MODELING AND STOCHASTIC ANALYSIS OF EMBEDDED SYSTEMS EMPHASIZING COINCIDENT FAILURES, FAILURE SEVERITY AND USAGE-PROFILES

By

KSHAMTA JERATH

A thesis submitted in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE
(COMPUTER SCIENCE)

WASHINGTON STATE UNIVERSITY
School of Electrical Engineering and Computer Science

AUGUST 2002

To the Faculty of Washington State University:

The members of the Committee appointed to examine the thesis of KSHAMTA JERATH find it satisfactory and recommend that it be accepted.

_____
Chair

_____

_____

ACKNOWLEDGEMENT

I am grateful to Dr. Frederick T. Sheldon, my major advisor, who channeled my research in the right direction and from whom I received precious pointers throughout my master's program. I am very thankful to his dedicated teaching and enthusiasm for work in this field.

Special thanks to Juergen S. Schwarz, the Technical Lead for Research Information and Communication System Safety at DaimlerChrysler, who provided invaluable feedback regarding the analysis of results and the discussion about modeling strategy. I am also thankful to my committee members, Dr. Zhe Dang and Dr. Curtis Dyreson for their insightful comments on my work.

I wish to thank all the members of the *Software Engineering for Dependable Systems* (SEDS) Lab, including Hye Yeon Kim, Zhihe Zhou, Shuren Wang, David Dugan, Yingxia Wang and Shuangshuang Jin for their help and support. I am also thankful to Ms. Ruby Young who was always very helpful and patiently answered all my questions concerning administrative procedures.

Finally, my sincere thanks to my parents and brother, who provided immeasurable moral support and prodded me on to work diligently, from across the miles. I am extremely thankful for their support that kept me going, and made this possible.

MODELING AND STOCHASTIC ANALYSIS OF EMBEDDED SYSTEMS EMPHASIZING COINCIDENT FAILURES, FAILURE SEVERITY AND USAGE-PROFILES

ABSTRACT

by Kshamta Jerath, M.S.
Washington State University
August 2002

Chair: Frederick T. Sheldon

The increasingly ubiquitous use of software systems has created the need of being able to depend on them more than before, and of being able to measure just how dependable they are. Knowing that the system is *reliable* is absolutely necessary for safety-critical systems, where any kind of failure may result in an unacceptable loss of human life. This study models and analyzes the Anti-lock Braking System of a passenger vehicle. Special emphasis is laid on modeling extra-functional characteristics of coincident failures, severity of failures and usage-profiles - the goal is to develop an approach that is realistic, generic and extensible for this application domain. Components in a system generally interact with each other during operation, and a faulty component can affect the probability of failure of other correlated components. The severity of a failure is the impact it has on the operation of the system. This is closely related to the notion of hazard which defines what undesirable consequence will potentially result from the incorrect system operation (i.e., threat). Usage profile characterizes how the system is used for the purpose of modeling and reliability analysis. Only those failures that occur

during active use are considered in reliability calculations. The strategy of modeling these characteristics (using empirical data) is innovative in terms of the approach used to integrate them into the Stochastic Petri Net and Stochastic Activity Network formalisms. The validation approach compares the results from the two separate models using the two different modeling formalisms. The results were found to be comparable and confirm that the effect of modeling coincident failures, failure severity and usage-profiles is noticeable in determining overall system reliability. The contribution of this research to the automotive industry is substantial as it offers a greater insight into the strategy for developing realistic models. This work also provides a solid basis for modeling more complex systems and carrying out further analyses.

LIST OF PUBLICATIONS

Kshamta Jerath and Frederick T. Sheldon. “Reliability Analysis of an Anti-lock Braking System using Stochastic Petri Nets.” *Fifth International Workshop on Performability Modeling of Computer and Communication Systems (PMCCS’5)*, Erlangen, Germany. September, 2001.

Frederick T. Sheldon, Kshamta Jerath and Stefan Greiner. “Examining Coincident Failures and Usage Profiles in Reliability Analysis of an Embedded Vehicle Sub-System.” *16th European Simulation Multiconference (ESM’02)*, Darmstadt, Germany. June 3-5, 2002.

Frederick T. Sheldon and Kshamta Jerath. “Predicting Reliability of an Embedded Vehicle System by modeling Coincident Failures and Usage-Profiles.” *13th International Symposium on Software Reliability Engineering (ISSRE’02)*, Annapolis, MD, November 12-15, 2002. (Submitted)

Kshamta Jerath and Frederick T. Sheldon. “Stochastic Modeling and Analysis of an Embedded Vehicle Sub-system with Emphasis on Coincident Failures, Failure Severity and Usage-Profiles.” IEEE Transactions on Reliability. (In preparation for submission).

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

And here Alice began to get rather sleepy, and went on saying to herself, in a dreamy sort of way, `Do cats eat bats? Do cats eat bats?' and sometimes, `Do bats eat cats?' for, you see, as she couldn't answer either question, it didn't much matter which way she put it.

- Alice in Wonderland

1.1 Problem Definition

This study models and analyzes the Anti-lock Braking system of a passenger vehicle using two different stochastic formalisms - Stochastic Petri Nets (SPN) and Stochastic Activity Networks (SAN). Special emphasis is laid on modeling extra-functional characteristics of coincident failures, severity of failures and usage-profiles. The goal is to develop a realistic and extensible model of the system (useful as a framework for future analysis), to carry out its reliability analysis using the two different formalisms and to compare the results. The modeling approach must overcome the two most common challenges in modeling using Markov models – large state space and stiffness. Further, the strengths and weaknesses of each of the two tools used (Stochastic Petri Net Package for solving SPN models and UltraSAN for solving SAN models) need to be examined so that they can be used to achieve robust models and accurate analysis results.

1.2 Motivation

The increasingly ubiquitous use of software systems has created the need of being able to depend on them more than before; and being able to measure just how dependable they are. Knowing that the system is reliable is absolutely necessary for safety-critical systems, where any kind of failure may result in an unacceptable loss of human life.

Reliability is the probability that a system will deliver its intended functionality and quality for a specified period of "time" and under specific conditions, given that the system was functioning properly at the start of this "time" period [1]. Structured models of reliability allow the reliability of a system to be derived from the reliabilities of its components.

A complex embedded vehicle system (like the Anti-lock Braking System) is composed of numerous components and the probability that the system survives (efficient or acceptable degraded performance) depends directly on each of the constituent components. The reliability analysis of a vehicle system can provide an understanding about the likelihood of failures occurring in the system and an increased insight to manufacturers about inherent "weaknesses" in the system [2].

1.2.1 Need for a Realistic and Extensible Model

In [3], the authors presented Stochastic Petri Net (SPN) models of a vehicle dynamic driving regulation (DDR) system. Sub-system representations of the Anti-lock Braking System (ABS), the Electronic Steering Assistance (ESA), the Automatic Slip Reduction (ASR) and a combined model were developed and analyzed for critical failures. The main theoretical idea stated for future work was the decomposition of the stochastic problem into a finite number of manageable scenario sub-problems and the coordination of their solutions by specially designed algorithms. They asserted that there is a great deal of disconnectedness among the steps needed to (1) understand the problem, (2) break it into manageable sub-problems, (3) develop models that are realistic in terms of the sub-problems they represent and, (4) combining them into the larger more complex context.

In this study, I have focused on modeling and analyzing the Anti-lock Braking System. Naturally, this is but one component of the total system and there is an implicit requirement that the developed model be easily extensible and fit into a larger complex context. Further, the model needs to be realistic enough to take into consideration certain extra-functional relationships among components of the system, as discussed in the next two sub sections.

1.2.2 Importance of Modeling Coincident Failures and Severity of Failures

If a system does not contain any redundancy – that is, if every component must function properly for the system to work - and if component failures are statistically independent, then the system reliability is simply the product of the component reliabilities. Furthermore, the failure rate of the system is simply the sum of the failure rates of the individual components [4]. The assumption that failures occur independently (in a statistical sense) in hardware components is a widely used and often successful model for predicting the reliability of hardware devices. However, components generally interact with each other during operation, and a faulty component can affect the probability of failure of other components too [5]. Such failures are coincident in the sense that failure of one component increases the probability of failure of another. Thus, for the model to be realistic, it is important to consider coincident failures.

Another aspect of modeling failures occurring in the system is their severity. Severity of a failure is the impact it has on the operation of the system. It is closely related to the threat (hazard) the problem poses, in functional terms, to the correct operation of the system [1]. Predicting the reliability/availability based on the characteristics of a model of the system provides more objective and concrete information that can be used in

assessing the risk tradeoffs and integrity levels. Severity is an important candidate to weight the data used in reliability calculations and must be incorporated into the model to determine the probability that the system survives, including efficient or acceptable degraded operation.

1.2.3 Importance of Modeling Usage-Profiles

A software-based product's reliability depends on just how a customer will use it. The operational profile – quantitative characterization of how a system will be used – is essential in software reliability engineering [6]. The same basic concept can be extended and applied for predicting *system* reliability. We extend the idea of operational profiles – considering the *use* of a software system during testing; into usage profiles – the usage of the system (hardware **and** software) for modeling and reliability analysis.

The reliability of a system depends on its usage profile - users interact with the system in an intermittent fashion, resulting in operational workload profiles that alternate between periods of "active" and "passive" use. Reliability is concerned with the service that is actually delivered by the system as opposed to a system's capacity to deliver such service [7]. Intermittent use influences the mean time to failure and reliability of the system. Specifically, while considering usage profiles, faults need not necessarily cause failures since they can be repaired; failures occurring during "active" use of the system only should contribute to reliability calculations.

1.2.4 Comparing Results from two Stochastic Formalisms

Markov Models are a basic and powerful tool for modeling systems composed of several processes (such as a failure process and a repair process). They are a tool for both reliability and availability modeling. However, when using a model, there is always the

question of whether it accurately reflects the important facets of the system for the purpose of the decisions to be made. Since the model is just an abstraction of the real world problem, predictions based on the model should be validated against actual measurements. A poor validation (lack of correlation between what is predicted and what is empirically observed) may suggest modifications to the original model [8].

Since validation by comparison against actual measurements is beyond the scope of this research (see Section 5.3 for further discussion), the objective was to compare the results obtained by modeling the same sub-system using two different stochastic formalisms: Stochastic Petri Nets (SPNs) and Stochastic Activity Networks (SANs). SPNs are a powerful tool for the description and the analysis of systems that exhibit concurrency, synchronization and conflicts [9]. SANs are a generalization of SPNs that permit a more expressive, general and flexible representation of concurrency, timeliness, fault tolerance, and degradable performance in a single model [10].

1.3 Organization

The rest of this thesis is organized as follows: Chapter 2 provides a survey of related work. Chapter 3 introduces an embedded system – the Anti-lock Braking System which is used as a representative example to explain the modeling strategy. Chapter 4 enumerates the modeling philosophy using both SPNs and SANs for representing severity of failure, coincident failures and usage-profiles. Chapter 5 presents the results and a discussion of the analyses. Chapter 6 concludes with a brief summary and direction for future research. There are two brief appendices that provide some code listings and models developed, an appendix that lists the various symbols used in SPNs and SANs and an appendix that provides a discussion on failure severity levels.

CHAPTER TWO

RELATED RESEARCH

It sounded an excellent plan, no doubt, and very neatly and simply arranged; the only difficulty was, that she had not the smallest idea how to set about it.

- Alice in Wonderland

2.1 Introduction to Modeling and Analysis

Systems consist of hardware and software, and it is common to fully design, implement and functionally test them before an attempt is made to determine their performance characteristics. At the same time, the redesign of both hardware and software is costly and may cause late system delivery [11]. Complexity in such systems is one of, if not the most important properties that make the design and the implementation of high assurance systems so difficult. Furthermore, as the complexity of future systems increases, the more important it becomes to evaluate and predict their behavior. To better understand the complexity, it is common practice to create a model [12]. A model is a representation of the system and is studied as a surrogate for the actual system [13].

A model is an abstraction of a system that includes sufficient detail to facilitate an understanding of system behavior. To be useful, the model should reflect important system characteristics such as fault tolerance, automatic reconfiguration and repair, contention for resources, concurrency and synchronization, deadlines imposed on tasks, and graceful degradation [14].

2.1.1 Defining Important Terms

To optimize and enhance systems, various forms of evaluation are carried out on

models and as these systems become more complex, evaluation of these models do the same. The two most basic aspects of evaluation are: **performance** and **dependability**.

Performance is defined as "quality of service, provided the system is correct." Dependability is "the property of a system which allows reliance to be justifiably placed on the service it delivers" [15]. Dependability encompasses failure, reconfiguration and repair related aspects of system behavior. Reliability, availability, safety and related measures are collectively known as dependability.

Reliability is the probability that a system, or a system component, will deliver its intended functionality and quality for a specified period of time, and under specified conditions, given that the system was functioning properly at the start of this time period. Availability is the probability that a system, or a system component, will be available to start a mission at a specified time [1].

2.1.2 Evaluation Methodology

To assure an appropriate performance, today's evaluation methodology includes the following steps as shown in Figure 1 (Figure 3 in [11] reproduced with permission of the publisher). Workload characterization and system parameter specification are the first sensitive steps.

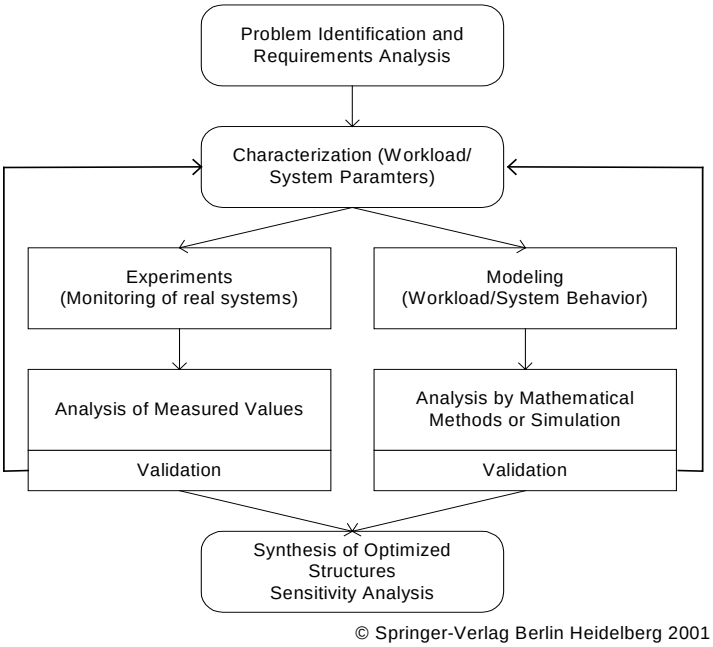


Figure 1: Overview of Performance Evaluation Methodology

Determining these values requires care and knowledge about both the application and the technical system components. Next, the design methodology distinguishes between two totally different but complementary approaches: experiments on the real system (measurements) and modeling. Both are followed by analysis steps using methods of statistics, stochastic processes and simulation. The validation of experimental and modeling results follows next and is very important. Finally, system structures and operating modes are synthesized; systematic parameter variation and mathematical optimization techniques guarantee good system design.

2.1.3 Modeling and Analysis Techniques

Due to the recent development in model generation and solution techniques, and the availability of software tools, large and realistic models can be developed and studied. A system designer has a wide range of different types of models to choose from. Each type has its strengths and weaknesses in terms of accessibility, ease of construction, efficiency and accuracy of solution algorithms, and availability of software tools. The most appropriate type of model depends upon the complexity of the system, the questions to be studied, the accuracy required, and the resources available for study [14].

For example, combinatorial models such as fault-trees and reliability block diagrams are efficient in both specification and evaluation of systems models. But it is difficult, if not impossible, to allow for various types of dependency (such as repair dependency and near-coincident-fault type dependency), transient and intermittent faults, and so forth. Markov models can capture such interesting system behavior. The model, thus, can be developed using a formalism appropriate for the system under study.

Once a mathematical model has been built, it must then be examined to see how it can be used to answer the questions of interest about the system it is supposed to represent. There are two basic methods used to solve the system model: mathematical and system simulation [12]. While mathematical solution methods allow one to obtain *exact* information on questions of interest, simulation evaluates a model numerically in order to *estimate* the desired true characteristics of the system [13]. The mathematical solution method may be further classified into analytical (non-state-space based) and numerical (state-space based). The mathematical method works by solving a system (or set) of linear or differential equations while a simulation is differentiated into discrete event simulation and continuous simulation.

2.1.3.1 Analytical Solution Methods

Reliability block diagrams, fault trees and reliability graphs are non-state-space methods commonly used to study dependability of systems. They are concise, easy to understand and have efficient solution methods. However, realistic features such as non-independent behavior of components, imperfect coverage, non-zero reconfiguration delays, and combination with performance cannot be captured by these models [14].

2.1.3.2 Numerical Solution Methods

State-space based models enable us to overcome the limitations of the non-state-space models in modeling complicated interactions between measures of interest. Most commonly used state space models are Markov chains. They provide great flexibility for modeling dependability, performance and combined dependability and performance measures [14].

2.2 Stochastic Processes and Models

A family of random variables[1] that is indexed by a parameter such as time is known as a stochastic process. A stochastic process $\{X(t) \mid t \in T\}$ is defined over a given probability space and is indexed by the parameter t (time), where t varies over an index set T. The values assumed by the random variable X(t) are called states, and the set of all possible values forms the state space of the process [8].

2.2.1 Markov Process

A Markov Process is a stochastic process whose dynamic behavior is such that probability distributions for its future development depend only on the present state and not on how the process arrived in that state (the so called memory-less property). If we assume that the state space is discrete (finite or countably infinite), then the Markov process is known as a Markov chain. If we further assume that the parameter space, T, is also discrete, then we have a discrete-parameter Markov chain; otherwise a continuous-parameter Markov chain [8].

2.2.2 Applicability of Markov Chains

These days, Markov chains and stochastic processes form the basis for model-based system evaluations in many areas of science and engineering. They find applicability, for instance, in biology to model growth and decay of populations, in physics to model interactions between elementary particles, in chemical engineering to model (chain) reactions between molecules or to model mixing processes, in management sciences to model the flow of commodities in logistic or flexible manufacturing systems or to model the availability of production lines and, most notably, in computer and communication

[1] A random variable is a rule that assigns a numerical value to each possible outcome of an experiment.

science and engineering to model system performance and dependability in a wide variety of settings [16].

2.2.3 Performability and Markov Reward Models

Performability is a fabricated word that combines the two terms: performance and reliability. The performability discipline tries to merge these two modeling paradigms. The systems under consideration are so-called degradable systems, meaning the system may be able to survive the failure of one or more system components. Once a system component fails, the system may continue to operate with a reduced performance. In such cases, it is necessary to consider both performance and reliability together [12, 17].

2.2.3.1 Performability Model Specification

A specification of a performability model can be regarded as having three major ingredients [18] shown in Figure 2.

S1. Specification of what is to be learned about the object system from its (model-based) evaluation, i.e., the performability measures of interest.

S2. Specification of a stochastic process on which the evaluation is to be based (a base model of the total system).

S3. Specification of how S2 relates to S1 in a manner that permits the base model (after construction) to support solution of the

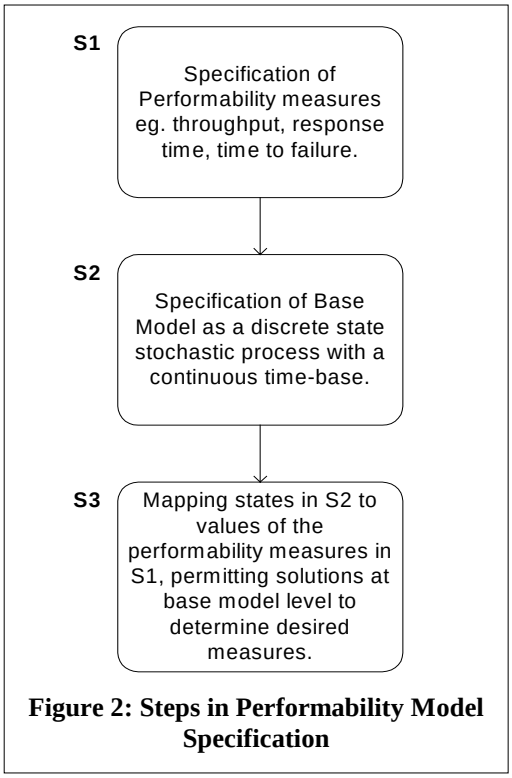


Figure 2: Steps in Performability Model Specification

specified measures.

Moreover, given that the recipient of the above is a model-based evaluation tool, languages used to express S1-S3 must be sufficiently formal to permit their unambiguous interpretation and subsequent automated realization by the tool. Naturally, the results obtained after analysis may instigate subsequent enhancements to the model, requiring the steps S1-S3 to be repeated for a more robust model.

2.2.3.2 Markov Reward Models

The most common solution method for performability is based on reward models. This model associates reward rates with state occupancies. The reward rate can be thought of as the work accomplished in that specific state. By combining the model of a stochastic process for a given system with the reward rates for that system, a reward model results. The total reward accumulated over a given time period is the performance of the system. Performability then results by combining this performance with a Markov process representing the dependability of the system [15]. Markov reward models are the most common technique for modeling degradable systems.

Markov Reward Models (MRM) have the potential to reflect concurrency, contention, fault-tolerance, and degradable performance; they can be used to obtain not only system performance and system reliability/availability measures, but also combined measures of performance and reliability/availability [19].

2.2.4 Challenges in Modeling

Structured models of reliability allow the reliability of a system to be derived from the reliabilities of its components. The reliabilities of individual components are often easier to estimate or are known before the system is even built. Markov Models have

been used successfully in numerous instances to specify and evaluate the performance/reliability of systems. However, practical issues that stand in the way of developing such models include: (1) obtaining reliability data of components, (2) a simple model being able to capture only limited interactions among components, (3) the need to estimate fault correlation between components, and (4) reliability depends on how the system is used, thereby usage information being an important part of reliability evaluation [20].

Further, two distinct problems that arise while using Markov processes are largeness and stiffness [14]. The size of a Markov Model for the evaluation of a system grows exponentially with the number of components in the system. If there are n components, the Markov Model may have up to 2^n states. This causes the analysis to take a great deal of time. Stiffness is due to the different orders of magnitude (sometimes 10^6 times) between the rates of occurrence of performance-related events and the rates of rare, failure-related events. Stiffness leads to difficulty in the solution of the model and numerical instability. Any attempt at modeling using Markov models must address these two problems. These challenges have been overcome in the current study as described in Sections 4.1.1.2, 4.1.2.2, 4.2.1.2 and 4.2.2.2.

2.3 Stochastic Modeling Techniques and Tools

As discussed above, state-space based models such as Markov models are capable of capturing the various kinds of dependencies that affect the prediction of reliability/ availability for a given system or proposed system. The sizes of these Markov models tend to be very large for complex systems and hence are difficult to specify and manage. A number of techniques exist that can be used to generate the (large) underlying Markov

chains automatically from a concise description of the system. Two of them – Stochastic Petri Nets and Stochastic Activity Networks, are discussed here.

2.3.1 Stochastic Petri Nets

Petri Nets (PNs) are abstract formal models that have been developed in search for natural, simple and powerful methods for describing and analyzing the flow of information and control in systems [9]. They are a graphical and mathematical tool for describing and studying information processing systems that are characterized as being concurrent, asynchronous, distributed, parallel, non-deterministic and/or stochastic [21].

2.3.1.1 Basic Overview

A PN is a bipartite directed graph whose nodes are divided into two disjoint sets *places* and *transitions* [19]. Places (drawn as circles) represent conditions, and transitions (drawn as bars) represent events. A marked Petri net is obtained by associating *tokens* with places. Tokens (drawn as small filled circles) are moved from place to place when the transitions fire, and are used to denote the conditions holding at any given time. As an event is usually enabled by a combination of conditions, a transition is enabled by a combination of tokens in places. An *arc* is drawn from a place to a transition (input arc) or from a transition to a place (output arc). Arcs are used to signify which combination of conditions must hold for the event to occur and which combination of conditions holds after the event occurs. A *cardinality* may be associated with these arcs. A transition is enabled if each input place contains at least one token (or at least equal to the cardinality of the input arc from that place); an enabled transition *fires* by removing a token from each input place and depositing a token in each output place [22].

The Stochastic Petri Net (SPN) model is obtained from the Petri net model by associating a probability distribution function to the firing time of each transition. Transitions with an associated exponential distribution function are said to be *timed*; transitions with zero time distribution are said to be *immediate* [23]. An SPN can be analyzed by considering all possible markings (enumerations of the tokens in each place) and solving the resulting reachability graph as a Markov chain. The symbols used to represent the various components of an SPN are present in Appendix C.1.

2.3.1.2 An Example SPN Model

Consider the well-known example of a producer-consumer system with two processes, one that produces data and places it into the (infinite) queue and the second that reads the data from the queue and consumes it. Figure 3 shows the SPN model of this system. Places *process_1* and *process_2* model the state when either process is ready to write and read from the queue respectively (denoted by the presence of a token in those places). Transitions *write* and *read* perform the function of actually writing data and reading data from the queue respectively. The temporal characterization of these two transitions is based on assumptions about the duration of such operations; the choice of immediate transitions here amounts to neglecting the delays inherent in such operations. The queue is denoted by the place *queue*. The number

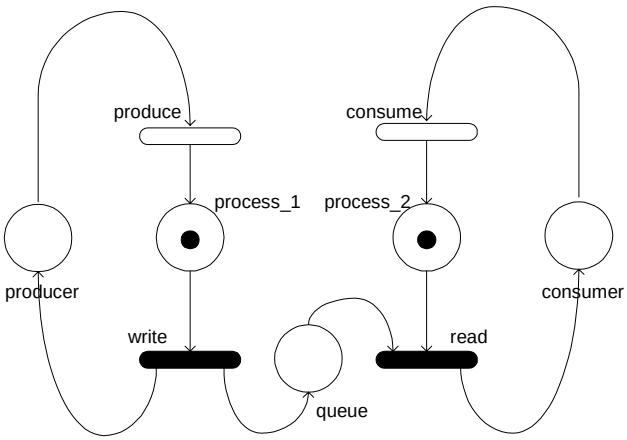


Figure 3: Example SPN Model

of tokens in this place indicates the number of data values available for reading. When there is no token in this place, the transition *read* is not enabled and hence nothing can be read from the queue. Places *producer* and *consumer* indicate the state when the processes are ready to produce the data and process the data read respectively. Transitions *produce* and *consume* perform the function of actually producing and consuming the data. The temporal characterization of these two transitions is again derived by assumptions about the duration of such processing.

2.3.1.3 Stochastic Petri Net Package

A number of software packages exist that enable the performability and reliability analysis of SPNs. The Stochastic Petri Net Package[2] (SPNP) has been developed by Ciardo *et al.* at Duke University [23, 24].

The model type used for input is a stochastic reward net (SRN). SRNs incorporate several structural extensions to SPNs such as marking dependencies (marking dependent arc cardinalities, enabling functions etc.) and allow reward rates to be associated with each marking. The reward function can be marking dependent as well. There is no interactive interface, but a graphical interface exists [25].

SRNs are specified using CSPL (C based Stochastic Petri Net Language) which is an extension of C with additional constructs for describing the SPN models. SRN specifications are automatically converted into an MRM, which is then solved to compute a variety of transient, steady state, cumulative, and sensitivity measures.

Thus, SPNP allows the specification of Stochastic Reward Models, the computation of steady state, transient, cumulative, time-averaged and "up-to-absorption" measures and

[2] SPNP is written in C and runs on a variety of operating systems including UNIX, AIX, OS/2 and VMS.

sensitivities of these measures. Efficient and numerically stable algorithms employing sparse matrix techniques are used to solve the underlying Markov chain. Parametric sensitivity analysis of Stochastic Petri Net models is also implemented.

2.3.2 Stochastic Activity Networks

Stochastic Petri Nets (SPNs) are limited in their expressive power, and these limited operations make it very difficult to model complex interactions. More general and flexible formalisms are needed to represent real systems. The need for a more expressive modeling language has led to several extensions to SPNs. Stochastic Activity Networks (SANs) are one such extension, defined with the express purpose of facilitating unified performance/dependability (performability) evaluation as well as more traditional performance and dependability evaluation [26].

Specifically, SANs permit both the representation of complex interactions among concurrent activities (as can be represented in SPNs) and non-determinism in actions taken at the completion of some activity (this type of uncertainty does not have a natural representation in SPNs). SPNs exhibit non-deterministic behavior as the consequence of *temporal* uncertainty i.e., among a set of enabled transitions, there is uncertainty as to which transition will fire. When modeling the structure and behavior of complex systems, one wants to represent *spatial* uncertainty as well as temporal uncertainty e.g., on the completion of an activity, the uncertainty about the next state of the system. SANs permit the representation of both temporal and spatial uncertainty in a natural, well-defined manner [27], using output cases associated with each activity. On the other hand, the only way of representing such uncertainty in SPNs is to model it as a conflict among immediate transitions.

2.3.2.1 Basic Overview

SANs, a generalization of SPNs, permit the representation of concurrency, fault tolerance, and degradable performance in a single model [10]. Using graphical primitives, SANs are more compact and provide greater insight into the behavior of the network.

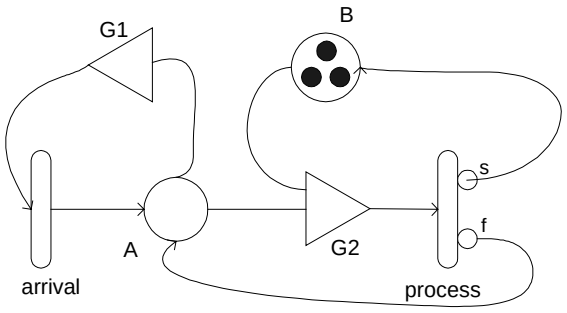
Structurally, SANs consist of four primitive objects: *places*, *activities*, *input gates* and *output gates* [28, 29]. Places represent the state of the modeled system. They are represented graphically as circles. Each place contains a certain number of tokens, which represents the marking of the place. The set of all place markings represents the marking of the network. Activities represent actions in the modeled system that take some specified amount of time to complete. They are of two types: *timed* and *instantaneous*. Timed activities have durations that impact the performance of the modeled system, and are represented as hollow ovals. Instantaneous activities represent actions that complete in a negligible amount of time compared to the other activities in the system. *Case probabilities*, represented graphically as circles on the right side of an activity, model uncertainty associated with the completion of an activity.

Input gates control the enabling of activities and define the marking changes that will occur when an activity completes. They are represented graphically as triangles with their point connected to the activity they control. Like input gates, output gates define the marking changes that will occur when activities complete. The only difference is that output gates are associated with a single case. They are represented graphically as triangles with their flat side connected to an activity or a case. The symbols used to represent the various components of a SAN are present in Appendix C.2.

For solution, a SAN is converted into a state-level representation (via markings) called Stochastic Activity System (SAS). If this is Markovian in nature, a Markov model is generated and solved.

2.3.2.2 An Example SAN Model

Consider the example of a (M/M/3) system that consists of three processors and an arrival queue for tasks. With probability s, a processor successfully completes the task assigned to it. With probability f, a processor fails and is unrepairable, and the task returns to the queue. Figure 4 shows the SAN model of the system.



Gate	Enabling Predicate	Function
G1	MARK(A) < 5	;
G2	-	MARK(A)=MARK(A)-1; MARK(B)=MARK(B)-1;

Activity	Rate	Probability	
		Case1	Case2
arrival	lambda	1	-
process	mu	s	f

Figure 4: Example SAN Model

The three tokens in place B indicate the three working processors; the place A indicates the arrival queue of tasks. The activities *arrival* and *process* are timed activities with one and two cases respectively and with rates and probabilities as indicated in the diagram. Gate G1 is an input gate to monitor capacity of queue: only when there are less than 5 tasks in the arrival queue is the arrival activity enabled. G2 is an input gate, that enables the *process* activity when there are tokens in both places A and B (implicit condition).

2.3.2.3 UltraSAN

UltraSAN is an X-window based software tool for evaluating systems that are represented as Stochastic Activity Networks. UltraSAN[3] has been developed by Sanders *et al.* [29] at the University of Arizona.

Three main tools are used for model specification: the SAN editor, the composed model editor, and the performance model editor [29]. The SAN editor expedites the specification of the SAN sub-models by allowing the user to enter the SAN graphically. The composed model editor is used to draw a tree representing the connection of the sub-models. Finally, the performance variable editor is used to specify reward variables. Rewards may be specified for activity completions or may be based on specific markings of the model. Figure 5 [29] (reproduced with permission of the publisher) shows the organization and data flow in UltraSAN.

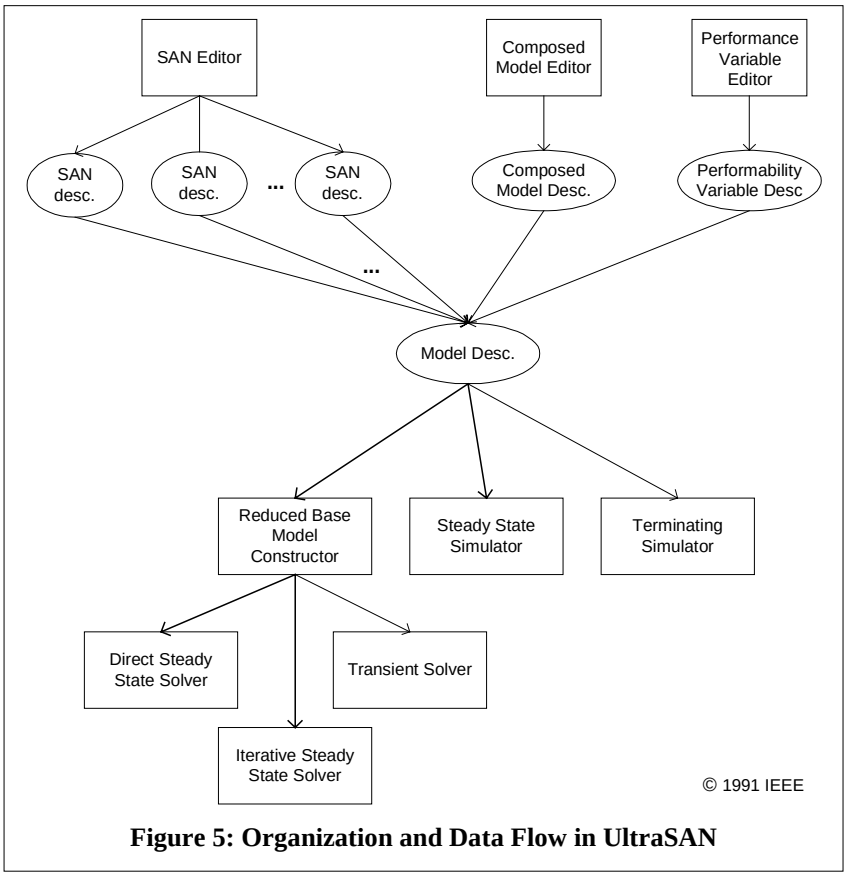


Figure 5: Organization and Data Flow in UltraSAN

[3] UltraSAN is written using C and X-window interface library and runs on UNIX on DEC, SUN and AT&T workstations. Extensive support is provided for performability analysis.

Both analytical solvers and simulators are provided, and the tool also has a report generator, which facilitates the generation of graphs and tables from the obtained results. Steady-state and transient solutions are possible. Largeness of state space is overcome by constructing a reduced base model [30]. This model retains only the necessary information for a desired output measure.

2.4 Related Work on Severity and Coincident Failures

When a system in operation does not deliver its intended functionality and quality, it is said to *fail*. A failure is an observed departure of the external result of operation from requirements or user expectations [31]. Failures can be caused by hardware or software faults (defects), or by how-to-use errors.

2.4.1 Severity of Failures

Severity of a failure is the impact it has on the operation of a system. Severity is usually closely related to the threat the problem poses in functional (service) terms, economic (cost) terms, or in case of critical failures, to human life. This is related to the notion of hazard, which defines what undesirable consequence will potentially result from the incorrect system operation. An example of a service impact classification is: critical, major and minor failure. Severity of failures is sometimes used to partition the operational failure data, and thus make decisions regarding failures of a particular severity, or to weight the data used in reliability and availability calculations [1].

Severity of failures has been studied in the context of *gracefully degrading systems* [32]. In contrast to ultra-reliable systems, which usually achieve a high level of performance by masking out failures or by switching in spares to replace failed resources, gracefully degrading systems are designed to provide a high level of service by

reconfiguring the system and/or reallocating resources when a failure occurs. The paper developed models based on Markov processes for modeling severity (as well as workload) for a multiprocessor system.

Degraded modes of operation have been handled through the concept of a reward function associated with the Markov process in [33]. A portion of an air traffic control system (a set of radars) was modeled; as the number of failed radars increased, the reward (airspace surveillance coverage) decreased.

Predicting the reliability/availability based on the characteristics of a model of the system provides more objective and concrete information that can be used in assessing the risk tradeoffs and integrity levels. Appendix D provides a discussion on risk classification and safety integrity levels that are used to classify failures into different levels of severity (tolerable/intolerable/acceptable) based on quantitative or qualitative methods and the kind of demand of operation.

Clearly, severity is an important candidate to weight the data used in reliability calculations and must be incorporated into the model to determine the probability that the system survives, including efficient or acceptable degraded operation.

2.4.2 Coincident Failures

Components generally interact with each other during operation, and a faulty component can affect the probability of failure of other components too [5]. Such coincident/correlated failures should be modeled in order to get a realistic picture of system reliability.

2.4.2.1 Dependencies within a System

In real systems, there are several kinds of dependencies. Some of these are [34]:

- *Repair dependence*. Two or more components or subsystems may share a repair person.
- *State-dependent failure rates*. It is possible that the failure rate of a component may depend on the past history of the system. For example, it is possible that the repair of a component does not restore it to its original state. In that case, the component may have a larger failure rate after it has been repaired.
- *Near-coincident fault dependence*. The design of a system may be such that near-coincident faults cause a system failure, while faults that are separated in time can be handled individually without overall system failure.

These dependencies/interactions result, for example, from components communicating for functional purposes, or from the structure of the system, mainly the distribution of the software components onto the hardware components, or from fault tolerance and maintenance strategies. They induce dependencies between at least two components that are usually stochastic in nature. As a result, system dependability (including reliability) cannot be obtained by combining the dependability of its components. An overall model accounting for these dependencies is thus needed [35].

2.4.2.2 Modeling Correlation between Failures

Several researchers have considered the problem of modeling correlation between failures. Two schools of thought emerged, differentiated by the definition of the basic

events of interest, the two approaches are called *Correlated failures* and *Differentiated causes* [36].

The correlated failures approach was first considered by Eckhardt and Lee [37], and their work was later extended by Littlewood and Miller [38]. For the case of two simultaneous failures, the correlated failures model considers that there are two (possibly correlated) basic events that are not independent and proposes a modeling framework to account for the correlation between events. Nicolas and Goyal proposed the use of the Beta-binomial distribution for modeling correlation within this same framework [39].

The differentiated causes approach was first proposed by Arlat, Kanoun and Laprie [40] and later adopted by others. This approach differentiates between unrelated and related faults. For the case of two simultaneous failures, the differentiated causes model considers that there are three independent basic events.

The correlated causes model provides a good fit to data sets, there are only two parameters to be considered, and it reduces the amount of simulated execution associated with the experiments. On the other hand, the differentiated causes model maintains the statistical independence of the basic events, allowing the use of readily available tools for analysis.

2.4.2.3 Limitations in Modeling Coincident Failures

The common approach to modeling systems that possess some kind of dependence is to use a global Markov model. The Markov model of coincident failures in a DEC-VAX cluster multi-computer system has been developed in [41]. The main problem in this approach is the state explosion. As stated in Section 2.2.4, the size of a Markov model for

the evaluation of a complex system grows exponentially with the number of components in the system, and developing a *global* Markov model is particularly tedious.

Given the limitations imposed by non-independence, it is important to develop a reliability model that accounts for coincident errors. Two possibilities exist: either the model includes all possible terms including those that cannot be measured within feasible amounts of time, or the model includes only those parameters which can be measured within feasible amounts of time. It has been stated that the development of a coincident error model which can be used to estimate system reliability (for an ultra-reliable system) within feasible amounts of time is not possible [42].

2.5 Related Work on Usage-Profiles

A software-based product's reliability depends on just how a customer will use it. The operational profile – quantitative characterization of how a system will be used – is essential in *software* reliability engineering [6]. The same basic concept can be extended and applied for predicting the *system* reliability. The idea of operational profiles – considering the use of a software system during testing; is extended into usage profiles – the usage of the system (hardware and software) for modeling and reliability analysis.

2.5.1 Usage-Profiles and Performability

It has long been recognized that the workload of a system can influence its *performance*. There is also growing recognition that workload can affect system *dependability*. In many applications, users interact with a system in an intermittent fashion, resulting in operational workload profiles that alternate between periods of "Active" and "Passive" use. Moreover, a user's specification of desired service often refers exclusively to the behavior experienced while use is active. Assuming this premise,

a system's behavior during passive periods, particularly how its behavior may be affected (i.e. altered with respect to its expected/required behavior) by design defects[4] and/or operational faults[5], has no direct effect on the quality of the desired service. Accordingly, the extent to which "usage" is intermittent can affect user-oriented measures of service quality such as time-to-failure and reliability [7].

Reliability is concerned with the service that is actually delivered by the system as opposed to a system's *capacity* to deliver such service. Specifically, while considering usage profiles, faults need not necessarily cause failures since they can be repaired; failures occurring during "active" use of the system *only* should contribute to reliability calculations.

2.5.2 Modeling Usage-Profiles or Workload

Accounting for computational demand in analyzing a system's performance requires that some type of analytical model be used to represent the system workload, or the demand for system resources. Representing a system's workload can be very difficult; approaches to this problem range from assuming deterministic system inputs to generating system inputs from specific probability distributions. A workload model is a model which combines both system structure and demand [32].

2.5.2.1 Experimental Investigations

Some investigations have been mainly experimental using empirical data from measurements of real systems to correlate workload with various measures of

[4] Design defects, whether in hardware or software, are those caused by improper translation of a concept into an operational realization. Note: hardware unlike software is subject to wear out (mechanical/physical processes that cause the useful lifetime of a hardware component to end).
[5] Operational faults, whether in hardware of software, result from failure of components, physical interference from the environment and/or operator error.

dependability, e.g. Hsueh *et al.* [43] developed a semi-Markov model to describe the resource-usage/error/recovery process in a large mainframe system. They developed a state-transition model to describe the variation in system activity characterized by measuring a number of resource usage parameters. The separate workload, error and recovery models developed were then combined into a single model. Their results, from measurements and real data, indicate that it is important to consider the resource-usage as much as error rates while analyzing performability of the system. Further, the results were validated against direct calculations from the actual data, providing support for the model structure identification method employed in the research.

Castillo and Siewiorek [44] developed a new modeling technology to characterize failure processes in Time-Sharing systems due to hardware transients and software errors, recognizing workload-fault interaction for operational faults as well as design faults. In this work, it is clear that there is a reinforcement effect between workload and lack of reliability. Higher workload implies that the Kernel of the operating system has to take more decisions per unit time, increasing the probability of a system failure.

2.5.2.2 Analytic Investigations

On the analytic side, probabilistic models have been used to obtain workload-related dependability measures. In [45], Markov renewal processes were employed to analyze the interplay between workload and system fault tolerance mechanisms under the assumption of instantaneous processing times. Of particular importance was their analysis of the workload influence on mean time to failure, and the fact that time to failure can usually be approximated by an exponential random variable. Further, they

concluded that the kind of models developed to analyze the influence of workload on performability could be extended to degradable systems.

Gay [32] developed models based on Markov processes for modeling severity and workload for a multiprocessor system. The workload model combined both system structure and demand, the general idea being that failures primarily affect system structure and cause the system to operate at reduced performance levels, while the demand on the system determines to what degree the reduced performance is acceptable. The research demonstrated how capacity and workload models could be used to evaluate the performance of gracefully degrading systems.

Malhis *et al.* [46] illustrated a method for determining the performability of group-oriented multicast protocols, specifically Psync, using Stochastic Activity Networks, under a wide variety of workload and message loss probabilities. Their analysis showed that the protocol works well when message transmissions are frequent, but exhibits extremely long message stabilization times when transmissions are infrequent and message losses occur. The work presents useful information regarding performability under a wide range of workloads, and the appropriateness of SANs for analytically predicting the performance of group-oriented multicast protocols.

Qureshi and Sanders [47] analytically investigated the effect of workload on the performance and availability of Voting Algorithms. They used Stochastic Activity Networks to model and analyze a networked LAN environment utilizing particular static and dynamic voting algorithms. Their work stated that the effect of workload could be significant, since failures of system components are not important unless they are needed to deliver a service.

The importance of considering workload has also been recognized in [48] where a methodology for evaluating fault-tolerant systems was presented assuming workloads and fault arrivals to be non time-homogeneous; in [49] where the effects of shared use on the dependability of modular software was evaluated in terms of a generally defined stochastic model; among several others. Recently, a new metric has been designed for predicting the performance of an application under a growing workload [50].

The studies cited above demonstrate that workload should indeed be accounted for in the context of dependability evaluation. They also indicate that such evaluations are generally more difficult than those involving traditional structure-oriented measures.

This research contributes by incorporating the concept of failure severity, coincident failures and usage-profiles into the model developed for the Anti-lock Braking System of a passenger vehicle. These characteristics have never been modeled together for this system, generating a potentially more realistic model (with real data being used to model failure rates). The strategy of modeling failure severity as spatial uncertainty, coincident failures as correlation between the failure rates of components and the way usage-profiles have been incorporated are all innovative in terms of the approach employed to integrate them into the Stochastic Petri Net and Stochastic Activity Network formalisms (Chapter 4). Further, this study establishes the degree of complexity and the level of abstraction that is feasible to model and solve utilizing the available resources.

The contribution of this research to the automotive industry is substantial as it offers a greater insight into the strategy for developing realistic models, and acts as a stepping-stone for modeling more complex systems and carrying out further analyses (Chapter 6).

CHAPTER THREE

AN EXAMPLE EMBEDDED SYSTEM

And she tried to fancy what the flame of a candle is like after the candle is blown out, for she could not remember ever having seen such a thing.

- *Alice in Wonderland*

3.1 Basic Overview

The embedded system considered for stochastic modeling and reliability analysis is the Anti-lock Braking System (ABS), an integrated part of the total braking system in a vehicle, which avoids locking of tires when brakes are applied and maintains the driver's ability to steer. The various components in the ABS are also shared by two other sub-systems: the Automatic Slip Reduction (ASR) sub-system and the Electronic Steering Assistance (ESA) sub-system. Figure 6 depicts a state transition diagram of the system.

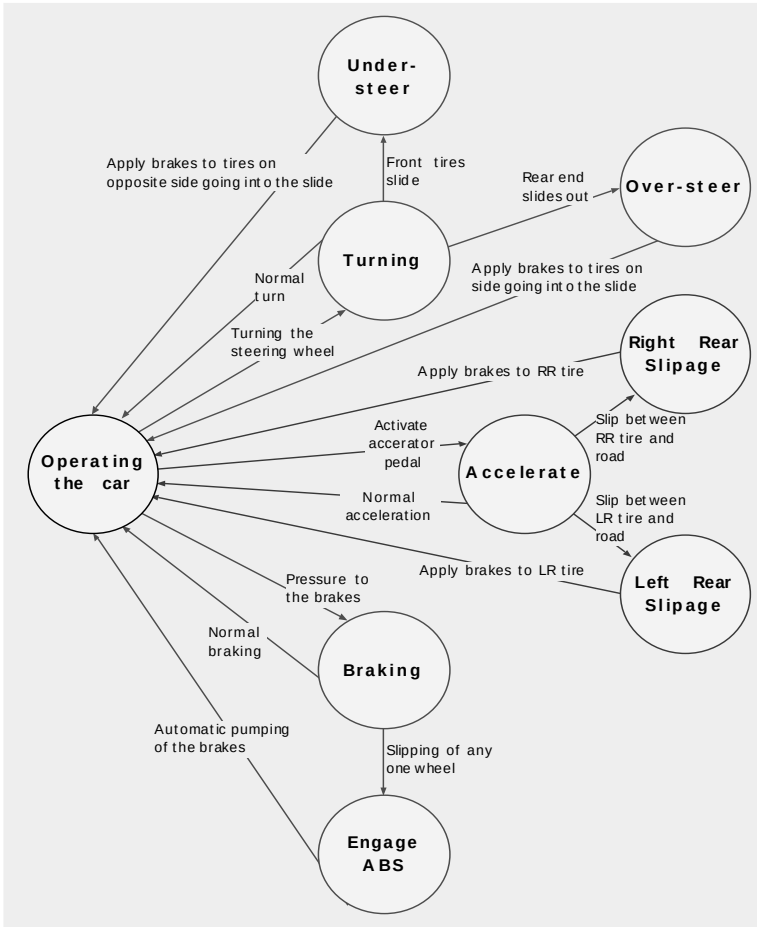


Figure 6: Top-level State Transition Diagram

3.2 The Anti-lock Braking System Description

Anti-lock Braking System (ABS) is an integrated part of the total braking system in a vehicle. Applying excessive pressure on the brake pedal, or panic slamming the brake pedal, can cause wheels to lock up and possibly send the vehicle careening into a terrifying skid. Excessive brake pedal pressure often occurs in an emergency or adverse situation, such as wet or icy roads [51]. The ABS prevents wheel lockup during an emergency stop by modulating the brake pressure and permits the driver to maintain steering control while braking. Figure 7 shows a logical view of the system operation, based on the SADT (Structured Analysis and Design Technique). It indicates a choice by a circle in the upper right hand corner of the box that describes an alternate activity. All boxes on the same level indicate a sequence of activities starting with the leftmost box.

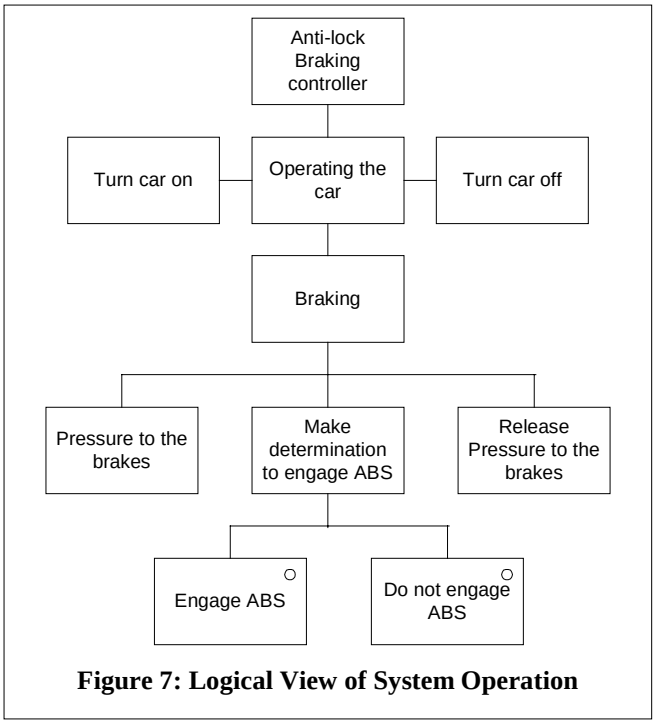


Figure 7: Logical View of System Operation

3.2.1 Components of the ABS

The ABS prevents the wheels of a vehicle from locking up. This is achieved by a control unit that reduces and increases pressure on the brake cylinders based on the measured rotational speeds of the wheels through appropriate actions of valves and pumps. Besides the hydraulic system, it comprises a subsystem for sensing the wheel speeds and transmitting the respective signals to the control unit [52].

The ABS consists of the following major components [53].

- Wheel Speed Sensors - These measure wheel-speed and transmit information to an electronic control unit.
- Electronic Control Unit (Controller) - This receives information from the sensors, determines when a wheel is about to lock up and controls the hydraulic control unit.
- Hydraulic Control Unit (Hydraulic Pump) - This controls the pressure in the brake lines of the vehicle.
- Valves - Valves are present in the brake line of each brake and are controlled by the hydraulic control unit to regulate the pressure in the brake lines.

Figure 8 displays the top-level schematic of the system showing the interconnections

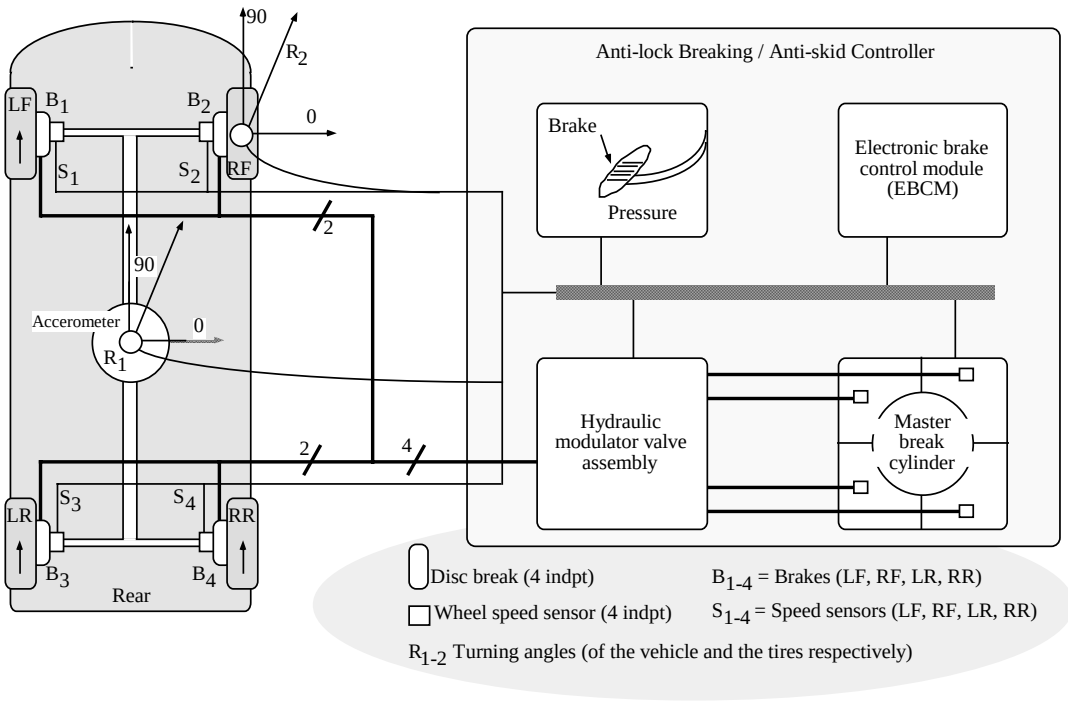


Figure 8: Top-level Schematic showing Sensors, Processing and Actuators

between the components. Anti-lock Braking systems use three different schemes depending on the type of brake in use [54]. (1) Four channel, four sensor ABS – There is a speed sensor on all four wheels and a separate valve for all four wheels; (2) Three channel, three sensor ABS – There is a speed sensor and a valve for each of the front wheels with one speed sensor and valve for both rear wheels; (3) Two channel, two sensor ABS – There are two speed sensors and valves for each of the two rear wheels.

3.2.2 Functioning of the ABS

When a driver applies brakes on an ABS-equipped vehicle, wheel sensors monitor the rotational speed of each wheel. The electronic control unit (ECU) "reads" signals from the sensors and compares the speed of each wheel. If one wheel is slowing at a faster rate than the others, the ECU sees that the wheel is beginning to lock up. The ECU then orders the hydraulic control unit (HCU) to reduce the line pressure to that wheel's brakes. The HCU reduces the pressure in that particular brake line by controlling the valves present there. Once the wheel resumes normal operation, the controls restore pressure to its brake. Depending on the system, this "pulsing" of brake line pressure can occur at up to 15 times per second. The result is that the tire slows at the same rate as the vehicle, with the brakes keeping the tires very near the point at which they will start to lock up. This gives the

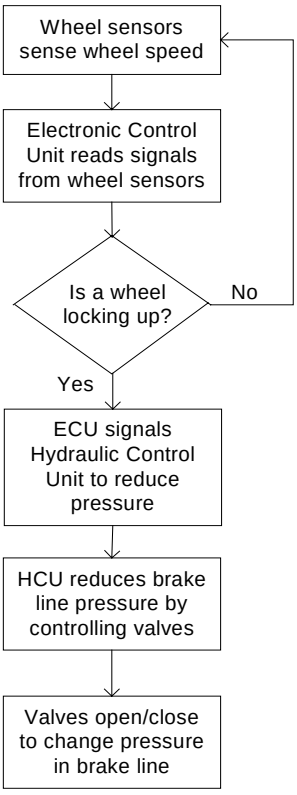


Figure 9: Control flow in ABS Functioning

system the highest steering capability. This flow of control in the ABS functioning is shown in Figure 9.

3.2.3 Component Failure Rates

The basic modeling philosophy begins by identifying the essential components of the system (Section 3.2.1) and the different ways in which they interact. The data collected from system measurements are used to parameterize the abstract model. System measurements can help in the process of deciding which components of the system are important in regards to the measure of interest [12]. Table 1 provides a list of the parts that are considered in this analysis, along with the respective failure rates[6] associated with critical failure states. The number of components is for a four channel four sensor ABS scheme.

Table 1: Component Failure Rates associated with critical failure states

Component	Number	Failure Rate
Wheel Speed Sensor	4	2.00E-11
Pressure Sensor	4	1.50E-11
Main Brake Cylinder	1	1.00E-11
Pressure Limiting Valve	2	6.00E-13
Inlet Valve	4	6.00E-13
Drain Valve	4	6.00E-13
Toggle Switching Valve	2	6.00E-13
Hydraulic Pump	2	6.80E-11
Pressure Tank	2	2.00E-12
Controller	1	6.00E-12
Tubing	1	3.00E-12
Piping	1	4.00E-12

[6] The data was obtained from DaimlerChysler. The failure rates listed in Table 1 however are dummy values. The real values we had are protected under a non-disclosure agreement. The same is true for the data shown in Table 2.

3.3 System Assumptions

Since the system here is very complex, this prevents us from making a direct analysis. A series of abstraction steps are needed to obtain system measures from the real system. Initially the system model is created at an abstract level and the data collected from system measurements (as shown in Table 1) are used to parameterize the abstract model. In the second abstraction step the computational model is created which allows an easier and more efficient system analysis [12]. The key element therefore in our modeling approach was to identify the essential components of the system, the different ways in which they interact and introduce various assumptions. The details of the assumptions made are discussed here.

3.3.1 Modes of Operation

As stated in Section 2.4.1, severity of a failure is the impact it has on the operation of a system and is usually closely related to the threat the problem poses in functional terms. For the purpose of this discussion, the three different modes of operation of the system (in presence/absence of failures of different severity) are assumed to be: (1) *normal operation*, (2) *degraded operation*, and (3) *lost stability mode*; in increasing order of severity. Critical failures seriously impact the operation of the system, and are assumed to cause *loss of vehicle*. Further, if sufficient components of the system have failed to impact the system operation (either degraded operation or lost stability mode), the sum of those failures is assumed to be critical, causing loss of vehicle.

3.3.2 Lifetime of a Passenger Vehicle

Essentially the average hours of operation for a passenger vehicle range from 300-600 hours/year and the average lifetime is 10-15 years. Thus, the average life span of a

passenger vehicle ranges from 3000 – 9000 hours. This estimate is important while considering the duration for which to carry out the reliability analysis.

3.3.3 Inter-relationships between Components

To model coincident failures, several dependencies among system components is assumed. Only those inter-relationships between components depicted as solid lines in Figure 10 are explicitly modeled in the stochastic models. All other possible inter-relationships between components (only some of them depicted as dashed lines in the figure) have been ignored.

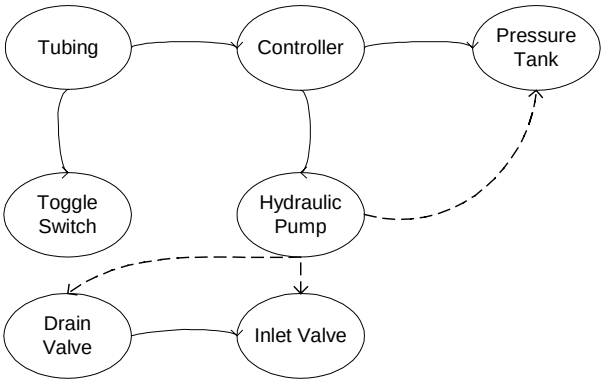


Figure 10: Inter-dependencies between Components

Further, for modeling purposes, we assume a four channel four sensor ABS [54]. The model can be easily modified to represent other ABS schemes.

CHAPTER FOUR

STOCHASTIC MODELING FORMALISMS

`Do you mean that you think you can find out the answer to it?' said the March Hare. `Exactly so,' said Alice.

- Alice in Wonderland

4.1 Stochastic Petri Net Models

In this section, the Stochastic Petri Net (SPN) models developed to model severity of failures and coincident failures for the ABS, as well as the SPN models developed to model usage-profiles for the ABS are presented. The extensibility of the models developed is also discussed.

4.1.1 Modeling Coincident Failures and Severity

The SPNs were input to the Stochastic Petri Net Package tool in CSPL (C-based Stochastic Petri net Language). Here, the models are discussed in Petri Net form for clarity. Code is presented for explanation wherever necessary. The entire code listing and PNs modeling coincident failures and severity can be found in Appendix A.1.

4.1.1.1 Assumptions

To model complex systems, assumptions need to be made. All simplifying assumptions that were made for modeling coincident failures and severity of failures are discussed in this section.

4.1.1.1.1 Exponential Failure Rates

To allow a Markov chain analysis, the time to failure of all components is assumed to have an exponential distribution. This signifies that the distribution of the remaining life of a component does not depend on how long the component has been operating. The

component does not “age” or it forgets how long it has been operating, and its eventual breakdown is the result of some suddenly appearing failure, not of gradual deterioration [8].

How do things fail? Reasons for failures occurring in an embedded system (in the domain being considered) can be categorized into the following four classes: (1) Communication failure between distributed components, processing units, sensors and actuators, (2) Mechanical failures (including wear and tear) of components like the piping, brake cylinders and axles, (3) Timing errors attributed to software, and (4) Software design errors that lie dormant and manifest themselves when the right input triggers them.

Hardware components. The assumption of an exponential failure rate holds true for electronic components and has been widely accepted for modeling hardware failures. At the same time, the failure of other mechanical parts like valves might occur due to gradual deterioration. However, mechanical parts are generally replaced at regular intervals and essentially can be assumed not to age. Hence, the assumption of an exponential distribution of failures for all hardware components is justified.

Software components. Normally, software is considered to have a negative exponential failure curve. Repeated testing and fixing of errors in the software makes it more robust and the probability of a failure decreases with time. However, for the development phase being considered (beyond the testing phase), any residual software design error present after testing will remain in the product. The probability of any lurking defect being triggered by the right input increases with the passage of time.

Hence, the assumption of an exponential distribution of failures for all software components is justified.

4.1.1.1.2 Impact of Severity and Coincident Failures

To consider the severity of failures, every component is assumed to operate in three modes (See Section 3.3.1): normal operation, degraded operation or causing loss of stability. The system is assumed to *fail* when more than five components function in a *degraded state*, or more than three components cause *loss of stability*, or the failure of an important component causes the *loss of the vehicle*. A component operating in a degraded condition causes its failure rate to increase by *two* orders of magnitude, while a component causing loss of stability causes the failure rate to increase by *four* orders of magnitude. (See Appendix D for a discussion on severity integrity levels.)

The correlation between failure rates of two "related" components (to model coincident failures) is consistent with the above scheme. The correlated failures considered for the purpose of this study include only hardware components. Coincident failures are also important in software because errors can be propagated and it is difficult to predict how, when and where they will manifest. The actual behavior may not relate very well to the root cause. While the correlation in hardware failure is physical, the correlation in software errors is mainly informational. Thus, I have not attempted to model coincident failures in software in this research. However, any such correlation between software components can be modeled in an identical manner. The details of how failure severity and coincident failures are modeled are presented in the next section.

4.1.1.2 The SPN Model

To model the inter-dependence in the Anti-lock Braking System it is important to use a global Markov model. The ABS is represented as a combination of all the important components it consists of, as shown in Figure 11 (The actual Petri Net Model is shown in Appendix A.1). The components are sorted into two groups: *central* and *axle*. The components under *axle* are further segregated according to the corresponding *wheel* – FRWheel (Front Right Wheel), FLWheel (Front Left Wheel), RRWheel (Rear Right Wheel) and RLWheel (Rear Left Wheel). This division into groups is representative of the number of a given component present in the system. A component like the Wheel Speed Sensor, one for each wheel, finds its place under each of the four wheel categories (FRWheel, FLWheel, RRWheel and RLWheel). A component like the Hydraulic Pump, one for each axle, finds its place in the *axleCentral* group under the *axle* place. A component like the Main Brake Cylinder, of which there is only one instance, finds its place under the *central* category.

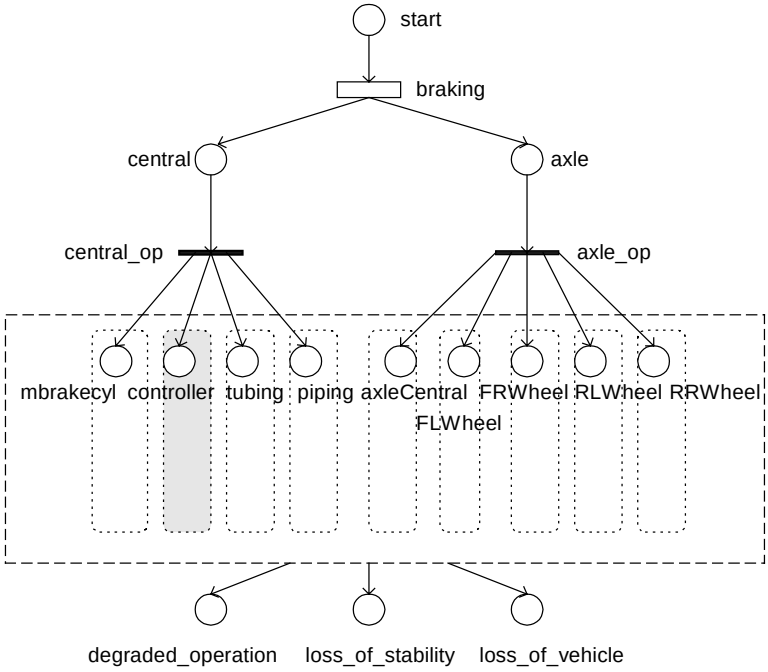


Figure 11: The SPN Model for ABS

The problem of large state space has been handled by avoiding the use of multiple places to denote multiple instances of the same component (where possible). Each component has its own model, shown as dashed rectangles in Figure 11. The model for the *controller* component (shaded rectangle) is depicted in Figure 12 and discussed in the next section.

4.1.1.2.1 Modeling Severity of Failures

The model shown in Figure 11 also depicts the operation of the ABS under normal, degraded and lost stability conditions. The places *degraded_operation*, *loss_of_stability* and *loss_of_vehicle* model the severity of failure. The system is functioning normally when there are no tokens in any of these three places. Loss of vehicle (indicated by a token in the *loss_of_vehicle* place), extreme loss of stability (indicated by three tokens in *loss_of_stability* place) or extreme degraded operation (indicated by five tokens in *degraded_operation* place) signify critical failures and determine the halting condition for the model. The model is instantiated with a single token in the *start* place. When the *central_op* and the *axle_op* transitions fire, a token is deposited in each place that represents a component of the ABS. The operation of each component is now independent of every other component (except where coincident failures are modeled

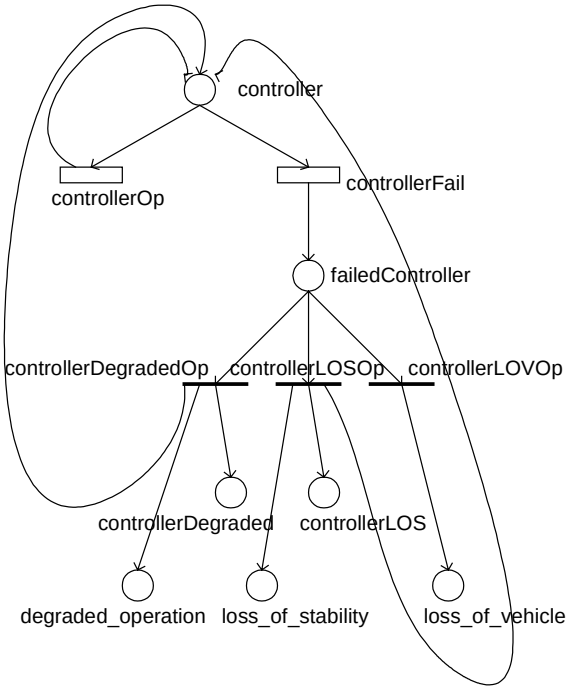


Figure 12: The SPN Model With Coincident Failures and Severity

explicitly e.g., the coincidence between the *controller* and the *tubing* is modeled explicitly while defining the failure rate for controller. See Section 4.1.1.2.2 for details).

The model of a component of the ABS is shown in Figure 12. The component depicted here is the controller. Every component either functions "normally" as shown by the *controllerOp* transition or "fails" as shown by the *controllerFail* transition. A failed component may either cause degraded operation, loss of stability or loss of vehicle (as represented by the *controllerDegradedOp*, *controllerLOSOp* and *controllerLOV* immediate transitions respectively). The probability of any one of these three transitions occurring (obtained from measures on the real system) is different for each component. Table 2 lists the failure rates and probabilities of failure of different severities for all components (the data is protected under a non-disclosure agreement and has been falsified). When the failure causes either degraded operation or loss of stability, the component continues to operate (token recycled back to the *controller* place), though the failure rate increases by two and four orders of magnitude respectively.

Table 2: Probability of failures of different severity

Component	#	Base Failure Rate	Probability		
			Degraded operation	Loss of Stability	Loss of Vehicle
Wheel Speed Sensor	4	2.00E-11	0.38	0.62	-
Pressure Sensor	4	1.50E-11	0.64	0.36	-
Main Brake Cylinder	1	1.00E-11	-	-	1.0
Pressure Limiting Valve	2	6.00E-13	-	0.22	0.78
Inlet Valve	4	6.00E-13	-	0.18	0.82
Drain Valve	4	6.00E-13	-	0.19	0.81
Toggle Switching Valve	2	6.00E-13	1.0	-	-
Hydraulic Pump	2	6.80E-11	-	-	1.0
Pressure Tank	2	2.00E-12	-	-	1.0
Controller	1	6.00E-12	0.2	0.4	0.4
Tubing	1	3.00E-12	0.33	-	0.67
Piping	1	4.00E-12	0.33	-	0.67

4.1.1.2.2 Modeling Coincident Failures

To model coincident failures, several dependencies among system components are assumed, as shown in Figure 10 (Section 3.3.3). Coincident failures are modeled in a manner similar to severity of failures. "Coincidence" of failures of two components is modeled by causing the failure of one component (to degraded operation or loss of stability) to increase the failure rate of the dependent component. The rule for calculating failure rates is shown in Figure 13. The failure of a component A to a degraded mode causes the failure rate of a "related" component B to increase by two orders of magnitude. The failure of component A to a lost stability mode causes the failure rate of a "related" component B to increase by four orders of magnitude. (There was no data available to confirm or validate this assumption for modeling coincident failures.)

```
function failureRateForB()
{
      // other calculations for severity of failure

      // coincident failures
      if failed(A)(degraded) then
            failureB = failureB * 100;
      else if failed(A)(loss of stability) then
            failureB = failureB * 10000;
}
```

Figure 13: Rule for Calculating Failure Rates

SPNP provides a function **void ratefun(char* t, double (*_func_()));** that defines the firing rate of transition t to be the value of marking-dependent function *func*, evaluated in the current marking.

```
double controllerRate()
{
      double controller_rate = 0.0000006;

      if (mark("controllerLOS") > 0) return controller_rate * 10000;
      if ((mark("controllerDegraded") > 0) || (mark("tubingDegraded") > 0))
            return controller_rate * 100;
      return controller_rate;
}
```

Figure 14: Variable Rate to Model Coincident Failures

The function that calculates the failure rate of the transition *controllerFail* is shown in Figure 14. It is assumed that a tubing malfunction affects the

operation of the controller. Hence, while calculating the failure rate of the controller, the normal rate is increased by two orders of magnitude if the tubing has failed causing degraded operation (indicated by a token in the *tubingDegraded* place).

Only a few coincident failures have been represented in the model. However, coincident failures between other components (or among more than two components) can be easily modeled by suitably modifying the failure rate function of the relevant components using the rule shown in Figure 13. The importance of this rule in the context of extensibility of the developed model is discussed in Section 4.1.4.

4.1.2 Modeling Usage-profiles

The Stochastic Petri Nets were encoded using CSPL (C-based Stochastic Petri net Language) and subsequently analyzed using the Stochastic Petri Net Package (SPNP) tool. Here, the models are discussed in Petri net form for clarity. Code is presented for explanation wherever necessary.

4.1.2.1 Assumptions

To model complex systems, assumptions need to be made. All simplifying assumptions that were made for modeling usage-profiles are discussed in this section.

4.1.2.1.1 Infinite Repair Rate

Unlike traditional reliability models where repair of components is not considered, when considering intermittent use it is important to note that faults need not necessarily cause failures. Faults occurring only during the active use cause failures while those occurring during passive use can be repaired. Hence, repair can affect reliability calculations. For simplicity, we assume an infinite repair rate of all components, implying that all repairs, if any, occur instantaneously [7].

4.1.2.1.2 Usage Profiles: Low Usage and High Usage

To comprehend the significance of intermittent use on reliability, we assume two usage-profiles exceedingly different in degree. The first profile (Low Usage) models sparse use of the Anti-lock Braking System e.g., a driver who is extremely cautious while driving the vehicle (longer periods of passive use). The second usage profile (High Usage) models dense use of the anti-lock braking system e.g., a driver in perilous conditions like driving over ice (frequent active use periods). The second usage-profile is assumed to have a rate two orders of magnitude greater than the first usage profile.

4.1.2.1.3 Exponentially Distributed Workload and Failure Rates

For simplicity and to allow Markovian analysis, the active period is assumed to be exponentially distributed, as are the failure rates of the components. To work around the stiffness problem in Petri nets caused by the difference in magnitude between the failure rates of the components and the active period duration distribution rates, the duration distribution rates are assumed to be factored by the failure rates of individual components. The details of how this is accomplished are the topic of the next section.

4.1.2.2 The SPN Model

The global model of the ABS, represented as a combination of all the important components it consists of, remains unchanged (Figure 11**)**. To incorporate the usage-profiles in the ABS model, the model of each individual component (like the *controller* depicted in Figure 12) is extended as shown in Figure 15. The figure shows the *controller*, with the bold lines indicating the additions to the model. In case of a failure (*failedController*), the model differentiates between the two situations regarding whether the system was in active use (along the branch to transition labeled *mu*) or not (along the

branch to transition labeled *alpha*). The parameter 1/*mu* indicates the mean duration of active use while the parameter 1/*alpha* indicates the mean duration of passive use. As stated earlier, the active period is assumed to be exponentially distributed.

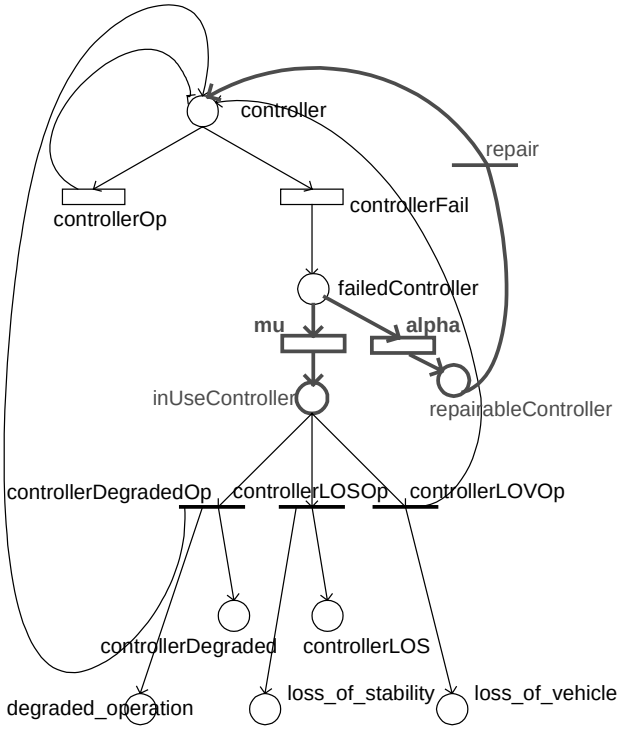


Figure 15: SPN Model with Usage-Parameters

In the case where the failure occurs during the active period (*inUseController*), the system either continues to operate in the degraded mode (*controllerDegradedOp*), or lost stability mode (*controllerLOSOp*), or causes loss of vehicle (*controllerLOVOp*). In the case where the failure occurs during passive use of the system (*repairableController*), the fault can be repaired and an infinite repair rate is assumed. The system continues to operate as if no failure had occurred. The model can be extended to associate a cost with each time the failed component must be repaired, if required.

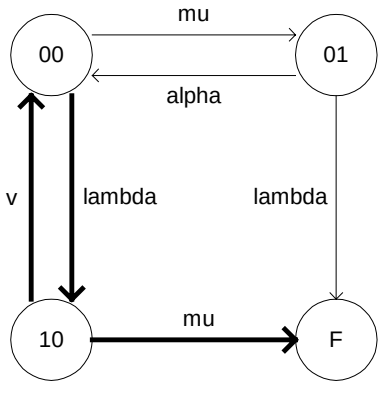


Figure 16: State Diagram for Reliability Evaluation

Generally, each component's model is updated to match the **bold** arrows in the state diagram shown in Figure 16 [7] (adapted with permission from original author). Prior to

failure, each component is assumed to be in one of the three states: fault-free and passive (00), fault-free and active (01), and fault but passive (10). The parameter *mu* indicates transition to active state, the parameter *alpha* indicates transition to passive phase, while the parameter *lambda* indicates a failure and the parameter *v* indicates repair.

To work around the state explosion problem occurring due the evident increase in the number of states in the model (as shown in Figure 15), it is simplified to incorporate the usage parameters while calculating the failure rate itself for each component. The modified function for calculating the failure rate in light of the usage-profile is shown in Figure 17. Essentially, the failure rate (considering only usage-parameters) is the sum of the failure rates *mu* and *lambda*. As stated in the assumptions, the value of these usage parameters was factored by the actual failure rate of the component to avoid stiffness in the model. The value of *mu* is assumed to be 2.5 for infrequent active use periods (low-usage) and 250 for frequent active use periods (high-usage). If more information about usage becomes available, then it is easy to revise the values of *mu* to make the model more realistic.

```
double controllerRate()
{
        double controller_rate = 0.0000006;

        // usage parameter
        controller_rate += controller_rate * mu();

        if (mark("controllerLOS") > 0) return controller_rate * 10000;
        if ((mark("controllerDegraded") > 0) || (mark("tubingDegraded") > 0))
                return controller_rate * 100;
        return controller_rate;
}
```

Figure 17: Variable Rate to Model Usage Parameter

4.1.3 Specifying Reliability Measures and Halting Condition

For a Stochastic Reward Net (SRN), all output measures are expressed in terms of the expected values of reward rate functions [19]. Depending on the quantity of interest, an

appropriate reward rate is defined. To study the reliability of the system at time t, it is sufficient to define a single set of 0/1 reward rates. Figure 18 shows the function used to compute the reward rate for determining the reliability of the system, for each of the SPN models described in the above two sections (4.1.1 and 4.1.2).

```
double reliab()
{
 double reward;
 if((mark("loss_of_vehicle") >= 1) ||
            (mark("loss_of_stability") >= 3) ||
                        (mark("degraded_operation") >= 5))
       reward = 0;
 else
       reward = 1;
 return reward;
}
```

Figure 18: Function to Calculate Reliability Reward

This function is used as an input argument to the function **void pr_expected(char* string, double (****func***)());** provided by SPNP that computes the expected value of the measure returned by the function *func*. The expected value of the reliability at different instances of time is calculated by making a call to the function **void solve(double t));** to solve the Markov chain at time t before making a call to function pr_expected().

Since, the SPN models described above recycle tokens when the system is either operating in normal mode or degraded mode, it is necessary to explicitly impose a halting condition to indicate an absorbing state. The function **void halting_condition(int (****gfunc***)());** defines the halting condition *gfunc* for the SPN. When this function evaluates to zero, the marking is considered to be absorbing.

```
int halt()
{
    if((mark("loss_of_vehicle") >= 1) ||
          (mark("loss_of_stability") >= 3) ||
             (mark("degraded_operation") >= 5))
        return 0;
    else
        return 1;
 }
```

Figure 19: Function to Evaluate for Halting Condition

As stated before, the system is assumed to *fail* (absorbing state) when more than five components function in a *degraded state*, or more than three components cause *loss of stability*, or the

failure of an important component causes the *loss of the vehicle*. Figure 19 shows the function used to evaluate the present marking for the halting condition.

4.1.4 Extensibility of the SPN Model

The SPN models developed for modeling coincident failures and severity (described in Section 4.1.1) and usage-profiles (described in Section 4.1.2) are easily extensible. The global SPN Model can be extended to include other components deemed relevant to the ABS by including their corresponding sub-models. The sub-models, in turn, would be simple reproductions of the sub-models for other components with different failure rates and probabilities. The model, developed for the four channel four sensor ABS, can be adapted to model other schemes of the ABS, by suitably changing the numbers of the relevant components modeled (by either removing/adding the respective place, or updating the failure rate).

Inter-dependencies between other components (or among more than two components) culminating in coincident-failures can be modeled by updating the failure rates of the relevant components using the rule for calculating the failure rates shown in Figure 13. Different categorizations for severity of failure can be used by simply updating the sub-models of the components to include the necessary places depicting the severity level (replacing the *degraded_operation*, *loss_of_stability* and *loss_of_vehicle* places). The SPN model representing usage profiles can be updated to represent different usage-parameters or intensity of workload by simply changing the value of *mu*. The model can be extended to associate a *cost* with each time the failed component must be repaired, by adding an additional place to keep track of the number of times the component has been repaired denoted by the number of tokens in this place. Cost is important in the

commercial world where optimizing the cost for production runs of 200K-500K units can have significant savings impact.

However, since the model is an abstraction of a real world problem, predictions based on the model must be validated against actual measurements collected from the real phenomena. A poor validation may suggest modifications to the original model [8]. The results from the analyses of each of the above-discussed models using SPNP (Stochastic Petri Net Package) version 6 are presented in Section 5.1.

4.2 Stochastic Activity Network models

In this section, the Stochastic Activity Network (SAN) models developed to model severity of failures and coincident failures for the ABS, as well as the SAN models developed to model usage-profiles for the ABS are presented. The extensibility of the models developed is also discussed.

4.2.1 Modeling Coincident Failures and Severity

The SANs were input to the UltraSAN tool graphically. Here, the composed model and the individual subnet *Central_2* are discussed. Code is presented for clarity, wherever necessary. The entire array of subnet models developed and the corresponding code snippets for modeling coincident failures and severity can be found in Appendix B.1.

4.2.1.1 Assumptions

To model complex systems, assumptions need to be made. All simplifying assumptions that were made for modeling coincident failures and severity of failures are discussed in this section.

4.2.1.1.1 Exponential Failure Rates

To allow a Markov chain analysis, the time to failure of all components is assumed to have an exponential distribution. This signifies that the distribution of the remaining life of a component does not depend on how long the component has been operating. The component does not "age" or it forgets how long it has been operating, and its eventual breakdown is the result of some suddenly appearing failure, not of gradual deterioration [8].

4.2.1.1.2 Impact of Severity and Coincident Failures

To consider the severity of failures, every component is assumed to operate in three modes: normal operation, degraded operation or causing loss of stability. The system is assumed to *fail* when more than five components function in a *degraded state*, or more than three components cause *loss of stability*, or the failure of an important component causes the *loss of the vehicle*. A component operating in a degraded condition causes its failure rate to increase by *two* orders of magnitude, while a component causing loss of stability causes the failure rate to increase by *four* orders of magnitude. The correlation between failure rates of two "related" components (to model coincident failures) is consistent with the above scheme. The details of how this is accomplished are the topic of the next section.

4.2.1.2 The SAN Model

As stated before, to model the inter-dependence in the Anti-lock Braking System it is important to use a global Markov model. The composed model for the ABS is shown in Figure 20. The model consists of three individual SAN sub models: *Central_1*, *Central_2* and *Wheel*. The *Wheel* subnet is replicated four times to model the four wheels

of the vehicle. The division into these three sub-categories is done to facilitate the representation of coincident failures. As depicted in Figure 10, the inlet valve and the drain valve are correlated, and so are the components listed under the group *Central_2*. All components under *Central_1* are assumed to be independent of each other (for the purpose of this study). Such a distribution/categorization avoids replicating of subnets where unnecessary (for modeling severity and coincident failures) and thereby prevents the potential state explosion problem.

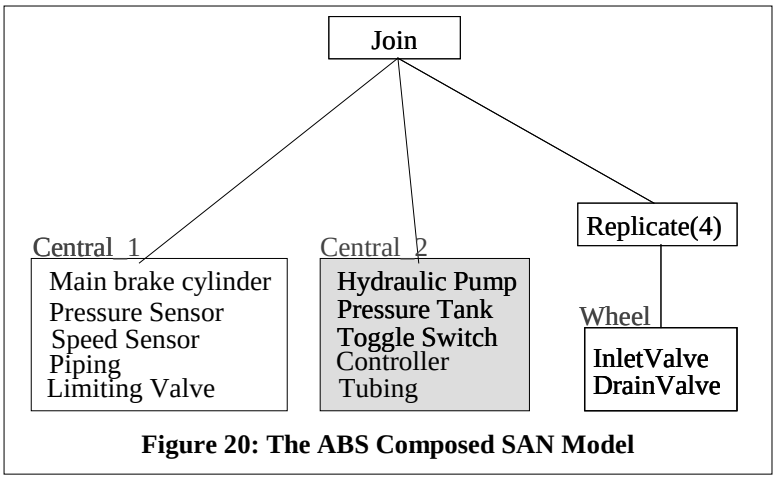


Figure 20: The ABS Composed SAN Model

4.2.1.2.1 Modeling Severity of Failures

All subnets when combined to form the composed model share some common places: *degraded*, *LOS*, *LOV* and *halted*. The first three places model the severity of failure, while the *halted* place is relevant in context of the halting condition and is discussed in Section 4.2.3. The *Central_2* subnet is shown in Figure 21. The presence of tokens in *degraded*, *LOS* and *LOV* models the system operation under degraded mode, loss of stability and loss of vehicle respectively (the same concept as in the SPN models). The system is operating normally when there are no tokens in any of these three places.

The subnet is instantiated with a single token in the *central_2* place. The *central2_op* activity fires and deposits a token in each of the five places: *hydraulicPump*, *pressureTank*, *toggleSwitch*, *controller* and *tubing*. The portion of the subnet for the

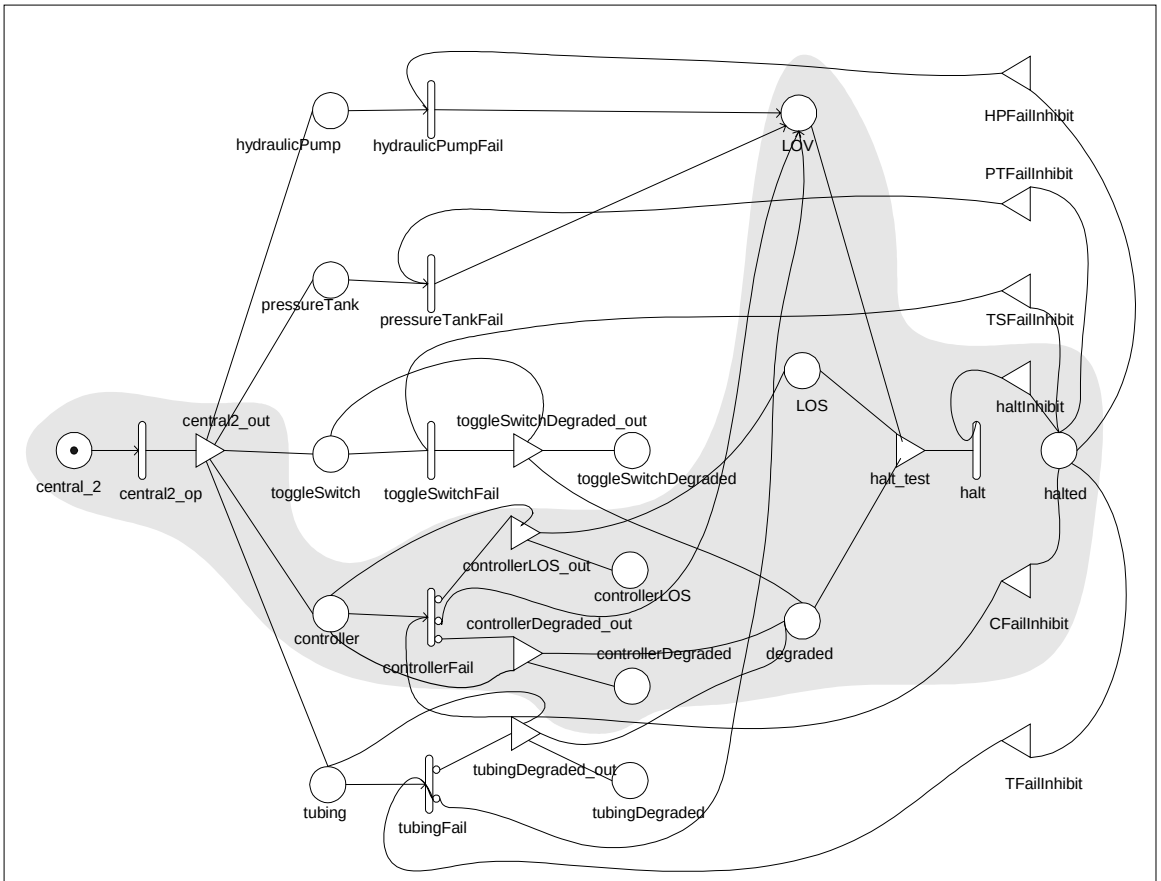


Figure 21: *Central_2* Subnet with the Controller Component Highlighted

controller component is highlighted in Figure 21 and discussed here in the context of severity of failures. The *controllerFail* activity models the failure of the controller. There are three possible outcomes of this activity. The *controller* either fails causing degraded operation (with probability 0.2, output gate *controllerDegraded_out*), or causes loss of stability (with probability 0.4, output gate *controllerLOS_out*), or causes loss of vehicle (with probability 0.4, output to *LOV*). In the former two cases the controller continues to operate in a degraded manner, as is evident by the recycling back of the token to the *controller* place. Further, the failure rate in this situation increases by two (for degraded) and four (for loss of stability) orders of magnitude respectively. The code snippet that achieves this is shown in Figure 22.

UltraSAN requires the failure rate to be specified in a single statement, hence the use of the special if-then-else construct available in the C programming language. Consider the *controllerFail* activity. If the controller fails causing degraded operation (i.e., MARK(controllerDegraded)!=0), it continues to function manifested by recycling the token back to the *controller* place, and the failure rate for the *controllerFail* activity increases by two orders of magnitude (i.e. controllerRate*100). Similarly, if the controller fails causing loss of stability (i.e., MARK(controllerLOS)!=0), it continues to function manifested by recycling the token back to the *controller* place, and the failure rate for the *controllerFail* activity increases by four orders of magnitude (i.e. controllerRate*10000).

Activity	Rate	Probability		
		Case1	Case2	Case3
controllerFail	MARK(controllerLOS) !=0? controllerRate*10000: (MARK(controllerDegraded) !=0 || MARK(tubingDegraded) !=0 ?controllerRate*100 :controllerRate)	0.4	0.4	0.2
hydraulicPump Fail	MARK(controllerLOS) !=0? hydraulicPumpRate*10000: (MARK(controllerDegraded) !=0 ?hydraulicPumpRate*100 :hydraulicPumpRate)	1.0	-	-

Figure 22: Activity Rates Model Severity and Coincident Failures

4.2.1.2.2 Modeling Coincident Failures

As in the SPN models, “Coincidence” of failures of two components is modeled by causing the failure of one component (to degraded operation or loss of stability) to increase the failure rate of the dependent component. The failure of a component A to a degraded mode causes the failure rate of a “related” component B to increase by two orders of magnitude. The failure of component A to a lost stability mode causes the

failure rate of a "related" component B to increase by four orders of magnitude. Figure 22 shows the rates for the activities modeling the failure of the controller and the hydraulic pump (other component failure rates as modeled in a similar manner). Case 1, 2 and 3 represent the probabilities of the failure causing loss of vehicle, loss of stability and degraded mode respectively.

Since UltraSAN requires the failure rate to be specified in a single statement, the special if-then-else construct available in the C programming language is used. Consider the *controllerFail* activity. Since a failed tubing (in degraded mode) is assumed to affect the failure rate of the controller, if the number of tokens in the *tubingDegraded* place is not zero (i.e., MARK(tubingDegraded)!=0), the failure rate for the controller increases by two orders of magnitude (i.e., controllerRate*100). Similarly, for the *hydraulicPumpFail* activity, I have assumed that a failed controller affects the failure rate of the hydraulic pump. Thus, the failure rate for the hydraulic pump increases by four orders of magnitude if the controller has failed causing loss of stability, and increases by two orders of magnitude if the controller is operating in a degraded mode.

Only a few coincident errors have been modeled (as shown in Figure 10). However, coincident failures between other components (or among more than two components) can be modeled in a similar fashion. The general programming construct for modeling coincident failures between two components A and B (where A is dependent on B) is

```
MARK(componentBLOS)!=0 ?
componentARate*10000 :
(MARK(componentBDegraded)!=0 ?
componentARate*100 : componentARate).

                Is Equivalent to:

if(MARK(componentBLOS)!=0)
        return componentARate*10000;
else if(MARK(componentBDegraded)!=0)
        return componentARate*100;
else    return componentARate;
```

Figure 23: Construct to Model Coincident Failures

shown in Figure 23. The translation to the simple C if-else statement is also presented for clarity.

4.2.2 Modeling Usage-profiles

The SANs were input to the UltraSAN tool graphically. Here, the composed model and the individual subnets are discussed. Code is presented for clarity, wherever necessary.

4.2.2.1 Assumptions

To model complex systems, assumptions need to be made. All simplifying assumptions that were made for modeling usage-profiles are discussed in this section.

4.2.2.1.1 Usage Profiles: Low Usage and High Usage

To comprehend the significance of intermittent use on reliability, we assume two usage-profiles exceedingly different in degree. The first profile (Low Usage) models sparse use of the Anti-lock Braking System e.g. a driver who is extremely cautious while driving the vehicle (longer periods of passive use). The second usage profile (High Usage) models dense use of the anti-lock braking system e.g. a driver in perilous conditions like driving over ice (frequent active use periods).

4.2.2.1.2 Exponentially Distributed Workload and Failure Rates

For simplicity and to allow Markovian analysis, the active period is assumed to be exponentially distributed, as are the failure rates of the components. The second usage-profile is assumed to have a rate *one* order of magnitude greater than the first usage profile. To work around the stiffness problem in Petri nets caused by the difference in magnitude between the failure rates of the components and the active period duration distribution rates, the duration distribution rates are assumed to be factored by the failure

rates of individual components. The details of how this is accomplished are the topic of the next section.

4.2.2.2 The SAN Model

The composed SAN model for the ABS remains unchanged while modeling usage-profiles. The individual component's model within each subnet needs to be updated to handle usage, as depicted in Figure 16 and discussed in section 4.1.2.

To work around the state explosion problem occurring due the evident increase in the number of states in the model (for a component in passive and active modes), the model is simplified to incorporate the usage parameters while calculating the failure rate itself for each component. The modified construct calculating the rate for each failure activity in light of the usage-profile is shown in Figure 24. The translation to the simple C if-else statement is also presented for clarity.

```
MARK(componentBLOS)!=0 ?
(componentARate+componentARate*mu)*10000 :
(MARK(componentBDegraded)!=0 ?
(componentARate+componentARate*mu)*100 :
(componentARate+componentARate*mu)).
```

Is Equivalent to:

```
if(MARK(componentBLOS)!=0)
   return (componentARate+componentARate*mu)*10000;
else if(MARK(componentBDegraded)!=0)
   return (componentARate+componentARate*mu)*100;
else       return (componentARate+componentARate*mu);
```

Figure 24: Construct to Model Usage-Profiles

The parameter 1/*mu* indicates the mean duration of active use of a given component. To calculate the failure rate of the component, the actual failure rate is added to the active usage rate (*mu* factored by the actual failure rate to avoid stiffness due to the evident difference in the orders of magnitude). The remaining constructs for severity and coincident failures remain unchanged. The value of *mu* is assumed to be 2.5 for infrequent active use periods (low-usage) and 25 for frequent active use periods (high-usage).

4.2.3 Specifying Reliability Measures and Halting Condition

The required reliability measure is defined as a reward rate function. While it is sufficient to define a single set of 0/1 reward rates to study the reliability of the system at time t (the way the reward rate was defined for the SPN models in Section 4.1.3), the reward rates for the SAN model are defined to take the degraded operation of the system into consideration.

In UltraSAN, reward rates are specified using a predicate and a function. The function represents the rate at which the reward is accumulated in the states when the predicate evaluates to true. The reward is 0 when the predicate evaluates to false by default [28]. Figure 25 shows the reward rate used to calculate reliability. As long as the system is functioning (i.e., not in an absorbing state), the reward rate accumulates as a function of the number of tokens in the *degraded*, *LOS* and *LOV* places. The function evaluates to 1.0 when there are no tokens in any of those three places indicating normal operation and complete reliability. The reliability is 0 when the system has stopped functioning (in absorbing state). For all other states, the reliability ranges from 1.0 to 0.0 depending on how degraded the system is (indicated by the number of tokens in those three places).

Predicate:
MARK(halted)==0

Function:
1.0/(1+MARK(degraded)+MARK(LOS)+MARK(LOV))

Figure 25: Reward Rate to Calculate Reliability

Since, the SAN models described above recycle tokens when the system is either operating in normal mode or degraded mode, it is necessary to explicitly impose a halting condition to indicate an absorbing state. The *halted* place common to all the subnets is used to specify the halting condition for the model. Five or more tokens in *degraded*, or

three or more tokens in *LOS*, or one or more token in *LOV*, causes a token to appear in *halted*. The presence of a token in this place is the indication of an absorbing state in the corresponding stochastic activity system. This is achieved by having an input condition on each activity stating that the activity is enabled only if there is no token in the *halted* place (i.e., MARK(*halted*)==0). The presence of a token in *halted* thus disables all the activities in the model, thereby causing an absorbing state.

4.2.4 Extensibility of the SAN Model

The SAN models developed for modeling coincident failures and severity (described in Section 4.2.1) and usage-profiles (described in Section 4.2.2) are easily extensible. The composed SAN model can be extended to include other components deemed relevant to the ABS by adding other subnets, or including the components as part of an existing subnet. A component is modeled just like other existing components, with its own failure activity and the corresponding output cases and probabilities.

Adding more components into the model can however lead to the state explosion problem because of the interleaving of the different token configurations within the subnet itself as well as among all the subnets in the composed model. The best way to handle such cases is to avoid using multiple places to denote multiple instances of the same component where possible (e.g., instead of using two places to denote the two axles, use a single place with the associated activity having a failure rate twice the failure rate for one axle). If it is absolutely imperative to model the two axles separately, then use Replicate, because then UltraSAN can take advantage of the State Lumping Theorem that allows the generation of a smaller state space.

The model, developed for the four channel four sensor ABS, can be adapted to model

other schemes of the ABS, by suitably changing the numbers of all the relevant components modeled (by either removing/adding the respective place, or updating the failure rate as described above).

Different categorizations for severity of failure can be used by simply updating the sub-models of the components to include the necessary places depicting the severity level. The levels of severity can also be altered by changing the number of tokens in each of the "severity" indicating places necessary to cause the system to halt (i.e., the halting condition). Different levels of severity can also be modeled by multiplying the failure rate of the affected component by a different scalar (other than 100 and 10000 for degraded mode and lost stability respectively). Inter-dependencies between other components (or among more than two components) which cause coincident-failures can be modeled by updating the rates of the activities that model failure of those components using the construct shown in Figure 23.

The SAN model representing usage profiles can be updated to represent different usage-parameters or intensity of workload by simply changing the value of *mu* in Figure 24. The model can be extended to associate a cost with each time the failed component must be repaired, by adding an additional place to keep track of the number of times the component has been repaired (denoted by the number of tokens in this place).

However, since the model is an abstraction of a real world problem, predictions based on the model must be validated against actual measurements collected from the real phenomena. A poor validation may suggest modifications to the original model [8]. The results from the analyses of each of the above-discussed models using UltraSAN version 3.5 are presented in Section 5.2.

4.3 Comparing the SPN and the SAN Models

The differences in the modeling strategies for the SPN and the SAN models as well as the difference in the models themselves, which have been described in Sections 4.1 and 4.2 respectively, are discussed here.

4.3.1 Modeling conflicts: Temporal uncertainty vs. Spatial uncertainty

As discussed in Section 2.3.2, while SANs have a natural way of representing spatial uncertainty as well as temporal uncertainty, SPNs on the other hand permit the representation of only temporal uncertainty. This distinction affects the way failure transitions are represented in either formalism.

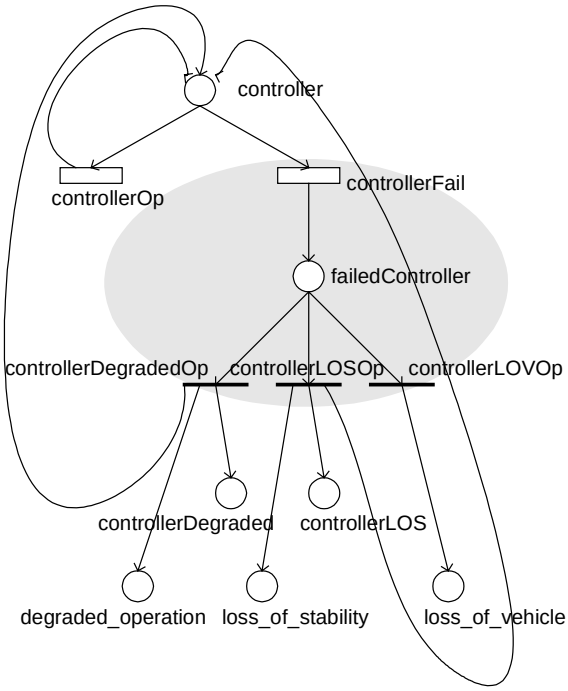


Figure 26: Modeling Uncertainty - SPN Model of Controller

When a component fails, there are up to three possibilities – either it causes the system to operate in a degraded mode, or in lost stability mode, or causes loss of vehicle. Being limited to temporal uncertainty in SPNs, this has been modeled as a conflict between three immediate transitions with different probabilities associated with each (using the **void probval(char*, double)** function available in SPNP which associates an un-normalized probability to an immediate transition) and is highlighted in the SPN model for the *controller* component in Figure 26. This mechanism consists of a discrete probability

distribution function associated with the set of conflicting transitions, and the conflict among immediate transitions are randomly solved.

On the other hand, SANs have a more intuitive way of modeling this uncertainty. *Case probabilities*, represented graphically as circles on the right side of an activity, model uncertainty associated with the completion of an activity. There is a probability distribution associated with each of the output cases, and this models the uncertainty (in the outcome) at completion time of the associated activity. The uncertainty upon completion of an activity modeling a failure is hence easily represented as illustrated in Figure 27 for the *controller* component.

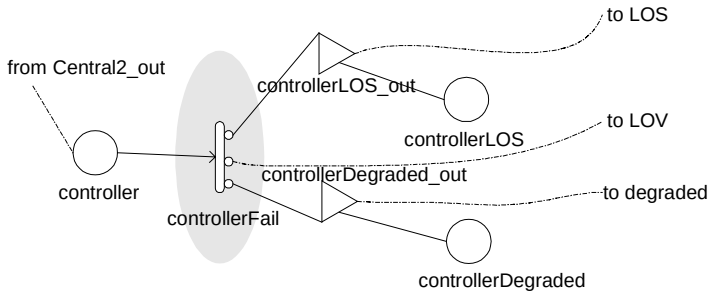


Figure 27: Modeling uncertainty- SAN Model of Controller

4.3.2 Specifying the Halting Condition

Since, both the SPN and the SAN models of the ABS recycle tokens when the system is either operating in normal mode or degraded mode, or even after a component is repaired under passive-use conditions, it is necessary to explicitly impose a halting condition to indicate an absorbing state. The function **void halting_condition(int (**gfunc*)());** defines the halting condition *gfunc* for the SPN. When this function evaluates to zero, the marking is considered to be absorbing. Figure 19 shows the halting condition for the SPN models.

However, specifying the halting condition for the SAN models isn't so straightforward. Figure 28 illustrates the construct to determine the halting condition in

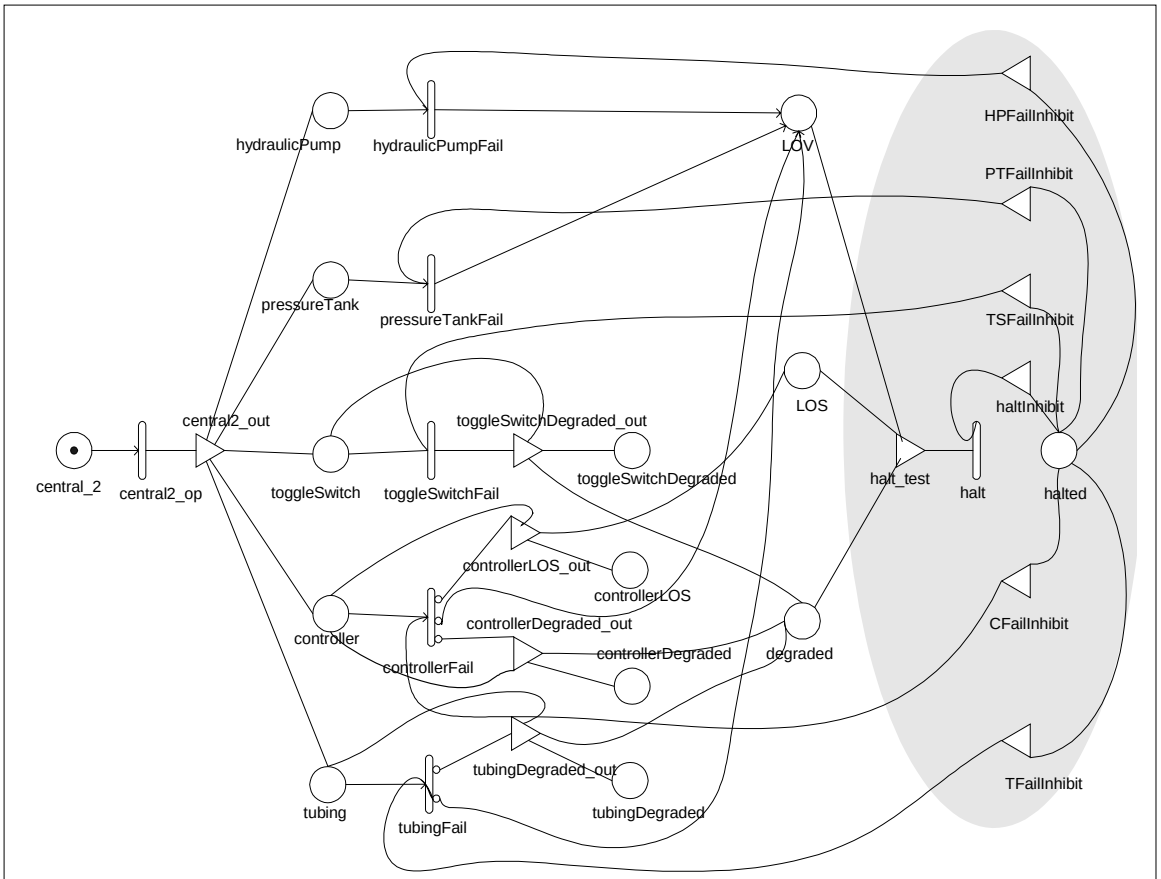


Figure 28: Construct to Determine Halting Condition in SAN Model

the SAN model. As discussed in Section 4.2.3 The function in the input gate *halt_test* deposits a token in *halted* place when there are five or more tokens in *degraded*, or three or more tokens in *LOS*, or one or more token in *LOV*. The presence of a token in this place is an indication of an absorbing state in the corresponding stochastic activity system. This is achieved by having inhibiting conditions (modeled in input gates *HPFailInhibit*, *PTFailInhibit*, *TSFailInhibit*, *haltInhibit*, *CFailInhibit* and *TFailInhibit*) on each activity stating that the activity is enabled only if there is no token in the *halted* place (i.e., MARK(*halted*)==0).

4.3.3 Composed Model Specification

It is cumbersome to specify the large ABS system in terms of a single SPN. Consider the complex and cryptic global SPN model as illustrated in Appendix A.1. SAN models, however, can be combined with replicate and join as shown in Figure 20 to form a composed model. Consequently, while the wheel is modeled four times in the SPN model as FRWheel (Front Right Wheel), FLWheel (Front Left Wheel), RRWheel (Rear Right Wheel) and RLWheel (Rear Left Wheel); in the SAN model, the wheel subnet is conveniently replicated four times.

Besides simplicity of specification, composed SAN models also offer advantages in the area of model solution. For analytic solution, composed models exploit the symmetries in the model to reduce the number of reachable states [28].

4.3.4 Definition of Reliability Reward Rates

The reliability reward rates have been defined differently for the SPN models and the SAN models (Figure 18 and Figure 25 respectively). For the SPN models the reliability measure is defined only in terms of when the system is “up” or “down”, i.e. reliability is defined in terms of when the system (or sub-system) is in operation (up or ‘1’) or has halted (down or ‘0’). The SAN formalism facilitates the notion of degraded operation, and defining the output measures in their terms. Hence, for the SAN models, the reliability measure takes into consideration the degraded modes of operation of the system as well (denoted by the presence of tokens in the *degraded*, *LOS* and *LOV* places).

4.3.5 Compactness and Clarity

Consider the SPN and SAN models illustrated in Appendix A and B respectively. Clearly SANs, with their more expressive graphical primitives, allow a more compact

representation of the system. However, a lot of detail is hidden inside the operation of the primitives, which might not be obvious at first glance.

The results from reliability analysis of the developed models are presented in Chapter 5 and how the results from the SPN models and the SAN models compare to each other is discussed in Section 5.3.

CHAPTER FIVE

RESULTS AND DISCUSSION

`What do you mean by that?' said the Caterpillar sternly. `Explain yourself!'

- *Alice in Wonderland*

The results from the analyses of the models discussed in Chapter 4 are presented here. Section 5.1 presents and discusses the results from the reliability analysis of the SPN models and Section 5.2 presents and discusses the results from the reliability analysis of the SAN models. The results from these two formalisms are then compared and analyzed in Section 5.3.

5.1 Reliability Analysis of SPN models

The reliability of the system at time *t* is computed as the expected instantaneous reward rate at time *t*. To determine the reliability of the system, transient analysis of the developed SPN models was carried out and the reliability measured between 0 and 50K hours. The time duration was deliberately conservative, even though the average life span of a passenger vehicle ranges from 3000 – 9000 hours, the reliability measures were determined for up to 50K hours.

5.1.1 Transient Analysis using SPNP

The transient analysis of the developed SPN models was carried out using the Stochastic Petri Net Package (SPNP) version 6 on a Sun Ultra 10 (400 MHz) with 500MB memory. The number of states that can be handled by the solvers available in the package is only limited by the size of memory available. The package allows options that

specify the way of solving the Stochastic Reward Net (SRN) to be specified in the **void options(void)** function in the input CSPL file. SPNP can perform transient and sensitivity analysis only by reducing the SRN to a CTMC (Continuous-Time Markov Chain). This is achieved by setting the IOP_MC option to VAL_CTMC. Further, the transient-state solution method (IOP_TSMETHOD) for the CTMC was specified as VAL_TSUNIF which stands for Transient Solution using Standard Uniformization[7] [24].

The transient analysis of the developed SPN models resulted in 164,209 tangible markings, of which 91,880 were absorbing. (A marking is a bag representing the configuration of tokens in the places of the Petri net. It is also called the state of the Petri net. A marking is *tangible* if it enables no immediate transition; a marking that does not enable any transition is *absorbing*.) There were 36 immediate transitions. (See Appendix A.1 for details). The approximate running time of the solver on the models was 144-168 hours.

5.1.2 Results for Models Representing Coincident Failures and Severity of Failure

The reliability measure was predicted at 169 different points along the range of 0-50K hours. The interval between the points did not remain constant along the entire time range; instead the time range was divided into four segments. Each of these segments had a different time interval. The expected values of reliability at various time instances were plotted as a function of time.

In Figure 29, the Y-axis gives the measure of interest - the reliability; while the time range (0 to 50K hours) is shown along the X-axis. As expected, the reliability steadily

[7] Uniformization is based on the randomization of the CTMC and exploits the sparse structure of the matrix. In this method, the CTMC is reduced to a DTMC subordinated to a Poisson process. The time-dependent probabilities are given as an infinite series. This method can easily handle large state spaces and is numerically stable but not efficient for stiff problems.

decreases with time. The dashed line indicates the reliability function when coincident failures are modeled and the complete line (**completely overlapped**) indicates the reliability function when coincident failures are not modeled. In other words there is no visible difference between the two curves. The box highlights the range of the average lifetime period (3K-9K hours) of the vehicle.

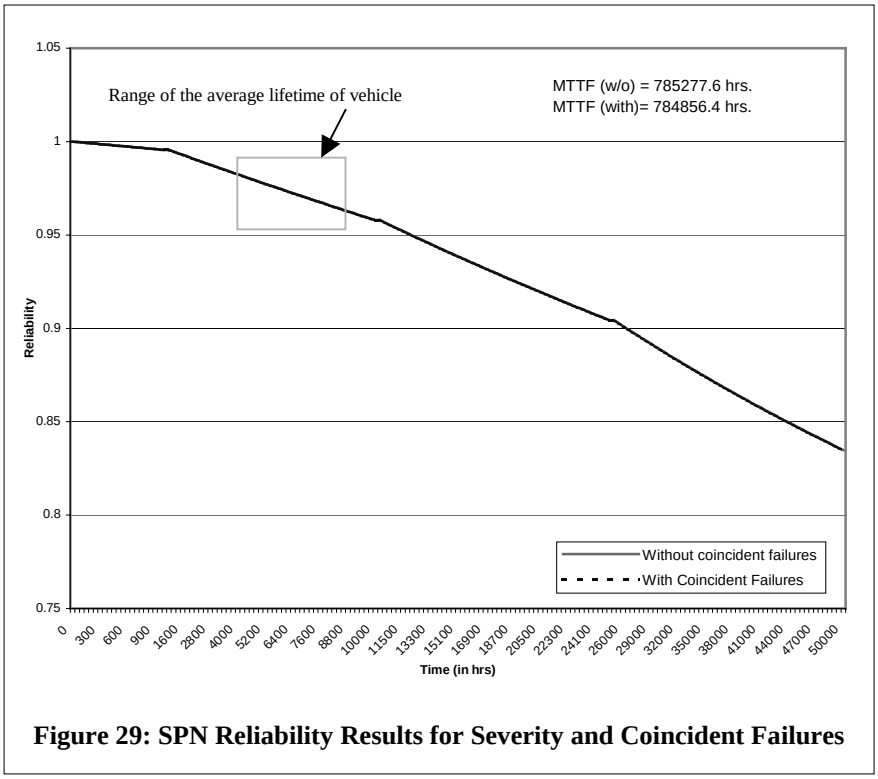


Figure 29: SPN Reliability Results for Severity and Coincident Failures

Figure 30 subtly displays the difference between the two reliability functions. This figure highlights the difference between the reliability functions (reliability for the model without coincident failures **minus** the reliability for the model with coincident failures at each time point) plotted with respect to time, and hence the contribution of modeling coincident failures in the analysis of the system. If there were no difference, the line would not slope upwards but would be at the X-axis. The reliability functions diverge starting around 350 hours of operation, and the difference becomes discernible after

around 13K hours of operation (after the expected life time period a the vehicle). The difference continues to increase with time.

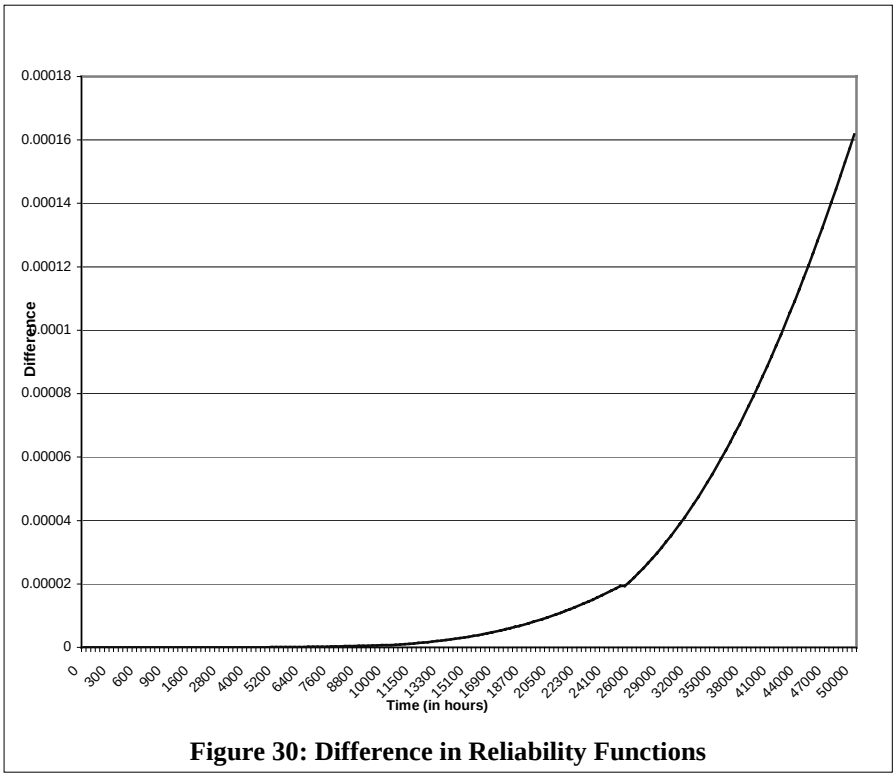


Figure 30: Difference in Reliability Functions

5.1.2.1 Interpretation of Results

For the *limited*[8] number of coincident failures that were modeled, it is clear that the Mean Time to Failure (MTTF) for the model with coincident failures (784,856 hrs) is approximately 421 hours less than the model without coincident failures (785,277 hrs). It is interesting to note that the difference in Mean Time To Failure between the two cases becomes marked only beyond the average lifetime of the vehicle. For the limited number of coincident failures that have been modeled, the difference of 421 hours in the two cases is considered well within the confidence interval. However, it is evident that representing severity and coincident failures in the model contributes to predicting the

[8] One may speculate that there is some kind of relationship (perhaps linear) between the number of dependencies modeled and the difference observed in the graphs and the MTTF values.

system reliability that may be closer to how the real system will behave under the assumptions that have been made (Section 4.1.1.1). This may be borne out by increasing the faithfulness of the model, which in turn may make it more intractable.

5.1.3 Results for Models Representing Usage-Profiles

In Figure 31, the Y-axis gives the measure of interest - the reliability; while the time range (0 to 50K hours) is shown along the X-axis. As expected, the reliability steadily decreases with time. The complete (upper) line indicates the reliability function when the usage of the system is infrequent and the dashed (lower) line indicates the reliability function when the usage of the system is frequent.

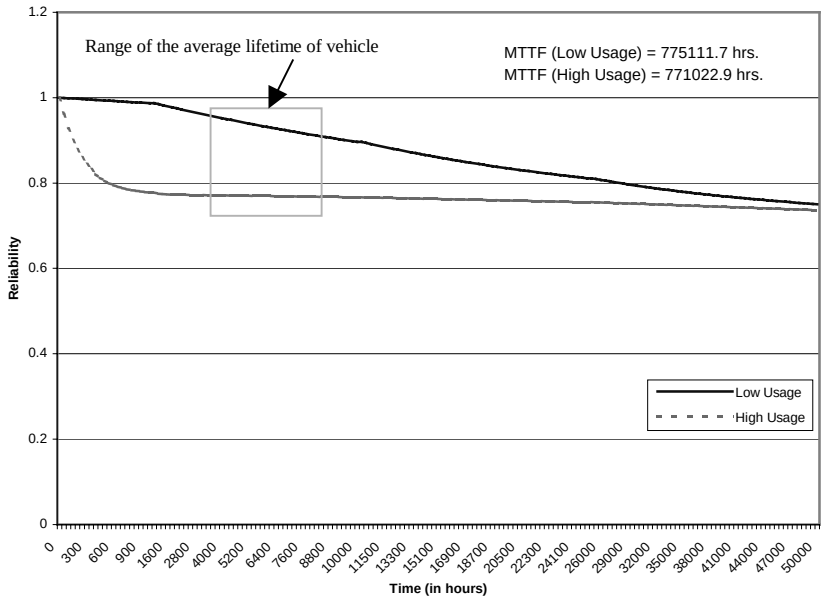


Figure 31: SPN Reliability Results for Usage Profiles

5.1.3.1 Interpretation of Results

Interestingly, the reliability of the system with heavy usage decreases alarmingly within the first 1K hours of operation, while the reliability of the system with moderate

usage decreases perceptibly only after 2.5K hours of operation and then steadily afterwards. Also, the mean time to failure (MTTF) for the high usage case is approximately 771,023 hours as opposed to 775,112 hours for the low usage case, a difference roughly to within an hour of 4,089 hours.

While the two curves are markedly different at the outset, they converge to around the same point at 50K hours. This can be explained by arguing that the plot depends on the way the reliability reward measure has been defined (Figure 18), and it puts a limit to how low the reliability can get in a given time interval. Since the number of absorbing states is the same in both cases (because the only difference in the two models is the value of *mu*), the range of reliability value is the same, yet the rate at which the reward is accumulated (how soon the system degraded/fails) depends on *mu* and hence is different. It is significant to note that the reliability of the system converges to approximately 0.74 at 50K hours (well beyond the lifetime of the vehicle) irrespective of the type of usage. Naturally, concerned about the reliability within the average lifetime of the vehicle, it is clear that aggressive use of the system, causes the reliability to drop much more rapidly than when the system is used conservatively.

Also, consider that some components are used only for a few minutes during the entire lifetime of the vehicle (10-15 years) while other components like the tubing are used all of the time during that period. Hence, the usage of different components is different even within a given usage profile and will affect the actual reliability. What is important is the approach we used and the results (considering the different underlying assumptions - Section 4.1.2.1) clearly indicate that it is important to consider the usage profiles while determining the reliability for any given system. Further refinement and

fidelity is a natural extension (see Section 4.1.4 for the discussion on the extensibility of the model) of the overall approach used here.

5.2 Reliability Analysis of SAN models

The reliability of the system at time *t* is computed as the expected instantaneous reward rate at time *t*. To determine the reliability of the ABS, transient analysis of the developed SAN models was carried out using the instant-of-time transient solver available in the UltraSAN tool. The reliability was measured between 0 and 50K hours. The time duration was deliberately conservative, even though the average life span of a passenger vehicle ranges from 3000 – 9000 hours, the reliability measures were determined for up to 50K hours (as previously mentioned).

5.2.1 Transient Analysis using UltraSAN

The transient analysis of the developed SAN models was carried out using UltraSAN version 3.5 on a Sun Ultra 10 (400 MHz) with 500MB memory. All the analytic solvers in UltraSAN require the explicit generation of the state space and the state transitions of the stochastic process using the state space generator (using Reduced Base Model Generation). The transient solver (trs) solves for instant of time variables with $t < \infty$, using randomization (also known as uniformization). The accumulated reward solver (ars) was used to determine the reward accumulated over 50K hours. This measure, when extrapolated for time going to infinity, represents the mean time to failure of the system.

Uniformization is based on the idea of subordinating a Markov chain to a Poisson process (as stated before). It is computationally efficient, preserves matrix sparsity, and solves to user specified tolerances. The developed SAN models were solved at a tolerance of 1e-09, and resulted in the generation of 859,958 states. The approximate

running time of the solver on the models was 120-144 hours. The results are discussed in detail here.

5.2.2 Results for Models Representing Coincident Failures and Severity of Failure

The reliability measure was predicted at 11 different points along the range of 0-50K hours. The interval between the points did not remain constant along the entire time range and therefore the X-axis is not linear and should be taken into account when viewing the results graphs. The expected values of reliability at various time instances were plotted as a function of time.

In Figure 32, the Y-axis gives the measure of interest - the reliability; while the time range (0 to 50K hours) is shown along the X-axis. As expected, the reliability steadily decreases with time. The dashed line indicates the reliability function when coincident failures are modeled and the complete line indicates the reliability function when

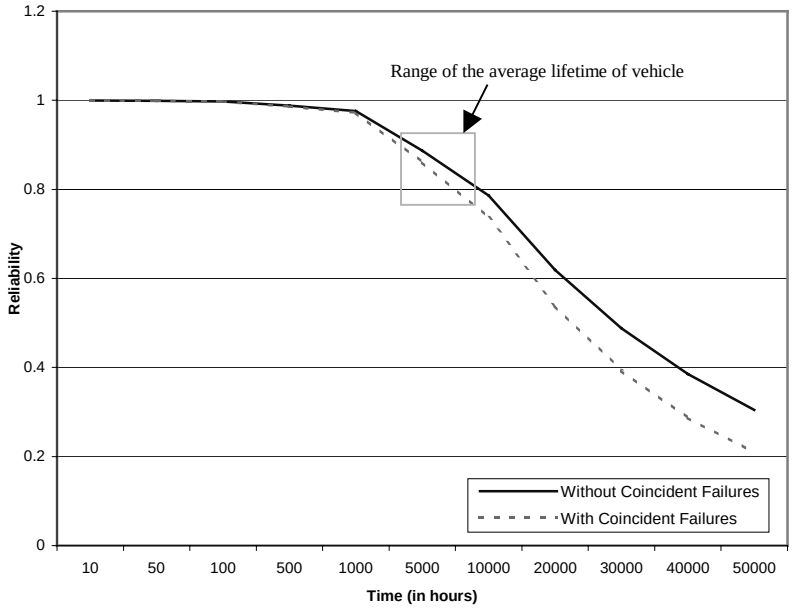


Figure 32: SAN Reliability Results for Severity and Coincident Failures

coincident failures are not modeled.

5.2.2.1 Interpretation of Results

The reliability functions diverge perceptibly after around 1K hours of operation, and the difference continues to increase with time. At 50K hours, the reliability has dropped down to 0.21 when coincident failures are modeled, and down to 0.30 when coincident failures are not modeled, a difference of 0.09 in reliability in the two cases within 50K hours.

Considering the time period approximately around the expected lifetime of the vehicle (3,000-9,000 hours), the difference in reliability after 5K hours of operation is approximately 0.0253 and after 10K hours is 0.0493. This clearly indicates that representing severity and coincident failures in the model contributes to predicting the system reliability that may be closer to how the real system will behave considering the underlying different assumptions.

The Mean Time to Failure calculated at 50K hours in the case where coincident failures are not modeled is 29,167 hours, and in the case where coincident failures are modeled is 25,409 hours, a difference of 3,758 hours. It is important to realize that these results are only for the limited number of coincident failures and levels of severity that have been modeled. Clearly, modeling severity and coincident failures have a significant contribution in determining the system reliability at any given instant of time.

5.2.3 Results for Models Representing Usage-Profiles

In Figure 33, the Y-axis gives the measure of interest - the reliability; while the time range (0 to 50K hours) is shown along the X-axis. As expected, the reliability steadily decreases with time. The complete (upper) line indicates the reliability function when the

usage of the system is infrequent and the dashed (lower) line indicates the reliability function when the usage of the system is frequent.

5.2.3.1 Interpretation of Results

Interestingly, the reliability of the system with heavy usage starts decreasing alarmingly after the first 100 hours of operation, while the reliability of the system with not so heavy usage decreases only perceptibly after 100 hours of operation and then steadily afterwards. Considering the time period within the expected lifetime of the vehicle (3,000-9,000 hours), the reliability for the high-usage profile drops from around 0.55 down to approximately 0.05. For the same duration, the reliability for the low usage drops from 0.9 to only 0.5, a difference of approximately 0.45 after 10K hours of operation.

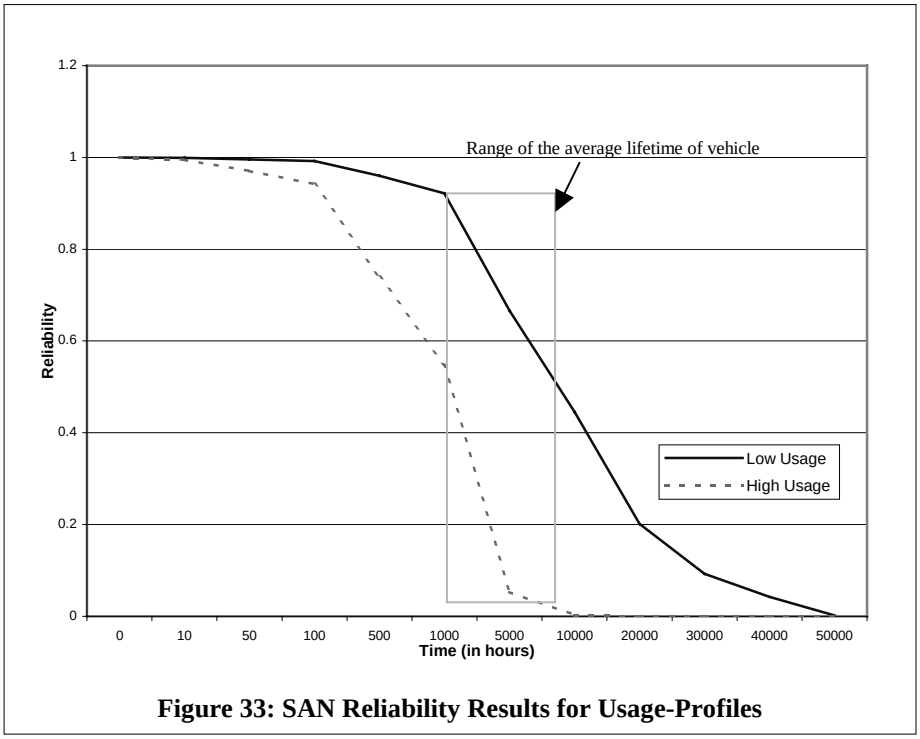


Figure 33: SAN Reliability Results for Usage-Profiles

Also, the mean time to failure (MTTF) calculated at 50K hours for the high usage case is approximately 1,687 hours as opposed to 12,262 hours for the low usage case, a

difference of 10,575 hours. It is significant to note that the reliability of the system drops to 0 at 50K hours (well beyond the lifetime of the vehicle) irrespective of the type of usage. However, it is clear that aggressive use of the system, causes the reliability to drop much more rapidly than when the system is used conservatively.

5.3 Comparison of Results from Analysis using the Two Different Stochastic Tools

A model is always a compromise between faithfulness and simplicity. Figure 34 shows the relation between the realism (or faithfulness) and simplicity (tractability) of a model [12]. How closely a model mirrors its originator or the vision of the system is in direct conflict with how easily and efficiently the model can be analyzed (i.e., solved with respect to its predicted behavior). The models described in Chapter 4 were built incrementally to achieve the best balance between faithfulness to the real system and keeping the model tractable at the same time. As a result models of higher fidelity (more realistic) were created progressively.

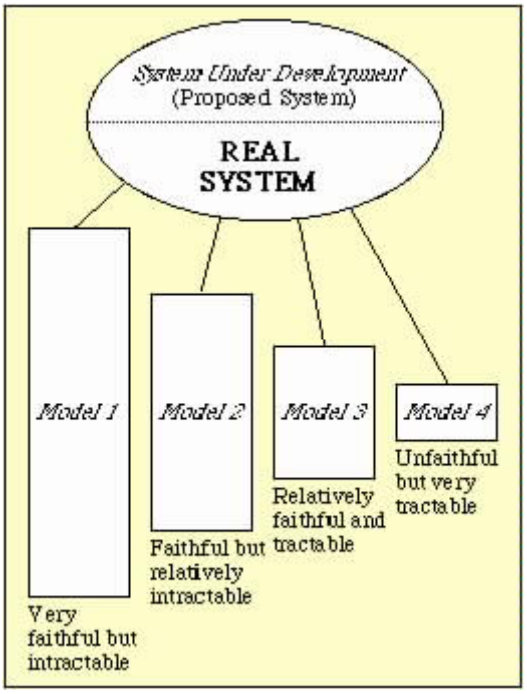


Figure 34: Model Faithfulness vs. Simplicity

A survey of performance/performability analysis studies over the past five years brought up one paper that validated results obtained against those obtained by experimental techniques [55], another that used mathematical reasoning for arguing correctness of results [56], and two that used results from two different methods to

compare them and hence prove their correctness - [57] compared results from analytic and simulation modeling, while [58] compared results from software implemented fault injection and simulated fault injection on a high-speed network. Since, it is beyond the scope (and the means[9]) of this research to validate the results from the analytic experiments against real data, two different stochastic formalisms have been used to carry out the reliability analysis. The results, from the analyses of each of the (SPN and SAN) models developed, using the SPNP and the UltraSAN tools respectively, are discussed and compared in this section.

5.3.1 Comparing Results for Models Representing Severity and Coincident Failures

The results for the SPN models representing severity and coincident failures are shown in Figure 29 and for the SAN models representing severity and coincident failures

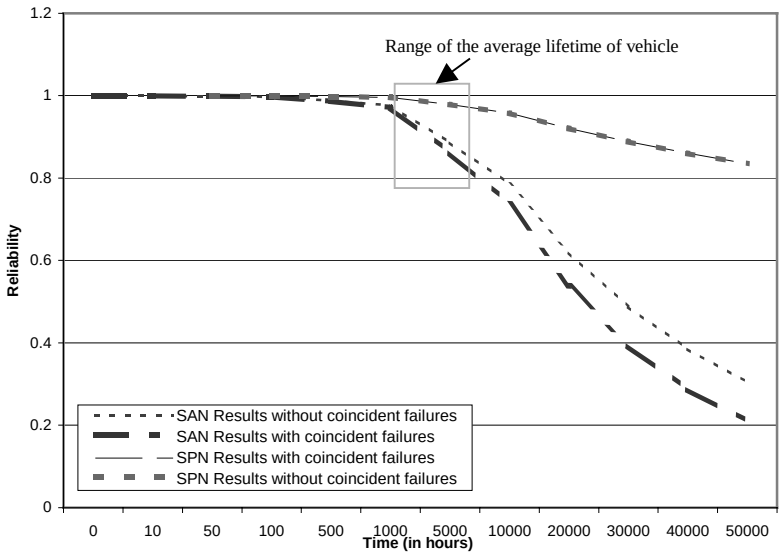


Figure 35: Comparison of SPN and SAN Results for the Models Representing Severity and Coincident Failures

[9] We were lucky to get the reliability data of the components, which is confidential, and had to falsify the data to publish it as part of this research.

are shown in Figure 32. The curves for both these results have now been plotted in one graph and are shown in Figure 35. The curves for the two different SPN models for coincident failures (the model representing severity and coincident failures and the model not representing them) are completely overlapped, the distinction between the results for the SAN models representing severity and coincident failures and the SAN models not representing them is clearly visible. The box highlights the average lifetime period of a vehicle.

The difference in the range of actual reliability values between the SPN and SAN models may be attributed to the different ways in which the reliability reward is defined. The SPN reward rate was defined as a single set of discrete 0/1 values (Section 4.1.3), the SAN reward rate function models a range between 0 and 1 (a function of the number of tokens in the *degraded*, *LOS* and *LOV* places - Section 4.2.3). Therefore, the different rewards accumulate at different rates, and this explains the disparity in the reliability values at any given point in time in the two cases.

5.3.2 Comparing Results for Models Representing Usage-Profiles

The results for the SPN models representing usage-profiles are shown in Figure 31 and for the SAN models representing usage-profiles are shown in Figure 33. In Figure 36 all the results are provided in one graph, making it is easier to compare them. Again, the difference in the range of actual reliability values between the SPN and SAN models may be attributed to the way in which the reliability reward is defined in either case.

5.3.3 Comparing Results to Compensate for Lack of Validation

The above discussion focuses on comparing the results obtained from analyzing both types of models. It should be noticed that results from both models agree on the fact that

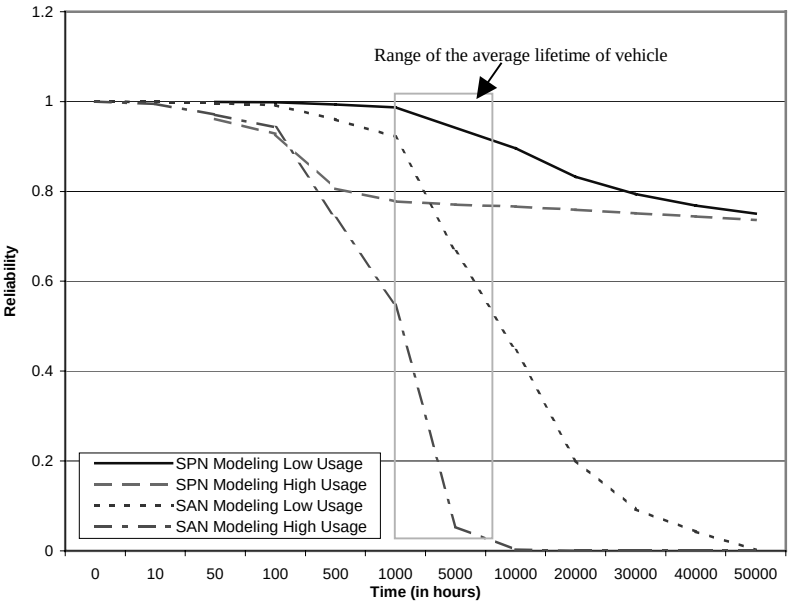


Figure 36: Comparison of SPN and SAN Results for the Models Representing Usage-Profiles

failure severity, coincident failures and usage-profiles contribute significantly to predicting system reliability. The reward rates are defined differently for the two formalisms (Figure 18 and Figure 25), consequently the results don't agree on the exact values of reliability.

Since it is best not to trust results from a single modeling formalism (including the built-in solvers), two different formalisms were employed to predict the reliability for the given system (ABS). If the results had agreed on all accounts, it would have corroborated the modeling strategy. Nevertheless, more information has been gathered by using two different formalisms.

Which of these results is more realistic? This question cannot be easily answered because there exists no data that takes into account the contribution of failure severity,

coincident failures and/or usage-profiles. If such data did exist one could be able to conclude which model yielded more realistic results. Complete validation of these results would require numerous different experiments on vehicles, subject to different usage-profiles and tests to identify and gather data regarding correlated failures and levels of severity. Clearly, this is beyond the scope (and the means) of this research.

However, considering the expressive power of SANs and the way the reliability reward measure has been defined (taking into consideration degraded performance), the SAN models seem to be intuitively more realistic (of course, under the assumptions which are the same for both of the stochastic formalisms). The results from the SAN models show a clear divergence in system reliability where coincident failures have and have not been modeled, as well as in the two different usage-profiles. Considering this, the importance of modeling severity, coincident failures and usage-profiles for a more realistic representation of the system has been reasonably justified.

Ideally, one would like to have data collected that was comprehensive enough to account for the contribution of coincident failures and usage information. Given the data that was available, the next best thing to do was to compare results from the two different modeling formalisms. This gap between predicted and actual results exists in many scientific fields and this approach (of comparing results) is used commonly by researchers. Thus, two different formalisms have been used to counter balance the absence of a feasible validation procedure, and their results have been compared and discussed.

CHAPTER SIX

CONCLUSIONS

`Oh, I've had such a curious dream!' said Alice, and she told her sister, as well as she could remember them, all these strange Adventures of hers that you have just been reading about.

- Alice in Wonderland

6.1 Summary

The objective of this research was to develop a generic (including extensible and realistic) framework for analyzing an embedded vehicle system focusing on severity of failures, coincident failures and usage-profiles. This objective was met by modeling these characteristics in the special case of an Anti-lock Braking System (ABS) of a passenger vehicle. Modeling challenges of state explosion and stiffness were met and overcome. The basis of creating the models was an industrial-strength system characterized by empirical data. The modeling strategy was explained and the extensibility of the models developed has also been discussed. Two different stochastic formalisms – Stochastic Petri Nets and Stochastic Activity Networks, were used to analyze the developed models for reliability measures. The results from these two modeling formalisms were compared and discussed, in the absence of another validation procedure.

6.2 Conclusion

The characteristics of failure severity, coincident failures and usage-profiles were successfully incorporated into the model developed for the ABS of a passenger vehicle. The models evolved over successive iterations of modeling, increasingly refined in their ability to represent different factors that affect the measure of interest (i.e. system

reliability). This resulted in generating a potentially more realistic model (with real data being used to model failure rates). Consequently, the weaknesses in the available data in terms of providing adequate basis to model the extra-functional characteristics under study were identified. This study also established the degree of complexity and the level of abstraction that can be modeled and solved utilizing the available resources.

Comparison of the models developed using the two separate modeling formalisms, SPNs and SANs, indicate that there is no clear winner between the two formalisms. Even though SANs provide more expressive power and compactness to the models developed, the details that are clearly visible in the SPN models are hidden in the SAN models, making them more cryptic.

The results obtained from the analyses showed that the reliability measures were different when the extra-functional characteristics of severity and coincident failures were incorporated. The difference in reliability measures for markedly different system workloads was also identified. Since the model is an abstraction of a real world problem, predictions based on the model should be validated against actual measurements observed from the real phenomena.

Suitable validation procedures (discussed in Section 6.3.4) can provide helpful feedback for refining the model and making it even more realistic. Analysis of such a realistic model provides basis for verification that the requirements for system safety (in terms of required safety functions[10] and safety integrity[11]) have been achieved. Using two different stochastic formalisms for analysis and comparing the results did not compensate

[10] Safety functions are the required functions necessary to achieve or maintain a safe state for the equipment under control.

[11] Safety integrity is defined as the probability of a safety-related system satisfactorily performing the required safety functions under all stated conditions within a stated period of time.

for the lack of validation procedures for analysis results. Yet, the results justified the modeling strategy adopted and highlighted the importance of modeling severity, coincident failures and usage-profiles while examining system reliability.

The goal of developing an approach that is generic and extensible for this application domain was achieved by discussing the extensibility of the developed models to incorporate greater complexity and other relevant features in the given context (e.g., severity of failure, and the effects of different usage profiles). Further extension to the developed models to include other closely related sub-systems is the topic of future work (Section 6.3.2).

The contribution of this research to the automotive industry is substantial as it offers a greater insight into the strategy for developing realistic models. The information from the results is also useful in making design decisions. Results provide greater insight to manufacturers about “weak links” in the system, and increased understanding of which components need to be highly reliable to potentially increase overall system reliability.

This research has successfully established a framework for investigating subsystem reliability. Strategies for modeling severity, coincident failures and usage-profiles were explored, and the models with an adequate level of abstraction and a degree of complexity that were feasible to solve have been presented and discussed. This study can be the basis of numerous other studies, building up on the framework provided and investigating other areas of interest, as presented in the next section.

6.3 Future Work

This section highlights the scope of future work which may be conducted on the basis

of the work presented here.

6.3.1 Sensitivity analysis

Sensitivity analysis is the analysis of the effect of small variations in system parameters on the output measures and can be studied by computing the derivatives of the output measures with respect to the parameter. If a small change in a parameter results in relatively large changes in the outcomes, the outcomes are said to be sensitive to that parameter. System optimization is an important application of sensitivity analysis.

Both the SPN and SAN models lend themselves to sensitivity analysis at various levels: (1) changes in the structure, (2) arbitrary changes in the initial marking, (3) changes in the initial number of tokens in a place, and (4) changes in a parameter involved in the definition of the rate or probability of one or more transitions. While (1), (2) and (3) require the reevaluation of the entire model, since they modify the underlying reachability graphs, in (4) the structure of the underlying reachability graphs is unaffected [24]. Both SPNP and UltraSAN allow the computation of the sensitivities of various parameters.

The models developed so far can be used to carry out sensitivity analysis of the system under study/development to identify the components that are most likely to fail, thereby making the system susceptible to critical failures. Armed with such knowledge, system and software architects can make more informed decisions as to the inherently reliable and safe choices and/or make economic/cost tradeoffs.

6.3.2 Model the Entire System

The Anti-lock Braking system is a small part of the DDR (Dynamic Driving Regulation) system which consists of subsystems like the Anti-lock Braking System

(ABS), the Electronic Steering Assistance (ESA), and the Automatic Slip Reduction (ASR). The models discussed so far can be considered to be details inside the Anti-lock Braking System box in Figure 37. Similar models can be developed for the ESA and the ASR sub-systems as well. To achieve a more realistic model, this work can be extended to incorporate the other closely related sub-systems and analyze the composed model for reliability/availability and sensitivity.

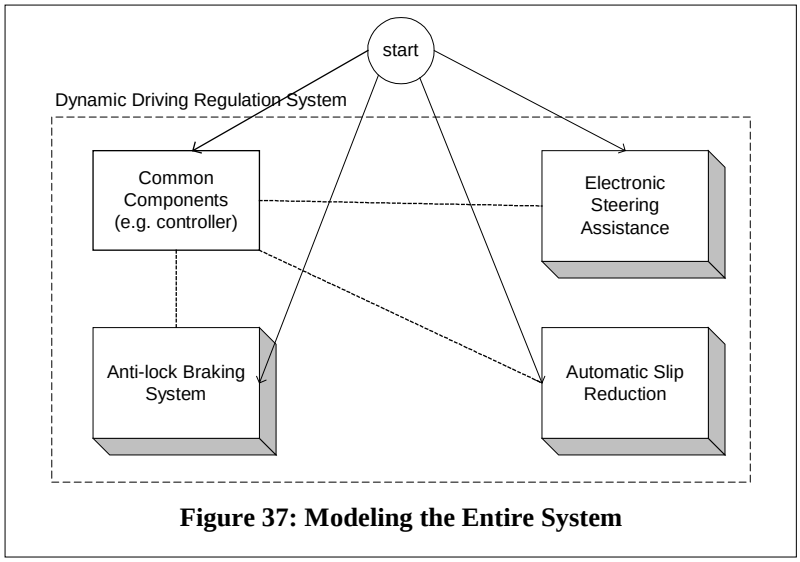


Figure 37: Modeling the Entire System

However, all three sub-systems work closely and share common components. To extend the framework established for modeling the ABS sub-system for dealing with the composition of multiple sub-systems, a new strategy would need to be developed because the model already exploits the maximum degree of complexity that is feasible to solve. The sub-system models can be modeled and analyzed separately and the results combined to get a picture of the overall system reliability. Further, the components that are shared by the different sub-systems can be modeled taking advantage of symmetry in the system and the modeling formalism. Also, different parts of the system can be modeled at different levels of abstraction to focus on a specific point in the system.

6.3.3 Discrete Event Simulation

The DDR system is a very complex system, and a model capturing all its essential

features/characteristics would itself be very complex, precluding any possibility of an analytical solution. In this case, the model must be studied by means of simulation i.e., numerically exercising the model for the inputs in question to determine how they affect the output measures of performance [13]. Discrete Event Simulation concerns the modeling of a system as it evolves over time by a representation in which the state variables change instantaneously at separate points in time.

UltraSAN supports simulation of the SAN model, and makes use of the structure of the SAN and the choice of performability variables to speed up the solution of the system. A choice between a transient solver and a steady-state solver is available [29]. Both simulation solvers provide estimates of the accuracy of the result, using an iterative method for estimating the confidence interval at a user specified interval.

The UltraSAN tool is now being superceded by the Möbius Tool [59]. It provides the capabilities and features available in previous versions of UltraSAN, while at the same time supporting an internal framework design that is more advanced than that of its predecessor. Möbius has the ability to import the existing models from the UltraSAN software, and provides the ability to view intermediate results as a simulation executes. The Discrete Event Simulator offered by Möbius can be used for simulating the composed model developed, instead of using UltraSAN.

6.3.4 Validation of Results

Modeling and experiments on the real system (measurements) are two totally different but complementary approaches in performance evaluation methodology as was shown in Figure 1. The results achieved from analysis of mathematical methods should be compared with the results from analysis of measured values. Comparison of these

values gives essential feedback to the step where the system parameters are characterized in the model being developed, and this process incrementally results in a more faithful model.

The automotive industry has several experimental studies in progress that record the effect of various system components and their failure rates on the vehicle's safety and reliability properties. The technical strategy for validation includes the measures (techniques) and procedures that would be used for confirming that each safety function conforms to the specified system safety requirements [60]. One would like to have data collected that was comprehensive enough to account for the contribution of coincident failures and usage information, such as data about (1) the effect of degraded operation/loss of stability on component failure rate, (2) the correlation of failures between components, (3) the effect of demand/usage on failure rates and, (4) quantization of workload durations. Such data can provide sufficient evidence to corroborate and/or validate the results obtained from these analyses or the basis for evolution of the developed models to more realistic models characterized by measurements from real experiments.

BIBLIOGRAPHY

`Begin at the beginning,' the King said gravely, `and go on till you come to the end: then stop.'

- Alice in Wonderland

1. Vouk, M.A. *Software Reliability Engineering*. in *2000 Annual RELIABILITY and MAINTAINABILITY Symposium*. 2000. Los Angeles, CA: IEEE Computer Society.
2. Jerath, K. and F.T. Sheldon. *Reliability Analysis of an Anti-lock Braking System using Stochastic Petri Nets*. in *PMCCS5*. 2001. Erlangen, Germany: Springer Verlag. p. 56-60.
3. Sheldon, F.T., S. Greiner, and M. Benzinger. *Specification, Safety and Reliability Analysis Using Stochastic Petri Net Models*. in *Tenth International Workshop on Software Specification and Design*. 2000. San Diego, California: IEEE Computer Society. p. 123-132.
4. Siewiorek, D.P. and R.S. Swarz, *Reliable Computer Systems: Design and Evaluation*. 2 ed. 1992: Digital Press. 908.
5. Balbo, G., Professor, Universita di Torino, Italy. Personal Communication at EEF-Summerschool Formal Methods and Performance Analysis, Netherlands, July 2000.
6. Musa, J.D., *Operational Profiles in Software-Reliability Engineering.* IEEE Software, 1993. **10**(2): p. 14-32.
7. Meyer, J., Professor, University of Michigan, Ann Arbor, MI. Personal Communication at PMCCS5, Erlangen, Germany, September 2001.

8. Trivedi, K., *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. 2 ed. 2001, New York: John Wiley & Sons. 848.

9. Balbo, G. *Introduction to Stochastic Petri Nets*. in *FMPA*. 2000. The Netherlands: Springer-Verlag. LNCS 2090. p. 84-155.

10. Meyer, J.F., A. Movaghar, and W.H. Sanders. *Stochastic Activity Networks: Structure, Behavior and Application*. in *Proc. of International Workshop on Timed Petri Nets*. 1985. Turin, Italy: IEEE Computer Society. p. 106-115.

11. Herzog, U. *Formal Methods for Performance Evaluation*. in *FMPA*. 2000. The Netherlands: Springer-Verlag. LNCS 2090. p. 1-37.

12. Sheldon, F.T. and S. Greiner, *Composing, analyzing and validating software models to assess the performability of competing design candidates.* Annals of Software Engineering, 1999. **8**: p. 239-287.

13. Law, A.M. and W.D. Kelton, *Simulation Modeling and Analysis*. 3 ed. 2000, New York: McGraw Hill. 760.

14. Popstojanova, K.G. and K. Trivedi. *Stochastic Modeling Formalisms for Dependability, Performance and Performability*. in *Performance Evaluation: Origins and Directions*. 2000: Springer-Verlag. LNCS 1769. p. 403-422.

15. Meyer, J.F., *Performability: a retrospective and some pointers to the future.* Performance Evaluation, 1992. **14**: p. 139-156.

16. Haverkort, B.R. *Markovian Models for Performance and Dependability Evaluation*. in *FMPA*. 2000. The Netherlands: Springer Verlag. LNCS 2090. p. 38-83.

17. Meyer, J. *Unified performance-reliability evaluation*. in *Proc. 1984 America*

Control Conference. 1984. San Diego, CA. p. 1771-1778.

18. Meyer, J.F. and W.H. Sanders, *Specification and Construction of Performability Models*, in *Performability Modeling: Techniques and Tools*, B.R. Haverkort, et al., Editors. 2001, Wiley. p. 179-222.
19. Muppala, J.K., G. Ciardo, and K. Trivedi, *Stochastic Reward Nets for Reliability Prediction.* Communications in Reliability, Maintainability and Serviceability, 1994. **1**(2): p. 9-20.
20. Littlewood, B. and L. Strigini. *Software reliability and dependability: a roadmap*. in *Proc. of International Conference on Software Engineering*. 2000. Limerick, Ireland: ACM Press. 22. p. 175-188.
21. Murata, T., *Petri Nets: Properties, analysis and applications.* Proceedings of the IEEE, 1989. **77**(4): p. 541-580.
22. Dugan, J.B. and G. Ciardo, *Stochastic Petri Net Analysis of a Replicated File System.* IEEE Transactions on Software Engineering, 1989. **15**(4): p. 394-401.
23. Trivedi, K., *SPNP User's Manual Version 6.0*. 1999, Duke University: Durham, NC, USA.
24. Ciardo, G., J.K. Muppala, and K. Trivedi. *SPNP: Stochastic Petri Net Package.* in *Proc. of Intl. Workshop on Petri Nets and Performance Models*. 1989. Kyoto, Japan: IEEE Computer Society Press. p. 142-151.
25. Trivedi, K. and M. Malhotra. *Reliability and Performability Techniques and Tools: A Survey*. in *Proc. 7th ITG/GI Conference on Measurement, Modeling and Evaluation of Computer and Communication Systems*. 1993. Aachen University of Technology. p. 27-48.

26. Sanders, W.H. and J.F. Meyer. *Stochastic Activity Networks: Formal Definitions and Concepts*. in *FMPA*. 2000. Nijmegen, the Netherlands: Springer Verlag. LNCS 2090. p. 315-343.

27. Movaghar, A. and J. Meyer. *Performability Modeling with Stochastic Activity Networks*. in *Proc. of the IEEE Real-Time Systems Symposium*. 1984. Austin, Texas: IEEE Computer Society. p. 215-224.

28. Sanders, W.H., *UltraSAN User's Manual Version 3.0*. 1994-95, University of Illinois at Urbana-Champaign: Urbana, IL.

29. Couvillion, J., et al., *Performability Modeling with UltraSAN*. IEEE Software, 1991. **8**(5): p. 69-80.

30. Sanders, W.H. and J. Meyer, *Reduced Base Model Construction Methods for Stochastic Activity Networks.* IEEE Journal on Selected Areas in Communications, 1991. **9**(1): p. 25-36.

31. IEEE, *IEEE Standard Glossary of Software Engineering Terminology*. 1990: IEEE Standard 610.12-1990.

32. Gay, F.A. *Performance Evaluation for Gracefully Degrading Systems*. in *Proc. of 9th Annual Int'l Symposium on Fault-Tolerant Computing (FTCS-9)*. 1979. Madison, Wisconsin: IEEE Computer Society. p. 51-58.

33. Hecht Myron, Tang Dong, and H. Hecht. *Quantitative Reliability and Availability Assessment for Critical Systems Including Software*. in *Proc. of the 12th Annual Conference on Computer Assurance*. 1997. Gaitherburg, Maryland.

34. Sahner, R.A. and K. Trivedi. *A hierarchical, combinatorial-Markov model of solving complex reliability models*. in *Proc. of ACM/IEEE Fall Joint Computer*

Conference. 1986. Dallas, Texas: IEEE Computer Society.

35. Kanoun, K. and M. Borrel. *Dependability of Fault-Tolerant Systems - Explicit Modeling of the Interactions Between Hardware and Software Components*. in *Proc. of 2nd Int'l Computer Performance and Dependability Symposium (IPDS)*. 1996. Urbana-Champaign: IEEE Computer Society. p. 252-261.
36. Dugan, J.B. *Experimental analysis of models for correlation in multiversion software*. in *Proc. of 5th Int'l Symposium on Software Reliability Engineering*. 1994. Los Alamitos, CA: IEEE Computer Society. p. 36-44.
37. Eckhardt, D.E. and L.D. Lee, *Theoretical Basis for the Analysis of Multiversion Software Subject to Coincident Errors.* IEEE Transactions on Software Engineering, 1985. **11**(12): p. 1511-1517.
38. Littlewood, B. and D.R. Miller, *Conceptual Modeling of Coincident Failures in Multiversion Software.* IEEE Transactions on Software Engineering, 1989. **15**(12): p. 1596-1614.
39. Nicola, V.F. and A. Goyal, *Modeling of Correlated Failures and Community Error Recovery in Multiversion Software.* IEEE Transactions on Software Engineering, 1990. **16**(3): p. 350-359.
40. Arlat, J., K. Kanoun, and J.-C. Laprie, *Dependability Modeling and Evaluation of Software Fault-Tolerant Systems.* IEEE Transactions on Computers, 1990. **39**(4): p. 504-513.
41. Dong, T. and R.K. Iyer, *Analysis and Modeling of Correlated Failures in Multicomputer Systems.* IEEE Transactions on Computers, 1992. **41**(5): p. 567-577.

42. Butler, R.W. and G.B. Finelli. *The infeasibility of experimental quantification of life-critical software reliability*. in *Proc. of the conference on Software for critical systems*. 1991. New Orleans, Louisiana: ACM Press. p. 66-76.

43. Hsueh, M.C., R.K. Iyer, and K. Trivedi, *Performability modeling based on real data: A case study.* IEEE Transactions on Computers, 1988. **37**(4): p. 478-484.

44. Castillo, X. and D.P. Siewiorek. *Workload, Performance and Reliability of Digital Computing Systems*. in *Proc. of IEEE 11th Int'l Symposium on Fault Tolerant Computing*. 1981. Portland, ME. p. 84-89.

45. Meyer, J. and L. Wei. *Analysis of Workload Influence on Dependability*. in *Proc. of IEEE 18th Int'l Symposium on Fault Tolerant Computing*. 1988. Tokyo, Japan. p. 84-89.

46. Malhis, L.M., W.H. Sanders, and R.D. Schlichting. *Numerical Evaluation of a Group-Oriented Multicast Protocol using Stochastic Activity Networks*. in *Proc. of 6th International Workshop on Petri Nets and Performance Models*. 1995. Durham, NC: IEEE Computer Society. p. 63-72.

47. Qureshi, M.A. and W.H. Sanders. *The Effect of Workload on the Performance and Availability of Voting Algorithms*. in *Proc. of 3rd International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'95)*. 1995. Durham, NC: IEEE Computer Society. p. 217-224.

48. Aupperle, B.E., J. Meyer, and L. Wei. *Evaluation of Fault-Tolerant Systems with Nonhomogeneous Workloads*. in *Proc. of 19th International Symposium on Fault Tolerant Computing (FTCS-19)*. 1989. Chicago, IL: IEEE Computer Society. p. 159-166.

49. Meyer, J., B. Littlewood, and D.R. Wright. *Dependability of Modular Software in a Multiuser Operational Environment*. in *Proc. of 6th International Symposium on Software Reliability Engineering*. 1995. Toulouse, France: IEEE Computer Society. p. 170-179.

50. Weyuker, E.J. and A. Avritzer, *A metric for predicting the performance of an application under a growing workload.* IBM Systems Journal, 2002. **41**(1): p. 45-54.

51. Kolsky, M., *ABS: Understanding Anti-Lock Brakes*. 1997. http://www.abrn.com/archives/0797tech.htm.

52. Struss, P. and A. Malik. *Automated Diagnosis of Car-Subsystems Based on Qualitative Models*. in *Proc. of German Conference on Knowledge-Based Systems (XPS)*. 1997. Bad Honnef, Germany. p. 157-166.

53. Nice, K., *How Anti-Lock Brakes Work*. 2001. http://www.howstuffworks.com/anti-lock-brake.htm.

54. Bosch, R., *Automotive Handbook*. 4 ed, ed. U. Adler and H. Bauer. 1993: Bentley Pubs. 852.

55. Dahlgren, F., M. Dubois, and P. Stenstrom, *Performance Evaluation and Cost Analysis of Cache Protocol Extensions for Shared-Memory Multiprocessors.* IEEE Transactions on Computers, 1998. **47**(10): p. 1041-1055.

56. Meyer, J., *Performability of an Admission for Connection Admission Control.* IEEE Transactions on Computers, 2001. **50**(7): p. 724-733.

57. Han, G., R.H. Klenke, and J.H. Aylor, *Performance Modeling of Hierarchical Crossbar-Based Multicomputer Systems.* IEEE Transactions on Computers, 2001.

50(9): p. 877-890.

58. Stott, D.T., et al., *Dependability Analysis of a High-Speed Network Using Software-Implemented Fault Injection and Simulated Injection.* IEEE Transactions on Computers, 1998. **47**(1): p. 108-119.

59. Clark, G., et al. *The Möbius Modeling Tool*. in *Proc. of 9th International Workshop on Petri Nets and Performance Models*. 2001. Aachen, Germany: IEEE Computer Society. p. 241-250.

60. IEC, *CEI/IEC 61508. Functional safety of electrical/electronic/programmable electronic safety-related systems*. 1998: International Electrotechnical Commission.

APPENDIX A

STOCHASTIC PETRI NET MODELS FOR ABS

A.1 Modeling Severity of Failure and Coincident Failures

This section presents the CSPL code representative of the SPNs modeling severity of failure and coincident failures, along with the actual SPNs constructed from this code. See Section 4.1 for explanation. The CSPL code representative of the SPNs modeling usage-profiles is not very different and is not presented here.

A.1.1 The CSPL code listing

The code presented here defines the global SPN model representing severity of failures and coincident failures. A CSPL file must contain the following five basic functions [23]: options(), net(), assert(), ac_init(), ac_reach() and ac_final(). The code listing presented here has 9 major sections:

- the **options()** function sets the options (like specifying the type of solver to use) which affect the way of describing and solving the model,
- the **halt()** function defines the halting condition of the model,
- followed by the functions that define the variable firing rate of the failure transitions for all components,
- the **net()** function calls a set of functions (like place(), init(), ratefun() etc.) to define the global SRN,
- **assert()** is a Boolean marking-dependent function called by SPNP during the reachability graph construction to check the validity of each newly found marking,
- **ac_init()** is called before starting the reachability graph construction,

- **ac_reach()** is called after the reachability graph construction has completed,
- **reliab()** and **calc_reliab()** are user defined functions that specify the reward rate and the time instants when the reward is to be evaluated respectively,
- **ac_final()** allows the user to flexibly define outputs, calc_reliab() in this case, using some built-in functions provided by SPNP.

```
/*****************************************************************
     * SPNP File Name: ABSWithCoincident.c
     * Run this file as follows:
     *   spnp ABSWithCoincident
*****************************************************************/

#include "user.h"

options()
{
     iopt(IOP_PR_MARK_ORDER, VAL_CANONIC);
     iopt(IOP_PR_MERG_MARK, VAL_YES);
     iopt(IOP_PR_MC_ORDER, VAL_FROMTO);
     iopt(IOP_MC, VAL_CTMC);
     iopt(IOP_OK_ABSMARK, VAL_YES);
     iopt(IOP_OK_VANLOOP, VAL_NO);
     iopt(IOP_OK_TRANS_M0, VAL_YES);
     iopt(IOP_ITERATIONS, 2000);
     fopt(FOP_ABS_RET_M0, 0.000000);
     fopt(FOP_PRECISION, 0.000001);
     iopt(IOP_TSMETHOD,VAL_TSUNIF);
     iopt(IOP_PR_DOT,VAL_YES);
}

int halt()
{
if((mark("loss_of_vehicle") >= 1) || (mark("loss_of_stability") >=
                                3) || (mark("degraded_operation") >= 5))
     return 0;
      else
     return 1;
}

double controllerRate()
{
       double controller_rate = 0.0000006;

       if (mark("controllerLOS") > 0) return controller_rate * 10000;
       if ((mark("controllerDegraded") > 0) || (mark("tubingDegraded") > 0))
              return controller_rate * 100;
       return controller_rate;
}

double tubingRate()
{
       double tubing_rate = 0.0000003;

       if(mark("tubingDegraded") > 0) return tubing_rate * 100;
```

```
	else return tubing_rate;

}

double pipingRate()
{
	double piping_rate = 0.0000004;

	if(mark("pipingDegraded") > 0) return piping_rate * 100;
	else return piping_rate;
}

double hydraulicPumpRate()
{
	double hydraulicPump_rate = 0.0000136;

	if (mark("controllerLOS") > 0) return hydraulicPump_rate * 10000;
	if (mark("controllerDegraded") > 0) return hydraulicPump_rate * 100;
	return hydraulicPump_rate ;
}

double pressureTankRate()
{
	double pressureTank_rate = 0.0000004;

	if (mark("controllerLOS") > 0) return pressureTank_rate * 10000;
	if (mark("controllerDegraded") > 0) return pressureTank_rate * 100;
	return pressureTank_rate;
}

double limitingValveRate()
{
	double limitingValve_rate = 0.00000012;

	if(mark("limitingValveLOS") > 0) return limitingValve_rate * 10000;
	else return limitingValve_rate;
}

double toggleSwitchRate()
{
	double toggleSwitch_rate = 0.0000013;

	if((mark("toggleSwitchDegraded") > 0) || (mark("tubingDegraded") > 0))
		return toggleSwitch_rate * 100;
	else return toggleSwitch_rate;
}

double speedSensorRate()
{
	double speedSensor_rate = 0.000008;

	if(mark("speedSensorDegraded") > 0) return speedSensor_rate * 100;
	else return speedSensor_rate;
}

double pressureSensorRate()
{
	double pressureSensor_rate = 0.000006;

	if(mark("pressureSensorDegraded") > 0) return pressureSensor_rate * 100;
	else return pressureSensor_rate;
}
```

```
double rlValveRate()
{
	double rlValve_rate = 0.00000006;

	if((mark("rlDrainValveLOS") > 0) || (mark("rlInletValveLOS") > 0))
		return rlValve_rate * 10000;

	return rlValve_rate;
}

double rrValveRate()
{
	double rrValve_rate = 0.00000006;

	if((mark("rrDrainValveLOS") > 0) || (mark("rrInletValveLOS") > 0))
		return rrValve_rate * 10000;

	return rrValve_rate;
}

double flValveRate()
{
	double flValve_rate = 0.00000006;

	if((mark("flDrainValveLOS") > 0) || (mark("flInletValveLOS") > 0))
		return flValve_rate * 10000;

	return flValve_rate;
}

double frValveRate()
{
	double frValve_rate = 0.00000006;

	if((mark("frDrainValveLOS") > 0) || (mark("frInletValveLOS") > 0))
		return frValve_rate * 10000;

	return frValve_rate;
}

net()
{
	place("start");
	init("start",1);

	place("loss_of_vehicle");
	place("loss_of_stability");
	place("degraded_operation");

	rateval("braking",0.5);

	iarc("braking","start");

	place("central");
	place("axle");


	oarc("braking","axle");
	oarc("braking","central");


	/* Central Control*/
```

```
imm("central_op");  probval("central_op",1);

place("mbrakecyl");
place("controller");
place("tubing");
place("piping");

iarc("central_op","central");
oarc("central_op","mbrakecyl");
oarc("central_op","controller");
oarc("central_op","tubing");
oarc("central_op","piping");

/* Main Brake Cylinder*/

rateval("mbcOp",1.0);
rateval("mbcFail",0.000001);

iarc("mbcOp","mbrakecyl");
oarc("mbcOp","mbrakecyl");

iarc("mbcFail","mbrakecyl");
oarc("mbcFail","loss_of_vehicle");

/* Controller */

rateval("controllerOp",1.0);
ratefun("controllerFail",controllerRate);

iarc("controllerOp","controller");
oarc("controllerOp","controller");

place("failedController");

iarc("controllerFail","controller");
oarc("controllerFail","failedController");

imm("controllerDegradedOp");     probval("controllerDegradedOp",0.2);
imm("controllerLOSOp");          probval("controllerLOSOp",0.4);
imm("controllerLOVOp");          probval("controllerLOVOp",0.4);

place("controllerDegraded");
place("controllerLOS");

iarc("controllerDegradedOp","failedController");
oarc("controllerDegradedOp","degraded_operation");
oarc("controllerDegradedOp","controllerDegraded");
oarc("controllerDegradedOp","controller");

iarc("controllerLOSOp","failedController");
oarc("controllerLOSOp","loss_of_stability");
oarc("controllerLOSOp","controllerLOS");
oarc("controllerLOSOp","controller");

iarc("controllerLOVOp","failedController");
oarc("controllerLOVOp","loss_of_vehicle");

/* Tubing */

rateval("tubingOp",1.0);
ratefun("tubingFail",tubingRate);

iarc("tubingOp","tubing");
```

```
oarc("tubingOp","tubing");

place("failedTubing");

iarc("tubingFail","tubing");
oarc("tubingFail","failedTubing");

imm("tubingDegradedOp");   probval("tubingDegradedOp", 0.33);
imm("tubingLOVOp");        probval("tubingLOVOp",0.67);

place("tubingDegraded");

iarc("tubingDegradedOp","failedTubing");
oarc("tubingDegradedOp","degraded_operation");
oarc("tubingDegradedOp","tubingDegraded");
oarc("tubingDegradedOp","tubing");

iarc("tubingLOVOp","failedTubing");
oarc("tubingLOVOp","loss_of_vehicle");

/* Piping */

rateval("pipingOp",1.0);
ratefun("pipingFail",pipingRate);

iarc("pipingOp","piping");
oarc("pipingOp","piping");

place("failedPiping");

iarc("pipingFail","piping");
oarc("pipingFail","failedPiping");

imm("pipingDegradedOp");   probval("pipingDegradedOp", 0.33);
imm("pipingLOVOp");        probval("pipingLOVOp",0.67);

place("pipingDegraded");

iarc("pipingDegradedOp","failedPiping");
oarc("pipingDegradedOp","degraded_operation");
oarc("pipingDegradedOp","pipingDegraded");
oarc("pipingDegradedOp","piping");

iarc("pipingLOVOp","failedPiping");
      oarc("pipingLOVOp","loss_of_vehicle");

/* Axle */

imm("axle_op");     probval("axle_op",1);

place("axleCentral");
place("FLWheel");
place("FRWheel");
place("RLWheel");
place("RRWheel");

iarc("axle_op","axle");
oarc("axle_op","axleCentral");
oarc("axle_op","FLWheel");
oarc("axle_op","FRWheel");
oarc("axle_op","RLWheel");
oarc("axle_op","RRWheel");
```

```
/* Axle Central */
imm("axle_central_op");    probval("axle_central_op",1);

place("hydraulicPump");
place("pressureTank");
place("limitingValve");
place("toggleSwitch");
place("speedSensor");
place("pressureSensor");

iarc("axle_central_op","axleCentral");
oarc("axle_central_op","hydraulicPump");
oarc("axle_central_op","pressureTank");
oarc("axle_central_op","limitingValve");
oarc("axle_central_op","toggleSwitch");
oarc("axle_central_op","speedSensor");
oarc("axle_central_op","pressureSensor");

/* Hydraulic Pump */

rateval("hydraulicPumpOp",1.0);
ratefun("hydraulicPumpFail",hydraulicPumpRate);

iarc("hydraulicPumpOp","hydraulicPump");
oarc("hydraulicPumpOp","hydraulicPump");

iarc("hydraulicPumpFail","hydraulicPump");
oarc("hydraulicPumpFail","loss_of_vehicle");

/* Pressure Tank */

rateval("pressureTankOp",1.0);
ratefun("pressureTankFail",pressureTankRate);

iarc("pressureTankOp","pressureTank");
oarc("pressureTankOp","pressureTank");

iarc("pressureTankFail","pressureTank");
oarc("pressureTankFail","loss_of_vehicle");

/* Limiting Valve */

rateval("limitingValveOp",1.0);
ratefun("limitingValveFail",limitingValveRate);

iarc("limitingValveOp","limitingValve");
oarc("limitingValveOp","limitingValve");

place("failedLimitingValve");

iarc("limitingValveFail","limitingValve");
oarc("limitingValveFail","failedLimitingValve");

imm("limitingValveLOSOp"); probval("limitingValveLOSOp",0.22);
imm("limitingValveLOVOp"); probval("limitingValveLOVOp",0.78);

place("limitingValveLOS");

iarc("limitingValveLOSOp","failedLimitingValve");
oarc("limitingValveLOSOp","loss_of_stability");
oarc("limitingValveLOSOp","limitingValveLOS");
oarc("limitingValveLOSOp","limitingValve");
```

```
iarc("limitingValveLOVOp","failedLimitingValve");
oarc("limitingValveLOVOp","loss_of_vehicle");

/* Toggle Switch */

rateval("toggleSwitchOp",1.0);
ratefun("toggleSwitchFail",toggleSwitchRate);

iarc("toggleSwitchOp","toggleSwitch");
oarc("toggleSwitchOp","toggleSwitch");

place("toggleSwitchDegraded");

iarc("toggleSwitchFail","toggleSwitch");
oarc("toggleSwitchFail","degraded_operation");
oarc("toggleSwitchFail","toggleSwitchDegraded");
oarc("toggleSwitchFail","toggleSwitch");

/* Speed Sensor */

rateval("speedSensorOp",1.0);
ratefun("speedSensorFail",speedSensorRate);

iarc("speedSensorOp","speedSensor");
oarc("speedSensorOp","speedSensor");

place("failedSpeedSensor");

iarc("speedSensorFail","speedSensor");
oarc("speedSensorFail","failedSpeedSensor");

imm("speedSensorDegradedOp");
probval("speedSensorDegradedOp",0.38);

imm("speedSensorLOSOp");   probval("speedSensorLOSOp",0.62);

place("speedSensorDegraded");
place("speedSensorLOS");

iarc("speedSensorDegradedOp","failedSpeedSensor");
oarc("speedSensorDegradedOp","degraded_operation");
oarc("speedSensorDegradedOp","speedSensorDegraded");
oarc("speedSensorDegradedOp","speedSensor");

iarc("speedSensorLOSOp","failedSpeedSensor");
oarc("speedSensorLOSOp","loss_of_stability");
oarc("speedSensorLOSOp","speedSensorLOS");
oarc("speedSensorLOSOp","speedSensor");

/* Pressure Sensor */

rateval("pressureSensorOp",1.0);
ratefun("pressureSensorFail",pressureSensorRate);

iarc("pressureSensorOp","pressureSensor");
oarc("pressureSensorOp","pressureSensor");

place("failedPressureSensor");

iarc("pressureSensorFail","pressureSensor");
oarc("pressureSensorFail","failedPressureSensor");

imm("pressureSensorDegradedOp");
```

```
probval("pressureSensorDegradedOp",0.64);

imm("pressureSensorLOSOp");      probval("pressureSensorLOSOp",0.36);

place("pressureSensorDegraded");
place("pressureSensorLOS");

iarc("pressureSensorDegradedOp","failedPressureSensor");
oarc("pressureSensorDegradedOp","degraded_operation");
oarc("pressureSensorDegradedOp","pressureSensorDegraded");
oarc("pressureSensorDegradedOp","pressureSensor");

iarc("pressureSensorLOSOp","failedPressureSensor");
oarc("pressureSensorLOSOp","loss_of_stability");
oarc("pressureSensorLOSOp","pressureSensorLOS");
oarc("pressureSensorLOSOp","pressureSensor");

/* Rear Left Wheel */
imm("rear_left_wheel_op"); probval("rear_left_wheel_op",1);

place("rlInletValve");
place("rlDrainValve");

iarc("rear_left_wheel_op","RLWheel");
oarc("rear_left_wheel_op","rlInletValve");
oarc("rear_left_wheel_op","rlDrainValve");

/* Rear Left Inlet Valve */
rateval("rlInletValveOp",1.0);
ratefun("rlInletValveFail",rlValveRate);

iarc("rlInletValveOp","rlInletValve");
oarc("rlInletValveOp","rlInletValve");

place("failedRLInletValve");

iarc("rlInletValveFail","rlInletValve");
oarc("rlInletValveFail","failedRLInletValve");

imm("rlInletValveLOSOp");  probval("rlInletValveLOSOp",0.18);
imm("rlInletValveLOVOp");  probval("rlInletValveLOVOp",0.82);

place("rlInletValveLOS");

iarc("rlInletValveLOSOp","failedRLInletValve");
oarc("rlInletValveLOSOp","loss_of_stability");
oarc("rlInletValveLOSOp","rlInletValveLOS");
oarc("rlInletValveLOSOp","rlInletValve");

iarc("rlInletValveLOVOp","failedRLInletValve");
oarc("rlInletValveLOVOp","loss_of_vehicle");

/* Rear Left Drain Valve */
rateval("rlDrainValveOp",1.0);
ratefun("rlDrainValveFail",rlValveRate);

iarc("rlDrainValveOp","rlDrainValve");
oarc("rlDrainValveOp","rlDrainValve");

place("failedRLDrainValve");

iarc("rlDrainValveFail","rlDrainValve");
oarc("rlDrainValveFail","failedRLDrainValve");
```

```
imm("rlDrainValveLOSOp");  probval("rlDrainValveLOSOp",0.19);
imm("rlDrainValveLOVOp");  probval("rlDrainValveLOVOp",0.81);

place("rlDrainValveLOS");

iarc("rlDrainValveLOSOp","failedRLDrainValve");
oarc("rlDrainValveLOSOp","loss_of_stability");
oarc("rlDrainValveLOSOp","rlDrainValveLOS");
oarc("rlDrainValveLOSOp","rlDrainValve");

iarc("rlDrainValveLOVOp","failedRLDrainValve");
oarc("rlDrainValveLOVOp","loss_of_vehicle");

/* Rear Right Wheel */
imm("rear_right_wheel_op");      probval("rear_right_wheel_op",1);

place("rrInletValve");
place("rrDrainValve");

iarc("rear_right_wheel_op","RRWheel");
oarc("rear_right_wheel_op","rrInletValve");
oarc("rear_right_wheel_op","rrDrainValve");

/* Rear Right Inlet Valve */
rateval("rrInletValveOp",1.0);
ratefun("rrInletValveFail", rrValveRate);

iarc("rrInletValveOp","rrInletValve");
oarc("rrInletValveOp","rrInletValve");

place("failedRRInletValve");

iarc("rrInletValveFail","rrInletValve");
oarc("rrInletValveFail","failedRRInletValve");

imm("rrInletValveLOSOp");  probval("rrInletValveLOSOp",0.18);
imm("rrInletValveLOVOp");  probval("rrInletValveLOVOp",0.82);

place("rrInletValveLOS");

iarc("rrInletValveLOSOp","failedRRInletValve");
oarc("rrInletValveLOSOp","loss_of_stability");
oarc("rrInletValveLOSOp","rrInletValveLOS");
oarc("rrInletValveLOSOp","rrInletValve");

iarc("rrInletValveLOVOp","failedRRInletValve");
oarc("rrInletValveLOVOp","loss_of_vehicle");

/* Rear Right Drain Valve */
rateval("rrDrainValveOp",1.0);
ratefun("rrDrainValveFail", rrValveRate);

iarc("rrDrainValveOp","rrDrainValve");
oarc("rrDrainValveOp","rrDrainValve");

place("failedRRDrainValve");

iarc("rrDrainValveFail","rrDrainValve");
oarc("rrDrainValveFail","failedRRDrainValve");

imm("rrDrainValveLOSOp");  probval("rrDrainValveLOSOp",0.19);
imm("rrDrainValveLOVOp");  probval("rrDrainValveLOVOp",0.81);
```

```
place("rrDrainValveLOS");

iarc("rrDrainValveLOSOp","failedRRDrainValve");
oarc("rrDrainValveLOSOp","loss_of_stability");
oarc("rrDrainValveLOSOp","rrDrainValveLOS");
oarc("rrDrainValveLOSOp","rrDrainValve");

iarc("rrDrainValveLOVOp","failedRRDrainValve");
oarc("rrDrainValveLOVOp","loss_of_vehicle");

/* Front Left Wheel */
imm("front_left_wheel_op");      probval("front_left_wheel_op",1);

place("flInletValve");
place("flDrainValve");

iarc("front_left_wheel_op","FLWheel");
oarc("front_left_wheel_op","flInletValve");
oarc("front_left_wheel_op","flDrainValve");


/* Front Left Inlet Valve */
rateval("flInletValveOp",1.0);
ratefun("flInletValveFail", flValveRate);

iarc("flInletValveOp","flInletValve");
oarc("flInletValveOp","flInletValve");

place("failedFLInletValve");

iarc("flInletValveFail","flInletValve");
oarc("flInletValveFail","failedFLInletValve");

imm("flInletValveLOSOp");  probval("flInletValveLOSOp",0.18);
imm("flInletValveLOVOp");  probval("flInletValveLOVOp",0.82);

place("flInletValveLOS");

iarc("flInletValveLOSOp","failedFLInletValve");
oarc("flInletValveLOSOp","loss_of_stability");
oarc("flInletValveLOSOp","flInletValveLOS");
oarc("flInletValveLOSOp","flInletValve");

iarc("flInletValveLOVOp","failedFLInletValve");
oarc("flInletValveLOVOp","loss_of_vehicle");

/* Front Left Drain Valve */
rateval("flDrainValveOp",1.0);
ratefun("flDrainValveFail",flValveRate);

iarc("flDrainValveOp","flDrainValve");
oarc("flDrainValveOp","flDrainValve");

place("failedFLDrainValve");

iarc("flDrainValveFail","flDrainValve");
oarc("flDrainValveFail","failedFLDrainValve");

imm("flDrainValveLOSOp");  probval("flDrainValveLOSOp",0.19);
imm("flDrainValveLOVOp");  probval("flDrainValveLOVOp",0.81);

place("flDrainValveLOS");
```

```
iarc("flDrainValveLOSOp","failedFLDrainValve");
oarc("flDrainValveLOSOp","loss_of_stability");
oarc("flDrainValveLOSOp","flDrainValveLOS");
oarc("flDrainValveLOSOp","flDrainValve");

iarc("flDrainValveLOVOp","failedFLDrainValve");
oarc("flDrainValveLOVOp","loss_of_vehicle");

/* Front Right Wheel */
imm("front_right_wheel_op");     probval("front_right_wheel_op",1);

place("frInletValve");
place("frDrainValve");

iarc("front_right_wheel_op","FRWheel");
oarc("front_right_wheel_op","frInletValve");
oarc("front_right_wheel_op","frDrainValve");

/* Front Right Inlet Valve */
rateval("frInletValveOp",1.0);
ratefun("frInletValveFail",frValveRate);

iarc("frInletValveOp","frInletValve");
oarc("frInletValveOp","frInletValve");

place("failedFRInletValve");

iarc("frInletValveFail","frInletValve");
oarc("frInletValveFail","failedFRInletValve");

imm("frInletValveLOSOp");  probval("frInletValveLOSOp",0.18);
imm("frInletValveLOVOp");  probval("frInletValveLOVOp",0.82);

place("frInletValveLOS");

iarc("frInletValveLOSOp","failedFRInletValve");
oarc("frInletValveLOSOp","loss_of_stability");
oarc("frInletValveLOSOp","frInletValveLOS");
oarc("frInletValveLOSOp","frInletValve");

iarc("frInletValveLOVOp","failedFRInletValve");
oarc("frInletValveLOVOp","loss_of_vehicle");

/* Front Right Drain Valve */
rateval("frDrainValveOp",1.0);
ratefun("frDrainValveFail", frValveRate);

iarc("frDrainValveOp","frDrainValve");
oarc("frDrainValveOp","frDrainValve");

place("failedFRDrainValve");

iarc("frDrainValveFail","frDrainValve");
oarc("frDrainValveFail","failedFRDrainValve");

imm("frDrainValveLOSOp");  probval("frDrainValveLOSOp",0.19);
imm("frDrainValveLOVOp");  probval("frDrainValveLOVOp",0.81);

place("frDrainValveLOS");

iarc("frDrainValveLOSOp","failedFRDrainValve");
oarc("frDrainValveLOSOp","loss_of_stability");
```

```
       oarc("frDrainValveLOSOp","frDrainValveLOS");
       oarc("frDrainValveLOSOp","frDrainValve");

       iarc("frDrainValveLOVOp","failedFRDrainValve");
       oarc("frDrainValveLOVOp","loss_of_vehicle");

       halting_condition(halt);
}

int assert()
{
       return(RES_NOERR);
}

void ac_init()
{
  fprintf(stderr,"\nAnti-Lock Braking/Anti-Skid Controller");
  fprintf(stderr,"\nGenerating SRN data ...\n\n");
  pr_net_info();
}

void ac_reach()
{
  fprintf(stderr,"\nThe reachability graph is being generated");
  fprintf(stderr," for the Anti-Lock Braking System...\n\n");
  pr_rg_info();
}

double reliab()
{
  double reward;
  if((mark("loss_of_vehicle") >= 1) || (mark("loss_of_stability") >=
                            3) || (mark("degraded_operation") >= 5))
     reward = 0;
  else
     reward = 1;
  return reward;
}

void calc_reliab(int start,int stop,int interval)
{
  int i;

  for(i = start; i <= stop; i+= interval)
  {
    solve((double)i);
    pr_expected("reliability",reliab);
  }
}

ac_final()
{
 calc_reliab(0,1000,50); /* 20 intervals */
 calc_reliab(1000,10000,200); /* 45 intervals*/
 calc_reliab(10000,25000,300); /* 50 intervals*/
 calc_reliab(25000,50000,500); /* 50 intervals */
 pr_mtta("mean time to failure");
}
```

A.1.2 The SPN Model

The actual Stochastic Petri Net Model is depicted in Figure A.1. A composite form of this global SPN model that showed only the key structure was presented in Figure 11, and the sub-model for the controller component was depicted in Figure 12. The figure here shows the details of all components glued together and present in the global model.

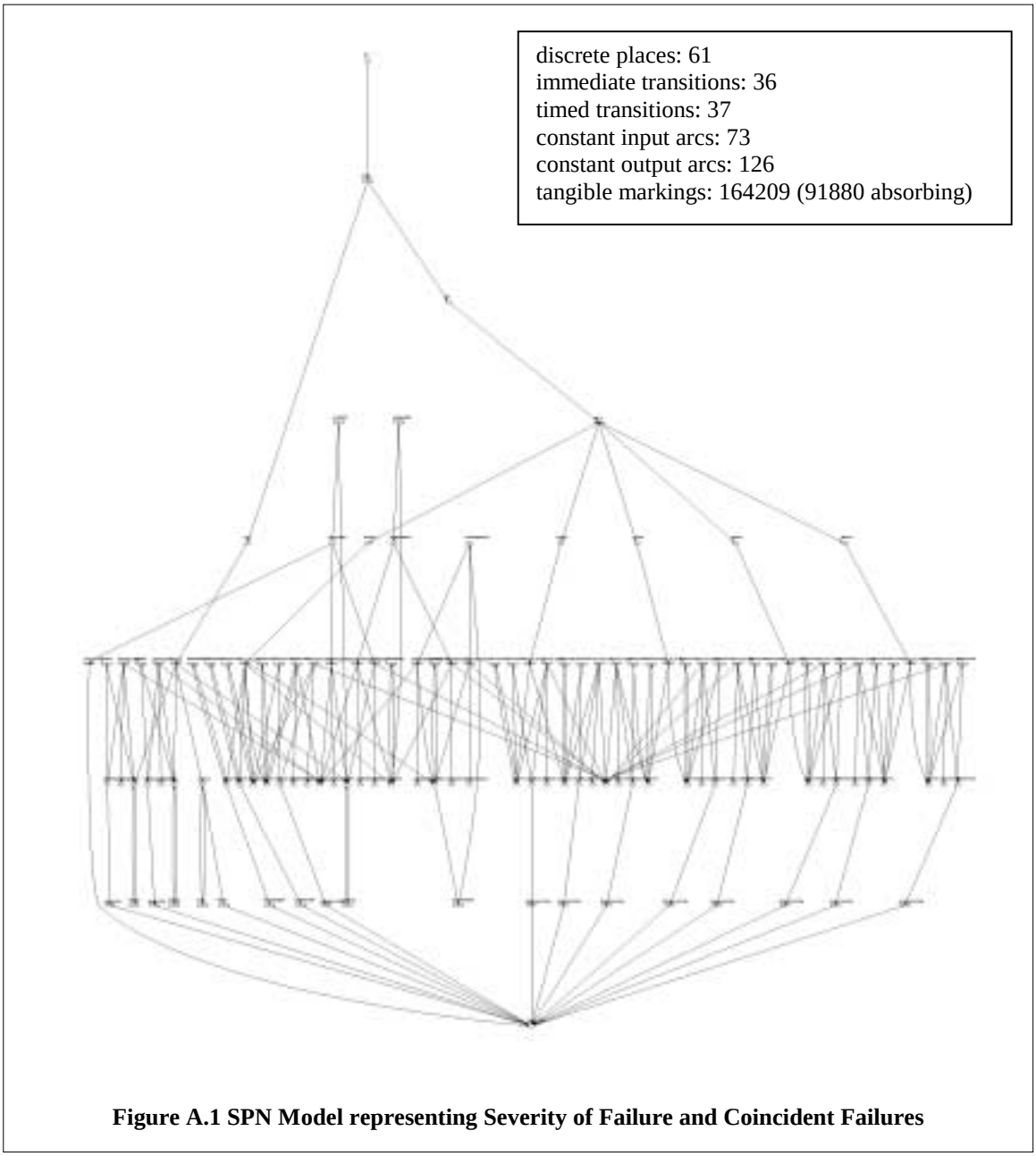


Figure A.1 SPN Model representing Severity of Failure and Coincident Failures

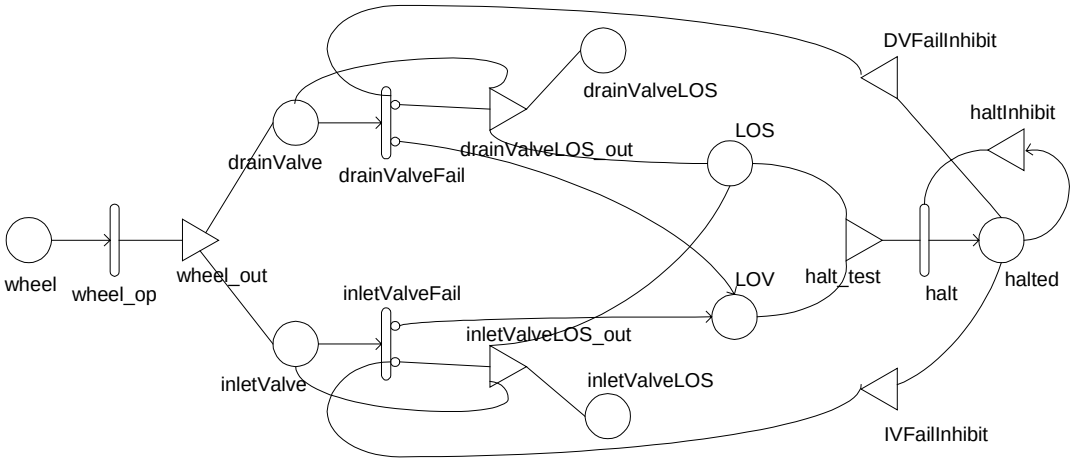
APPENDIX B

STOCHASTIC ACTIVITY NETWORK MODELS FOR ABS

B.1 Modeling Severity and Coincident Failures

This section presents the SAN models for representing severity of failure and coincident failures. See Section 4.2 for explanation. The SAN models representing usage-profiles are not very different these and are not presented here.

The composed SAN model for the ABS was depicted in Figure 20 and the *Central_2* subnet was presented in context of modeling severity and coincident failures in Figure 21. All three subnets are presented here, with the definitions of all the input and output gates and the failure activities. The *Wheel* subnet is depicted in Figure B.1.



Input Gate	Enabling Predicate	Function
haltInhibit	MARK(halted) == 0	1.0
DVFailInhibit	MARK(halted) == 0	1.0
IVFailInhibit	MARK(halted) == 0	1.0
halt_test	MARK(LOS) >=3 || MARK(LOV) >=1	MARK(LOS) = 3; MARK(LOV) = 1;

Output Gate	Output Function
wheel_out	MARK(drainValve) = 1; MARK(inletValve) = 1;

drainValveLOS_out	MARK(drainValve) = 1;
	MARK(drainValveLOS) = 1;
	if(MARK(LOS)>2)
	MARK(LOS) = 3;
	else
	MARK(LOS) += 1;

inletValveLOS_out	MARK(inletValve) = 1;
	MARK(inletValveLOS) = 1;
	if(MARK(LOS)>2)
	MARK(LOS) = 3;
	else
	MARK(LOS) += 1;

Activity	Rate	Probability	
		Case1	Case2
drainValveFail	MARK(drainValveLOS) !=0 || MARK(inletValveLOS) !=0 ?drainValveRate*10000 :drainValveRate	0.18	0.82
inletValveFail	MARK(drainValveLOS) !=0 || MARK(inletValveLOS) !=0 ?inletValveRate*10000 :inletValveRate	0.22	0.78

Figure B.1 The *Wheel* Subnet representing Severity of Failure and Coincident Failures

The *Central_2* subnet is depicted in Figure B.2a.

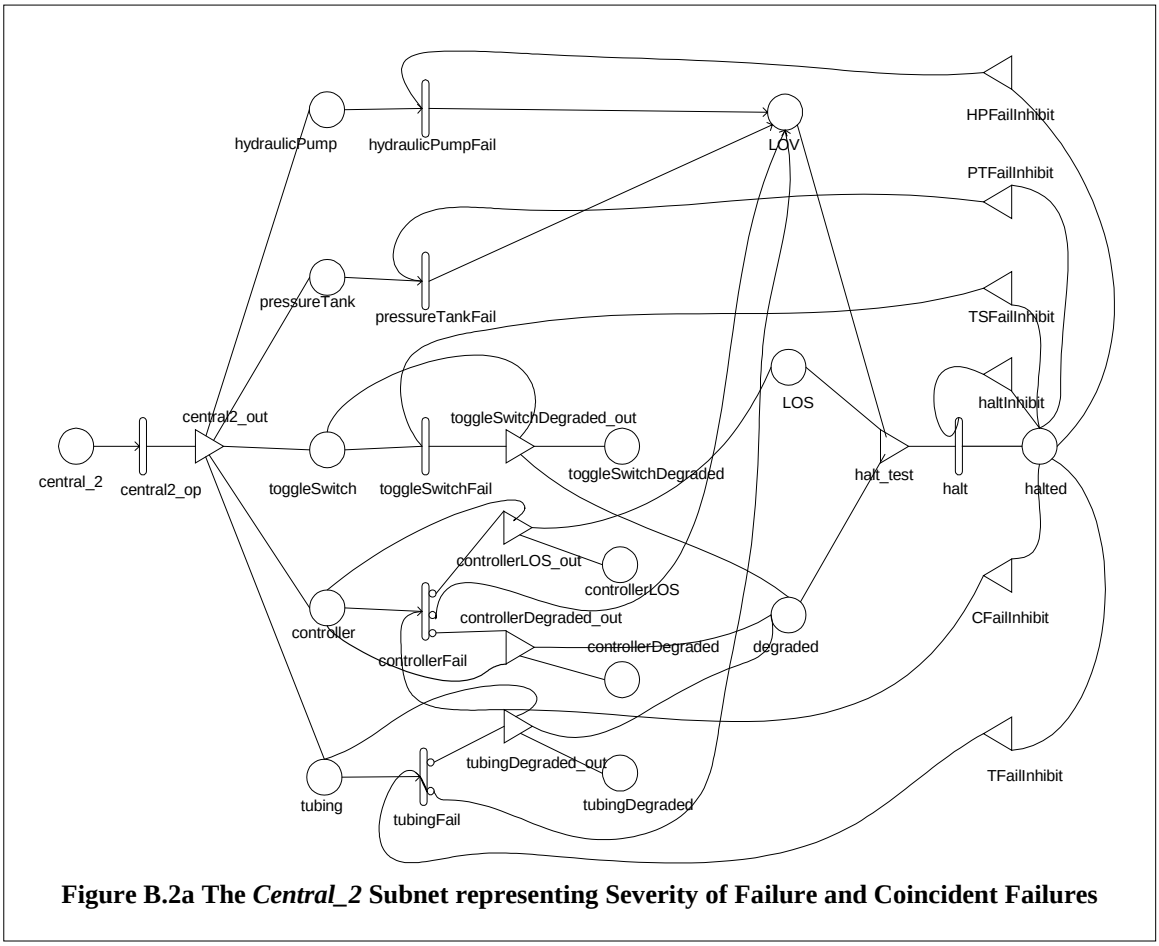


Figure B.2a The *Central_2* Subnet representing Severity of Failure and Coincident Failures

The input gates, output gates and activity rates for the Central_2 subnet are represented in Figure B.2b. Note the constructs specifying the activity rates for modeling severity and coincident failures as discussed in Section 4.2.1.2.

Input Gate	Enabling Predicate	Function
haltInhibit	MARK(halted) == 0	1.0
HPFailInhibit	MARK(halted) == 0	1.0
PTFailInhibit	MARK(halted) == 0	1.0
TSInhibit	MARK(halted) == 0	1.0
CFailInhibit	MARK(halted) == 0	1.0
TFailInhibit	MARK(halted) == 0	1.0
halt_test	MARK(degraded)>=5 || MARK(LOS) >=3 || MARK(LOV) >=1	MARK(degraded)=5; MARK(LOS) = 3; MARK(LOV) = 1;

Activity	Rate	Probability		
		Case1	Case2	Case3
hydraulicPumpFail	MARK(controllerLOS) !=0? hydraulicPumpRate*10000: (MARK(controllerDegraded) !=0 ?hydraulicPumpRate*100 :hydraulicPumpRate)	1.0	-	-
pressureTankFail	MARK(controllerLOS) !=0? pressureTankRate*10000: (MARK(controllerDegraded) !=0 ?pressureTankRate*100 :pressureTankRate)	1.0	-	-
toggleSwitchFail	MARK(tubingDegraded) !=0? toggleSwitchRate*100 :toggleSwitchRate)	-	-	1.0
controllerFail	MARK(controllerLOS) !=0? pressureTankRate*10000: (MARK(controllerDegraded) !=0 || MARK(tubingDegraded) !=0 ?pressureTankRate*100 :pressureTankRate)	0.4	0.4	0.2
tubingFail	MARK(tubingDegraded) !=0? tubingRate*100 :tubingRate)	0.67	-	0.33

Output Gate	Output Function
central2__out	MARK(hydraulicPump)=1; MARK(pressureTank)=1; MARK(toggleSwitch)=1; MARK(controller)=1; MARK(tubing)=1;

toggleSwitchDegraded_out	MARK(toggleSwitch) = 1;
	MARK(toggleSwitchDegraded) = 1;
	if(MARK(degraded)>4)
	MARK(degraded) = 5;
	else
	MARK(degraded) += 1;

controllerLOS_out	MARK(controller) = 1;
	MARK(controllerLOS) = 1;
	if(MARK(LOS)>2)
	MARK(LOS) = 3;
	else
	MARK(LOS) += 1;

controllerDegraded_out	MARK(controller) = 1;
	MARK(controllerDegraded) = 1;
	if(MARK(degraded)>4)
	MARK(degraded) = 5;
	else
	MARK(degraded) += 1;

tubingDegraded_out	MARK(tubing) = 1;
	MARK(tubingDegraded) = 1;
	if(MARK(degraded)>4)
	MARK(degraded) = 5;
	else
	MARK(degraded) += 1;

Figure B.2b The *Central_2* Subnet definitions for Activity Rates and Gates

The *Central_1* subnet is depicted in Figure B.3a.

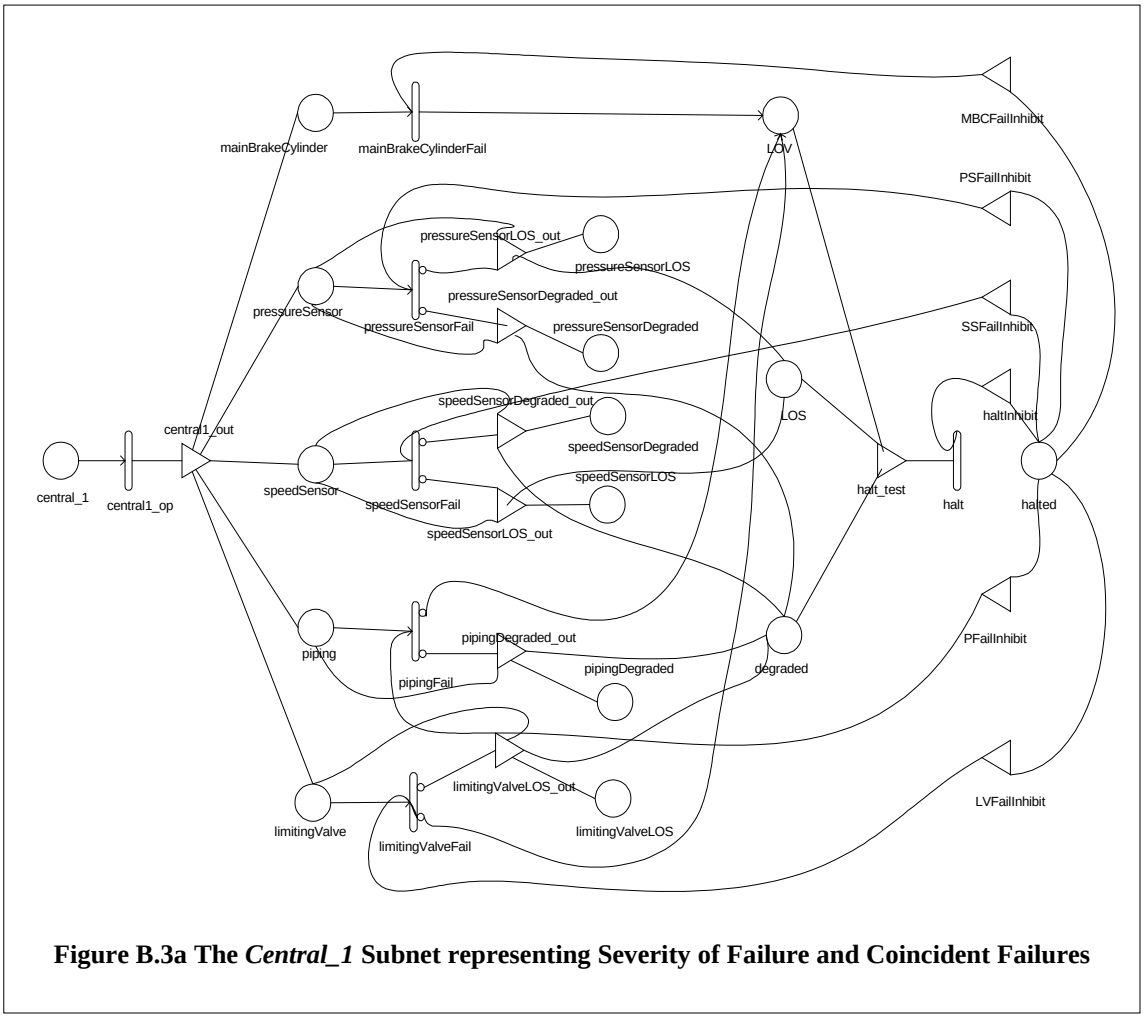


Figure B.3a The *Central_1* Subnet representing Severity of Failure and Coincident Failures

The input gates, output gates and activity rates for the Central_1 subnet are represented in Figure B.3b.

Input Gate	Enabling Predicate	Function
haltInhibit	MARK(halted) == 0	1.0
MBCFailInhibit	MARK(halted) == 0	1.0
PSFailInhibit	MARK(halted) == 0	1.0
SSInhibit	MARK(halted) == 0	1.0
PFailInhibit	MARK(halted) == 0	1.0
LVFailInhibit	MARK(halted) == 0	1.0
halt_test	MARK(degraded)>=5 || MARK(LOS) >=3 || MARK(LOV) >=1	MARK(degraded)=5; MARK(LOS) = 3; MARK(LOV) = 1;

Activity	Rate	Probability		
		Case1	Case2	Case3
mainBrakeCylinderFail	mainBrakeCylinderRate	1.0	-	-
PressureSensorFail	MARK(pressureSensorLOS) !=0? pressureSensorRate*10000: (MARK(pressureSensorDegraded) !=0 ?pressureSensorRate*100 :pressureSensorRate)	-	0.36	0.64
speedSensorFail	MARK(pressureSensorLOS) !=0? pressureSensorRate*10000: (MARK(pressureSensorDegraded) !=0 ?pressureSensorRate*100 :pressureSensorRate)	-	0.62	0.38
pipingFail	(MARK(pipingDegraded) !=0 ?pipingRate*100 :pipingRate)	0.67	-	0.33
limitingValveFail	MARK(limitingValveLOS) !=0?limitingValveRate*10000 :limitingValveRate)	0.78	0.22	-

Output Gate	Output Function
central1__out	MARK(mainBrakeCylinder)=1; MARK(pressureSensor)=1; MARK(speedSensor)=1; MARK(piping)=1; MARK(limitingValve)=1;

pressureSensorDegraded_out	MARK(pressureSensor) = 1;
	MARK(pressureSensorDegraded) = 1;
	if(MARK(degraded)>4)
	MARK(degraded) = 5;
	else
	MARK(degraded) += 1;

pressureSensorLOS_out	MARK(pressureSensor) = 1;
	MARK(pressureSensorLOS) = 1;
	if(MARK(LOS)>2)
	MARK(LOS) = 3;
	else
	MARK(LOS) += 1;

speedSensorDegraded_out	MARK(speedSensor) = 1;
	MARK(speedSensorDegraded) = 1;
	if(MARK(degraded)>4)
	MARK(degraded) = 5;
	else
	MARK(degraded) += 1;

speedSensorLOS_out	MARK(speedSensor) = 1;
	MARK(speedSensorLOS) = 1;
	if(MARK(LOS)>2)
	MARK(LOS) = 3;
	else
	MARK(LOS) += 1;

pipingDegraded_out	MARK(piping) = 1;
	MARK(pipingDegraded) = 1;
	if(MARK(degraded)>4)
	MARK(degraded) = 5;
	else
	MARK(degraded) += 1;

limitingValveLOS_out	MARK(limitingValve) = 1;
	MARK(limitingValveLOS) = 1;
	if(MARK(LOS)>2)
	MARK(LOS) = 3;
	else
	MARK(LOS) += 1;

Figure B.3b The *Central_1* Subnet definitions for Activity Rates and Gates

APPENDIX C

KEY TO SYMBOLS USED IN SPN AND SAN

C.1 Symbols used in Stochastic Petri Nets

The symbols used to describe the various components in a Stochastic Petri Net are listed in Figure C.1.

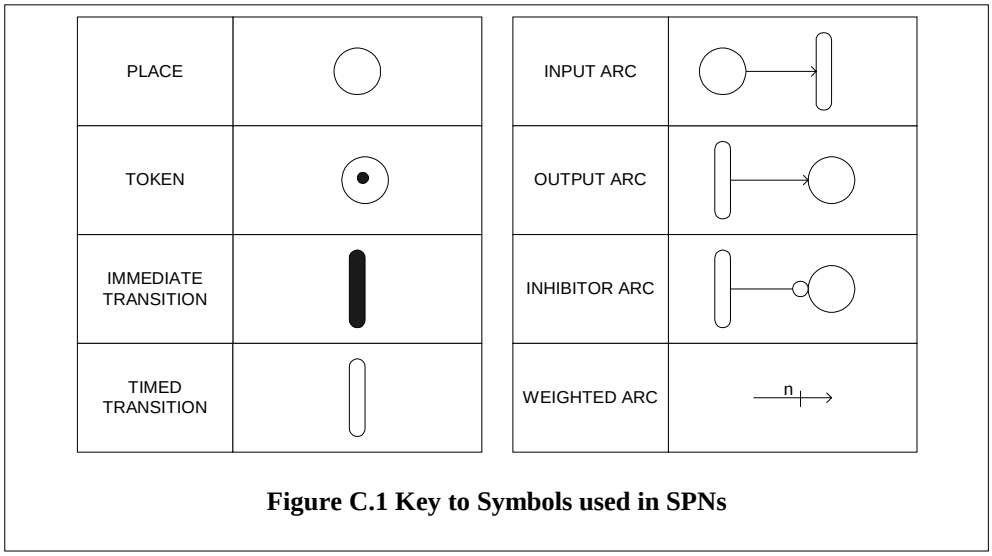


Figure C.1 Key to Symbols used in SPNs

C.2 Symbols used in Stochastic Activity Networks

The symbols used to describe the various components in a Stochastic Activity Networks are listed in Figure C.2.

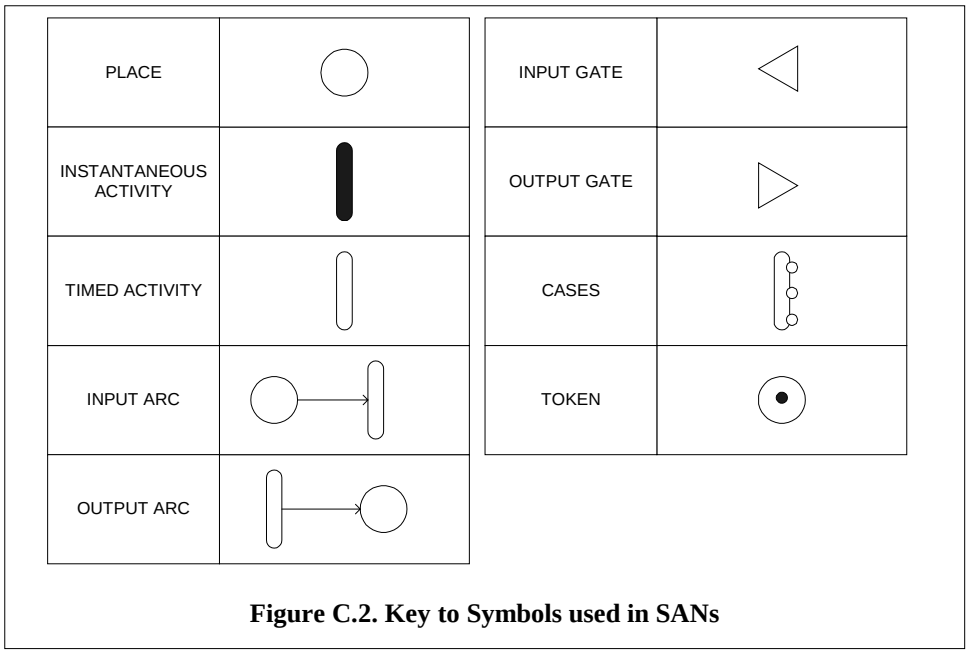


Figure C.2. Key to Symbols used in SANs

APPENDIX D

RISK AND SAFETY INTEGRITY LEVELS

The International Electrotechnical Commission (CEI/IEC) sets out a generic approach for all safety lifecycle activities for systems comprised of electrical and/or electronic and/or programmable electronic components (electrical/electronic/programmable electronic systems (E/E/PESs)) that are used to perform safety functions. This appendix provides information on risk and safety integrity from IEC Standard 61508 [60]. The framework provided by this thesis can provide the basis of quantitative risk analysis (including the determining the SIL of hazards, safety requirements elicitation/generation, risk mitigation and root cause analysis).

D.1 Risk and Safety Integrity

Risk is a measure of the probability and consequence of a specified hazardous event[12] occurring. This can be evaluated for different situations (EUC risk, risk required to meet the tolerable risk, actual risk). The purpose of determining the tolerable risk for a specific hazardous event is to state what is deemed reasonable with respect to both the frequency (or probability) of the hazardous event and its specific consequences. Safety integrity applies solely to E/E/PE safety-related systems, other technology safety related systems and external risk reduction facilities and is a measure of the likelihood of those systems/facilities satisfactorily achieving the necessary risk reduction in respect of the specified safety functions. Safety integrity is considered to be composed of two elements: hardware safety integrity and software safety integrity.

[12] A circumstance in which a person is exposed to potential source of harm and which results in physical injury or damage to the health of people either directly or indirectly as a result of damage to property or the environment.

Table D.1: Example of risk classification of accidents

Frequency	Consequence			
	Catastrophic	**Critical**	**Marginal**	**Negligible**
Frequent	I	I	I	II
Probable	I	I	II	III
Occasional	I	II	III	III
Remote	II	III	III	IV
Improbable	III	III	IV	IV
Incredible	IV	IV	IV	IV

Table D.1 is an example showing four risk classes (I, II, III and IV) for a number of consequences and frequencies. Table D.2 interprets each of the risk classes using the concept of ALARP (as low as reasonable practicable).

Table D.2: Interpretation of risk classes

Risk class	Interpretation
Class I	Intolerable risk
Class II	Undesirable risk, and tolerable only if risk reduction is impracticable or if the costs are grossly disproportionate to the improvement gained
Class III	Tolerable risk if the cost of risk reduction would exceed the improvement gained
Class IV	Negligible risk

D.2 Safety Integrity Levels

To cater for the wide range of necessary risk reductions that the safety-related systems have to achieve, it is useful to have available a number of safety integrity levels as a means of satisfying the safety integrity requirements of the safety functions allocated to the safety-related system. In the IEC 61508 standard, four safety integrity levels are specified, with safety integrity level 4 being the highest level and safety integrity 1 being the lowest. The safety integrity level target failure measures for the four safety integrity levels are specified in Tables D.3 and D.4 for safety-related systems operating in a low demand mode of operation and in a high demand mode of operation respectively.

Table D.3: Safety integration levels: target failure measures for a safety function operating in a low demand mode of operation

Safety integrity level	**Low demand mode of operation** (Average probability of failure to perform its design function on demand)
4	$\geq 10^{-5}$ to $< 10^{-4}$
3	$\geq 10^{-4}$ to $< 10^{-3}$
2	$\geq 10^{-3}$ to $< 10^{-2}$
1	$\geq 10^{-2}$ to $< 10^{-1}$

TableD.4: Safety integrity levels: target failure measures for a safety function operating in high demand or continuous mode of operation

Safety integrity level	**High demand mode of operation** (Probability of a dangerous failure per hour)
4	$\geq 10^{-9}$ to $< 10^{-8}$
3	$\geq 10^{-8}$ to $< 10^{-7}$
2	$\geq 10^{-7}$ to $< 10^{-6}$
1	$\geq 10^{-6}$ to $< 10^{-5}$

The safety integrity levels can be determined from the information present in Tables D.1 and D.2 using either quantitative methods or qualitative methods (e.g. risk graph and hazardous event severity matrix).