9. Comparing the GSPN models with and without repair, figures 8 & 14 and figures 9 & 15, we find that it is not enough to introduce only the repair transitions to model repair of components. In the GSPN models without repair (figures 8 & 11) the flow of tokens is 1-way and that keeps the net simple. However, when repair is introduced (figures 14 & 15) the flow of tokens is 2-way, and that requires more output arcs and inhibitor arcs in the mesh of immediate transitions and places. We also need to introduce a *complementary mirrored subnet* to remove appropriately the tokens from the places which indicate different subsystem failures in order to reflect component repair. We call this subnet complementary since the AND & OR dependencies of subsystems or components are complemented: AND becomes OR and vice-versa.

For example, consider place A in the GSPN in figure 15.

If processor P_2 fails

OR if both disks D_{21} AND D_{22} have failed

OR if both memory modules M_2 AND M_3 have failed,

then there will be a token in place A. After repair, suppose that none of the 3 conditions hold, then we must remove the token from place A; *ie*,

if 'P_2 is up' AND 'either of disks D_{21} OR D_{22} is up' AND 'either of M_2 OR M_3 is up',

then we must remove the token from place A. These conditions are complementary to the conditions which led to the deposit of a token in place A. The complementary subnet modeling these conditions is enclosed in the dashed rectangular box in figures 14 & 15. The leftmost immediate transition in these boxes has no output place; *ie*, the tokens disappear when this transition fires.

One of the other modifications required is the need of several inhibitor arcs on the immediate transitions (both in the original subnet and its complementary subnet) to prevent these transitions from continuously firing (since at least one of the input places to these transitions is also one of the output places). Thus, unless the inhibitor arcs are used, these transitions will fire indefinitely. An algorithm to convert an FTRE model where each component has its own repair facility into a GSPN model can be derived based on the arguments given here. It will be similar to the algorithm in table 1. Various output measures can be computed from this model. Steady-state probability of a token in place PF gives *steady-state* unavailability of the system. Transient probability of a token in place PF gives *instantaneous* unavailability of the system.

The complexity of the GSPN models with repair is:

$$\#(\text{places}) \geq 2\cdot\#(\text{components}) + 2\cdot\#(\text{gates})$$

$$\#(\text{timed transitions}) = 2\cdot\#(\text{components})$$

$$\#(\text{immediate transitions}) \geq \Sigma_{g\in \text{gates}} (\#(\text{inputs to gate } g) + 1)$$

The complexity of the complementary net (number of places & transitions) is nearly the same as the complexity of the original net (modeling the FTRE with no repair). Thus, the size of the GSPN nearly doubles in order to model repair.

6.2 SRN Models

The SRN models of the *series-parallel* system and the multiprocessor system where each component has an *s*-independent

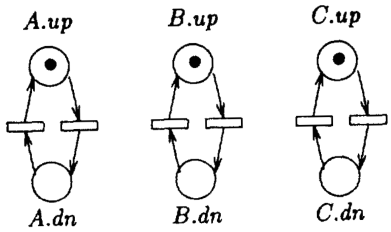


Figure 16. SRN Model of the *Series-Parallel* System (in Figure 4) with Repair

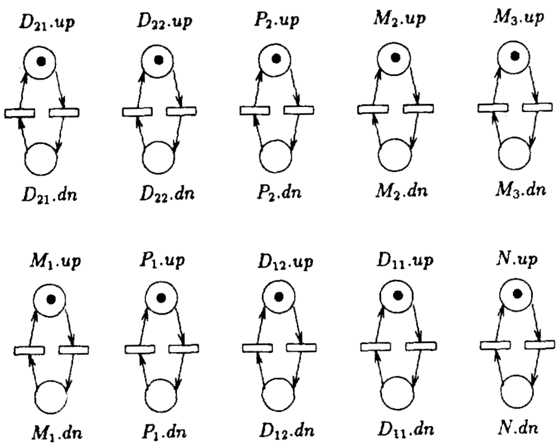


Figure 17. SRN Model of the Multiprocessor System (in Figure 6) with Repair

repair facility are shown in figures 16 & 17. Comparing these models with the GSPN models in figures 14 & 15, the usefulness of SRN over GSPN becomes obvious. The only modification needed for the SRN models in figures 12 & 13 is to add transitions for repair of components. The Boolean function bool(G_1) which was used to specify the reward function for reliability of the *series-parallel* and the multiprocessor system (figures 12 & 13 respectively) can also be used to specify the reward function for the availability of the systems. However, there is no halting condition in this case since the system is repairable (repair transitions must not be *disabled* after system failure). Instead, there are guards for each failure transition. These guards disable the failure transitions while the system is down (*ie*, components do not fail while the system is down). The guard for each failure transition in the SRN model in both figures 16 & 17 would be:

if (bool(G_1) == 1)
  then disable the transition\\
  else enable the transition
  endif

A guard is specified for each transition independently. The failure transitions are enabled once the system is operational again. Contrasting the SRN models with the equivalent GSPN models where a complementary subnet of about the same size as the original subnet must be constructed to model the repair of components, the SRN models more succinctly capture the failure-repair behavior of a system than do the GSPN models.

Besides the standard output measures such as instantaneous & steady-state availability, we can also compute *cumulative* up (or down) time of the system until time t. This is done by computing the accumulated reward in the system *up* (or *down*) states.

The complexity of the SRN models with repair is:

$$\#(\text{places}) = 2 \cdot \#(\text{components})$$

$$\#(\text{timed transitions}) = 2 \cdot \#(\text{components})$$

$$\#(\text{immediate transitions}) = 0.$$

7. MODELING REPAIR

(With Repair-Dependencies)

Section 6 considered the simple case where each component of a system has its s-independent repair facility. In practice, this is not the case. Usually, repair facilities are shared among components. If a component fails while the repair facility is busy, then it has to queue for service. Components that queue for service are serviced according to some scheduling policy. This section shows how to model a) such repair dependency, and b) various scheduling disciplines using GSPN & SRN models. Example 3-CPS is used in sections 7.1 - 7.4.

Example 3-CPS

A *parallel* system has 3 components C_1, C_2, C_3 that share a repair facility R. The repair discipline is different for each case. ◄

7.1 FCFS Repair Discipline

Example 3-CPS is used with the repair discipline: Components that arrive for repair at a repair facility are served in the order of arrival.

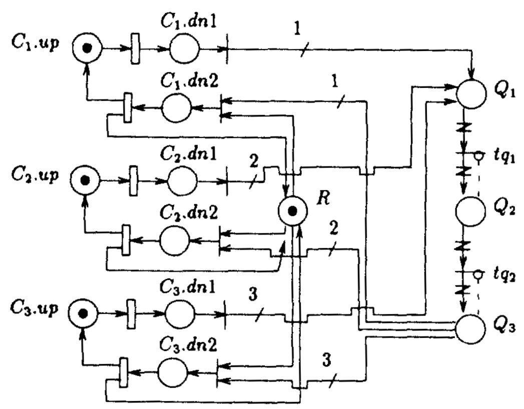


Figure 18. SRN Subnet for Modeling FCFS Repair Discipline

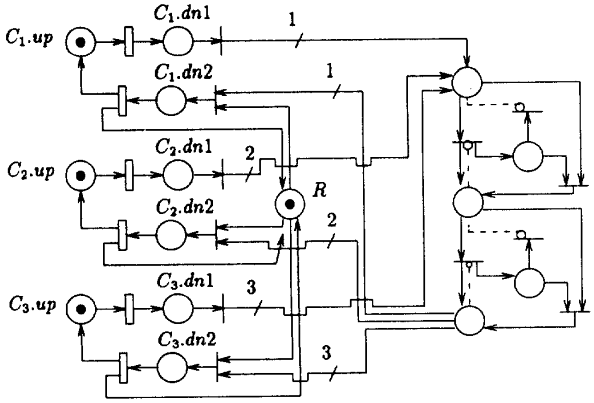


Figure 19. GSPN Subnet for Modeling FCFS Repair Discipline

Figure 18 shows the SRN subnet for this discipline. The component that arrives for repair first, grabs the token from place R (*ie*, grabs the repair facility) and releases it only after repair is completed. The FCFS queue is modeled by the places Q_1, Q_2, or Q_3 and the immediate transitions and inhibitor arcs between them. Each C_i is identified by i tokens in any of the places Q_1, Q_2, Q_3. The arcs with a '**Z**' like sign are variable cardinality arcs, a special feature of SRN. Each time transition tq_i $(i=1,2)$ fires, it removes as many tokens as present in Q_i and places them in Q_{i+1}. The reward rate r for availability of this system is:

if ((#tokens(C_1.up) $==$ 1) $\vee$ (#tokens(C_2.up) $==$ 1) $\vee$ (#tokens(C_3.up) $==$ 1))
then $r = 1$ (system is up)
else $r = 0$ (system is down)
endif

Figure 19 shows the GSPN subnet of the same system with FCFS repair queue. This differs from the SRN subnet only in the modeling of the FCFS queue. ◄

Since GSPN do not allow variable cardinality arcs, we need to model explicitly that behavior [12], and the net becomes appreciably complicated.

7.2 Preemptive Resume Priority (PRP) Repair Discipline

Example 3-CPS is used with the PRP repair discipline: Components that arrive for repair at a repair facility are served in the order of component priority — that priority decreases in the order C_1, C_2, C_3.

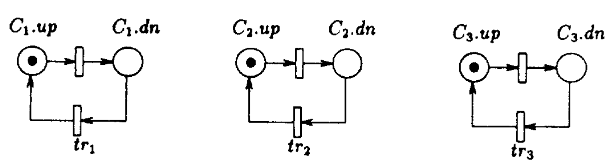


Figure 20. SRN Subnet for Modeling PRP Repair Discipline

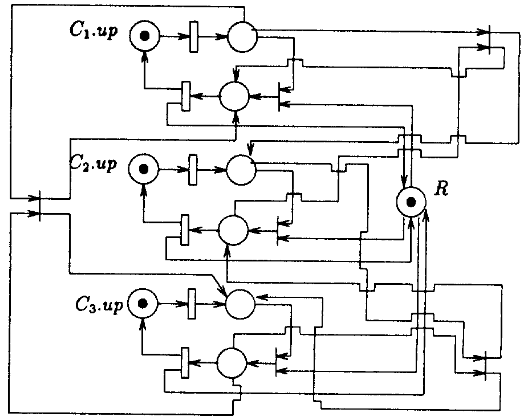


Figure 21. GSPN Subnet for Modeling PRP Repair Discipline

If a high priority component needs repair while a low priority component is being repaired, then the repair of low priority component is preempted and resumed after the repair of high priority component is completed. By virtue of memoryless property of exponential distribution, the amount of remaining repair time has the same distribution as the original repair time.

Figure 20 shows the SRN model for this system. Priority x_i, $i=1,2,3$ $(x_1 > x_2 > x_3)$ is assigned to the timed transition tr_i. An enabled timed transition is disabled if another timed transition with higher priority is enabled before this transition fires. This models the PRP repair discipline. Although the repair facility is a shared resource which is in contention when more than one component has failed, it is not explicitly modeled.

Figure 21 shows the equivalent GSPN subnet. Since GSPN does not allow priorities on timed transitions, the PRP repair discipline has to be explicitly modeled. Comparing the models in figures 20 & 21 shows that a simple feature of SRN results in a appreciably more-concise model specification than GSPN.

7.3 Non-Preemptive Priority Repair (Non-PRP) Discipline

Example 3-CPS is used with the Non-PRP repair discipline: Same as PRP except that the component which is being repaired currently is not preempted if a higher priority component arrives for repair. However, after the current repair completes, then the highest priority component from the queued components is selected for repair.

Figure 22 shows the GSPN model for this system. The priority is modeled by inhibitor arcs. For instance, these arcs guarantee that if C_1 and C_2 (or C_3) are waiting in the queue for repair, then C_1 begins repair first, and C_2 (or C_3) is repaired after C_1 finishes repair. This could also be modeled by simply assigning priorities x_1, x_2, x_3 $(x_1 > x_2 > x_3)$ respectively to the immediate transitions t_1, t_2, t_3. GSPN & SRN result in the same model specification.

7.4 Processor Sharing (PS) Repair Discipline

Example 3-CPS is used with the PS repair discipline: No queuing takes place at the repair facility. Instead, each failed component perceives the repair facility to be slowed by a factor of k if there are k failed components waiting to be repaired at any instant.

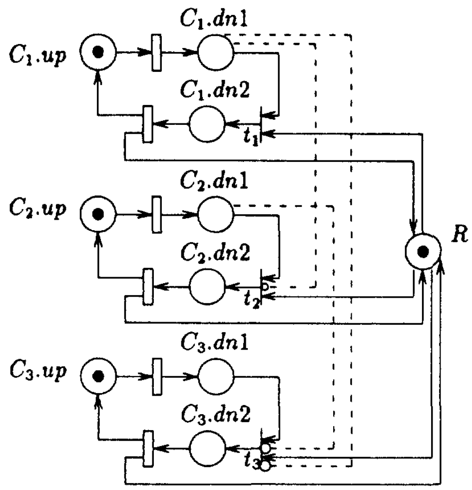
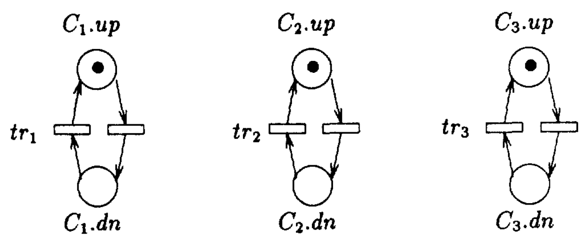


Figure 22. GSPN/SRN Subnet for Modeling Non-PRP Repair Discipline



Transition	Rate Function
tr_1	μ_1/(#tokens(C_1.dn) + #tokens(C_2.dn) + #tokens(C_3.dn))
tr_2	μ_2/(#tokens(C_1.dn) + #tokens(C_2.dn) + #tokens(C_3.dn))
tr_3	μ_3/(#tokens(C_1.dn) + #tokens(C_2.dn) + #tokens(C_3.dn))

Figure 23. GSPN/SRN Subnet for Modeling PS Repair Discipline

This is easily modeled by GSPN & SRN by assigning marking-dependent transition rates to transitions tr_1, tr_2, tr_3 as shown in figure 23; $1/\mu_i$ is the mean repair time for C_i. GSPN & SRN result in the same model specification.

7.5 Discussion

The overall GSPN models in sections 7.1 - 7.4 (different scheduling disciplines) contain the subnets shown in the corresponding figures, and the mesh of immediate transitions and places which models the operational dependency of the system onto its components. However, compared to the no-repair-dependency case (figure 15), this mesh is more complicated since now there is more than one place per component where a token indicates failure of a component. For example, a token in place C_1.dn1 or C_1.dn2 (figures 19 & 22) indicates that component C_1 is down.

ACKNOWLEDGMENT

This work was partially done while the first author was at Duke University. This work was supported in part by the US National Science Foundation under grant CCR-9108114 and by the US Naval Surface Warfare Center under grant N60921-92-C-0161.

REFERENCES

[1] G. Ciardo, J. Muppala, K. Trivedi, "Analyzing concurrent and fault-tolerant software using stochastic reward nets", *J. Parallel & Distributed Computing*, vol 15, 1992, pp 255-269.

[2] M. Ajmone-Marsan, G. Conte, G. Balbo, "A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems", *ACM Trans. Computer Systems*, vol 2, num 2, 1984, pp 93-122.

[3] M. Ajmone-Marsan, G. Balbo, G. Conte, *Performance Models of Multiprocessor Systems*, 1986; MIT Press.

[4] J. Meyer, A. Movaghar, W. Sanders, "Stochastic activity networks: Structure, behavior, and application", *Int'l Workshop Petri Nets and Performance Models*, 1985 Jul, pp 106-115; Torino, Italy.

[5] M. Molloy, "Performance analysis using stochastic Petri nets", *IEEE Trans. Computers*, vol C-31, 1982 Sep, pp 913-917.

[6] J. Muppala, "Performance and dependability modeling using stochastic reward nets", *PhD Thesis*, 1991 Apr; Duke Univ.

[7] W. Sanders, "Construction and solution of performability models based on stochastic activity networks", *PhD Thesis*, 1988; Univ. of Michigan.

[8] B. Beyaert, G. Florin, P. Lonc, S. Natkin, "Evaluation of computer system dependability using stochastic Petri nets", *Digest 11th Ann. Symp. Fault-Tolerant Computing*, 1981 Jun, pp 79-81; IEEE Computer Society Press.

[9] H. Kantz, K. Trivedi, "Reliability modeling of MARS system: A case study in the use of different tools and techniques", *Int'l Workshop Petri Nets and Performance Models*, 1991; Melbourne, Australia.

[10] J. Muppala, A. Sathaye, R. Howe, K. Trivedi, "Dependability modeling of a heterogenous VAXcluster system using stochastic reward nets", *Hardware & Software Fault Tolerance in Parallel Computing Systems* (D. Averesky, *Ed*), 1992; Ellis Horwood Ltd.

[11] G. Ciardo, A. Blakemore, P. F. Chimento, *et al*, "Automated generation and analysis of Markov reward models using stochastic reward nets", *Linear Algebra, Markov Chains, Queueing Models, IMA Volumes in Mathematics and its Applications* (C. Meyer & R.J. Plemmons, *Eds*), vol 48, 1993, pp 145-191; Springer-Verlag.

[12] G. Ciardo, J. Muppala, K. Trivedi, "SPNP: Stochastic Petri net package", *Proc. Int'l Workshop Petri Nets and Performance Models*, 1989 Dec, pp 142-150; IEEE Computer Society Press.

[13] G. Chiola, "A software package for the analysis of generalized stochastic Petri nets", *Proc. Int'l Workshop Timed* Petri Nets, 1985 Jul; Torino, Italy.

[14] M. Malhotra, K. Trivedi, "Power-hierarchy among dependability model types", *IEEE Trans. Reliability*, vol 43, 1994 Sep, pp 493-502.
[15] J. Arsenault, J. Roberts, *Reliability and Maintainability of Electronic Systems*, 1980; Computer Science Press.
[16] B. Dhillon, C. Singh, "Bibliography literature on fault-trees", *Microelectronics & Reliability*, vol 17, 1978, pp 501-503.
[17] M. Balakrishnan, K. Trivedi, "Stochastic Petri nets for reliability analysis of communication network applications with alternate routing", *Reliability Eng'g and System Safety*, 1995, (to be published).
[18] J. Muppala, G. Ciardo, K. Trivedi, "Stochastic reward nets for reliability prediction", *Comm. Reliability, Maintainability Serviceability*, vol 1, num 2, 1994, pp 9-20.
[19] L. Tomek, J. Muppala, K. Trivedi, "Modeling correlation in software recovery blocks", *IEEE Trans. Software Engineering*, vol 19, 1993, num 11, pp 1071-1086.
[20] J. Peterson, *Petri Net Theory and the Modeling of Systems*, 1981; Prentice-Hall.
[21] H. Choi, V. Kulkarni, K. Trivedi, "Markov regenerative stochastic Petri nets", *Performance Evaluation*, vol 20, 1994, pp 337-357.
[22] M. Malhotra, J. Muppala, K. Trivedi, "Stiffness-tolerant methods for transient analysis of stiff Markov chains", *Microelectronics & Reliability*, vol 34, num 11, 1994, pp 1825-1841.
[23] M. Malhotra, "A computationally efficient technique for transient analysis of repairable Markovian systems", *Performance Evaluation*, 1996, (to appear).
[24] W. Stewart, "A comparison of numerical techniques in Markov modeling", *Comm. ACM*, vol 21, 1978 Feb, pp 144-152.
[25] J. Dugan, S. Bavuso, M. Boyd, "Dynamic fault-tree models for fault-tolerant computer systems", *IEEE Trans. Reliability*, vol 41, 1992 Sep, pp 363-377.
[26] J. Dugan, K. Trivedi, M. Smotherman, R. Geist, "The hybrid automated reliability predictor", *AIAA J. Guidance, Control, Dynamics*, 1986, May-Jun, pp 319-331.

AUTHORS

Dr. M. Malhotra; AT&T Bell Labs; Holmdel, New Jersey 07733 USA.

Manish Malhotra received BTech (1989) in Computer Science from Indian Institute of Technology, Delhi, and MS (1990) and PhD (1993) from Duke University in Computer Science. Since 1993 June, he has been a member of technical staff at AT&T Bell Labs, Holmdel. His interests include reliability & performability modeling of telecommunication networks & services.

Dr. K.S. Trivedi; Dept. of Electrical Engineering; Duke University; Durham, North Carolina 27706 USA.

Kishor S. Trivedi received the B.Tech from the Indian Institute of Technology, Bombay, and MS & PhD in Computer Science from the University of Illinois, Urbana-Champaign. He is the author of *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. His research interests are in computer & communication system reliability & performance modeling. He has lectured & published extensively on these topics. He is an IEEE Fellow. He is a Professor of Electrical Engineering at Duke University, and the director of a joint NSF, Duke, NC State Industry/University Cooperative Research Center for Advanced Computing & Communication. He has served as a Principal Investigator on various AFOSR, ARO, Burroughs, IBM, DEC, NASA, NIH, ONR, NSWC, Boeing, Union Switch & Signals, NSF, and SPC funded projects and as a consultant to industry and research laboratories. He was an Editor of *IEEE Trans. Computers* from 1983-1987. He is a co-designer of HARP, SAVE, SHARPE and SPNP modeling packages. These packages have been widely circulated.

Manuscript received 1994 September 6.

IEEE Log Number 94-12417 ◄TR►

AR&MS AR&MS AR&MS AR&MS AR&MS AR&MS AR&MS AR&MS AR&MS AR&MS AR&MS AR&MS AR&MS

1995 *Annual* Reliability and Maintainability *Symposium*

The **Alan O. Plait Award** for Tutorial Excellence

was bestowed upon
Lawrence M. Leemis, *PhD*
for Best Tutorial
"Probabilistic-Models & Statistical-Methods in Reliability"

was bestowed upon
A. (Gus) Constantinides
for Continued Excellence
"Basic Reliability"

These awards are named for Alan O. Plait who instituted and took charge of the Tutorials program in 1975, and then guided and encouraged its growth for 17 years. Each tutorial is rated by attendees and the Tutorial Committee according to its presentation content, visual materials, speaker elocution, and printed text. The number of years a tutorial has been presented is an additional criterion for the *Continued Excellence* award.