RELIABILITY ANALYSIS OF CSP SPECIFICATIONS USING PETRI NETS AND MARKOV PROCESSES

Krishna M. Kavi, Frederick T. Sheldon, Behrooz Shirazi
University of Texas at Arlington
and
Ali R. Hurson
Pennsylvania State University

ABSTRACT

In our research we are developing methodologies and tools to permit stochastic analyses of CSP-based system specifications. In this regard, we have been developing morphisms between CSP-based models and Petri net-based stochastic models. This process has given us insight for further refinements to the original CSP specifications (i.e., identify potential failure processes and recovery actions). In order to create systems that meet user needs in terms of cost, functionality, performance and reliability, it is *essential* to relate the parameters needed for reliability analysis to the user level specification.

Keywords: Formal specification, CSP, Petri Nets, Reliability analysis, Markov models.

1. INTRODUCTION

Computers are increasingly used in every day life in today's society. These systems are monitoring and controlling complex and safety critical systems. It has been conjectured that formal mathematically precise methods should be used to design such systems. Among the benefits from using formal frameworks, we include [Ostroff 92]:

- In the process of formalizing informal requirements, ambiguities, omissions, and contradictions will often be discovered.
- A formal framework may lead to hierarchical semi-automated system development methods.
- A formal model can be verified for correctness by mathematical methods (rather than by exhaustive testing).
- A formally verified subsystem can be incorporated into a larger system with greater confidence that it will behave as specified.
- Different designs can be evaluated and compared.
- A clear specification of interactions among various subsystems may provide implementation insights for avoiding performance pitfalls.

"When it comes to the implementation of specifications formally, one does not do it by writing programs and then trying to prove that they meet the specifications. Instead, one constructs correct programs in small steps - each step taking the specification and producing something that is bit closer to the final program" [Hall 90]. At the other extreme, some formal specification and verification methods strive for "fool-proof" or "error-free" designs. A proof is <u>only</u> a demonstration that one formal statement follows from another, and the validity of a statement depends on the validity of the statement from which it is derived. Complex systems are placed in environments that are difficult to model accurately. Thus, it is not feasible (at least not cost-effective) to prove the correctness of a designed system in real environments. One must be satisfied by designing systems that will exhibit a high degree of dependability. Thus, future systems will be designed to tolerate unpredictable conditions and operate safely in the presence of hardware or software failures.

The research in formal specification and verification of complex systems has often ignored the specification of stochastic properties of the system. The normal practice is to derive designs and implementations of systems from formal specifications. Designers concurrently develop stochastic models of the target systems for the purpose of reliability and performance analyses. While detailed analyses require a clear understanding of the implementation (hardware/software failure modes, failure distributions, service distributions, workload, etc.), it is our belief that the cost of providing a desired level of reliability and performance can be related to user level specifications, even if only in terms of upper and lower bounds. It is also our belief that as specifications are refined into detailed designs and actual implementations, the reliability and performance requirements can also be refined to reveal the trade-offs in design alternatives.

Stochastic Petri-nets have been used to analyze complex distributed processing systems in terms of performance and reliability. Numerous tools have been developed for stochastic analysis of Petri nets (e.g., GSPN [Marsan 89], GreatSPN [Chiola 89], SPNP [Ciardo 89]). Petri nets however, are not very suitable for reasoning about the functional correctness of a system.

We have developed an initial set of rules for translating CSP (Communicating Sequential Processes) specifications

1060-3425/95 $4.00 © 1995 IEEE

into Petri nets [Kavi 93]. In this paper we will demonstrate, by using a simple example, (1) how CSP specifications can be converted into Petri nets, (2) how Petri nets can be embellished with failure modes, (3) how these failure modes can be converted into CSP processes so that the feasibility of certain failure modes can be examined by the user, and (4) how Petri nets can be analyzed for reliability (using user level information on failure rates).

2. FORMALISMS FOR SPECIFICATION AND ANALYSIS

Since CSP and other specification models are compositional, the usefulness of an analysis is improved by partitioning large systems into smaller subsystems whose reliability can then be approximated judiciously, giving greater comprehensibility and thereby reducing the analysis complexity. It is hoped that the insights gained will lead to a set of tools for the specification of functional and stochastic properties, as well as mechanical proofs and analyses for correctness, reliability and performance measures.

2.1 Communicating Sequential Processes. The CSP model was developed by Hoare and later extended by Olderog ([Hoare 85], [Olderog 86]). A program in CSP consists of n > 1 communicating processes; this is normally represented using the parallel composition operator (||), which is associative: $P = \{P_1 \| P_2 \| \ldots\ldots \| P_n\}$.

A process's actions are visible by means of its communications with other processes or the environment. The set of symbols representing the visible actions comprise the alphabet (S) for the process. Processes communicate synchronously by sending and receiving messages: the sending and receiving actions (or events) are indicated using the input (?) and output (!) actions. $P_i?x$ is the action of receiving a value sent by process P_i into variable x. $P_j!$ <expression> describes the action of sending the value of the expression to P_j. Synchronization is accomplished by using complementary input and output commands in the two communicating processes. Communication can be made selective by providing guards, where one of the alternative communication actions with a satisfied guard is selected. A guarded command has the general syntax of the form <guard> → <command list>. A command list is a set of commands defining a sequence of actions, alternative actions based on either deterministic or non-deterministic choice, recursive actions, or a STOP action. Stop terminates (or deadlocks) a process. The following summarizes CSP syntax:

$$P ::= \mathrm{STOP} \mid (a \rightarrow P) \mid (P\backslash b) \mid (P \sqcap Q)$$
$$\mid (P \,\square\, Q) \mid (P \,\|_b\, Q) \mid (P; Q) \mid (\mu x \bullet P)$$

In CSP, capitalized names are used for process names, and lower case characters are used to denote visible actions. Here, $(a \rightarrow P)$ means, action 'a' followed by P, $(P\backslash b)$ is the same as P except action b is hidden, $(P \sqcap Q)$ represents a non-deterministic choice between P and Q, $(P \,\square\, Q)$ represents a deterministic choice between P and Q, $(P \,\|_b\, Q)$ shows concurrent processes P and Q that synchronize on action b, (P; Q) a sequence between P and Q, $(\mu x \bullet P)$ is used for recursion.

2.1.1 THE CSP FOR A RAIL-ROAD CROSSING. In this example, a Rail-Road intersection is specified. The gate closes when a train arrives at the intersection and remains closed until the train leaves the intersection. Although the problem statement can be extended to handle multiple trains, only one train is specified here.

```
TRAIN =
        (IN_TRANSIT);
        (GATE ! a → AT_INTERSECTION);
        (GATE ! d → TRAIN)
GATE =
        (TRAIN ? a → CLOSE);
        (TRAIN ? d → OPEN→ GATE)
RAIL_ROAD_CROSSING =
        TRAIN ||{a,d} GATE
```

This specification shows two concurrent processes, the TRAIN and the GATE communicating via two activities, "a" and "d." The TRAIN outputs "a" (arriving) to the GATE as it approaches the intersection; proceeds through the intersection and outputs a "d" (departing) to the GATE as it leaves the intersection and continues to behave as a TRAIN. The GATE process receives an "a" from the TRAIN, closes the gate, waits for an input of "d" from the TRAIN before opening the gate and then behaves like a GATE. A few comments about the CSP specification are in order. The original CSP does not permit specification of time with actions, although some recent extensions to CSP permit the association of time with actions. Because CSP uses point-to-point communication it is awkward to describe the case where the GATE process accepts inputs from multiple TRAIN processes. Careful scrutiny reveals that the TRAIN process could enter the intersection (AT_INTERSECTION) before the gate closes which leads to unsafe behavior. Likewise, the train may depart while the gate is still closed which can be viewed as a fail-safe behavior. The Petri net equivalent reveals these flaws more readily (see Figures 1 and 2).

2.2 Stochastic Petri Nets. The Petri net was originally due to Carl Petri. In its simplest form, a Petri net is a directed bipartite graph, where the two types of nodes are known as places (shown as circles) and transitions (shown as bars). Places normally represent events while transitions represent actions. A transition is enabled if all its inputs contain at least one token (shown as dark circles inside places). Completion of the action defined by a transition causes a token to be assigned to each of its output places. When a place is the input to more than one transition, only one of the transitions is enabled based on a non-deterministic choice. The state of a Petri net is indicated by the number and location of tokens in places (known as a marking), and as transitions are enabled, the state of the Petri net moves from marking to marking. The complete set of markings of a Petri net can be obtained using reachability algorithms. When a Petri net is restricted to contain at most one token in a place (or a finite number of tokens, say k), such a Petri net is known as a safe net (or k-safe).

These initial concepts have been extended to permit probabilistic choices on the outputs of a place, inhibitor arcs to transitions (i.e., a transition is enabled in the absence of a token at its input place and such arcs can model zero testing), as well as the association of time and distributions with either places or transitions. See [Murata 89] for an excellent survey of Petri nets. We will rely on the stochastic Petri nets that permit the association of various probability distributions with transitions to model performance and reliability of the system. A stochastic Petri net (SPN) is a Petri net where each transition is associated with a random variable that expresses the delay from the enabling to the firing of the transition. When multiple transitions are enabled, the transition with a minimum delay fires first. When the random variable is exponential, the markings of the stochastic Petri net are isomorphic to the states of a finite Markov chain. The transition rate from state M_i to M_j = q_{ij} is given by q_{ij} = $\lambda_{i1} + \lambda_{i2} + . . .+\lambda_{im}$ where λ_{ik} is the delay in firing a transition t_k which takes the Petri net from marking M_i to M_j (when more than one transition can cause the transition from M_i to M_j). The performance and reliability analyses of the system represented by the Petri net can be achieved by using an equivalent Markov process.

2.3 Mapping of CSP-Level Specifications into Petri Nets. We have developed an initial set of rules for translating CSP specifications into Petri nets [Kavi 93]. The translation relies on the fact that CSP specifications are based on processes moving form one action to another. The *activities* which enable the actions of processes can be viewed as the events which are represented by places in a Petri net, while the *actions* are viewed as transitions in Petri nets. The translations between the CSP and Petri net models have not been formally verified to be isomorphic. However, we have developed rules which show the associated Petri net structure for the majority of CSP process structures and compositions. The Petri net equivalent of a CSP specification need not be unique, because of the need to introduce dummy places or transitions to maintain its bipartite nature. Intuitively, it is possible to reduce different Petri net equivalents into a canonical form. We plan to develop the necessary rules for producing canonical Petri net representations of CSP specifications.

Our goal is to demonstrate the feasibility of translating between CSP and Petri nets so that stochastic properties can be specified at the CSP level, and analyzed using stochastic Petri nets. Some example translations between CSP specifications and Petri nets are shown in the Appendix. Using these, we have converted the CSP example of the Rail-Road Crossing.

2.3.1 PETRI NET FOR THE RAIL-ROAD CROSSING EXAMPLE. The Rail-Road crossing presents a safety critical system where two tasks that operate independently must communicate in order to coordinate closing the gate when the train nears the crossing. The gate must remain closed until the train passes though. The Petri net is shown in Figure 1.

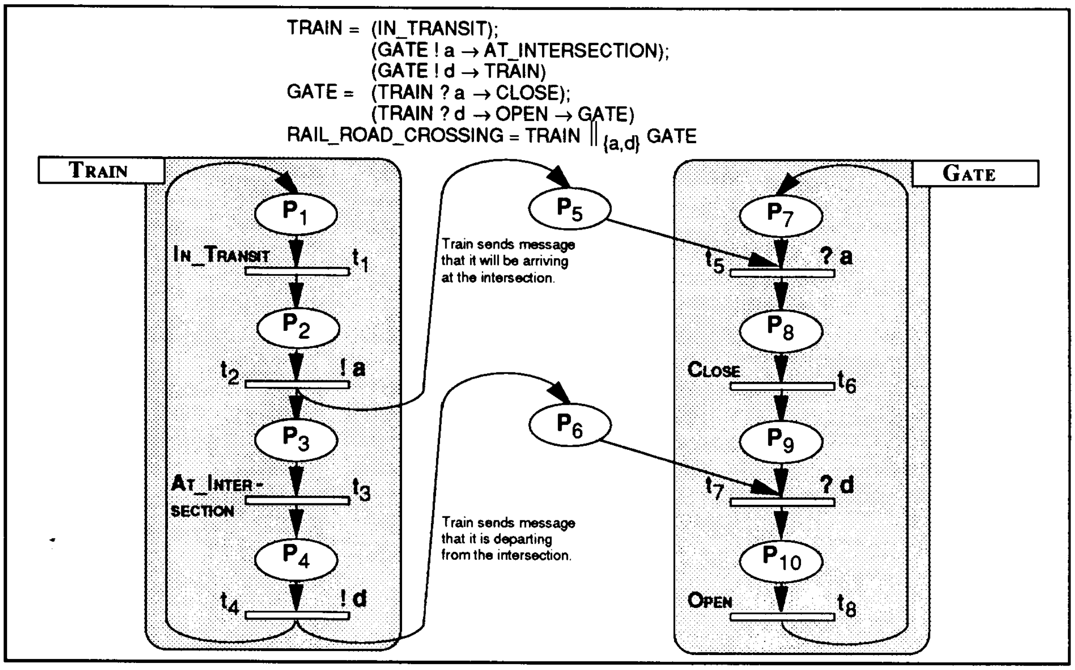
As stated earlier (¶2.1.1), a careful scrutiny reveals that the TRAIN process could enter the intersection before the gate closes, exposing the system to unsafe behavior. This potential flaw becomes immediately visible from the Petri net shown in Figure 1. If we assume that the gate always opens and closes sooner than the time it takes the train to reach the crossing, the Petri net can be viewed as hazard free.

Obviously some mechanism is needed to ensure that the train will not proceed unless the gate is closed. One way to redesign the system is to force the TRAIN process to wait until the GATE process completes CLOSEing the gate which will avoid such unsafe behavior. The Petri net of Figure 2 shows the additional synchronization (and its corresponding CSP) that is necessary to ensure the system will operate in such a manner.

In Figure 2, a failure of the communication related actions may lead to a deadlock (the train will halt), but synchronization between the TRAIN and GATE eliminates the possibility of trains passing through the intersection un-guarded by an open gate. Failure to OPEN the gate is not safety critical, yet should be avoided to prevent congestion of the associated infrastructure. It may be possible to use Reward nets (and performability analyses) to associate a cost with such delays in opening of the gate.

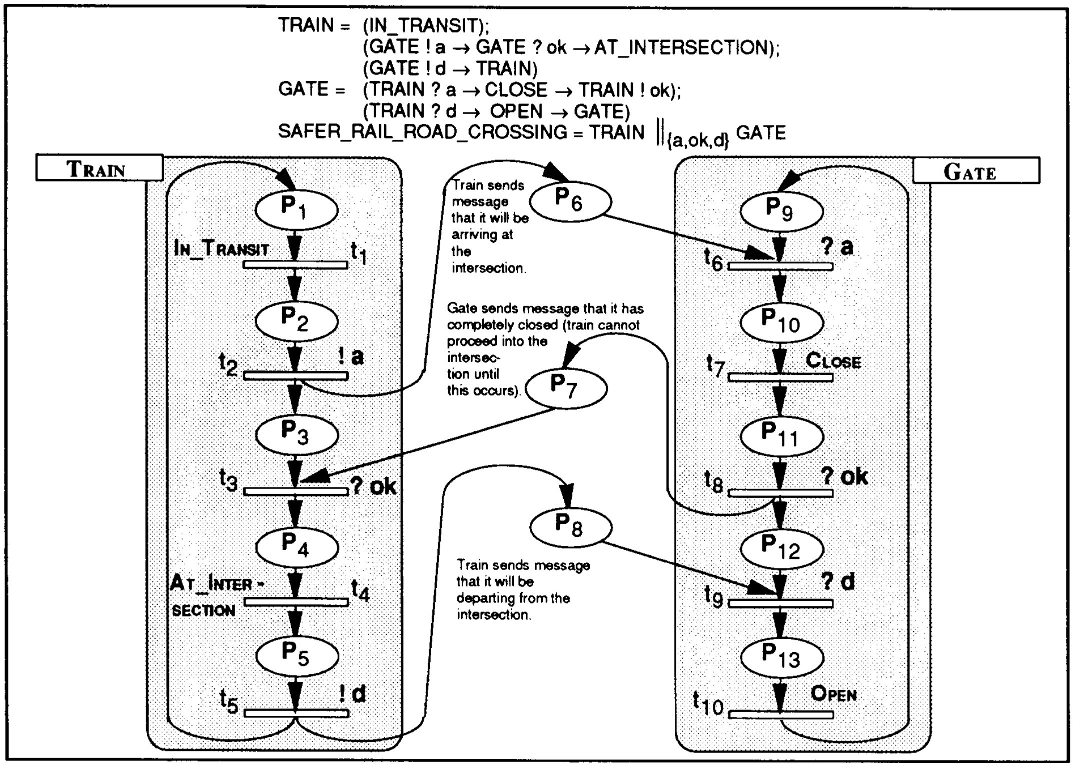
3. SPECIFICATION OF STOCHASTIC PROPERTIES

One of the major objectives of our research is to provide assistance to the user in specifying not only functionality but also reliability, performance and execution deadlines. In this paper we show how this is facilitated by the translation of CSP specifications into Stochastic Petri nets. One important benefit, as we have already shown, is how the Petri net (PN) equivalent of the Rail-Road crossing elucidated the need for additional synchronization to avoid a safety-critical failure.



```
TRAIN = (IN_TRANSIT);
        (GATE ! a → AT_INTERSECTION);
        (GATE ! d → TRAIN)
GATE =  (TRAIN ? a → CLOSE);
        (TRAIN ? d → OPEN → GATE)
RAIL_ROAD_CROSSING = TRAIN ||{a,d} GATE
```

Figure 1 Rail-Road Crossing with a Potential Hazard (unsafe PN Specification).



```
TRAIN = (IN_TRANSIT);
        (GATE ! a → GATE ? ok → AT_INTERSECTION);
        (GATE ! d → TRAIN)
GATE =  (TRAIN ? a → CLOSE → TRAIN ! ok);
        (TRAIN ? d → OPEN → GATE)
SAFER_RAIL_ROAD_CROSSING = TRAIN ||{a,ok,d} GATE
```

TRAIN
GATE
P_1
IN_TRANSIT
t_1
P_2
t_2
! a
P_3
t_3
? ok
P_4
AT_INTER-SECTION
t_4
P_5
t_5
! d
Train sends message that it will be arriving at the intersection.
P_6
Gate sends message that it has completely closed (train cannot proceed into the intersection until this occurs).
P_7
P_8
Train sends message that it will be departing from the intersection.
P_9
t_6
? a
P_{10}
t_7
CLOSE
P_{11}
t_8
? ok
P_{12}
t_9
? d
P_{13}
t_{10}
OPEN

Figure 2 Rail-Road Crossing with a Hazard Eliminated.

3.1 Failure Modes for the Rail-Road Crossing. Using the Petri net of Figure 1, we will assume that all transitions can fail. It should be noted however, that the Petri net of Figure 1 is unsafe, but for the purpose of the example here, we assume the Petri net to be safe (i.e., permit at most one token at a place). The Markings are shown in Figure 3. Note that P_{CF} and P_{NF} are places not identified in the Petri net but are used to designate critical and non-critical failure events respectively.

Markings M_5 and M_6 are critical markings resulting from *slow* firing of transitions (?a [t_5]) and (Close [t_6]) because it is possible for the train to enter the intersection before the gate has properly (or completely) closed. Similarly markings M_{11} and M_{12} occur due to a *slow* firing of transitions (?d [t_7]) and (Open [t_8]) because it is possible for the train to depart from the intersection and not have the gate properly (or completely) open, although these markings do not lead to a safety critical condition. Figure 4 shows the Markov process based on the stochastic Petri net. For analysis purposes, we will generally group failures into safety-critical (CF) and non-(safety-)critical (NF). A failure in sending or receiving the approaching ("a") message and the closing of the gate are safety-critical. The CSP specification (and the corresponding Petri net) can be augmented to show how such failures should be handled. For example, the communication failures can be handled using time-out and re-transmit techniques. The failure of the gate closing action can be handled by sounding a loud alarm to alert pedestrians and traffic.

3.2 Stochastic Analysis. Using conventional techniques or stochastic Petri net tools (e.g., SPNP), discrete and continuous analyses can be performed. For the purpose of this presentation, we have used Mathematica® to compute reliability of the rail-road crossing example with different failure rates (or probabilities) and service rates (e.g., speed of the train, rate at which the gate mechanism operates). The values used in this paper (and hence the results of the analysis) are only for illustrating the approach. It is not our intention to attach significance to the failure rates, MTTFs obtained, or the probability of detected and undetected failures. These analyses are useful in exploring different fault-handling mechanisms and the cost of providing fault tolerance. For example, more elaborate fault-handling and fault-recovery mechanisms could be used to tolerate or prevent safety critical failures, while less attention may be paid to non-safety critical failures (see runs 2, 3, and 4 in Figures 5-6). Failure to open the gate may anger people waiting at the crossing but such failures can be handled inexpensively by providing a mechanism to manually open the gate. On the other hand, failure to close the gate is more severe, so traffic at the crossing should be alerted reliably and automatically.

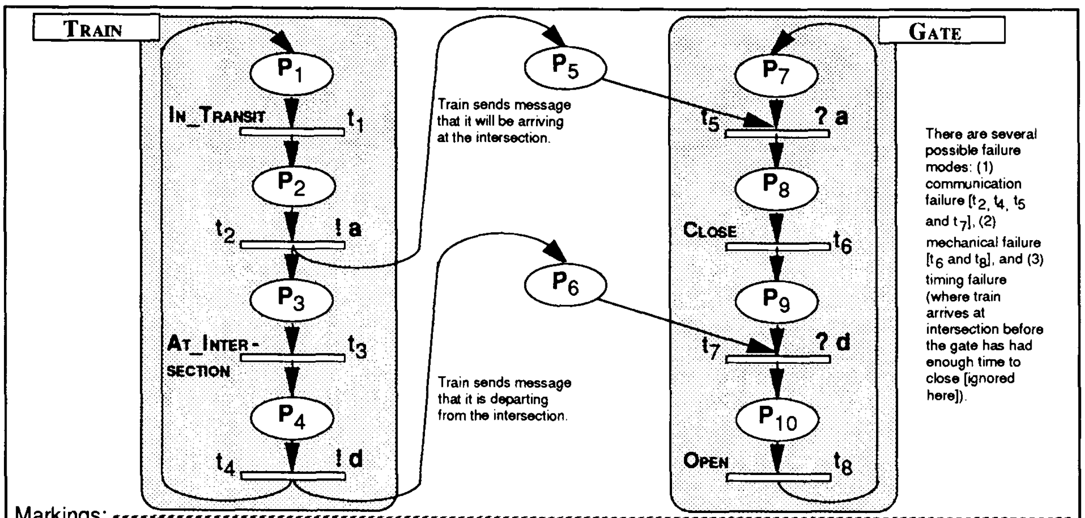
3.2.1 DISCRETE ANALYSIS. Table 1 presents the probability assignments for our test runs of the Rail-Road crossing example. Table 2 shows the results of the stochastic analysis. In all runs we assume that the mechanical failures have higher probabilities of failure than transmission failures. In order to reduce the probability of critical failures, in runs 2, 3, and 4 we assume that fault-tolerant mechanisms are utilized to improve the gate closing mechanism's reliability (as compared to the gate opening mechanism which is a non-critical failure) by the factors of 100, 3, and 5 respectively. This achieved a reduction in the probability of critical failures by the factors of 17.5, 1.24, and 1.75. Such an analysis of the improvements in the probability of critical failures can be used in deciding what level of fault tolerance is necessary.

3.2.2 CONTINUOUS ANALYSIS. The results of our continuous analysis are shown in Figure 5. We have also investigated the trade-offs between the rate of train arrivals (μ_1), the speed of the train (μ_3), the rate of the gate mechanism (μ_6, μ_8) and the failure rates associated with the signal transmission (λ_2, λ_4, λ_5, λ_7) and the mechanical failures (gate mechanism, λ_6, λ_8). These results are shown in Table 2. Since we assume that signal transmissions are more reliable than the gate's mechanical mechanism, we notice that the reliability of signal transmission does not significantly impact the MTTF. Thus, the mechanical failures of the gate and the rate of gate closing (opening) are greater contributors to the reliability (or unreliability) of system. An interesting result of this analysis is that when the train speed is such that it arrives at the intersection sooner than the gate has had time to close, then an improvement in the mechanical reliability will not improve the system's reliability. This supports our statements about the need for additional synchronization between the TRAIN and the GATE processes (Figure 2).

4. SUMMARY AND FUTURE WORK

Our objective in this paper is to show how CSP specifications can be translated into Stochastic Petri nets for the purpose of reliability and performance analyses. Such translations will give insight into the failure modes, and how fault handling mechanisms can be described as a part of the CSP specifications. We believe that our approach will provide the needed feedback to a designer so that judicious cost-benefit analysis in providing fault-tolerance can be made. In this paper we have illustrated this approach by using a simple example. The failure probabilities used in the examples (hence the results of the analysis) are for illustrating our approach, no other significance should be attached. Our only intention is to show the complete process of specification and analysis. We hope to develop a tool for automatically translating CSP specifications into Petri nets in order to use stochastic Petri net tools for the purpose of analysis.

Acknowledgments: Special thanks are extended to Sherman Reed of the Electrical Engineering Department at UTA and Patrica Howell at NASA Langley Research Center ofr their help with Mathematica®.

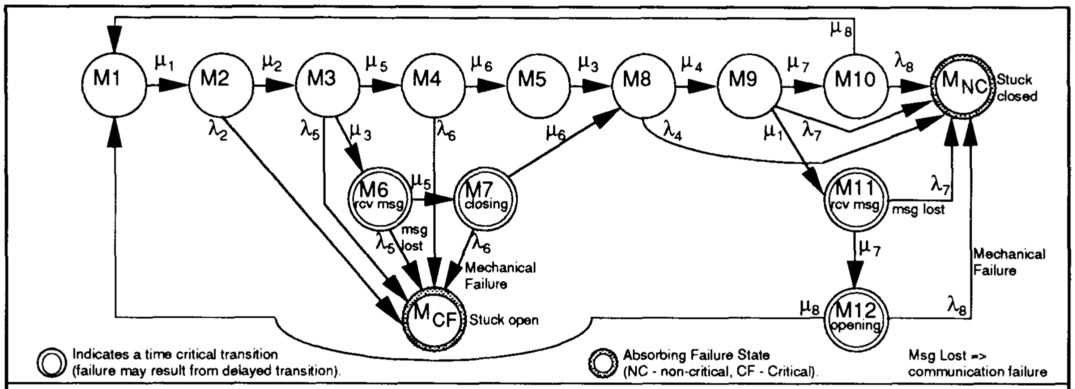

TRAIN
P1
IN_TRANSIT
t1
P2
t2
! a
P3
AT_INTER-SECTION
t3
P4
t4
! d
P5
Train sends message that it will be arriving at the intersection.
P6
Train sends message that it is departing from the intersection.
GATE
P7
t5
? a
P8
CLOSE
t6
P9
t7
? d
P10
OPEN
t8
There are several possible failure modes: (1) communication failure [t2, t4, t5 and t7], (2) mechanical failure [t6 and t8], and (3) timing failure (where train arrives at intersection before the gate has had enough time to close [ignored here]).

Markings:

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{CF}	P_{NF}	Description of marking (i.e., state of the system)
M1:	(1	0	0	0	0	0	1	0	0	0	0	0)	Train gone, gate open
M2:	(0	1	0	0	0	0	1	0	0	0	0	0)	Train in transit, gate open
M3:	(0	0	1	0	1	0	1	0	0	0	0	0)	Train sends *approaching* msg, gate open
M4:	(0	0	1	0	0	0	0	1	0	0	0	0)	Train approaching, msg rcv'd and gate closing
M5:	(0	0	1	0	0	0	0	0	1	0	0	0)	Train approaching/waiting, gate closed
M6*:	(0	0	0	1	1	0	1	0	0	0	0	0)	Train at intersection, approaching msg not rcv'd
M7*:	(0	0	0	1	0	0	0	1	0	0	0	0)	Train at intersection, msg rcv'd and gate closing
M8:	(0	0	0	1	0	0	0	0	1	0	0	0)	Train at intersection, gate closed
M9:	(1	0	0	0	0	1	0	0	1	0	0	0)	Train sends *departing* msg, gate closed
M10:	(1	0	0	0	0	0	0	0	0	1	0	0)	Train gone, gate received msg and is opening
M11**:	(0	1	0	0	0	1	0	0	1	0	0	0)	Train in transit, but gate hasn't received msg
M12**:	(0	1	0	0	0	0	0	0	0	1	0	0)	Train in transit, gate received msg and is opening

*Critical condition: Train at intersection but gate is still open **Non-critical condition: Train departed intersection but the gate is still closed

Figure 3 Markings (M1 - 12) Based on the Hazardous Rail-Road Crossing PN Specification.



{M2(λ_2), M3(λ_5), M4(λ_6), M6(λ_5), M7(λ_6)} → M_{CF}

{M8(λ_4), M9(λ_7), M10(λ_8), M11(λ_7), M12(λ_8)} → M_{NC}

Communication failure transitions: {t_5, t_7, t_4, t_2}
Mechanical failure in transitions: {t_6, t_8}
Absorb transitions: {t_1, t_3}

Figure 4 Markov Chain for Rail-Road crossing based on unsafe PN Specification.

Table 1 Discrete Analysis: Probability Assignments

State Transition Probabilities:	Run 1	Run 2	Run 3	Run 4	Transition Description:
	M_{CF}	M_{NC}	M_{CF}	M_{NC}	
P_1	1.0	1.0	1.0	1.0	
P_2	1.0 e -4	1.0 e -5	1.0 e -4	1.0 e -5	Probability of transmission failure
P_3	1.0	1.0	1.0	1.0	
P_4	1.0 e -4	1.0 e -5	1.0 e -4	1.0 e -5	Probability of transmission failure
P_5	1.0 e -2	1.0 e -5	1.0 e -2	1.0 e -3	Probability of gate closure failure
P_6	1.0 e -4	1.0 e -5	1.0 e -4	1.0 e -5	
P_7	1.0 e -4	1.0 e -5	1.0 e -4	1.0 e -5	
P_8	1.0 e -2	1.0 e -3	3.0 e -2	5.0 e -3	Probability of gate open failure

Table 2 Discrete Analysis: MTTF, Critical and Non-critical Failure Probabilities.

Run Number	MTTF	M_{CF}	M_{NC}
Run 1	392.20474	0.5026	0.4974
Run 2	7619.24626	0.0286	0.9714
Run 3	159.50404	0.4033	0.5967
Run 4	1138.50228	0.2862	0.7138

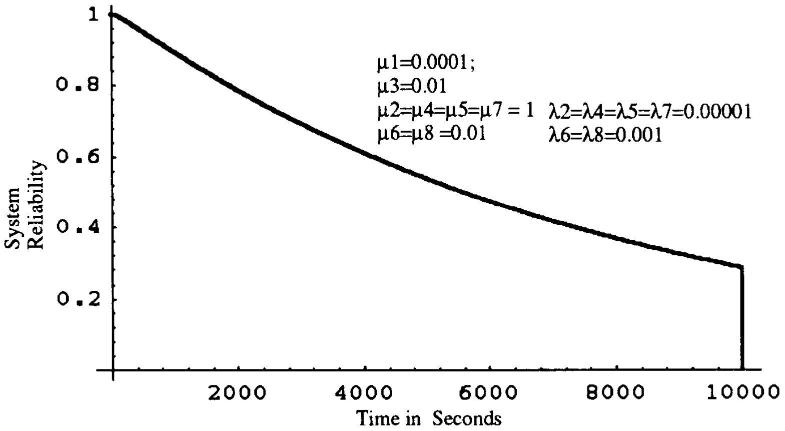


Figure 5 Continuous Analysis: System Reliability as a Function of Operational Time

5. REFERENCES

[Chiola 89] G. Chiola. "A software package for the analysis of generalized stochastic Petri net models," *IEEE Proceedings: Third Int'l Wkshp on Petri Nets and Performance Models*, Kyoto Japan, pp. 136-143, Dec. 1989.

[Ciardo 89] G. Ciardo, J. Muppala and K.S. Trivedi. "SPNP: Stochastic Petri Net Package," *IEEE Proceedings: third Int'l Wkshp on Petri Nets and Performance Models*, Kyoto Japan, pp. 142-151, Dec. 1989.

[Hall 90] A. Hall. "Seven myths of formal methods," *IEEE Software*, Sept. 1990, pp. 11-20.

[Hoare 85] C.A.R. Hoare. *Communicating sequential processes*, Prentice-Hall Int'l, Englewood Cliffs, NJ, 1985.

[Kavi 93] K.M. Kavi, and B.P. Buckles. "Formal methods for the specification and analysis of concurrent systems" Tutorial Notes, *1993 International Conference on Parallel Processing*, Lake Charles, IL., Aug. 20, 1993.

[Marsan 89] M. A. Marsan, S. Donatelli and F. Neri. "GSPN models of multiserver multiqueue systems," *IEEE Proc.: Third Int'l Wkshp on Petri Nets and Performance Models*, Kyoto Japan, pp. 19-28, Dec. 1989.

[Murata 89] T. Murata. "Petri nets: properties, analysis and applications," *Proceedings of the IEEE*, pp. 541-580, April 1989.

[Olderog 86] Olderog, E.R. *TCSP - Theory of communicating sequential processes*, LNCS-255, Springer Verlag, 1986.

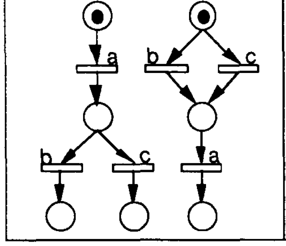
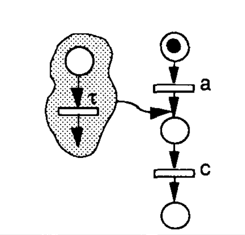
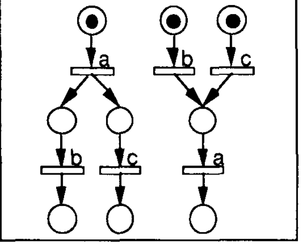
[Ostroff 92] Ostroff, J.S. "Formal methods for the specification and design of real-time safety critical systems," *The Journal of Systems and Software*, pp. 33-60, April 1992.

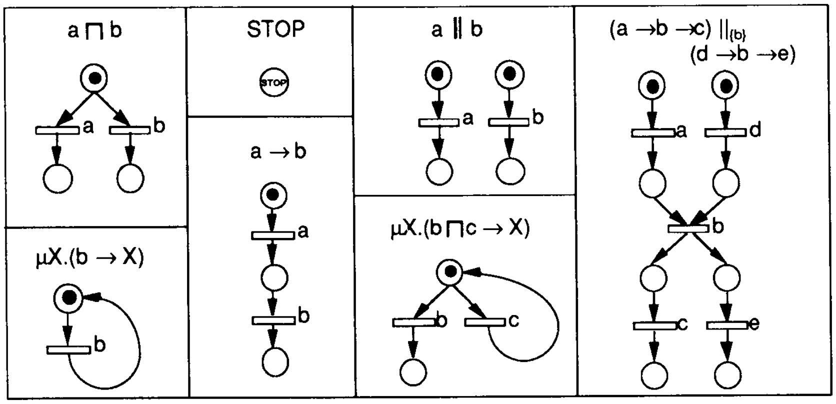
[Trivedi 82] Trivedi, K.S., *Probability and Statistics with reliability, queuing and computer science applications*, Prentice-Hall, Englewood, NJ, 1982.

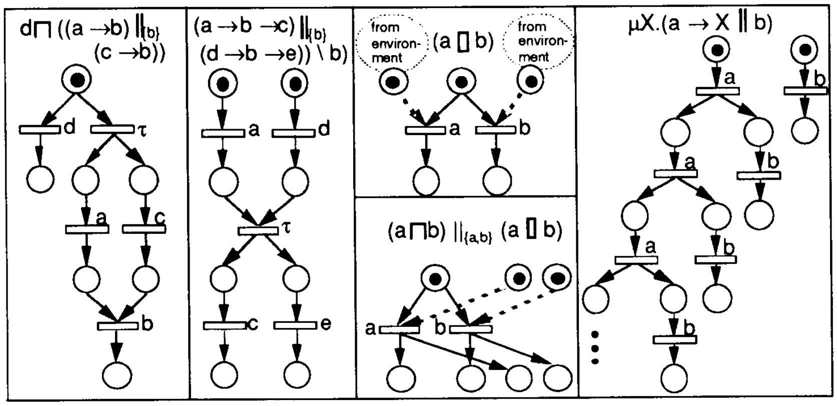
TABLE 3 Continuous Analysis: Sensitivity Analysis

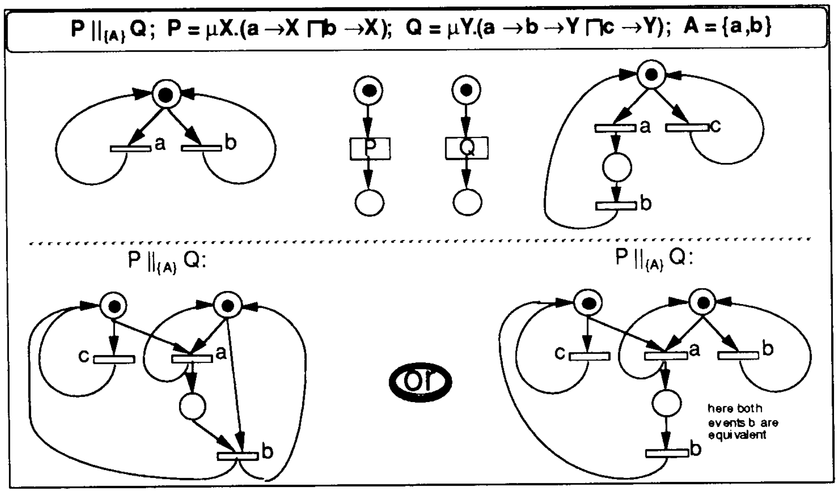
$\mu_1 = 0.0001$	$\lambda_2 = \lambda_4 = \lambda_5 = \lambda_7 = 0.00001$	MTTF	$\mu_1 = 0.0001$	$\lambda_2 = \lambda_4 = \lambda_5 = \lambda_7 = 0.00001$	MTTF
$\mu_2 = 0.01$; $\mu_6 = 0.01$	$\lambda_6 = \lambda_8 = 0.001$	5622	$\mu_2 = 0.0001$; $\mu_6 = 0.01$	$\lambda_6 = \lambda_8 = 0.001$	10107
	$\lambda_6 = \lambda_8 = 0.0001$	5201		$\lambda_6 = \lambda_8 = 0.0001$	10151
	$\lambda_6 = \lambda_8 = 0.0001$	5156		$\lambda_6 = \lambda_8 = 0.0001$	10549
$\mu_2 = 0.01$; $\mu_6 = 0.001$	$\lambda_6 = \lambda_8 = 0.001$	6096	$\mu_2 = \mu_6 = 0.001$	$\lambda_6 = \lambda_8 = 0.001$	9002
	$\lambda_6 = \lambda_8 = 0.0001$	6477		$\lambda_6 = \lambda_8 = 0.0001$	6925
	$\lambda_6 = \lambda_8 = 0.0001$	8642		$\lambda_6 = \lambda_8 = 0.0001$	6546
$\mu_2 = 0.01$; $\mu_6 = 0.0001$	$\lambda_6 = \lambda_8 = 0.001$	10912	$\mu_2 = 0.0001$; $\mu_6 = 0.001$	$\lambda_6 = \lambda_8 = 0.001$	10993
	$\lambda_6 = \lambda_8 = 0.0001$	14042		$\lambda_6 = \lambda_8 = 0.0001$	14402
	$\lambda_6 = \lambda_8 = 0.0001$	15029		$\lambda_6 = \lambda_8 = 0.0001$	15477
$\mu_2 = 0.001$; $\mu_6 = 0.01$	$\lambda_6 = \lambda_8 = 0.001$	6069	$\mu_2 = 0.001$; $\mu_6 = 0.0001$	$\lambda_6 = \lambda_8 = 0.001$	12602
	$\lambda_6 = \lambda_8 = 0.0001$	5651		$\lambda_6 = \lambda_8 = 0.0001$	11404
	$\lambda_6 = \lambda_8 = 0.0001$	5606.		$\lambda_6 = \lambda_8 = 0.0001$	11046

APPENDIX CSP TO PETRI NET TRANSLATIONS

$a \rightarrow (b \sqcap c)$ and $(b \sqcap c) \rightarrow a$	$(a \rightarrow b \rightarrow c) \setminus b)$	$a \rightarrow (b \parallel c)$ and $(b \parallel c) \rightarrow a$
		


a ⊓ b
a
b
STOP
STOP
a ∥ b
a
b
(a →b →c) ∥{b} (d →b →e)
a
d
b
c
e
μX.(b → X)
b
a → b
a
b
μX.(b ⊓ c → X)
b
c


d⊓ ((a →b) ∥{b} (c →b))
d
τ
a
c
b
(a →b →c) ∥{b} (d →b →e)) \ b)
a
d
τ
c
e
from environment
(a □ b)
from environment
a
b
(a⊓b) ∥{a,b} (a □ b)
a
b
μX.(a → X ∥ b)
a
b
a
b
a
b
b


P ∥{A} Q; P = μX.(a →X ⊓b →X); Q = μY.(a →b →Y ⊓c →Y); A = {a,b}
a
b
P
Q
a
c
b
P ∥{A} Q:
c
a
b
or
P ∥{A} Q:
c
a
b
b
here both events b are equivalent