To appear in the Int'l J. of Software Engineering and Knowledge Engineering, June 1996.

SPECIFICATION AND ANALYSIS OF REAL-TIME SYSTEMS USING CSP AND PETRI NETS

Krishna M. Kavi[1], Frederick T. Sheldon[2] and Sherman Reed[3]
The University of Texas at Arlington
916 Yates Ave., PO Box 19015
Arlington, Texas 76019

Abstract

Formal methods such as CSP (Communicating Sequential Processes) are widely used for reasoning about concurrency, communication, safety and liveness issues. Some of these models have been extended to permit reasoning about real-time constraints. Yet, the research in formal specification and verification of complex systems has often ignored the specification of stochastic properties of the system under study. We are developing methods and tools to permit stochastic analyses of CSP-based specifications. Our basic objective is to evaluate candidate design specifications by converting formal systems descriptions into the information needed for analysis. In doing so, we translate a CSP-based specification into a Petri net which is analyzed to predict system behavior in terms of reliability and performability as a function of observable parameters (e.g., topology, fault-tolerance, deadlines, communications and failure categories). This process can give insight into further refinements of the original specification (i.e., identify potential failure processes and recovery actions). Relating the parameters needed for performability analysis to user level specifications is essential for realizing systems that meet user needs in terms of cost, functionality, and other non-functional requirements.

An example translation is given (in addition, some general examples of CSP -> Petri net translations can be viewed in Appendix A). Based on this translation, we report both the discrete and continuous time Markovian analysis which provides reliability predictions for the candidate specification. The term "CSP-based" is used here to distinguish between the notation of Hoare's original CSP and our textual representations which are similar to occum. Our CSP-based grammar does not restrict consideration of the properties of CSP (traces, refusal sets, livelock, etc.), but we are not considering those properties. We are only interested that the structural properties are preserved. We define performability as a measure of the system's ability in meeting deadlines, in the presence of failures and variance in task execution times.

Keywords: Real-Time Systems, CSP, Stochastic Petri Nets, Performability and Reliability

---

[1] All correspondence should be addressed to Krishna M. Kavi, Dept. of Computer Science & Engineering, The University of Texas at Arlington, Arlington, TX 76019-0015, email: kavi@cse.uta.edu.
[2] Frederick Sheldon is supported by a NASA Graduate Fellowship from Langley Research Center (#NGT-50896).
[3] Sherman Reed is with the Electrical Engineering Department.

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 supposed that formal, mathematically precise methods should be used to design such systems. Indeed, formal specification of real-time systems has been the subject of intensive investigation over the past several years. See [Ostroff 92] for a survey of some of the formal approaches to the specification and reasoning about time. The research in formal methods, however, has often ignored the specification of stochastic properties of the system. While detailed analyses require a clear understanding of the implementation (hardware/software failure modes, failure distributions, service distributions, workload, etc.), the cost of providing real-time guarantees, and the desired level of reliability or performance should be related to user level specifications, even if only in terms of upper and lower bounds.

Real-time systems are characterized as those where correctness depends not only on the logical computation being performed, but also on the time at which the results are produced. Thus, for real-time systems, failures may be hardware related, software related, communications related, or due to a missed deadline (i.e., timing failure). Hard real-time systems can not tolerate missed deadlines. Therefore, tasks must be scheduled based on accurate worst-case execution time estimates which effectively leads to inefficient use of resources. This is problematic when the worst case behavior is rare. In soft real-time systems missed deadlines can be tolerated and worst case execution time bounds are not necessary (in most cases). Predicting how a system tolerates missed deadlines is a matter of *performability*. Performability is the probability that all tasks in the system will complete by a specified deadline and it can be improved by adding in slack time. Slack time is usually based on average case execution time (or some estimate to that effect). Slack is the extra time added to the average case time so that the system may tolerate the failure of *some* tasks to complete within their assigned execution times. Adding more slack time improves the chances that the system will not miss any deadlines. Thus, we can trade-off the guarantee of meeting all deadlines with the cost of such guarantees (in terms of expending

excessive amounts of processing resources). Moreover, for the type of real-time considered here, the system must also tolerate hardware, software and communication failures. Therefore, task deadlines must be planned to account for the possibility of re-executing tasks and/or migration of tasks from failed units, in addition to other possible delays. We use a simple model of performability to determine the probability that the system will meet its deadlines in the presence of hardware and software failures.[4]

Stochastic Petri-nets and discrete event simulations are typically 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 85], SPNP [Ciardo 89], [Geist 90]). Petri nets however, are not very suitable for reasoning about the functional correctness of a system. Although there have been extensions to Petri nets to permit such specification and verification, they have not gained popularity (e.g., predicate transition nets [Genrich 86, Vautherin 87]).

We have developed an initial set of rules for translating CSP (Communicating Sequential Processes) specifications into Petri nets [Kavi 93]. A translation is demonstrated here by a simple example that includes: (1) identifying system failure modes by inspecting the Petri nets (including *failure to meet task deadlines*), (2) identifying how failures can be handled, (3) specifying the appropriate fault handling mechanism (e.g., additional synchronization, time-out-retransmit) back at the CSP level (to be examined by the user/specifier), and (4) analysis of the Petri nets for reliability/performability based on failure probability estimates.

Section 2 introduces our performability model, and Section 3 describes the significant aspects of CSP and Stochastic Petri nets with respect to specification analysis. Finally, in Section 4 we illustrate how the CSP specifications are analyzed in terms of performability.

2. PERFORMABILITY OF REAL-TIME SYSTEMS

In addition to the traditional reliability and performance measures (e.g., MTBF, reliability, availability, throughput), we introduce a performability measure in our research. The term

[4] Thus, performability in our work involves more than just failure to meet deadlines.

performability was first introduced by Meyer to combine performance and reliability analyses of fault-tolerant systems [Meyer 79]. Informally, performability can be defined as the probability that a system performs at different levels of "accomplishment." Markov processes often are used to estimate the probability that a system is in one of several "capacity" states [Gay 79]. Chou and Abraham [Chou 80] provided an availability model for gracefully degraded systems with critically shared resources. However, these models did not include failure to meet deadlines.

In this paper we define performability of real-time systems as the probability that the set of tasks comprising the real-time system will complete their execution successfully by the deadline defined for the system [Kavi 94]. This definition permits the inclusion of accomplishment levels and graceful degradation. The failure of a task to successfully complete within its allocated time may be due to the following reasons.

- For a particular instance the input data required longer execution time.
- Due to failures, the task had to be re-executed, migrated and restarted.
- The task was delayed (hence missed the deadline) due to the failure of a preceding task to successfully complete.

We assume that tasks take different amounts of time to complete their execution for different input data, and the actual execution time for a specific run is described by a probability distribution E_i (Figure 1).[5] Such a task profile can be used to estimate execution time. In hard

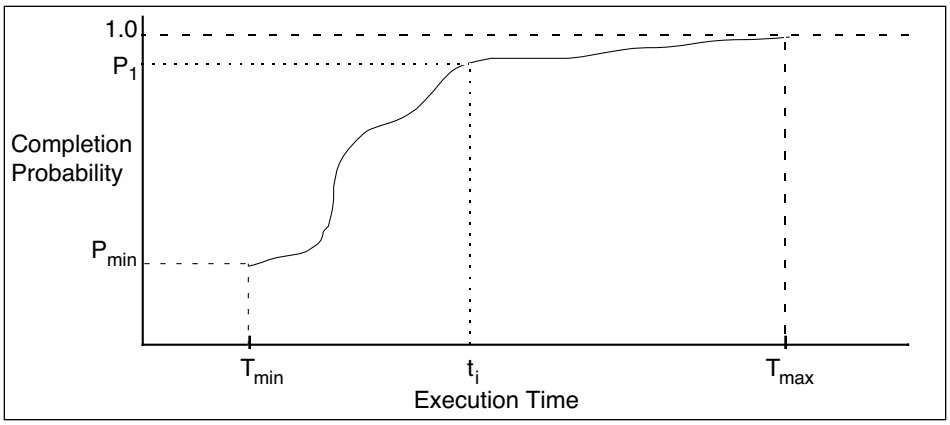


Figure 1 Execution Profile of a Task J_i.

[5] Researchers are actively developing methods to estimate task execution times and designing languages that facilitate execution time prediction (e.g., based on instruction counts and maximum number of loop iterations). These approaches are within the scope of our model. Figure 1 can be viewed as an empirical derivation of task execution times using actual execution time measurements (or based on instruction counts and loop iteration counts).

real-time systems, one would use T_{max} as the execution time estimate. In less stringent cases, one would use the p percentile execution time (i.e., the task will complete execution with probability p). Let us denote this estimated execution time for task J_i as t_i time units. Typically, deadlines D_i along with task execution time estimates t_i are used by scheduling algorithms (e.g., least-laxity-first, earliest deadline first) [Stankovic 88]. The performability of a real-time system given a set of tasks can be computed based on the *execution profile* of each individual task (see Figure 1), and the scheduling interdependencies among them.

It is possible to apply traditional stochastic processes to obtain an analytical solution to the performability of a soft real-time system that incorporates hardware/software failures and timing failures. However, it would be more useful to a real-time system designer if a tool were provided that can incorporate empirical data or that can simulate the various task execution times in the presence of injected hardware or software failures. Such a tool can be used to determine the temporal redundancy needed to achieve a desired level of performability.[6] In section 4, this approach is shown by varying task execution times to see the effect on system reliability.

3. FORMALISMS FOR SPECIFICATION AND ANALYSIS: CSP AND PETRI NETS

This section briefly introduces CSP and Petri nets and shows how CSP specifications can be translated into Petri nets for the purpose of stochastic (i.e., performability) analysis.

3.1. 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: $P = \{P_1 \,||\, P_2 \,||\, ...|| \, P_n\}$. Processes are assumed to have a disjoint set of variables (or local symbols). Processes communicate synchronously by sending and receiving messages: the sending and receiving actions (or events) are indicated using the input (?) and output (!) actions. Pi?x is the action of receiving a value sent by process P_i (or received on a channel P_i) into variable x. P_j!

[6] The design and implementation of the essential parts of our performability evaluation tool is complete. This implementation is being extended to consider different fault-tolerance strategies and evaluate their significance in achieving performability objectives.

<expression> describes the action of sending the value of the expression to P_j (or sending on a channel P_i). Synchronization is accomplished by using complementary input and output commands in the two communicating processes (or using the same channel for input and output actions). 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:

$$P ::= STOP \mid (a \rightarrow P) \cdot (P \backslash b) \cdot (P \sqcap Q) \cdot (P \,\Box\, Q) \cdot (P \,||_b\, Q) \cdot (P; Q) \cdot (\mu x \bullet P)$$

In CSP, capitalized names are used for process names, and lower case characters are used to denote visible actions. Here, (a → P) means, action 'a' followed by P, (P\b) is the same as P except action b is hidden, (P⊓Q) represents a non-deterministic choice between P and Q, (P□Q) represents a deterministic choice between P and Q, ($P||_bQ$) shows concurrent processes P and Q that synchronize on action b, (P; Q) a sequence between P and Q, (μx • P) is used for recursion. The quality of the reliability/performance analysis relies on the fidelity of the translation from CSP to Petri nets which is a semantic question. However, a detailed definition of CSP semantics is beyond the scope of the present paper and is unnecessary. Thus, we focus only on the structural aspects of CSP and give a simple example to illustrate such.

3.1.1. The CSP for a Railroad Crossing.

In this example, a railroad crossing intersection is specified.[7] The system to be developed operates a gate at a railroad crossing and is presented in the form of a Petri net in Figure 2. This problem presents two basic properties that the system must satisfy: (1) safety property – the gate is down during all occupancy intervals, (2) utility property – the gate is open when no train is in the crossing. This model encompasses the environment which includes the train(s) and the gate,

[7] A commonly used specification of the Railroad crossing uses three tasks, the Train, the Gate and the Controller [Heitmeyer 93, 94]. We are using a simplified version of this to illustrate our method. We assume only one train enters the intersection repeatedly.

as well as the interface between them. Thus, the gate closes when a train arrives at the intersection and remains closed until the train passes the intersection. Although the problem statement can be extended to handle multiple trains, only one train is specified here.

$$
\begin{aligned}
\text{TRAIN} &= (\text{IN\_TRANSIT}); \\
&\quad (\text{GATE} \,!\, a \rightarrow \text{AT\_INTERSECTION}); \\
&\quad (\text{GATE} \,!\, d \rightarrow \text{TRAIN}) \\
\text{GATE} &= (\text{TRAIN} \,?\, a \rightarrow \text{CLOSE}); \\
&\quad (\text{TRAIN} \,?\, d \rightarrow \text{OPEN} \rightarrow \text{GATE}) \\
\text{RAIL\_ROAD\_CROSSING} &= \text{TRAIN} \;||_{\{a,d\}}\; \text{GATE}
\end{aligned}
$$

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 then 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 did not permit specification of time with actions, although some recent extensions to CSP permit the association of time with actions [Ostroff 92]. A careful examination reveals that the TRAIN process could enter the intersection (AT_INTERSECTION) before the gate closes which leads to unsafe behavior. This can be eliminated by requiring the train to wait until the gate sends a message to the train as shown in ¶3.3.1, Figure 3. For the purpose of illustrating our performability model, we will not require such a synchronization. Likewise, the train may depart while the gate is still closed which can be viewed as non-critical behavior. If however, a new train enters the state of being IN_TRANSIT before the GATE has completely opened such behavior is deemed as non-critical *failure* behavior. Such erroneous behavior is characterized in Figure 4 (see the M_{nctf} markings).

For the purpose of performability analyses, we associate execution profiles with CSP processes. For example, we associate an execution profile with the gate CLOSing process and

with the train's IN_TRANSIT process.[8] We also associate execution times with the sending and receiving actions (i.e., it is possible to permit time-out and re-transmit actions specified at the CSP level). Likewise, other fault handling actions can be related to the CSP specification. The Petri net equivalent reveals these flaws more readily (compare Figures 2 and 3).

3.2. Stochastic Petri Nets.

The Petri net was originally due to Carl Petri. This abstract model has received considerable interest from researchers in concurrent processing, reliability, performance analyses, real-time specifications, and software specifications. There are numerous extensions to the original Petri net definition, which are embodied in tools to permit various analyses of systems using a Petri net model [Murata 89]. 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.[9] 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. We'll use the stochastic Petri nets that permit us to associate various probability distributions with transitions to model system performability and

[8] We will use these execution profiles of tasks to derive the probability that the gate's CLOSEing process meets its deadline (i.e., the gate closes before the train arrives at the intersection).
[9] For example, SPNP mentioned in ¶1, employs such algorithms.

reliability. A stochastic Petri net (SPN) is a Petri net where each transition is associated with random variables that expresses either the delay from the enabling to the firing of the transition or the failure of that transition. When multiple transitions are enabled, the transition with a minimum delay fires first. When the random variables are 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 $q_{ij} = \lambda_{i1} + \lambda_{i2} + \ldots + \lambda_{im}$ 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) or the failure of the transition t_k. The performability and reliability analyses of the system represented by the Petri net can be achieved by using the equivalent Markov process.

3.3. Mapping of CSP-Level Specifications into Petri Nets.

An initial set of rules for translating CSP specifications into Petri nets has been developed (see Appendix A) [Kavi 93]. The translation relies on the fact that CSP specifications are based on processes moving from (event) action to (event) action. The activities enabling actions of processes can be viewed as the events represented by places in a Petri net, while the actions can be viewed as transitions in a Petri net.[10] Although we have not formally verified an isomorphism between our CSP and Petri net models, we have developed a set of rules for transforming a majority of the CSP process structures and compositions. Nevertheless, the Petri net equivalent of a CSP specification need not be unique, because of the need to introduce dummy places or transitions in Petri nets to maintain its bipartite nature.[11] The dummy places and dummy transitions are important for connecting (or composing) the resultant CSP process structures (e.g., forking and joining) and do not affect the timeliness aspects of the analysis.

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.

---

[10] In modeling (see [Murata 89], page 542), using the concept of conditions and events, places represent conditions, and transitions represent events. A transition has a certain number of input places and output places representing the preconditions and post-conditions of an event. Using the notion in CSP of event-action pairings we have assumed a slightly different abstraction where the conditions are the events that cause actions (transitions) to take place.

[11] We believe, however, that it is possible to reduce different Petri net equivalents into a canonical form. We are developing the necessary rules to produce canonical Petri net representations of CSP specifications.

Some example translations between CSP specifications and Petri nets are shown in Appendix A. Using these examples, we have converted the CSP example of ¶3.1.1 into a Petri net.

3.3.1. Petri Net for the Train Crossing Example.

The railroad crossing described in ¶3.1.1 is presented in Figure 2 in the form of a Petri net. Our model consists of two tasks which operate independently (in parallel) and must communicate to coordinate closing the gate when the train nears the crossing. The gate must remain closed at all times while the train occupies the crossing (any violation is a safety critical failure).

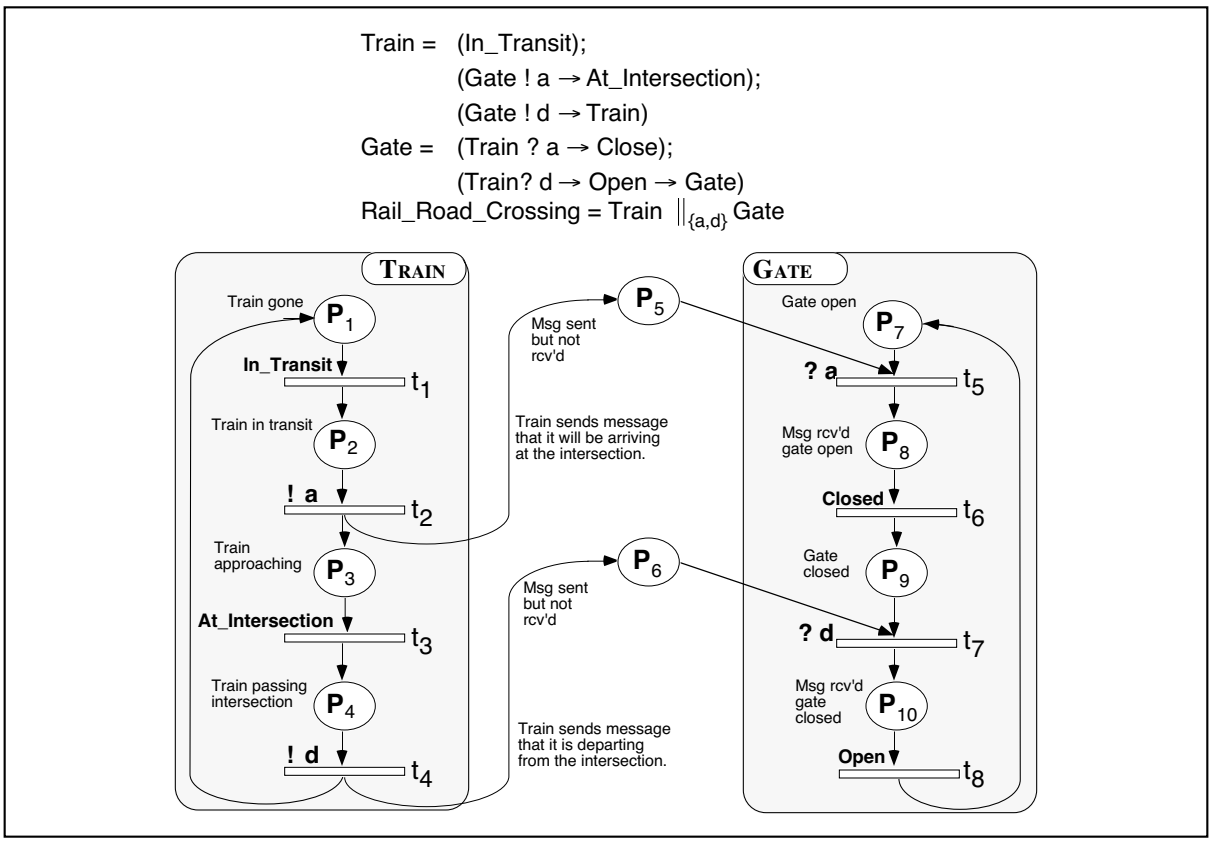


Figure 2. Train Crossing with Potential Hazard.

Though the Petri net of Figure 2 easily satisfies the CSP specification of ¶3.1.1, careful scrutiny reveals a flaw. In fact, the flaw is easily more visible (by inspection) in the Petri net than in the CSP specification. This flaw is safety critical because the TRAIN process *could* enter the intersection (AT_INTERSECTION) before the gate closes which may cause detrimental results. A timing failure such as this will occur if the gate doesn't meet its "closed" deadline. In other words, by stepping through each of the feasible markings of the Petri net we were able to

expose a weakness in the system's real-time constraints.[12] This timing hazard can be eliminated by requiring a real-time deadline guarantee at the gate mechanism level. In other words, we must require the gate be closed by the time it takes the train to arrive at the intersection after sending the "arriving" signal. Another way of avoiding this hazard is to redesign the system such that the TRAIN process will wait until the GATE process completes closing the gate. The Petri net of Figure 3 shows one possible way of implementing such a guarantee. This solution adds an additional synchronization point which, barring a communication failure, gives permission to the TRAIN to proceed by sending a message that says in effect: *the gate has completely closed and its safe to proceed* (i.e., GATE sends an "OK" signal to the TRAIN). One drawback is that a failure in any of the communication related actions may lead to a deadlock. In addition, failure

[12] This does not say that such constraints could not be detected at the CSP level, just that in general such constraints are more easily validated at the Petri net level.

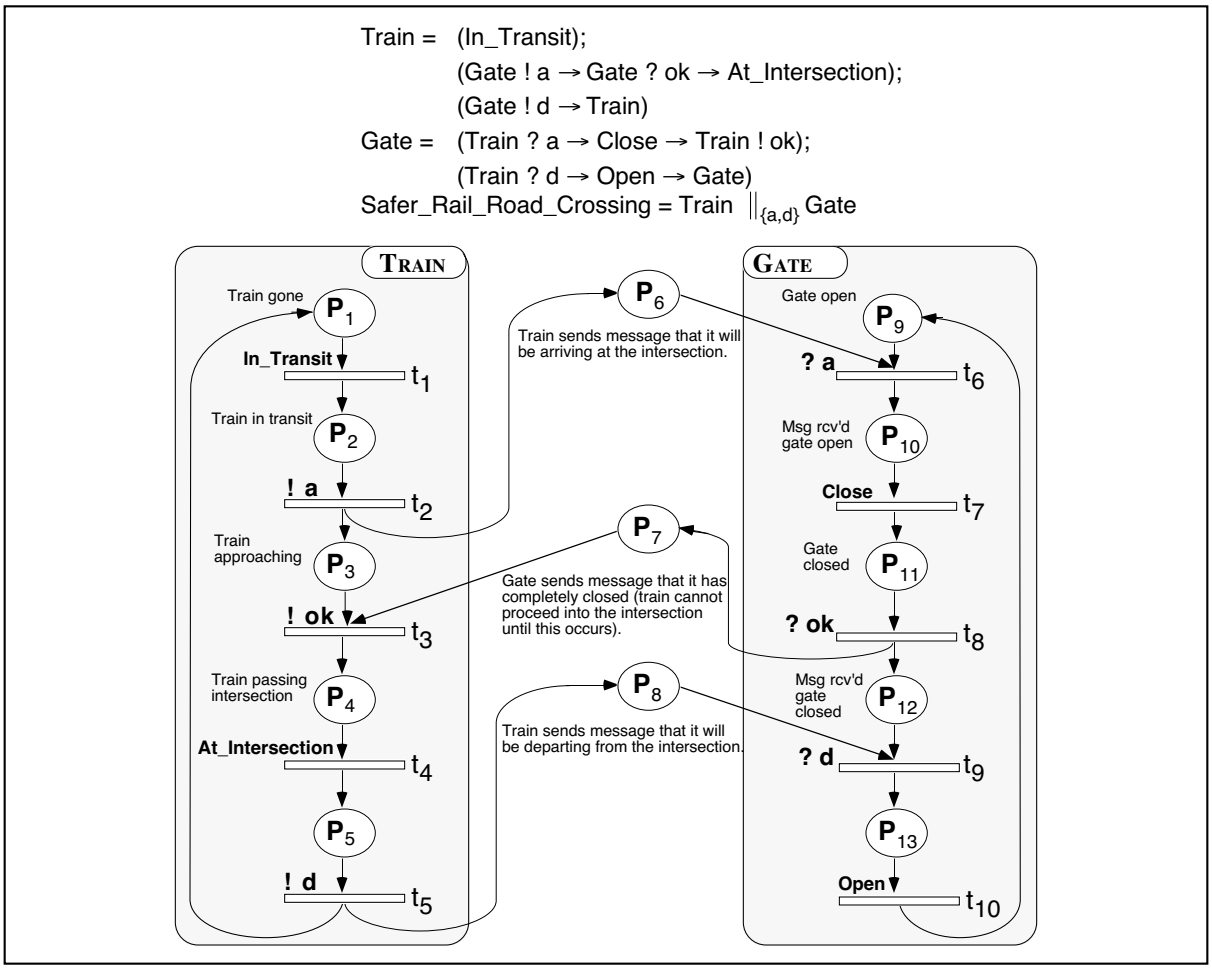


Figure 3. Train Crossing with Hazard Eliminated.

to OPEN the gate is not safety critical, yet should be avoided to maintain the system's utility property (i.e., prevent congestion of the associated infrastructure, traffic, etc.). Also notice that Figure 3 contains the CSP that corresponds to the "safer railroad crossing," which is an important improvement to the original specification.

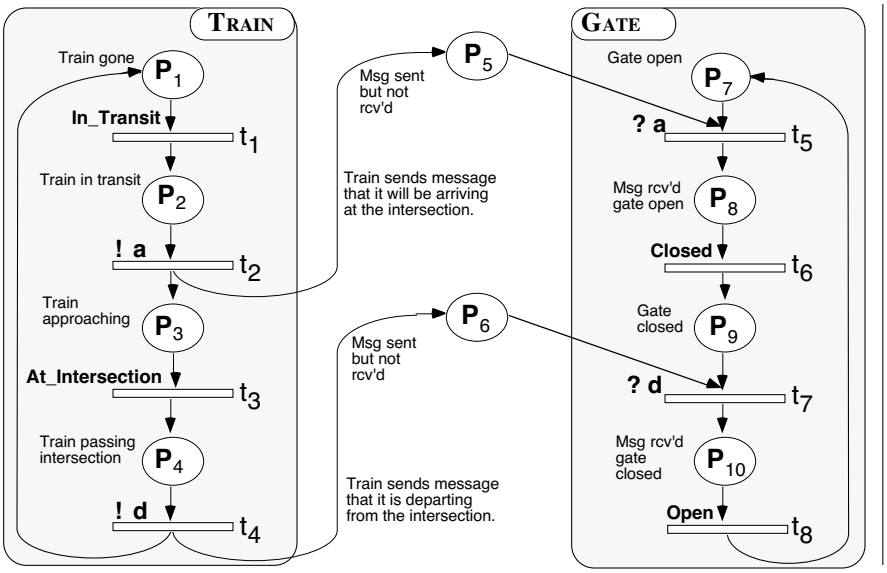
The performability of the original system can be improved by giving more "slack" time to the GATE process, and is achieved by having the TRAIN process send the "arriving" signal sooner. In ¶4.2.1 we analyze the reliability of the original CSP specification (from ¶3.1.1 based on the *hazardous* Petri net of Figure 2) under various failure modes. And, in ¶4.2.2 we analyze the performability of the same by changing the slack time.

4. SPECIFICATION OF STOCHASTIC PROPERTIES

One of the major objectives of this research is to provide assistance to the user in specifying not only functionality but also reliability, performance and execution deadlines. In doing so, we demonstrate the idea of modeling stochastic aspects and relating them back to the original CSP specifications. This approach is concerned only with translating the specification's structural properties as needed, and not all aspects of a CSP specification. The benefit, as we have presented in Section 3 (which is perhaps an oversimplified case), is in determining how the equivalent Petri net (PN) can elucidate the need to strengthen structural characteristics by adding (or possibly deleting) control structure(s) (e.g., synchronization points, redundancy, etc.). Figures 2 and 3 illustrate the case where, in order to avoid a safety-critical failure, an additional transmission is added (a message that the gate has closed and it is safe for the train to proceed).

4.1. Failure Modes for the Train Crossing

Using the Petri net of Figure 2, we will assume that all transitions can fail (except where noted). Furthermore, because transitions represent tasks of the system, task execution profiles (defined in Section 2, and Figure 1) can be associated with these transitions. It is then possible to determine the performability of the system using failure rates and execution profiles. The failure modes associated with transitions can be translated into failure modes of the corresponding CSP actions. When interpreting the failures of those actions, the user identifies failure modes that are critical. Users can easily eliminate improbable failures identified in the Petri net (i.e., some transitions may not realistically fail or can be reasonably tolerated [e.g., t_1 and t_3]). Such evaluations of the Railroad Crossing example could lead to the augmented stochastic Petri net shown in Figure 3. The Markings are also shown in the figure to provide for easy correlation between the markings and the different states of Petri nets. Note that P_{cf} and P_{ncf} are places not identified in the Petri



There are several possible failure modes: (1) communication failure [t_2, t_4, t_5 and t_7], (2) gate mechanical failure [t_6 and t_8], and (3) timing failure (train arrives at intersection before the gate has closed or the gate opens too slowly).

Markings (operational and failure states):

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{cf}	P_{ncf}		Description of marking (i.e., state of the system)
M1:	(1	0	0	0	0	0	1	0	0	0	0	0)	.....	Train gone (i.e., idle state), gate open.
M2:	(0	1	0	0	0	0	1	0	0	0	0	0)	.....	Train in transit (i.e., 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 open.
M5:	(0	0	1	0	0	0	0	0	1	0	0	0)	.....	Train approaching, msg rcv'd and gate closed.
M6:	(0	0	0	1	0	0	0	0	1	0	0	0)	.....	Train at intersection, gate closed.
Mcf:	(0	0	0	0	0	0	0	0	0	0	1	0)	.....	Critical (communication / mechanical) failure.
Mctf:*	(0	0	0	1	0	0	0	1	0	0	0	0)	.....	**Train at intersection, gate still open!**
M7:	(1	0	0	0	0	1	0	0	1	0	0	0)	.....	Train gone, *departing* msg sent and gate closed.
M8:	(1	0	0	0	0	0	0	0	0	1	0	0)	.....	Train gone, msg rcv'd and gate closed.
M9:	(1	0	0	0	0	0	1	0	0	0	0	0)	.....	Train gone, gate *opening* (dummy state = M1).
Mncf:	(1	0	0	0	0	0	1	0	0	0	0	1)	.....	Non-critical (communication / mechanical) failure.
Mnctf:**	(0	–	–	–	0	–	–	0	0	–	0	0)	.....	Six markings combined (non-critical timing failure).

*Critical condition: Train at intersection but gate is still open. **Non-critical conditions (see above): Train departs the intersection but the gate stays closed (– is a 0 or 1).

Figure 4. Failure Modes and Markings for the Railroad Crossing example.[13]

net but are used to designate critical and non-critical failure events respectively. These places represent failure states of the corresponding Markov process.

Markings M_3, to M_4 to M_5 are critical markings because the *slow* firing of transitions (?a [t_5]) and (Close [t_6]) will result in M_{tf} because, again, it is possible for the train to enter the intersection before the gate has properly (or completely) closed (violation of the safety property). Similarly marking M_{nctf} occurs due to the *slow* firing of transitions (?d [t_7]) and (Open [t_8]). In

[13] Marking Mnctf is a non-critical timing failure and can assume any of the following markings: a) 0100010010, b) 0010010010, c) 0001010010, d) 0100000001, e) 0010000001, and f) 0001000001

this second case, which is not safety critical, since the train may depart long before the gate has properly (or completely) closed, the utility property would be compromised. This Petri net is modeled by the Markov process of Figure 5 which comprehends failures due to hardware, communication related errors and failures due to missed deadlines.

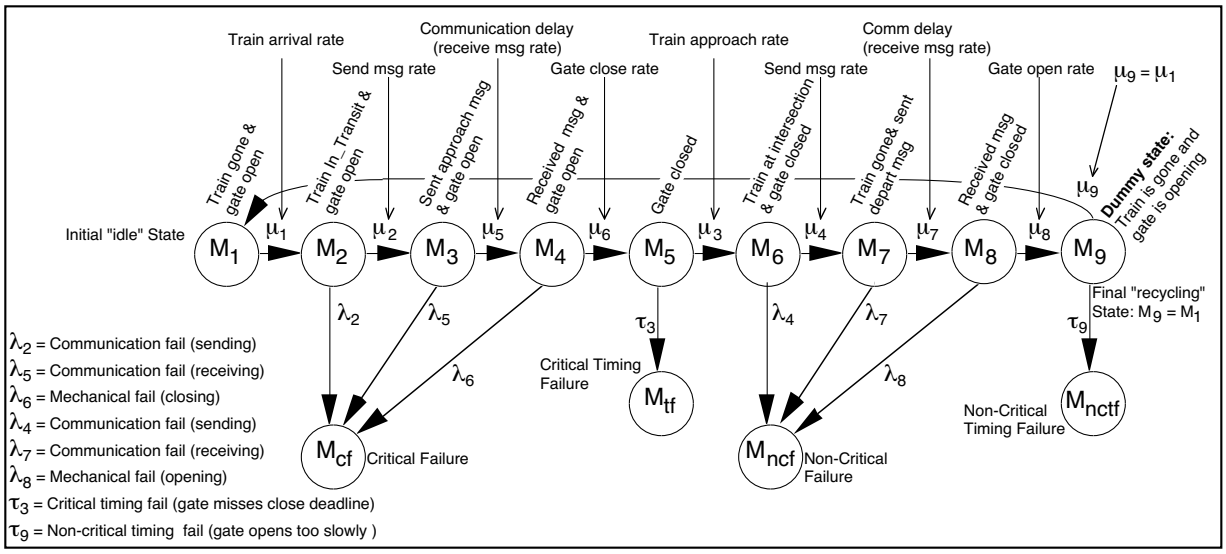


Figure 5. State diagram of the railroad (i.e., train/gate).

For this analysis we have grouped failures into deadline or time-related failures (TF), mechanical or communication, which are either safety-critical (CF) or non-(safety-)critical (NC). A failure in sending or receiving the approaching ("a") message and the closing of the gate are safety-critical whereas such failures for the departing ("d") message (including opening of the gate) are not. The CSP specification (and the corresponding Petri net) can be augmented to show how such failures should be handled. For example, communication failures can be handled using time-out and re-transmit techniques. Failure of the gate closing action can be handled by sounding a loud alarm to alert pedestrians and traffic. However, since this is a critical deadline and it is safety related, such a handling mechanism may not adequately satisfy the requirements. Tolerance to time-related failures can be maintained by permitting more (or less) slack time for the gate CLOSing process. For instance, requiring the Train to send the "arriving" message from a point further from the intersection adds more slack time to the gate's deadline. In other words, the gate closure task must complete execution in a time less than the following:

$$\frac{\textit{Distance to the gate from where the "arriving" message was sent}}{\textit{Speed of the train}}.$$

4.2. Stochastic Analysis.

Using conventional techniques or stochastic Petri net tools (e.g., SPNP), discrete and continuous analyses can be performed. For this analysis, the reliability of the Railroad Crossing example is presented based on differing the failure rates (or probabilities) and service rates (e.g., speed of the train, rate at which the gate mechanism operates). For the performability analysis we have evaluated the specification's sensitivity to the train speed (μ_3) and the speed with which the gate closes (μ_6).[14] In doing so, we have defined the following failure rates:

Mechanical failures of the gate:	λ_6 and λ_8
Communication failures (send/receive)	$\lambda_2, \lambda_5, \lambda_4$ and λ_7

This kind of analysis is useful in exploring different fault-handling mechanisms and the cost of providing fault-tolerance. For example, more elaborate fault-handling and fault-recovery mechanisms may be used to tolerate or prevent safety critical failures, while less attention may be paid to non-safety critical failures. Failure to open the gate violates the utility property for the system (not to mention all the angry people waiting at the crossing) but such failures can be handled inexpensively like providing a mode for the gate to be opened manually. On the other hand, failure to close the gate is much more detrimental and so traffic at the crossing must be alerted reliably and automatically.

In the following two sub-sections (¶4.2.1 and ¶4.2.2) the results from both the discrete and continuous analyses are presented. It is important to note that 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 any significance with respect to the failure rates or MTTFs we've obtained.

[14] The results from this analysis come from the core model used by the performability tool. This tool can be used to compute the performability of the system in assuring that the gate closes before the train arrives at the intersection with and without hardware and communication failures.

4.2.1. Discrete Analysis.

Tables 1-3 present our discrete analysis findings (i.e., top half of each table gives the input parameters and bottom half gives the results). In all three tables (all runs) the mechanical systems are assumed to have a higher failure probability than that of transmission failures (i.e., communication of synchronization messages). Table 1 shows the results ignoring deadline related failures. In the four runs shown the mechanical and communication failure probability is varied and in fact, there is a difference between gate closing and gate opening reliability. The communication failure probability is 0.0001 for runs 1 and 3 and is 0.00001 otherwise. This reduces, in effect, the probability of critical failures for runs 2, 3, and 4 (i.e., we can assume that fault-tolerant mechanisms are utilized to improve the gate closing mechanism's reliability *as compared to the gate opening mechanism*) by the factors of 100, 3, and 5 respectively.[15] This strategy achieves a reduction in the probability of critical failures by the factors: 17.59, 1.98, and 2.97 (respectively). This approach is useful because it shows the results from stepwise improvements in the probability of critical failures and can be easily correlated with the

Table 1. Discrete analysis without considering timing failures.[16]

	Run 1	**Run 2**	**Run 3**	**Run 4**	**Description**
P_{tf}	0.0	0.0	0.0	0.0	Probability of critical timing failure
P_{comf}	0.0001	0.00001	0.0001	0.00001	Probability of communication failure
P_{gcf}	0.01	0.00001	0.01	0.001	Probability of gate closure failure (mech.)
P_{gof}	0.01	0.001	0.03	0.005	Probability of gate open failure (mech.)
P_{nctf}	0.0	0.0	0.0	0.0	Probability of non-critical timing failure
MTTF	440.49	8570.67	222.50	1489.67	Mean-Time-To-Failure
P_{cf}	0.50256	0.02857	0.25442	0.16902	Critical failure probability
P_{ncf}	0.49744	0.97143	0.74558	0.83098	Non-critical failure probability

Key: P_{tf} = critical timing fail prob., P_{comf} = communication fail prob., P_{gof} = mechanical (open) fail prob., P_{gcf} = mechanical (close) fail prob., P_{nctf} = non-critical timing fail prob., P_{mechf} = mechanical fail prob.

incremental costs associated with determining the right level of fault-tolerance necessary for a given application.

---

[15] The ratio of P_{gcf}/P_{gof} is (runs 1→ 4) is 1, 100, 3 and 5.

[16] MTTF values are in number of discrete steps (or time units).

The significance of deadline-related failures (P_{tf} or P_{nctf}) can be seen in Table 2 which presents the MTTF values and overall failure probabilities associated with (1) critical, (2) non-critical and, (3) timing failures. Note, the *critical* timing failure probability (i.e., probability that [train arrival time < gate closing time]) is varied from 0.00001 up to 0.01.

Table 2. Discrete analysis considering all failures including timing failures.

	Run 1	**Run 2**	**Run 3**	**Run 4**	**Description**
$\mathbf{P_{tf}}$	0.00001	0.0001	0.001	0.01	Probability of critical timing failure
$\mathbf{P_{comf}}$	0.0001	0.0001	0.0001	0.0001	Probability of communication failure
$\mathbf{P_{gof}}$	0.01	0.01	0.01	0.01	Probability of gate open failure (**mech.**)
$\mathbf{P_{gcf}}$	0.01	0.01	0.01	0.01	Probability of gate closure failure (**mech.**)
$\mathbf{P_{nctf}}$	0.00001	0.00001	0.00001	0.00001	Probability of non-critical timing failure
MTTF	440.06	438.14	419.82	295.65	Mean-Time-To-Failure
$\mathbf{P_{cf}}$	0.50208	0.49991	0.47920	0.33882	Critical failure probability
$\mathbf{P_{tf}}$	0.00049	0.00485	0.04651	0.32885	Critical timing failure probability
$\mathbf{P_{ncf}}$	0.49695	0.49476	0.47383	0.33201	Non-critical failure probability
$\mathbf{P_{nctf}}$	0.00048	0.00048	0.00046	0.00032	Non-critical timing failure probability

To emphasize the significance of deadline-related failures (P_{tf} or P_{nctf}), all other failure rates (mechanical and communication) were set to zero in Table 3. This table illustrates the significance of assuring deadline guarantees even if no other failure exists.

Table 3. Discrete analysis considering only timing failure.

	Run 1	**Run 2**	**Run 3**	**Run 4**	**Description**
$\mathbf{P_{tf}}$	0.00001	0.0001	0.001	0.01	Probability of critical timing failure
$\mathbf{P_{comf}}$	0.0	0.0	0.0	0.0	Probability of communication failure
$\mathbf{P_{mechf}}$	0.0	0.0	0.0	0.0	Probability of mechanical failure: open/close
$\mathbf{P_{nctf}}$	0.00001	0.00001	0.00001	0.00001	Probability of non-critical timing failure
MTTF	450000.2	81815.2	8907.0	895.1	Mean-Time-To-Failure
$\mathbf{P_{tf}}$	0.50000	0.90910	0.99011	0.99901	Critical timing failure probability
$\mathbf{P_{cf\ /\ ncf}}$	0.0	0.0	0.0	0.0	Critical / Non-critical failure probability
$\mathbf{P_{nctf}}$	0.50000	0.09090	0.00989	0.00099	Non-critical timing failure probability

The input parameters used for the transition probabilities are related to the Lambda and Tau variables in the Markov model given in Figure 5 as follows:

P_{tf} =	critical timing failure probability:	τ_3
P_{comf} =	communication failure probability:	$\lambda_2, \lambda_5, \lambda_4$ and λ_7
P_{gof} =	mechanical (gate open) failure probability:	λ_8
P_{gcf} =	mechanical (gate close) failure probability:	λ_6
P_{mechf} =	mechanical failure probability:	λ_6 and λ_8
P_{nctf} =	non-critical timing failure probability:	τ_9

4.2.2. Continuous Analysis.

The results of the continuous analysis are shown in Figures 6 and 7. The sensitivity of the system failure rate (i.e., performability) to varying speeds of the train and the gate mechanism can be surmised from comparing the different curves (Runs 1-3) shown in Figure 6.

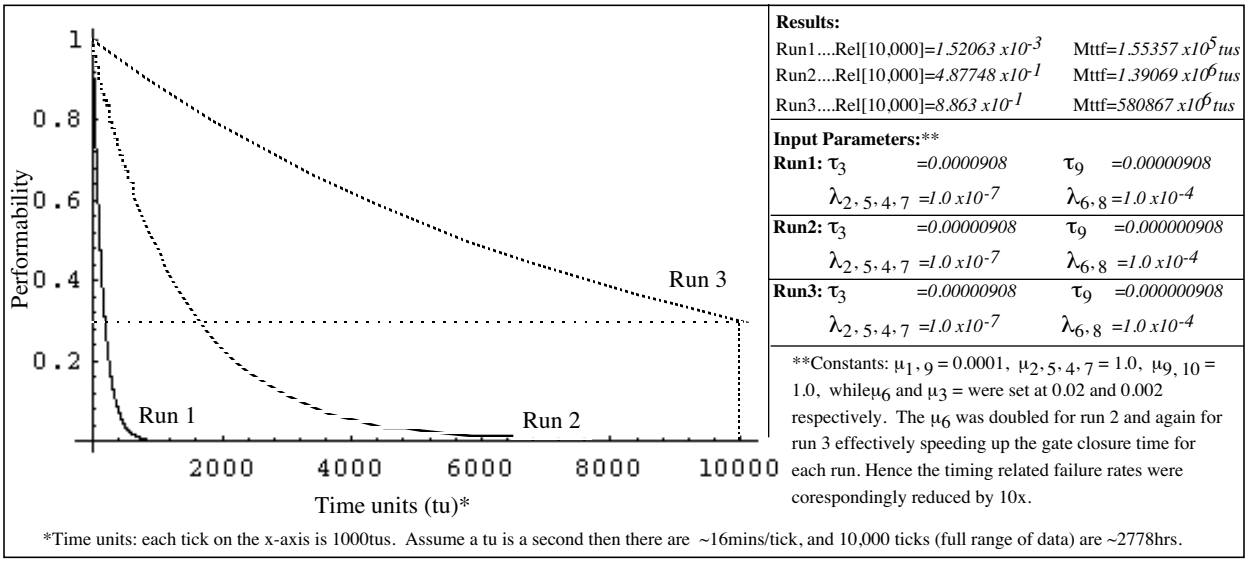


Figure 6. Performability for different gate closing speeds.

The following transition rates were assumed (see the figure caption for exact values).. The (constant) train arrival rate (i.e., rate at that trains enter the system) is μ_1 at 10^{-4}.(or one every 10,000 tus). It takes 1 time unit for sending or receiving a signal ($=1/\mu_2= 1/\mu_5= 1/\mu_4 = 1/\mu_7$) and includes any possible communication delays.. The average time it takes for the train to reach the intersection after sending the signal (i.e., train approach) is given by $1/\mu_3$ while the average time it takes for the gate to close (or open) is given by $1/\mu_6$ ($1/\mu_8$). Failure rates for signal

transmission of messages ($=\lambda_2 = \lambda_4 = \lambda_5 = \lambda_7$) are set equal to 10^{-7} while mechanical failures associated with the gate closing($= \lambda_6$) and opening ($=\lambda_8$) are set equal to 10^{-5}. In Figure 6 (Run1), it takes an average of 50 time units for the gate to close while it takes the train an average of 500 time units to approach (i.e., actually arrive at) the intersections. In Figure 6 (Run2) these numbers are 25 and 500 while in Figure 6 (Run3) they are 12.5 and 500. As can be seen from the graphs, the performability of the system increases as the slack time increases.

To emphasize the significance of timing failures (i.e., the train arriving at the intersection before the gate closes [and similarly for opening the gate]) we show the reliability of the system without timing failures in Figure 7 (Run2). The performability of the system with only timing failure and no mechanical or transmission failures is shown in Figure 7 (Run1). Note that Figure 6 (Run1) (which includes timing, transmission and mechanical failures combined) is set up identical to both runs in Figure 7 (except for the zeroing of either the τ's or the λ's). Clearly, the timing failures have a pronounced effect on the overall system performability. In this example they dominate the systems performance overall other types of failure.

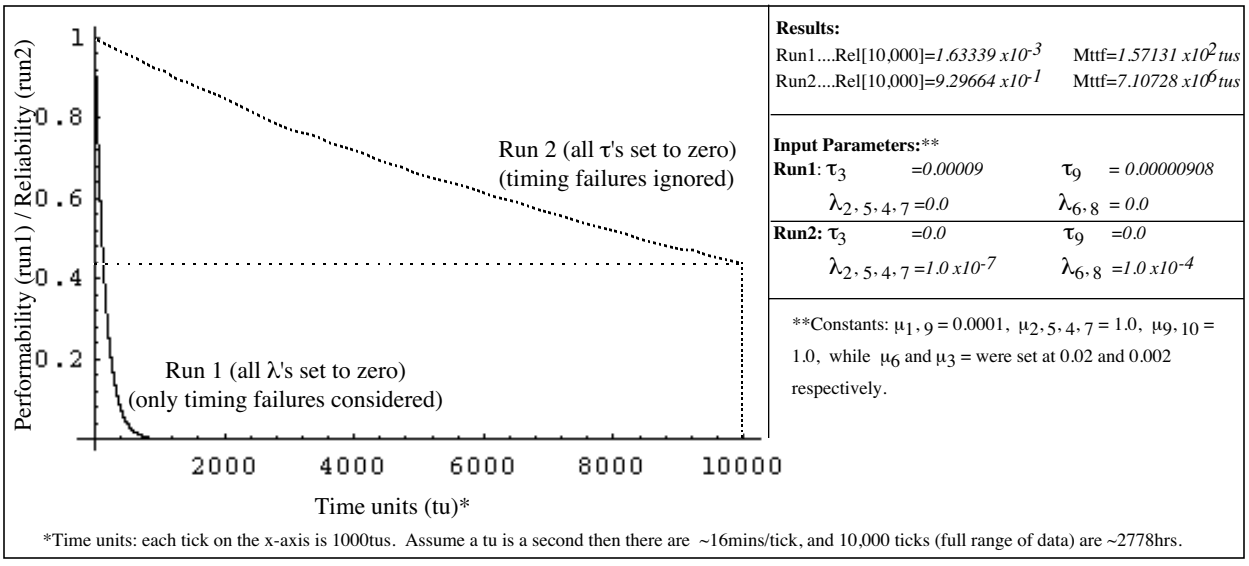


Figure 7. Comparison of system performability and reliability.

Once again, it is not our intention to attach any significance to the numbers; our only purpose is to illustrate the usefulness of these analyses in designing real-time systems with sufficient slack times and fault-tolerance to achieve a desired level of performability.

5. SUMMARY AND FUTURE WORK

Our objective in this paper was to show how the CSP specifications can be translated into Stochastic Petri nets for the purpose of performability and reliability analyses. Such translations can give insight into further refinements of the original specification (i.e., identify potential failure processes and recovery actions). Relating the parameters needed for performability analysis to user level specifications is essential for realizing systems that meet user needs in terms of cost, functionality, and other non-functional requirements [Sheldon 95a]. Our approach provides the needed feedback to a designer so that judicious cost-benefit analysis in providing fault-tolerance and deadline guarantees can be made. We have illustrated this approach in the railroad crossing example.[17] We recently completed a tool for automatically translating CSP specifications into Petri nets that enables us to utilize other stochastic analysis tools (viz., SPNP [Ciardo 89]). The canonical translation rules are slightly modified from those presented here (see [Sheldon 96]). The Performability tool is also being extended to permit evaluation of the various scheduling algorithms, and fault-tolerant mechanisms (e.g., task retries, task-migration). We hope to extend this environment to include reasoning about concurrency, safety, liveness and other real-time properties by employing the power of other tools that are available in this domain.

6. REFERENCES

[Chiola 85] 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.

[Chou 80] T.C.K. Chou and J.A. Abraham, "Performance/Availability model of shared resource multiprocessors," *IEEE Transactions on Reliability*, R-29, April, pp. 70-74.

[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.

[COSMOS 93] B.F. Lewis, et. al., "COSMOS Multicomputer operating system and development environment: Functional Specification," *NASA-JPL Memo.*

[Geist 90] R. Geist and K.S. Trivedi, "Reliability Estimation of Fault-Tolerant Systems: Tools and Techniques," *Computer*, (23)7, pp. 52-61, July 1990.

[17] Again, the failure probabilities, and task execution profiles 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.

[Gay 79] F.A. Gay and M.L. Ketelsen, "Performance evaluation for gracefully degrading systems," *Digest of 9th Int'l Symp on Fault-Tolerant Computing*, Madison, WI, pp. 51-58.

[Genrich 86] H.J. Genrich, "Pr/T-Nets," *Proc. of the Advanced Course on Petri Nets*, 1986.

[Heitmeyer 94] C. Heitmeyer, and N. Lynch, "The Generalized Railroad Crossing: A Case Study in Formal Verification of Real-Time Systems," *Proceedings of the Real-Time Systems Symposium*, December 7 - 9, 1994.

[Heitmeyer 93] C. Heitmeyer, R. Jeffords and B. Labaw, "Comparing different approaches for specifying and verifying real-time systems," *Proceedings of 10th IEEE Workshop on Real-Time Operating Systems and Software*, May 1993.

[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.

[Kavi 94] K.M. Kavi, H.Y. Youn, B. Shirazi and A.R. Hurson, "A performability model for soft real-time systems," *Proceedings the 27th Hawaii International Conference on Systems Sciences, HICSS-27* Vol. II pp 571-580 (Maui, HI, Jan. 4-7, 1994).

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

[Meyer 79] J.F. Meyer, D.G. Furchgott and L.T. Wu, "Performability evaluation of the SIFT Computer," *IEEE Transactions on Computers*, C-29, pp. 501-509.

[Murata 89] T. Murata, "Petri nets: Properties, Analysis and Applications," *Proceedings of the IEEE*, pp. 541-580, April 1989.

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

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

[Sheldon 96] Sheldon, F.T., "Specification and Analysis of Stochastic Properties for Concurrent Systems Expressed Using CSP," *Ph.D. Dissertation, U. of Texas at Arlington*, May 1996.

[Sheldon 95a] Sheldon, F.T., Kavi, K.M, and Kamangar, F.A., "Reliability Analysis of CSP Specifications: A New Method Using Petri Nets," *AIAA Proceedings Computing in Aerospace 10*, March 1995.

[Siewiorek 92] D.P. Siewiorek and R.S. Swarz, *Reliable Computer Systems: Design and Evaluation*, 2nd Edition, Digital Press, Bedford, MA.

[Stankovic 88] J.A. Stankovic and K. Ramamritham, *Tutorial on Hard Real-Time Systems*, IEEE CS Press.

[Stoyenko 92] A.D. Stoyenko, "The Evolution and State-of-the-Art of Real-Time Languages," *The Journal of Systems and Software*, April , pp. 61-84

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

[Vautherin 87] J. Vautherin , "Calculation of Semi-Flows for Pr/T-Nets," *IEEE Proc of the Int'l Wkshp on Petri Nets and Performance Models*, Madison, WI, pp. 174-183, August 1986.

APPENDIX A: GENERAL EXAMPLES OF CSP/PN TRANSLATIONS

A.1 CSP TO PN DEFINITIONS:

Figures A.1 through A.3 define a small set of equivalent CSP symbols/expressions and Petri net structures as a basis for the translations demonstrated in this paper. Also, there exists some set of composition rules that define how such structures should be composed (broad spectrum of interpretation). Our goal is to translate CSP specifications into irreducible PNs that are easily analyzed. Figure A.4 is a simple composition to illustrate this process.

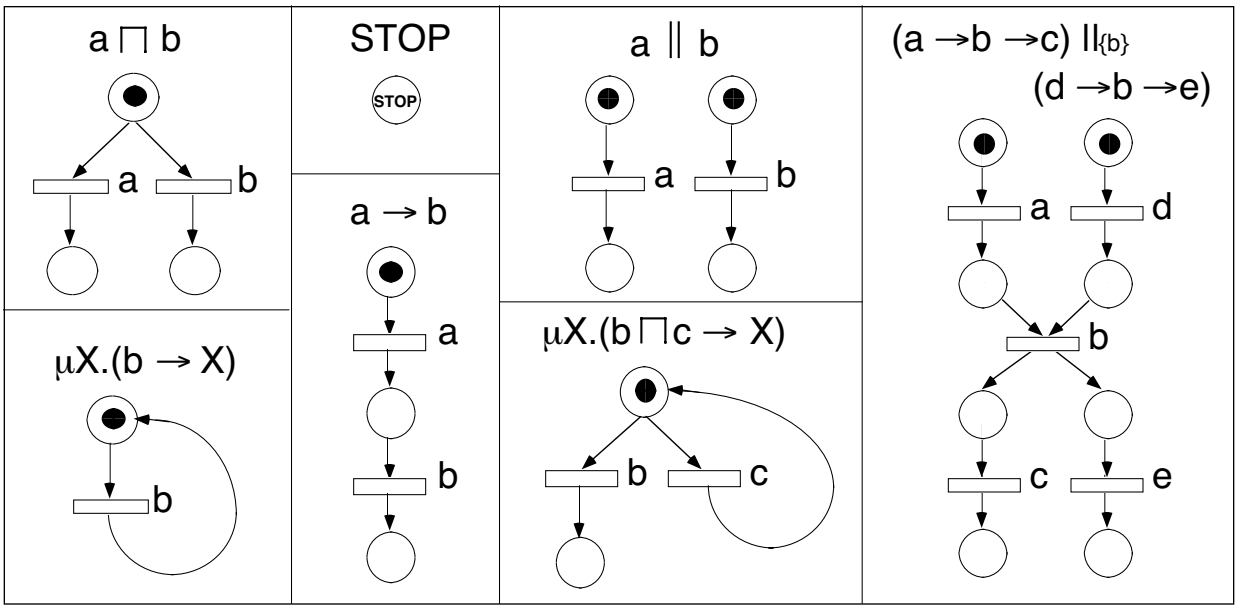


Figure A.1 CSP/PN Synonyms and Equivalent Expressions and Constructs.

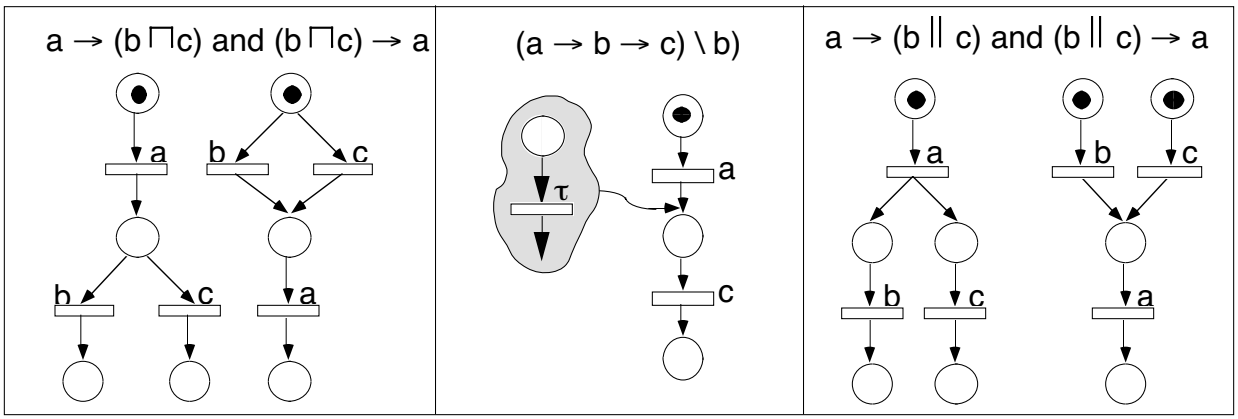


Figure A.2 CSP/PN Equivalent Expressions and Constructs (CSP top, PN bottom).

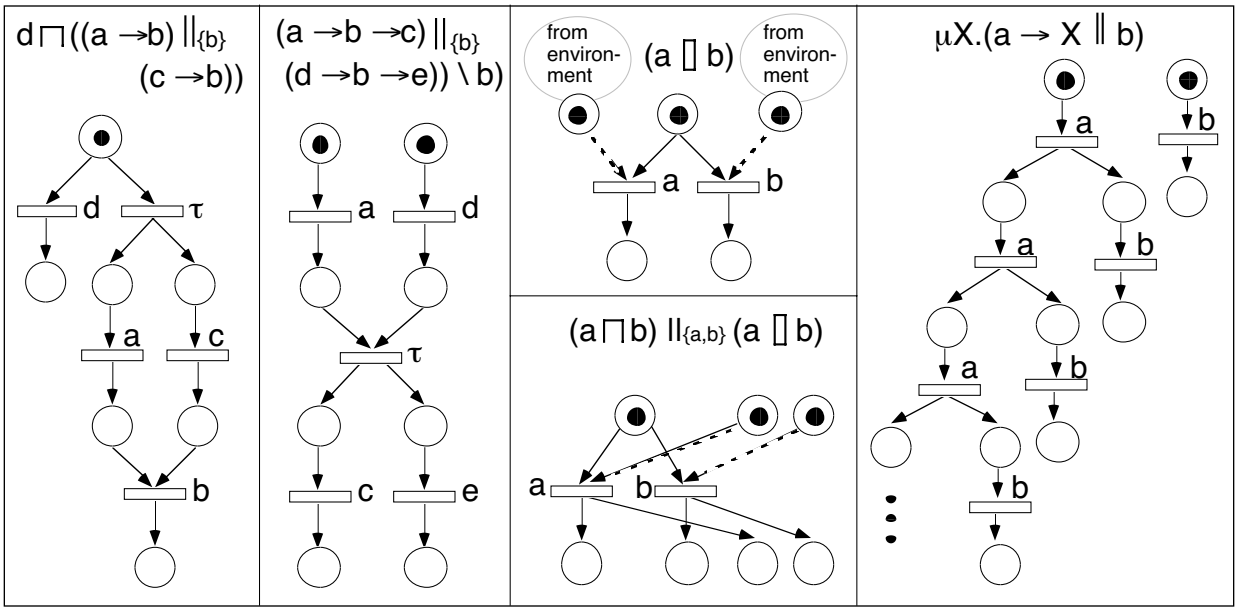


Figure A.3 CSP/PN Equivalent Expressions and Constructs (CSP top, PN bottom).

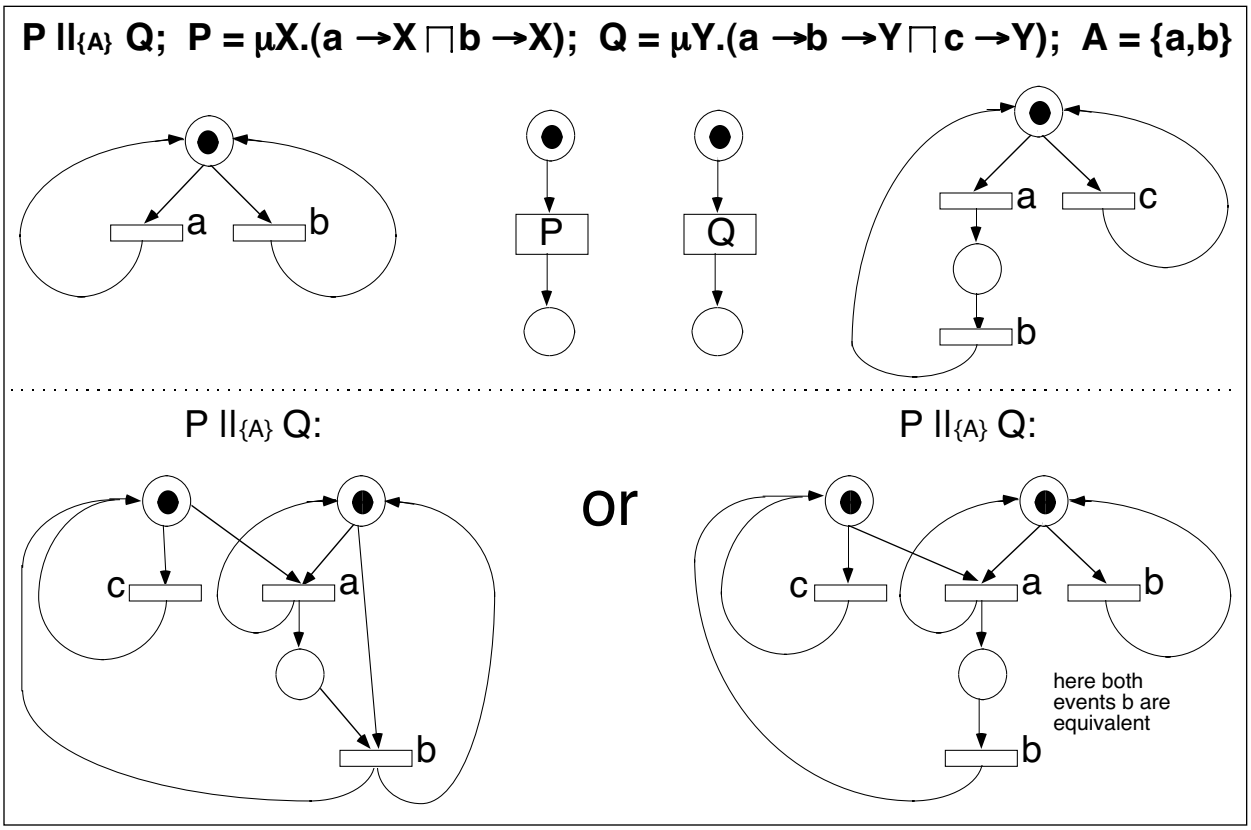


Figure A.4 CSP/PN Equivalent Expressions and Constructs (CSP top, PN bottom).

Listing of all figure and table captions
