

ABSTRACT

Title: A GUIDED SIMULATION METHODOLOGY
FOR DYNAMIC PROBABILISTIC RISK
ASSESSMENT OF COMPLEX SYSTEMS

YUNWEI HU
Doctor of Philosophy, 2005

Directed By: Dr. Ali Mosleh
Reliability Engineering Program
Department of Mechanical Engineering

Probabilistic risk assessment (PRA) is a systematic process of examining how engineered systems work to ensure safety. With the growth of the size of the dynamic systems and the complexity of the interactions between hardware, software, and humans, it is extremely difficult to enumerate the risky scenarios by the traditional PRA methods. Over the past 15 years, a host of DPRA methods have been proposed to serve as supplemental tools to traditional PRA to deal with complex dynamic systems. A new dynamic probabilistic risk assessment framework is proposed in this dissertation. In this framework a new exploration strategy is employed. The engineering knowledge of the system is explicitly used to guide the simulation to achieve higher efficiency and accuracy. The engineering knowledge is reflected in the “Planner” which is responsible for generating plans as a high level map to guide the simulation. A scheduler is responsible for guiding the simulation by controlling the

timing and occurrence of the random events. During the simulation the possible random events are proposed to the scheduler at branch points. The scheduler decides which events are to be simulated. Scheduler would favor the events with higher values. The value of a proposed event depends on the information gain from exploring that scenario, and the importance factor of the scenario. The information gain is measured by the information entropy, and the importance factor is based on the engineering judgment. The simulation results are recorded and grouped for later studies. The planner may “learn” from the simulation results, and update the plan to guide further simulation.

SIMPRA is the software package which implements the new methodology. It provides the users with a friendly interface and a rich DPRA library to aid in the construction of the simulation model. The engineering knowledge can be input into the Planner, which would generate a plan automatically. The scheduler would guide the simulation according to the plan. The simulation generates many accident event sequences and estimates of the end state probabilities.

A GUIDED SIMULATION METHODOLOGY FOR DYNAMIC PROBABILISTIC
RISK ASSESSMENT OF COMPLEX SYSTEMS

By
Yunwei Hu

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy
2005

Advisory Committee:
Professor Ali Mosleh, Chair
Professor Dave Akin
Professor Michel Cukier
Professor Mohammad Modarres
Professor Carol Smidts
Dr. Michael Stamatelatos

© Copyright by

Yunwei Hu

2005

Dedication

To my family.

Acknowledgements

I wish to express my sincere gratitude to Dr. Ali Mosleh for his support, patience, and encouragement throughout my graduate studies. Without his immense help in guiding my research and this dissertation would have been impossible.

I owe special thanks to the contributions of Dr. Frank Greon for his tremendous help during the research, as a colleague and as a friend.

I am fortunate to have been able to work on this project with a talented and dedicated team of UMD researchers consisting of Dr. Frank Greon, Thiago Pirest, Dongfeng Zhu, and Hamed Nejad.

I would like to thank Professor Michel Cukier, Professor Mohammad Modarres, Professor Carol Smidts, Professor Dave Akin and Dr. Michael Stamatelatos for agreeing to be on my committee.

Thanks to my wife, Xiang, for supporting me with love and understanding. My parents receive my deepest gratitude and love for their dedication and support.

Table of Contents

Dedication.....	ii
Acknowledgements.....	iii
Table of Contents.....	iv
List of Tables.....	xi
List of Figures.....	xii
List of Figures.....	xii
1. Introduction.....	1
1.1 Statement of Problem.....	1
1.2 Approach.....	3
1.3 Major Achievements.....	4
1.4 Outline of this Dissertation.....	7
2. Overview of Dynamic Probabilistic Risk Assessment.....	9
2.1 Brief history background of PRA.....	9
2.2 Why DPRA is Necessary.....	10
2.3 Methodologies for DPRA.....	13
2.3.1 Development of DPRA.....	13
2.3.2 Theory of Probability Dynamics.....	14
2.3.3 Graphical Models.....	16

2.3.4	Discrete Dynamic Event Tree:.....	18
2.3.5	Monte Carlo Simulation.....	20
2.3.6	Discrete Event Simulation	22
2.3.7	Improvements to Simulation Methods.....	23
2.4	Summary	28
3.	DPRA as Exploration of Event Sequence Space	29
3.1	Introduction.....	29
3.2	Characterization of the Dynamic PRA Process	32
3.2.1	Basic Terminology.....	32
3.2.2	Event Sequences vs. Scenarios	36
3.3	Exploration of the Event Sequence Space	39
3.3.1	Systematic Exploration	41
3.3.2	Random Exploration	44
4.	Enhanced DPRA Framework.....	48
4.1	Problem Statement.....	48
4.2	Adaptive Exploration	49
4.2.1	Traditional Exploration Strategy.....	49
4.2.2	Bayesian Adaptive Exploration	50
4.3	Outline of a New DPRA Methodology.....	52
4.3.1	The Framework.....	52

4.3.2	Key Elements	55
4.4	Implementation of the Planner of DPRA Simulations.....	57
4.5	Implementation of the Scheduler of DPRA Simulations.....	58
4.6	Interactions Between Planner and Scheduler.....	60
4.6.1	Load Plan into Scheduler	60
4.6.2	Update Plan Based on Simulation Result	61
5.	Scheduler Algorithms	64
5.1	Problem Definition.....	64
5.2	Scheduler Overview	66
5.3	Representation of the Plan in Scheduler	68
5.4	Branch Point Generation for Stochastic Events.....	71
5.4.1	Probability-based Branch Point Generation.....	72
5.4.2	Time-based branch generation.....	75
5.4.3	Branching Point Generation for Demand Based Event	76
5.5	Scheduling Algorithm Based on Value.....	77
5.5.1	Entropy as Measure of Information	78
5.5.2	Expected Entropy Gain Through Experiment.....	83
5.5.3	Principle of Evaluating the Value of Exploring a Scenario	90
5.5.4	Algorithm for Evaluating the Value of Proposed Event.....	91
5.5.5	Exploration of Branches	93

5.6	Event Sequence Quantification.....	94
5.7	Estimator of End State Probabilities.....	101
5.8	Simple Test Case.....	102
5.8.1	The plan.	103
5.8.2	End State Probability Estimates.....	104
5.8.3	Distribution of Sequences.....	106
5.8.4	The Impact of Importance Factor	107
6.	Introduction to SIMPRA.....	109
6.1	Overview.....	109
6.1.1	Framework of SIMPRA	109
6.1.2	Object-Oriented Paradigm	110
6.2	Planner	114
6.3	Scheduler.....	117
6.3.1	Functions of Scheduler	117
6.3.2	Systematic Exploration.....	118
6.4	Structure of the Simulation Model.....	121
6.4.1	Simulation Model.....	121
6.4.2	Interactions between Planner, Scheduler and Simulation Model	123
6.5	Simulation Model Building	124
6.5.1	The Library to Build the Simulation Model:	125

6.5.2	Running the Simulation:	131
6.5.3	Hardware Component Failure Modeling	133
6.5.4	Event Notification.....	135
6.5.5	System State Block	136
6.5.6	End State Notification.....	138
6.5.7	Human Behavior Modeling.....	139
6.5.8	Software Modeling.....	142
7.	Application I – Hold Up Tank	145
7.1	Introduction.....	145
7.1.1	Outline of the Holdup Tank	145
7.1.2	Dynamic Feature of the Holdup Tank Problem.....	147
7.2	Case I	149
7.2.1	Problem Statement.....	149
7.2.2	Analysis in Previous Work	150
7.2.3	Simulation with SIMPRA	153
7.2.4	Scenario Analysis.....	155
7.3	Case II.....	158
7.3.1	Problem Statement.....	158
7.3.2	Scenario Analysis.....	159
7.3.3	Simulation with SIMPRA.....	163

7.4	Comparison between SIMPRA results and other approaches:	165
8.	Application II - Satellite Telecommunication Example	167
8.1	Introduction.....	167
8.2	Scheduler/Simulator side	171
8.3	Result Analysis	173
8.3.1	End State Probability Estimation	173
8.3.2	Allocation of samples over the planed scenarios.....	174
9.	Application II - Space Shuttle Ascent Phase	175
9.1	Summary of the Shuttle Ascent Phase:.....	175
9.2	Building Then Simulation Model	179
9.2.1	Software Model.....	185
9.2.2	Crew Behavior Model.....	186
9.3	Analysis Results:.....	189
9.3.1	Exploration Methods.....	189
9.3.2	Event Sequences	191
9.3.3	End State Probabilities Estimation.....	193
9.3.4	Allocation of Event Sequences	194
9.4	Conclusion	196
10.	Summary and Future Research	198
10.1	Summary of Research Results	198

10.1.1	Overview.....	198
10.1.2	Comparison with Others' Work.....	201
10.2	Future Research	202
10.2.1	Planner	202
10.2.2	Multi-level Modeling.....	203
10.2.3	Human Modeling and Software Modeling	204
	Appendices.....	205
	Appendix A. Graphical Representation in DPRA	205
	Appendix B: Application of Dynamic Fault Tree Simulation.....	212

List of Tables

TABLE 3-1 COMPARISON OF EVENT SEQUENCES AND SCENARIOS	39
TABLE 5-1 EXPECTED INFORMATION GAIN OF BETA PRIOR.....	89
TABLE 5-2 EXPECTED INFORMATION GAIN FOR A DIRICHLET DISTRIBUTION	89
TABLE 5-3 DISTRIBUTION OF EVENT SEQUENCES	107
TABLE 5-4 DISTRIBUTION OF EVENT SEQUENCES	108
TABLE 6-1 EXAMPLES OF THE ELEMENTS OF SIMPRA LIBRARY.....	125
TABLE 6-2 DYNAMIC FAULT TREE.....	128
TABLE 6-3 DYNAMIC FAULT TREE IMPLEMENTED IN SIMPRA	130
TABLE 6-4 PARAMETERS USED IN CASE II.....	158
TABLE 6-5 PLAN FOR CASE II	164
TABLE 8-1 COMPONENT FUNCTIONALITY MATRIX.....	169
TABLE 8-2 PLAN FOR TELECOM SYSTEM.....	170
TABLE 9-1 MISSION ABORT PROCEDURES RULES	188
TABLE 9-2 PLAN FOR SHUTTLE MODEL	190

List of Figures

FIGURE 2.3.1 A DISCRETE DYNAMIC EVENT TREE	19
FIGURE 3.2.1 ILLUSTRATION OF DPRA TERMINOLOGY	36
FIGURE 3.3.1 AN EVENT SEQUENCE IN THE PROBABILISTIC EVENT SEQUENCE SPACE.....	40
FIGURE 3.3.2 ILLUSTRATION OF SYSTEMATIC EXPLORATION.....	44
FIGURE 3.3.3 ILLUSTRATION OF RANDOM EXPLORATION	45
FIGURE 4.2.1 ADAPTIVE LEARNING	50
FIGURE 4.3.1 PROPOSED ENHANCED USAGE OF THE INFORMATION IN THE DPRA EXPLORATION.	54
FIGURE 4.3.2 FRAMEWORK OF THE NEW MODEL BASED DPRA PLATFORM.....	56
FIGURE 5.2.1 THE SCHEDULER FRAMEWORK.....	67
FIGURE 5.3.1 SYSTEM EVOLUTION.....	71
FIGURE 5.4.1: CONSTRUCTION OF INTERVALS.....	74
FIGURE 5.4.2 CONVERSION FROM $U \Delta P$ TO $H(T_k)$	75
FIGURE 5.4.3 EXAMPLE OF GENERATED BRANCH POINT TIMES WITHIN EACH INTERVAL	75
FIGURE 5.4.4 ILLUSTRATION OF THE BRANCHING POINTS	76
FIGURE 5.5.1 STATES OF UNCERTAINTY ABOUT THE OUTCOME OF A SCENARIO.	79
FIGURE 5.5.2 INFORMATION MEASURE AS A FUNCTION OF BETA DISTRIBUTION.....	83
FIGURE 5.5.3 EXPECTED INFORMATION GAIN FOR EXPERIMENT WITH A BETA PRIOR	88
FIGURE 5.5.4 CHOOSING BRANCH BASED ON VALUE.....	94
FIGURE 5.6.1: EXAMPLE OF SCENARIO QUANTIFICATION.	100
FIGURE 5.8.1 FAULT TREE A SIMPLE TEST CASE	103
FIGURE 5.8.2 THE ESTIMATION OF THE PROBABILITY OF SYSTEM FAILURE.....	105
FIGURE 6.1.1 FRAMEWORK OF SIMPRA.....	110
FIGURE 6.2.1 COMPONENT TREE.....	115
FIGURE 6.2.2 STATE RELATIONSHIP DIAGRAM EDITOR	116

FIGURE 6.3.1 PARTIAL SYSTEMATIC SEARCH	120
FIGURE 6.4.1 THE INTERACTION BETWEEN THE DISCRETE MODEL AND CONTINUOUS MODEL.	122
FIGURE 6.5.1 SIMPRA LIBRARY	126
FIGURE 6.5.2 SIMPRA NAVIGATOR	131
FIGURE 6.5.3 RUNTIME GUI	132
FIGURE 6.5.4 END STATE DISPLAY	133
FIGURE 6.5.5 A TYPICAL HARDWARE RUNTIME FAILURE BLOCK.....	134
FIGURE 6.5.6 EVENT NOTIFICATION BLOCK	135
FIGURE 6.5.7 EXAMPLE OF HARDWARE STATE LOGIC	137
FIGURE 6.5.8 END STATE BLOCK.....	138
FIGURE 6.5.9 HIGH LEVEL VIEW OF THE IDAC RESPONSE MODEL (MOSLEH & CHANG, 2004)	140
FIGURE 6.5.10 THE BBN FRAMEWORK	142
FIGURE 7.1.1 HOLDUP TANK SYSTEM LAYOUT.....	146
FIGURE 7.1.2 LAYOUT OF THE SIMULATION MODEL OF HOLDUP TANK	148
FIGURE 7.2.1 A TYPICAL HISTORY OF TANK LEVEL EVOLUTION.....	150
FIGURE 7.2.2 OTHERS WORK	153
FIGURE 7.2.3 PROBABILITY ESTIMATE FROM SIMPRA	155
FIGURE 7.2.4 ACCIDENT SCENARIOS TRIGGERED BY PUMP2 FAILURE.....	156
FIGURE 7.3.1 ESD OF THE HOLDUP TANK, ADAPTED FROM (SIU, 1994).....	159
FIGURE 7.3.2 SCENARIOS OF CASE II	160
FIGURE 7.3.3 ACCIDENT SCENARIO UNDER DIFFERENT CONTROL LAWS	162
FIGURE 7.3.4 ALLOCATION OF EVENT SEQUENCES AMONG PLANS	164
FIGURE 8.1.1 STATE TRANSITION GRAPH	168
FIGURE 8.2.1 STATE DIAGRAM	172
FIGURE 8.3.1 ESTIMATION OF PROBABILITY OF DEGRADED (LEFT) AND FAILED (RIGHT) SYSTEM.....	173
FIGURE 8.3.2 AN EXAMPLE OF THE ALLOCATION OF EVENT SEQUENCES AMONG PLAN.	174
FIGURE 9.1.1 FIGURE SPACE SHUTTLE INTACT ABORT.	176

FIGURE 9.2.1 SPACE SHUTTLE EXAMPLE SIMULATION MODEL.....	179
FIGURE 9.2.2 HARDWARE SIMULATION MODULE.	180
FIGURE 9.2.3 SSME MODULE	181
FIGURE 9.2.4 HEIGHT VS. TIME FOR SPACE SHUTTLE.....	182
FIGURE 9.2.5 RUNTIME FAILURE SUBSYSTEM (INSIDE EACH COMPONENT MODULE)	183
FIGURE 9.2.6 FAULT TREE OF SSME SHUT DOWN.....	184
FIGURE 9.2.7 STATEFLOW® MODEL OF SHUTTLE GNC SOFTWARE	185
FIGURE 9.3.1 ESTIMATION OF THE PROBABILITY OF LOVC.....	194
FIGURE 9.3.2 ALLOCATION OF EVENT SEQUENCES	195

1. Introduction

1.1 Statement of Problem

Probabilistic risk assessment (PRA) is a systematic process of examining how engineered systems work to ensure safety. As the name suggests, this process is quantitative. Probabilities of events with potentially adverse consequences are calculated, as are the magnitudes of the consequences. Probabilistic risk assessment aims at estimating the probability of reaching a particular intermediate condition, or an undesirable end state, as well as the manner in which an event or a combination of events can cause or increase the chance of a particular undesirable end state being reached. The risk of such events is defined as the combination of the event probabilities and their consequences. Information on this risk and the failures that contribute the most to the risk level are of great value to the public, the regulatory agency, and the owner and operators of the system or facility.

In past three decades, through extensive studies conducted on nuclear weapons, nuclear reactors, and other hazardous processes, the PRA methods, particularly those for the assessment of the risks of low probability, high consequence accidents have evolved to a highly sophisticated level. The well-established PRA techniques integrate various reliability modeling tools, such as fault trees and event trees

numerically quantify the risks. The PRA methods typically rely on the risk analyst to identify the risk scenarios.

With the growth of the size of the dynamic systems and the complexity of the interactions between hardware, software, and humans, it is extremely difficult to enumerate the risky scenarios by the traditional ET/FT methods. Dynamic systems can be defined as “systems whose responses to initial perturbations evolve over time as system components interact with each other and with the environment.” (Siu 1994) Although, almost any real system is dynamic, here we use the word “dynamic” to emphasize the importance of timing and interactions. Over the past 15 years, a host of DPRA methods have been proposed to serve as supplemental tools to traditional PRA in such circumstances.

Simulation methods are often used to solve the DPRA problems. Due to the high reliability of many real systems, such methods are intrinsically rare event simulations. It is widely recognized that brutal force Monte Carlo simulation is highly inefficient and may result in generating a lot of histories without any information gain (Labeau, Smidts, & Swaminathan, 2000). A high efficiency simulation engine is often essential to treat realistic systems. Many of the techniques introduced in the DPRA literature focus on increasing efficiency through preset results and approximations. The practical applications to large systems are limited and mostly case-dependent. To the best of the author's knowledge, there is no generic platform for DPRA.

Acknowledging the problems, the objective of this research is to develop an integrated framework for general-purpose dynamic probabilistic risk analysis (DPRA), and an efficient model-based simulation engine for risk assessment of complex systems of hardware, software, and human elements.

1.2 Approach

The approach taken is essentially based on use of simulation models. It provides an environment for modeling and risk assessment of complex dynamic systems consisting of the software, hardware, and human. In order to efficiently cover the enormously large space of possible scenarios, a guided simulation process has been formulated that will avoid the slow convergence, which is common in the Monte Carlo simulations.

The dynamic PRA problem is interpreted as an exploration of the space of the possible event sequences to gain risk information.

Simulation methods have been widely used in dynamic PRA. Objective of the simulation is to understand the behaviors of the system under a variety of conditions, especially those leading to risky scenarios, and to provide insight into those behaviors to engineers/analysts. The simulation is probabilistic by its nature. Unlike most of the biased Monte Carlo methods in literature, which aim at finding an optimal sampling function, while disregarding the structure of the system under investigation, we

actively employ the engineering knowledge on the system through a plan as a high level guide of the simulation. The simulation is then directed according the plan toward scenarios which may are more likely to give us insight of the system vulnerabilities. The simulation can be interpreted as an exploration of space of possible event sequences. The exploration is similar to the exploration of a tree with many branches. The value of each branch is evaluated based on the expected information gain from that branch, while the information gain is measure by the Shannon information entropy. The branches with higher values are more likely to be explored.

To implement and test this methodology the Simulation-based probabilistic risk assessment (SIM-PRA), a general-purpose software package is developed. SIMPRA is applied to different models to demonstrate its capabilities for performing risk analysis on large complex dynamic systems during design and during mission.

1.3 Major Achievements

This research has proposed, developed and demonstrated the use of a new dynamic PRA framework is proposed. The core method is a new exploration strategy is employed. The engineering knowledge of the system is explicitly used to guide the simulation to achieve higher efficiency and accuracy. The engineering knowledge is reflected in the “Planner” which is responsible for generating plans as a high level

map to guide the simulation. A scheduler is developed to guide the simulation by controlling the timing and occurrence of the random events. During the simulation the possible random events are proposed to the scheduler at branch points. The scheduler decides which events are to be simulated. Scheduler would favor events with higher values. Value here is measure of how much we want to simulate a specific event. The value of a proposed event depends on the information gain from exploring that scenario, and the importance of the scenario. The information gain is measured by the information entropy, and the importance factor is based on the engineering judgment. In another word, the scheduler would favor the events which are expected to provide more information, more important by engineering judgment, and brings the system closer towards the scenarios of interest. The simulation results are recorded and grouped for later studies. The planner may “learn” from the simulation results, and update the plan to guide further simulation.

SIMPRA is the software package which implements the new framework. It provides the user with a friendly interface and a rich DPRA library to aid in the construction of the simulation model. User can input the engineering knowledge to the Planner, and the Planner would generate a plan automatically, according to which the simulation would be guided by the Scheduler. The simulation generates many accident event sequences and estimates of their end state probabilities.

SIMPRA has been applied to different systems to demonstrate its capability. A

small 2-out-of-3 system is analyzed, which demonstrates that simulation results converge to the analytical solution very quickly. It shows that the software is capable of guiding the simulation to efficiently generate risk scenarios and the probability estimates.

SIMPRA is applied to several larger, more complex systems. A hold-up tank example which has been investigated by many authors is reconstructed in SIMPRA. It is demonstrated that with guided simulation strategy SIMPRA not only provides an estimate of the system end state probabilities efficiently, but also makes it easier for the risk analysts to investigate the accident scenarios and find system vulnerabilities.

A satellite telecommunication system is also studied by SIMPRA when the system is still under design. With an abstract model of the system, SIMPRA generates possible accident sequences of the system and give a probability estimate the system failure and degradation.

A simulation of Space Shuttle mission in the ascent phase was also constructed in SIMPRA. The simulation model is an integrated one of hardware, software and human crew. It is very complex and highly interactive. The hardware failure, software malfunction and human error all contribute the accident scenarios.

The research on the new DPRA methodology is a team work. Some of my colleagues work on the planner, software modeling and human modeling. My contribution to this research includes the framework, the implementation of the

scheduler, building the simulation library block, constructing the hardware simulation block and integrating the software platform.

1.4 Outline of this Dissertation

The outline of this dissertation is as follows. Chapter 2 reviews dynamic PRA literature. With a short introduction of the history and recent developments in DPRA, a host of DPRA methods are reviewed regarding their advantages and limitations. Special attention is paid to discrete dynamic event tree (DDET) and simulation methods.

In Chapter 3, we interpret the DPRA problem as an exploration of the event sequences space. The DDET and Monte Carlo simulation introduced in chapter 2 represent two different exploration strategies: systematic and random exploration. Instead of focusing on obtaining a numerical result, we approach the problem of exploring the unknown space efficiently. Some of the terminology used throughout the dissertation is explained in this chapter.

In Chapter 4, we propose a new DPRA framework, which employs a new exploration strategy. In this framework, the knowledge of the system is explicitly used to guide the simulation to achieve higher efficiency. The knowledge is reflected in the “Planner” which is responsible for generating plans as a high level map to guide simulation. A scheduler is responsible for guiding the simulation toward the

scenarios, which may generate more information about the system.

The algorithm of scheduler is introduced in Chapter 5. During the simulation, the scheduler checks the status of the simulation and guides the simulation toward the scenarios of interest. The scheduler is also required to maintain a balance between the different scenarios.

In Chapter 6, we introduce SIM-PRA, the software package which implements the guided simulation methodology. It is a generic-purpose risk assessment platform, developed in Matlab and Java.

In Chapter 7, 8 and 9, we apply our methodology to solve the DPRA problems of three different models. The first one is the holdup tank problem in Chapter 7, which has been discussed frequently in the DPRA literatures. In Chapter 8, SIMPRA is applied to a satellite telecommunication system which is still under design. An abstract model is built, and accidents sequences and numerical estimation of the end state probabilities are generated. Another example is a hypothetical model of a space shuttle ascent phase. In the model, we observe complex interactions, between human, software, and hardware. The SIM-PRA very efficiently produces a model to depict the system and performs the risk analysis. With these two examples, we have shown the capability of our methodology and its difference from previous works.

2. Overview of Dynamic Probabilistic Risk Assessment

2.1 Brief history background of PRA

Probabilistic Risk Assessment (PRA), which is also called Quantitative Risk Analysis, has been applied to large complex systems for more than thirty years. The first full scale application of PRA methods was the Reactor Safety Study WASH-1400 (NRC, 1975).

The PRA methods have also been used in other industry sectors and military. After extensive review of NASA safety policy following the Challenger accident in 1986, NASA instituted a number of programs for quantitative risk analysis. An example is the risk assessment of Space Shuttle program (Fragola, 1995). Office of Safety and Mission Assurance at NASA headquarters published several handbooks to enhance the PRA expertise at NASA (Stamatelatos et al., 2002).

In some areas, PRA techniques are part the regulatory framework. In situations where risk management is critical to mission success, the PRA methods are increasingly playing an important support role for management decision making and regulatory agencies.

PRA tries to answer the three questions posed in (Kaplan & Garrick, 1981), which

can be represented as the set of triplets $\langle s_i, f_i, p_i \rangle$: “scenarios – frequencies – consequences”.

The classical PRA approach involves the construction of separate models describing the system vulnerabilities and risks, which is often performed by the risk analysts. Models are typically built in the form of fault trees and event trees, which are graphical representations of Boolean expressions describing the combinations of so-called basic events leading to system failure. Basic events typically represent the failure of some components or subsystems. The level of resolution in these models, e.g., the extent to which events are decomposed into the contributing basic events, is driven by the PRA objectives, as well as the availability of data to quantify the basic events (Mosleh & Bier, 1992). The knowledge required to solve the ET/FT is basic probability calculation, and commercial software packages are available to construct fault trees and conduct the computation.

2.2 Why DPRA is Necessary

PRA methodology has been successfully applied in different projects, but it has been recognized that it is hard to characterize some complex dynamical systems by solely applying such techniques as Event Tree/ Fault Tree analysis. Event trees or fault trees are implementations of logic. Primarily, the Boolean logic-based models are limited in terms of their capability to specify the timing of events or even the

order in which events occur. It is also difficult to model the dependency of the probability or rate of occurrence of events on scenarios or time.

In a Boolean logic based model, even with the “dynamic” expansions, it is the risk analysts who identify the interactions between the different parts of the system and their influence on the system safety. In a dynamic system such task is far from trivial.

In the fault tree analysis, an often used unstated assumption is that when the cut set occurs, the top event occurs simultaneously. In most cases, this assumption is legitimate, but in a dynamic system, especially when there are complex interactions between the hardware-software-human, the sojourn time or response time must be taken into account explicitly. This is illustrated in (Cojazzi, 1996).

Apart from the time-dependent analysis, leaving the system dynamics out of the picture is considered to be oversimplified in some cases. (Devooght, 1998) The event tree analysis can display the correct failure logic of dynamic systems, but due to ignoring the role of process variables explicitly, it cannot determine the distribution of time to an undesirable state. Event tree is basically a pictorial representation of Boolean logic, so the only way event tree can take the process variable into account is by discretizing the process variables ranges. When we need detail process variables or when the number of variables increases, the event tree may grow unmanageable. Without a physical model the event tree analysis has to involve subjective judgment of the interaction between variables. As a result the assessment of the probability to

achieve the absorbing state may be inaccurate. The stochastic process induced by the random hardware/software failures, coupled with the system dynamics and/or human intervention would possibly trigger other significant failures in the system.

Acknowledging such difficulties, a set of new methodologies were developed under the name “Dynamic reliability” or “Dynamic PRA”. Because of the diverse background of people working on this problem, it is sometimes hard to define the term “dynamic reliability”. Nevertheless, it is accepted that the following table lists the basic characteristics of dynamic reliability/PRA modeling: (Aldemir & Zio, 1998)

1. The dynamic phenomena have a strong influence on the system’s response (e.g. the operation of control/protection devices upon reaching assigned thresholds of the process variables values)
2. The hardware components failure behavior and on human operator actions depends on the process dynamics.
3. The complex interactions between human operator actions and hardware components influence the system’s response and failure behavior.
4. There are a variety of degraded modes related to multiple failure modes and the process dynamics.

The DPRA methodologies are not to replace the traditional PRA methods. The

dynamic methods are rather supplemental tools. With the DPRA tools, researchers would understand clearly the limits of classical approaches, and determine when the dynamic methods are needed.

2.3 Methodologies for DPRA

2.3.1 Development of DPRA

The analysis by (Amendola & Reina, 1981) explored the possibility of global treatment of the dynamic PRA. Later the DYLAM and ADS implementations were applied to treat DPRA problems in nuclear power plants and other areas (Chang, 1999; Cojazzi, 1996; Hsueh & Mosleh, 1996; Nivolianitou, Amendola, & Reina, 1986). Later, a more general mathematical framework was introduced for probabilistic dynamics (Devooght & Smidts, 1992). Probabilistic dynamics theory interprets the DPRA problems as problems equivalent to transport problems to be solved, for example, by Monte Carlo simulation. Even though the theoretical framework is sufficiently general but the complexity of numerical work to solve the equations is daunting.

The wide acceptance of traditional ET/FT methods has led some authors to propose extension to include some dynamic features in the FT framework. Others have introduced different graphical tools to capture the dynamical features, some of which have been use in applications. Examples are Petri Nets, Dynamic Flowgraph,

and Event Sequence Diagram. A detailed discussion of these techniques can be found in Appendix A. For broader overview of the dynamic PRA methodology, there are several nice review papers of the DPRA is available (Labeau, Smidts, & Swaminathan, 2000; Siu, 1994).

2.3.2 Theory of Probability Dynamics

The mathematical formulation of the DPRA problem was first attempted by Devooght 1992, and later expanded (Devooght & Smidts, 1992, 1996; Izquierdo & Labeau, 2004; Izquierdo, Melendez, & Devooght, 1996; Labeau, 1996).

In this framework the system configuration is indexed by an integer number $i=1, 2, 3 \dots n$. This implies that the human software and hardware models are considered to be multi-state components. The vector of process variables is denoted as \bar{x} . The evolution of the process vector in state i is deterministic and defined by a set of differential equation:

$$\frac{d\bar{x}}{dt} = \bar{f}_i(\bar{x}), \bar{x}(0) = \bar{x}_0, \bar{x} \in \mathfrak{R}^N \quad (1)$$

The deterministic evolution may be interrupted by a random walk from one state to another. The probability of that (\bar{x}, i) will change to (\bar{x}, j) per unit time is $p(i \rightarrow j | \bar{x})$, and the solution to equation (1) with initial condition \bar{x}_0 is $\bar{g}(t, \bar{x}_0)$

The objective of Dynamic PRA is to find the probability density function $\pi(\bar{x}, i, t)$,

which denoted the likelihood of find system at point \bar{x} and in state i at time t . The distribution is normalized as

$$\sum_{i=1}^{\alpha} \int_{\mathbb{R}^n} \pi(\bar{x}, i, t) \cdot d\bar{x} = 1$$

In a Markovian framework, the Chapman-Kolmogorov equations give,

$$\frac{\partial \pi(\bar{x}, i, t)}{\partial t} + \text{div}[\bar{f}_i(\bar{x}) \cdot \pi(\bar{x}, i, t)] + \lambda_i(\bar{x}) \cdot \pi(\bar{x}, i, t) = \sum_{j \neq i} p(i \rightarrow j | \bar{x}) \pi(\bar{x}, j, t)$$

Where $\lambda_i(\bar{x})$ is the total transition rate out of state i , given \bar{x} , such that

$$\sum_{j \neq i} p(i \rightarrow j | \bar{x})$$

Written in the integral formulation:

$$\pi(\bar{x}, i, t) = \int \pi(\bar{u}, i, 0) \delta(\bar{x} - \bar{g}_i(t, \bar{u})) e^{-\int_0^t \lambda_i(\bar{g}_i(s, \bar{u})) ds} d\bar{u} + \sum_{j \neq i} \int p(i \rightarrow j | \bar{u}) d\bar{u} \cdot \int_0^t \delta(\bar{x} - \bar{g}_i(t - \tau, \bar{u})) \times e^{-\int_0^{t-\tau} \lambda_i(\bar{g}_i(s, \bar{u})) ds} \pi(\bar{u}, j, \tau) d\tau$$

The integral formulations allow for a semi-Markov extension (Devooght & Smidts, 1996) and a unified treatment of transitions on demand and transitions in time.

Izquierdo et al. extend this theory to account for “stimulus” which may trigger automatic or manual actions (Izquierdo & Labeau, 2004). They call it stimulus-driven theory of probabilistic dynamics (SDTPD). The state space includes additional extension: stimulus activation states are considered. A stimulus is either an order for action from human or automatic control device, or the fulfillment of conditions that trigger a stochastic event.

The closed form of analytical solutions has been attempted several times. Some small systems with Markov assumption were solved (Cukier, 1991; Labeau, 1995; Labeau et al., 2000). ESDs can provide semi-analytical solutions while reducing the dimensionality of the problem by solving a sequence of integrals rather than a large system of PDEs (Swaminathan & Smidts, 1999). For larger systems, the analytical solution is hard to find. Discretization methods, such as Discrete Dynamic Event Tree, or simulation based methods present great potential to solve the DPRA problem.

2.3.3 Graphical Models

Graphs provide an intuitive representation of the system logic. To take advantage of the fact the risk analysts are familiar with classical Event Tress/ Fault Tree analysis, extended types of Fault Tree formalism have been proposed to address the dynamic evolution of systems. Other graphical frameworks dealing with dynamic systems which have been applied successfully in various engineering fields are introduced into reliability/ risk analysis. Examples are Petri Nets, (Chatelet, Chabot, & Dutuit, 1998; Dutuit, Chatelet, Signoret, & Thomas, 1997; Malhotra & Trivedi, 1995; Tombuyses, 1999; Vernez, Buchs, & Pierrehumbert, 2003; Volovoi, 2004), Dynamic Flowgraph Methodology (DFM), (Houtermans, Apostolakis, Brombacher, & Karydas, 2000, 2002; Kaufman, Bhide, & Johnson, 2000), GO-FLOW (Matsuoka,

2004; Matsuoka & Kobayashi, 1988) and Dynamic Event Sequence Diagram (Swaminathan & Smidts, 1999, 1999, 1999).

The graphical representation often serves as an input scheme of a numerical or mathematical procedure, e.g., Markov chain, which will be solved to obtain the numerical estimate of the system. Sometimes the graphs can encode Markov or semi-Markov processes. Petri Nets and GO-FLOW are examples which are modification of state graph to account for the accident specific features. ESDs which emerged from traditional PRA methods are also shown to be able to encode a Markov or semi-Markov process.

All these techniques enhance the ability to deal with dynamic reliability/PRA problem, but they also have their limitations. One common drawback of these approaches is that due to exponential explosion of the state space, the graphs rapidly grow unmanageable. Another limitation is that some graphical schemes rely on the Markovian assumption, which may not hold in most real systems, and the Markovian approximation may generate a distorted estimate.

This research would not follow this line of thought. One aim of this research is to ease the burden of risk analysts to identify all possible risk scenarios, which is exactly what is required by the graphical models. The intuitiveness of the graphical representation is still appealing. In fact, graphical models are supplementary tools in this study. For example, the planner (see Chapter 4) works partly based on knowledge

of the state transition diagrams. The graphical models used here, however, would not be directly used to generate the underlying probabilistic model which would be solved analytically or numerically.

2.3.4 Discrete Dynamic Event Tree:

Discrete Dynamic Event Trees (DDETs) are simulation methods implemented by forward branching event trees, the branch points are restricted at discrete times only. The knowledge of the physical system under study is contained in a numerical simulation, written by the analyst. The components of the system are modeled in terms of discrete states. All possible branches of the system evolution are tracked systematically (Cojazzi, 1996; Hsueh & Mosleh, 1996; Nivolianitou et al., 1986). One restriction of DDET is that the events (branches) only happen at predefined discrete time intervals. It is assumed that if the appropriate time step is chosen, DDETs would investigate all possible scenarios. It is a straightforward extension of the classical event trees. The binary logic restriction of classical event trees is removed. The construction of the tree is computerized. An example of DDET is given in **Error!**

Reference source not found.

The systematic branching would easily lead to such a huge number of sequences that the management of the output Event Tree becomes awkward. Measures have been taken to eliminate the explosion of branches. It can be done by increasing the

length of the time step, but this may be at the expense of the accuracy of the analysis. A cut-off probability P_{lim} was introduced in some implementations. The branches with a probability lower than P_{lim} would be discarded. (Amendola, 1988) suggested that when the number of failures in a sequence exceeds a user-defined value, further evolution along this sequence would be stopped. The determination of such parameters is problem-dependent.

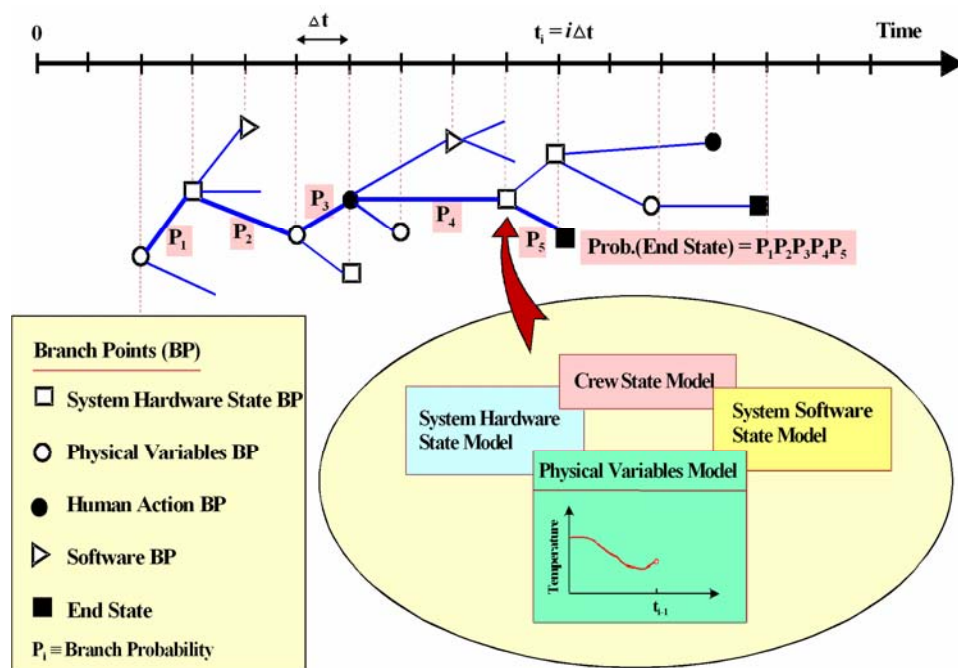


Figure 2.3.1 A Discrete Dynamic Event Tree

Implementations of DDETs include DYLAM,(Cacciabue, Carpignano, & Vivalda, 1992; Cacciabue & Cojazzi, 1994; Cojazzi, 1996; Nivolianitou et al., 1986) DETAM,(Siu, 1994) and ADS, (Hsueh & Mosleh, 1996), ADS-IDA, (Chang, 1999).

2.3.5 Monte Carlo Simulation

While DDETs require the events to occur at predefined discrete time only, the Monte Carlo simulation approaches allow events to happen at any time. This avoids the combinational explosion of DDETs. Monte Carlo methods is insensitive the complexity and dimension of the system. Any modeling assumption could be included, the non-fixed failure rate assumption, random delays, interaction between components and process dynamics, etc. Dubi claimed that MC is the only practical approach to solve the realistic systems (Dubi, 1998). Generally the MC methods estimate the system safety or reliability directly, expressed in form of a probabilistic distribution function $\pi(\bar{x}, i, t)$.

The analog between Monte Carlo simulation for PRA and the transport problem is often drawn, (Devooght & Smidts, 1992; Dubi, 1998; Smidts & Devooght, 1992). In the Monte Carlo simulation framework, a system is defined as a collection of components and state of the system is described by at least one real valued “system functions” as the function of its state vector and possibly other relevant parameters. The system function is defined on phase space $(\mathbf{B}; t)$, where state vector of the system is defined as: $\mathbf{B} = (b_1; b_2; b_3; \dots; b_n)$, and b_i is the state of component i . The phase space vector $(\mathbf{B}; t)$ indicates that the system entered the state vector \mathbf{B} at time t . The problem could be depicted as that of a “particle” moving in a phase space of states

and time. The behavior of systems is governed by an underlying transport equation.

The system transport kernel is the product of the free flight kernel and the collision kernel: $K(\bar{B}', t' \rightarrow \bar{B}, t) = T(\bar{B}'; t' \rightarrow t) \times C(t; \bar{B}' \rightarrow \bar{B})$.

The free flight kernel $T(\bar{B}'; t' \rightarrow t)$ is defined as the probability density that a system which entered state \bar{B}' at time t' will have a next event at time $t > t'$. The collision kernel, which is also called event kernel, $C(t; \bar{B}' \rightarrow \bar{B})$ is defined as the probability that upon an event at t in state \bar{B}' the system will change its state into \bar{B} .

Let $\{\bar{B}'\}$ be the set all state vector \bar{B}' from which it is possible to transfer to \bar{B} in a single event. The event density

$$\psi(\bar{B}, t) = P(\bar{B}_0) \cdot \delta(t) + \sum_{\{\bar{B}'\}} \int_{\tau} \psi(\bar{B}', t') \cdot K(\bar{B}', t' \rightarrow \bar{B}, t) \cdot d\tau$$

If we recall that for a system with n components, each of which has k_i different states, the dimension of the phase space would be $\prod_{i=1,n} k_i$. If the order of events is important, we will have to consider $n! \prod_{i=1,n} k_i$ different situations. The state explosion makes the analytical solution of the transport equation prohibitively difficult. The Monte Carlo simulation is almost only feasible solution.

We note that here, the state vector is defined as finite set of discrete states of the components, and that the continuous process variables were not considered. The

extension to account for continuous-time process variables can be found in many literatures. (Labeau & Zio, 1998; Smidts & Devooght, 1992; Tombuyses, DeLuca, & Smidts, 1998)

2.3.6 Discrete Event Simulation

Discrete Event Simulation is based on concept the concepts of state, events, activities and processes (Carson, 2004). In strict definition of discrete event simulation, the model state changes only at discrete times. Events represent the instantaneous change of system state. When an event occurs, it may trigger new events, activities and processes. Between events, the system is considered to be deterministic. A process is a sequence of events, activities and other time delays associated with one entity as it flows through a system.

Extension to treat continuous process variables is needed in most risk analysis problems (Dang, 1998). It is considered the evolution between events is deterministic. Like the Monte Carlo approach, this is implicit state-transition methodology, where there is no need to enumerate the possible system states, and the possible transition rates.

According to Siu, discrete event simulation has higher ability to deal with arbitrary complex systems than Markov analysis, DDET, or analog Monte Carlo (Siu, 1994).

2.3.7 Improvements to Simulation Methods

Despite all its advantages, the Monte Carlo and discrete event simulation methods display some drawbacks when applied to DPRA problems. When applied to dynamic PRA, due to the high reliability of most systems, the required number of simulation runs may become extremely large and impractical. Much research work has been conducted on using biased sampling for rare event simulation. In rare event simulation, great care must be taken to address the completeness of the search of scenarios. Another drawback is that the simulation results are hard to analyze. Various attempts have been made to tackle these problems. There are two classes of methods to improve the simulation efficiency. One class aims at improving the sampling efficiency, with the help of advanced biasing techniques. Thus we can get statistically better result with fewer runs of simulations. The other class attempts to accelerate the simulation speed, which will reduce the time consumption of the single run the simulation.

1. Biasing Techniques

a. Importance Sampling

Importance sampling is a standard variance reduction technique of Monte Carlo simulation. Consider the situation where we attempt to estimate:

$$\ell = E\{L(\bar{X})\}, \text{ where } \bar{X} \text{ is a random vector with independent components and the}$$

sample functions. An unbiased estimator of ℓ is: $\ell = \frac{1}{N} \sum_{i=1}^N L(\bar{X}_i)$. This estimator is

sometimes referred to as crude Monte Carlo estimator.

Let $G(\bar{y})$ be a cumulative distribution function such that $dG(\bar{y}) = g(\bar{y})d\bar{y}$, and $g(\bar{y})$ is a PDF for $\bar{y} \in \mathfrak{R}^n$. The density function $g(\bar{y})$ is called the *importance sampling biasing distribution*. Rewrite the target function:

$$\ell = E\{L(\bar{X})\} = \int L(\bar{x}) \cdot f(\bar{x}) d\bar{x} = \int L(\bar{x}) \cdot \frac{f(\bar{x})}{g(\bar{x})} g(\bar{x}) d\bar{x} = E_g \left\{ L(\bar{x}) \cdot \frac{f(\bar{x})}{g(\bar{x})} \right\}$$

We can form the *importance sampling estimator* as: $\ell = \frac{1}{N} \sum_{i=1}^N L(\bar{X}_i) \cdot W(\bar{X}_i)$;

where $W(\bar{x}) = f(\bar{x})/g(\bar{x})$. Here we can see that $g(\bar{y})$ should never be zero for any value of x , if $p(x)$ is positive. Mathematically, it implies that the support of $g(\cdot)$ must include the support of $f(\cdot)$. However, there would only be a problem where $L(x)$ is nonzero and the ratio of $f(\cdot)$ and $g(\cdot)$ is infinity, thus the requirement is:

$$\text{sup port}(f \cdot L) \subset \text{sup port}(g \cdot L)$$

The likelihood ratio $W(x)$ can be interpreted as a “correction factor” or “statistical weight” necessitated by the change of measure from f to g . To minimize the variance

of the estimator, with respect to the pdf g , that is: $\min \text{var}_g \left\{ L(\bar{X}) \frac{f(\bar{X})}{g(\bar{X})} \right\}$.

It has been shown that the minimizing g is given by: $g^*(\bar{x}) = \frac{|L(\bar{x})| \cdot f(\bar{x})}{\int |L(\bar{x})| \cdot f(\bar{x}) d\bar{x}}$.

If $L(x) > 0$, then $g^*(\bar{x}) = \frac{L(\bar{x}) \cdot f(\bar{x})}{\ell}$, whence $\text{var}_g \left\{ \frac{L(\bar{X}) \cdot f(\bar{X})}{g(\bar{X})} \right\} = 0$

The main difficulty of finding the optimal importance sampling is that knowledge of g^* implies knowledge of $\ell = E\{L(\bar{X})\}$, which is precisely the quantity we wish to estimate. The fact that the analytical expression for the sample performance L is unknown in most simulation may worsen the situation. The construction of g^* is complicated and time-consuming, especially when g^* is a high-dimensional pdf.

The biasing techniques introduced later in this chapter can be viewed as implementation of the importance sampling techniques. Only a few of biasing techniques are reviewed here. More general discussion of variance reduction techniques and rare event simulation can be found in (Bucklew, 2004; Rubinstein & Melamed, 1998)

b. System-based vs. Component-based.

Labeau et al. (Labeau & Zio, 2002) compared the indirect Monte Carlo method and direct MC methods. The system-based, indirect MC is derived from the transport analogy we discussed above. First the next transition time for the whole system is sampled, and the transition of system configuration is determined. New system

configurations can be analyzed and compared to a cut set. The component based, direct Monte Carlo method samples the next transition time of every component, and the earliest transition is chosen as the actual one.

Theoretically, if the individual components properties are known, the probability distribution function of the next system transition can be deduced. The sampling from such PDF is complex, especially when the system size grows larger, and the exponential assumption for the individual components is lifted. In such cases, the CB approach is more straightforward and allows for time dependencies and component interactions. A special case where the component failure time pdfs are analytical invertible is investigated, and the component based sampling turns out to be more efficient.

c. Biasing Toward Top event

Marseguerra et al. have discussed the Monte Carlo methods to estimate the reliability and availability of a complex system (Marseguerra & Zio, 1993). It is recognized that in the analog Monte Carlo many simulated histories do not yield much information. In order to improve the computational efficiency, they introduced a new biasing technique, which aims at driving the system toward a cut set configuration, which is more interesting but highly improbable. The concept of ‘distance’ between current system configuration and the ones pertaining to the cut-

sets was introduced to get a variance reduction technique. All possible transitions of the system were classified according whether the transition would bring the system closer or farther to the top-event (cut-sets). By doing this, the biasing techniques favor not the failures, but the transitions which lead toward top event of the fault tree.

d. Exponential biasing vs. Uniform biasing

In (Marseguerra, Zio, & Cadini, 2002) the Biased Monte Carlo simulation of time-dependent failure is discussed. It is shown that if biased sampling from exponential failure rate may result in a distorted estimate if the sampling failure rate is too high, on the other hand, if the sampling failure rate is too low, the result may have large variance. Sampling from a uniform distribution can get a better distribution throughout the mission time. They found that the sampling from a discrete uniform distribution generates “extremely satisfactory” results. The conclusion was drawn based on case studies of small-size problems. Whether the conclusion still holds with larger systems needs further investigations.

2. Accelerating the Numerical Simulation

The computation of system dynamics is time-consuming. Simplified dynamics is alternative to detailed numerical calculation. Trained artificial neural networks provide a fast approximation of the system dynamics. Artificial Neural Networks is able to model the non-linear system behavior with sufficient accuracy and substantial

reduction in computing time. However, the fact that there is no general way of training an artificial neural network limits the application of this method. (Chatelet, Zio, & Pasquet, 1998; Marseguerra, Masini, Zio, & Cojazzi, 2003; Marseguerra, Zio, Devooght, & Labeau, 1998)

Another way is the memorization-based methods. This approach starts with memorizing the system trajectories prior to simulation. This is achieved either by computing a grid of discretized process variable space (Marseguerra & Zio, 1995), or just memorizing the faultless trajectory. (Labeau & Zio, 1998)

2.4 Summary

The Dynamic Probabilistic Risk Assessment (DPRA) methodology has been evolving in the last two decades. DPRA methodologies are capable of handling interactions between components and the process variables, they provide more realistic modeling of the dynamic systems for the purpose of risk analysis. There is a growing recognition in the risk community of the potentials of these methods. Discrete Dynamic Event Tree and Monte Carlo simulation are two classes of methods that have been widely used. In next section we will focus on these two classes of methods, and propose a new methodology for Dynamic Probabilistic Risk Assessment.

3. DPRA as Exploration of Event Sequence Space

3.1 Introduction

A primary goal of DPRA is to identify vulnerabilities of the system, which is achieved by simulating a variety of sequences of events that are representative of all possible behaviors of the real system. The event sequences typically share a single initial condition, but are varied by introducing possible deviations. Such deviations, which may happen at various times, may be caused by hardware and software failures, as well as human actions. The set of simulated sequences is then analyzed to gain insight into events leading to undesirable end states, and their likelihood.

In the problem setting, we assume we understand the rules (e.g. physical laws) governing the system evolution, so that we can build a simulation model which represents the behavior of the system under different circumstances. Due to the large scale of realistic large systems and the complexity of their internal and external interactions in the systems, we cannot predict the system evolution for sure, even when we are equipped with a good understanding of all the underlying laws. The simulation model, which includes hardware, human crew and software, is a combination of deterministic and stochastic models. The deterministic (mathematical)

models are used to simulate the behavior of a system and physical processes taking place within the system, as a function of time. The stochastic elements are typically used to represent such events as the random failure of hardware systems and instruments, as well as the uncertain actions of human operators.

The occurrence and timing of the random events are controlled by the simulation program. In between the points of occurrence of these random events, the behavior of the system is modeled by deterministic models describing the physical and other processes taking place in the system.

Typical “stopping conditions”, i.e. absorbing states, for a sequence are mission time or the attainment of some condition in the system. Based on the stopping condition, the sequence of events can be classified as belonging to one of a set of predefined end states, representing the type and severity of the outcome of the particular event sequence. These end states are determined by the simulation model based on the process variables and/or the component states. When the sequence is completed, the analysis will continue by the simulation of another sequence.

The manner in which the event sequences to be simulated depends on the type of DPRA method. The Discrete Dynamic Event Tree methods e.g. (Cojazzi, 1996; Hsueh & Mosleh, 1996; Nivolianitou et al., 1986) systematically explore a large number of scenarios by introducing, at set points in time, branch points whose branches represent distinct courses of events, thus leading to distinct sequences of

events. The event sequences are usually explored using a depth first or breath first approach, and the analysis terminates when all the event sequences are exhausted.

Error! Reference source not found. illustrates the structure of DDET.

In another class of methods, the Continuous Dynamic Event Tree (CDET) methodology (Devooght & Smidts, 1992; Smidts, 1994), event sequences are randomly explored in the space of all possible event sequences. The exact manner in which sequences are generated varies. The analysis is terminated after a predetermined number of scenarios, or when some statistical objective is met.

The CDET class of methods is typically used to obtain estimates of system failure probabilities. Given that many of the stochastic elements in the system model represent rare events, the applicability of these methods depends heavily on the use of variance reduction techniques such as “importance sampling” (Campioni, Scardovelli, & Vestrucci, 2005; Dubi & Gerstl, 1980; Labeau & Zio, 2001; Labeau & Zio, 2002; Marseguerra & Zio, 1996; Marseguerra et al., 2002; Tombuyses et al., 1998).

The discussion of importance sampling techniques can be found in 2.3.7. In principle, the quality of importance sampling depends heavily on the understanding the problem. The zero variance is achieved when we understand the problem perfectly. In literature the biasing techniques used for DPRA focus on simply accelerating the failure at component level. One of the few, if not only, exceptions is (Marseguerra & Zio, 1993).

3.2 *Characterization of the Dynamic PRA Process*

The previous section described dynamic PRA as the analysis of risks and vulnerabilities by means of the simulation of many event sequences. The event sequences are generated by controlling the occurrence, and timing of stochastic elements in the model, such as hardware failure and human actions. In between the points of occurrence of these random events, the behavior of the system is typically modeled by the deterministic models. This description of the process applies to Dynamic PRA frameworks such as (Acosta & Siu, 1993; Cojazzi, 1996; Devooght & Smidts, 1992; Hsueh & Mosleh, 1996).

In this section, we present a characterization of Dynamic PRA that was developed in order to support the development of DPRA algorithms.

3.2.1 **Basic Terminology**

We first consider some of the basic terminologies that will be used throughout the discussion in this dissertation.

Model: an abstraction of the real-life system. Models are used to obtain predictions of the behavior of real system, especially how one or more changes in various aspects of the modeled system would affect the other aspects of the system.

Time: $T = \mathbb{R}^+$

System Configuration: It is assumed that all the components in system have only finite number of states. The system configuration, which is determined by the component states, can be indexed by a positive integer $C = N$.

System Status: System status includes both continuous process variables, (a real number vector \bar{X}) and discrete system configuration (a positive integer i). It is defined on $S = \mathfrak{R}^n \times N$

The process variables are governed by a set of deterministic equations

$$\frac{d\bar{x}}{dt} = \bar{f}_i(\bar{x}), \bar{x}(0) = \bar{x}_0, \bar{x} \in \mathfrak{R}^n. \text{ These equations are implied by the model. The explicit}$$

expression of the equations may not be available for all aspects of the system behavior.

Event: following the convention of discrete event simulation, an event is defined as an instantaneous occurrence that changes the system configuration (Carson, 2004).

There are two kinds of events. The event is defined as transition of system configuration from state i to state j at time t . $\delta_{int} : C \rightarrow C$

- **Random Events** are the events whose occurrences are depicted by a stochastic model and can be controlled by the simulation environment. Such events are not necessarily induced by the behavioral rules of the simulation model. An example of random events is a time distributed

component failure modeled by the Weibull model.

- **Deterministic Events** are induced by the deterministic rules. An example of deterministic events is that a threshold pressure or temperature is reached.

Event Sequence: a system trajectory, generated by the simulation model. It consists of a sequence of events, with deterministic behavior in between. Every event sequence should be unique. It is an instance of the system status evolution through time line. $ES = \delta_{\text{int}} \times T$

Event Sequence Space: the set of all possible event sequences. The definition of the event sequence space is implicit, i.e. follows from the definition of the simulation model. Event sequences in an event sequence space are considered to be mutually exclusive, even though they may partially overlap, since they are assumed to originate from a single initial state of the system. $SP = \{ES_i\}$

Sequence Generation: the process of simulating one or more event sequences, equivalent to the, possibly random, drawing of realizations of event sequence from the event sequence space.

Scheduling: the process of controlling the generation of event sequences. It is done by deciding on the occurrence and timing of the random events in the model.

Branch Point: a point in the simulation of the system at which the occurrence of

a random event is considered by the algorithm controlling the simulation. Each branch point will have two or more branches, corresponding to occurrence of possible events.

Scenario: a simplified representation of a group of event sequences with some common features. These features concern the (non-)occurrence, and possibly the timing, of events. The sequences belonging to a scenario are therefore considered to be similar, to the extent that they share the features implied by the scenario.

End State: a classification of the condition of the system at the end of an event sequence. It is an absorbing state of the simulation. Within the context of (D)PRA, end states are normally specified as one of the discrete end state types, which typically indicate the severity of the condition.

Some of these concepts are illustrates in **Error! Reference source not found.**

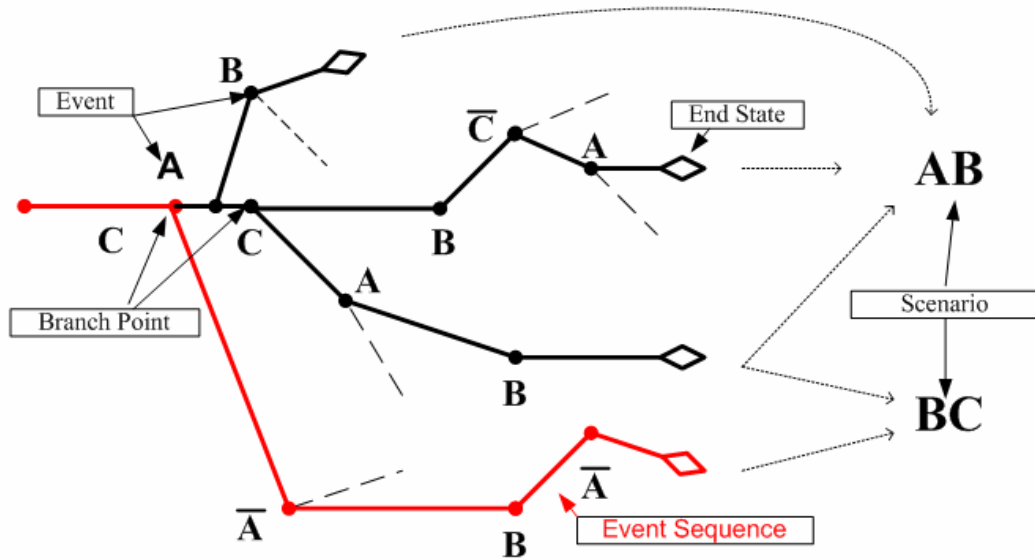


Figure 3.2.1 Illustration of DPR Terminology

3.2.2 Event Sequences vs. Scenarios

As part of our characterization of DPR processes, a distinction is introduced between event sequences and scenarios. An event sequence has been defined as a system trajectory, possible within the behavioral rules of the executable simulation model, consisting of a unique sequence of random events, with deterministic behavior in between.

The term scenario is referred to as a group of event sequences with some common features. These common features concern the (non-)occurrence, and possibly the timing or time ordering, of events. The sequences belonging to a scenario are

therefore considered to be similar, to the extent that they share the features implied by the definition of that scenario.

The difference between an event sequence and a scenario is therefore that the scenario is not fully specific about the occurrence, timing, or time ordering of events taking place during the event sequences, which allows for the grouping of the sequences, as well as a simplification of their representation. In case a scenario is defined solely based on the combination of event occurrence, ignoring all timing, Boolean expressions can be used to describe it.

A last simplification of event sequence descriptions, as implemented by scenarios, is the grouping of events that are considered equivalent. Particularly when redundant system elements are considered, it may be sufficient to know that one of the redundant systems encountered an event, e.g., a failure, whereas the particular system that encountered the event is irrelevant. Leaving out the unnecessary details leads to a simplification of the description.

Descriptions of scenarios involving the timing of events can be achieved using, for instance, temporal logic (Shults & Kuipers., 1997). These types of expressions allow combinations, or the sequences, of events to be described with their temporal order and characteristics. These logics provide a formal basis for scenario specification. Based on these logics, set operations can be defined over the space of event sequences, which will be used in the development of exploration algorithms.

Note that dynamic PRA approaches do not always implement a clear separation between event sequences and scenarios. For instance, Discrete Dynamic Event Trees, mentioned in a previous section, consist of a collection of event sequences that are represented in the form of a tree structure, reflecting the systematic fashion in which the event sequences were generated. However, the event sequences are implicitly taken as being representative of groups of event sequences. This is evidenced by the fact that finite probabilities are assigned to the event sequences, even though the probability of occurrence of individual event sequences is infinitely small. The probability of a scenario, i.e. the combined probability of groups of event sequences, can however assume finite values.

The removal of details regarding the occurrence and/or timing of events from analysis is a common practice in the classical Probabilistic Risk Assessment approach. First, the fault tree and event tree models used to represent the system failure logic generally do not represent time. Second, cut-set analysis can be interpreted as a way of finding the least specific description of combinations of events that are expected to result in the failure or an end state of the system.

A distinction is however that in classical PRA applications, cut-sets are specifically associated with a failure of the system. In contrast, no direct association is assumed between scenarios and a particular end state. Scenarios are not necessarily sufficiently specific to imply that all sequences would result in an identical end state.

Therefore, the end states of the event sequences belonging to a given scenario may well be different.

As we will see, the uncertainty about the end states of sequences belonging to a scenario will play an important role in the specification of rules for the exploration of the event sequence space. The term ‘outcome’ will therefore be used to distinguish between the end state of an event sequence, and the variation of end states of sequences belonging to a scenario.

Table 3-1 Comparison of Event Sequences and Scenarios

	Event Sequence	Scenario
Detail of Representation	Complete detail	Less specific
‘Consequence’	Uniquely defined end state	Possibly uncertain outcome
Probability	Not Defined	Defined

3.3 Exploration of the Event Sequence Space

As defined in a previous section, an event sequence space is the set of all possible event sequences. Each sequence then represents a unique combination of timing and occurrence of the events. This way it is possible to conceive of many, if not an infinite number of, event sequences originating from a single starting condition. Figure 3.3.1 illustrates the concept of event sequence in a probabilistic event

sequence space.

The process of generating event sequences by simulation can be viewed as an exploration of the event sequence space. The objective is to identify how the sequences lead to an undesirable end state, as well as an estimate of the probability that such sequences be realized.

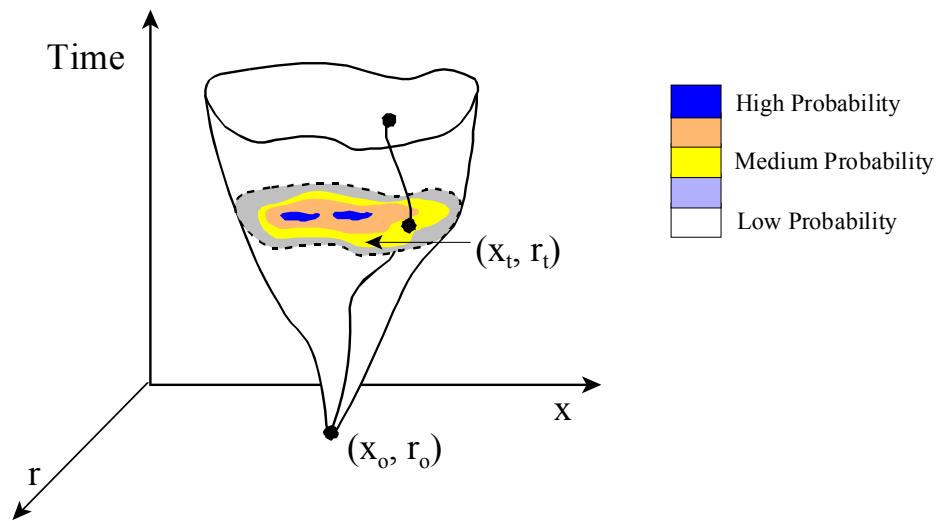


Figure 3.3.1 An event sequence in the probabilistic event sequence space.

At the start of the exploration, we know the behavioral rules of the physical processes, the software logic, and the operators that make up the system, as well as the starting system condition from which the simulation of sequences is initiated (boundary conditions and initial conditions). The actual specification of sequences then takes place by varying the occurrence and timing of the random events in the model. Doing so is the responsibility of a separate element in the simulation

environment that is referred to as the scheduler, consistent with the terminology used in the ADS and DYLAM DPRA environments (Hsueh & Mosleh, 1996).

The exploration/scheduling algorithm, which is separated from the numerical simulation model, decides how the event sequences are generated, generally by controlling the occurrence of the random events in the model.

Simply taking all combinations of all possible events may not give the correct extent of event sequence space, as the event sequence space is constrained by the behavioral rules of the physical processes, the software logic, and the operators' actions. These constraints determine if and when certain events, or combinations of events, can take place.

A combinatorial approach towards the scheduling of the simulation, analogue to the combinatorial design of experiments (Mason, Gunst, & Hess, 2003), is therefore not directly possible, as many combinations of events are rendered impossible by the rules that model the actual behavior of the system, and the possibility of particular combinations of events can not be anticipated in advance in many cases.

Instead, current DPRA frameworks largely rely on two strategies that will be referred to as systematic and random exploration.

3.3.1 Systematic Exploration

Systematic exploration is implemented by frameworks such as ADS and

DYLAM. The systematic approach operates by considering, at discrete points during the simulation, which events could possibly happen at that point in time, and to systematically explore, i.e., simulate, each of those options. The result of this strategy, which is typically implemented using a depth-first traversal of the event sequences, is a tree structure corresponding to the repeated branching of the event sequences, which is referred to as the Discrete Dynamic Event Tree (DDET).

While this approach has the advantage that shared sections of the event sequences only need to be simulated once, it is computationally demanding since the repeated branching of sequences leads to a combinatorial explosion of the number of sequences. Stopping the exploration before it is complete, after the generation of a predetermined number of sequences, or predetermined amount of computation time, would leave part of the sequence space untouched by the exploration.

The combinatorial explosion is typically counteracted by limiting the number of time points at which branching of the sequences can occur, as well as by stopping the simulation of sequences as soon as their likelihood of occurrence falls below a preset threshold value. It is known that these control measures introduce a bias in the risk estimates, i.e. sequence probabilities. (Smidts & Devooght, 1992).

The discretization of the event sequence space may affect the credibility of the results in other ways as well. By discretizing the event sequence space, single event sequences are taken as representative of entire groups of event sequences. For

example, a sequence involving the occurrence of an event at a predetermined point in time is taken as representative of any sequence involving the occurrence of the event sometime during an interval. While this does lead to a reduction of the computational load by allowing reuse of simulated segments, it affects the ability to identify vulnerabilities of the system, since it does not allow the sensitivity to small changes in the sequences, such as the timing of events, to be identified. This while, depending on the type of system, the end state of an event sequence may be highly sensitive to the particulars of the sequence, an analogy to one of the basic concepts behind Chaos Theory. The discretization of the sequence space makes it hard to detect such sensitivities.

In the implementation of systematic explorations (DYLAM, DETAM, ADS), the discrete time points at which the branching may take place is predefined, and in many cases, by fixed time step. There is always a trade off between precision and computation cost when setting the time points. Too many branching time points would not only increase the computational cost but also make the resulting DDET too large to manage. Situation may grow more difficult, if the simulated mission time is long. In the cases where automatic control and safety devices are ubiquitous and in critical situations the response time would be crucial, a predefined time step may be inadequate.

3.3.2 Random Exploration

Random Exploration is an alternative strategy and is less sensitive to combinatorial explosion. It does not involve the discretization of the event sequence space. This type of exploration consists of Monte Carlo experiments in which event sequences are randomly generated by randomly deciding on the occurrence and timing of events.

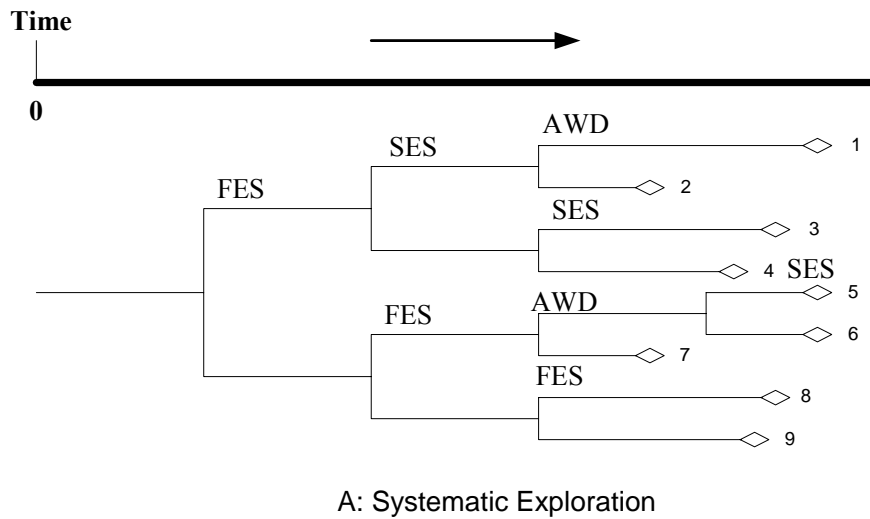
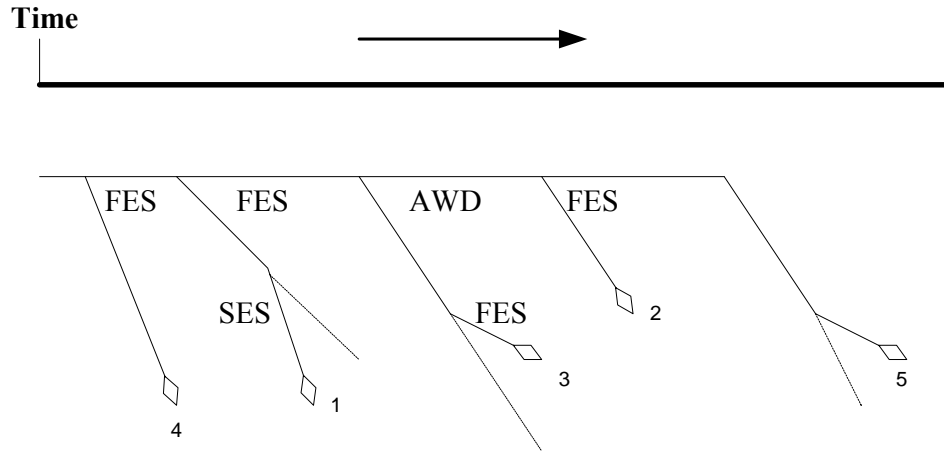


Figure 3.3.2 Illustration of Systematic Exploration



B: Random Exploration

Figure 3.3.3 Illustration of Random Exploration

Events are allowed to take place at any time and not just at predetermined points as is the case in systematic search. The exploration can in principle be stopped after any numbers of generated sequences, even though a higher number of sequences will generally improve the statistical quality of the results, e.g. reduce the variance in probability estimates.

DPRA approaches based on a random exploration strategy (Dubi, 1998; Labeau, 1996; Smidts & Devooght, 1992) typically apply variance reduction techniques such as importance sampling (Campioni et al., 2005; Labeau & Zio, 2001; Labeau, 1998; Labeau & Zio, 2002; Marseguerra & Zio, 2000; Marseguerra et al., 2002). These techniques usually rely on forcing the simulation of component-level failures, based

on the understanding that component-level failures will typically bring the system itself closer to a failure. As long as system-level failures of a system can indeed be interpreted as a combination of component-level failures, and component-level failures are by nature a rare occurrence, the acceleration of component-level failures will result in an increase in the number of sequences ending in system failure. This in turn may help improve the system failure probability estimates.

It is recognized that the biasing techniques have to be designed with great care. (Marseguerra et al., 2002) showed that poorly designed biasing technique may lead to distorted estimate. If the system has several competing failure modes, and /or the system is non-coherent, the biasing toward component failure strategy should be designed with even greater care. Marseguerra et al. attempted to take the fault tree into account (Marseguerra & Zio, 1993). Instead of simple favoring component failure, they tried to favor the events leading to a cut-set of the fault tree.

The random exploration approach is less equipped however to support another objective of the DPRA process, namely the identification of vulnerabilities in the system that could bring the system to an undesirable state. In a random exploration sequences are generated largely independent of each other: each time a new sequence is generated, it is done without consideration of sequences that have already been generated. Consequently, the sequences can not easily be organized in a structure like the discrete dynamic event tree.

The storage of details on the nature of the generated sequences is therefore not practical, and generally does not take place. This makes it hard to perform such tasks as the identification of classes of events with a major contribution to system risk.

4. Enhanced DPRA Framework

4.1 *Problem Statement*

As discussed in chapter 3, DPRA problem can be interpreted as exploration of the space of event sequences to discover the system vulnerabilities and provide an estimate of the likelihood, if possible. In the realistic high reliability system, risk scenarios are rare, and the exploration scheme may spend a lot of time in the areas which do not give the risk analysts much insight into what may lead to undesirable conditions. The exploration strategy should be able to avoid such areas where provide little information (**efficiency**). The effort should also be distributed fairly between all risk scenarios (**fairness**). Spending most of the effort in one scenario and leaving other scenarios unexplored or explored only few times is undesirable. At the same time, we want the exploration to be able to cover all possible scenarios (**completeness**). No scenario should be left unexplored. The event sequences space is infinitively large in most cases. The practical objective is then to explore all representative scenarios, not all event sequences.

4.2 Adaptive Exploration

4.2.1 Traditional Exploration Strategy

The DPRA methods which we have discussed so far largely rely on mechanistic procedures such as systematic exploration and Monte Carlo experiments, where exploration are controlled according to preset rules. Little or no effort has been made to take into account the impact of events, or of combinations of events, in the context of the system's behavior.

It is acknowledged that not all parts of the event sequences space have the same importance from risk perspective. It is natural that we want to guide our simulation toward the parts which represent higher priority. DDET does not take this into account. Most biased Monte Carlo simulation tried this on a component level, based on the belief that the component failures would more likely lead to risk scenarios. The knowledge of the system, which is obtained by traditional PRA and reliability analysis, is seldom taken into account. One of the few exceptions is in (Marseguerra & Zio, 1993) where Marseguerra et al tried to drive the system towards the more interesting but highly improbable cut set configurations.

An active understanding of the meaning of observed behaviors and significance of control actions, as well as a dynamic adjustment of exploration strategies is a promising alternative to current strategies. So far it has not been discussed in

literature.

4.2.2 Bayesian Adaptive Exploration

Human brain behaves in an adaptive and self-adjusting way. We learn from experience incrementally, make decisions and adjust questions through the learning process. In the DPRA settings, what has been learned from past data could be used to alter the exploration strategy of future to more efficiently address the questions of interest. A general framework of adaptive learning procedure is depicted in Figure 4.2.1.

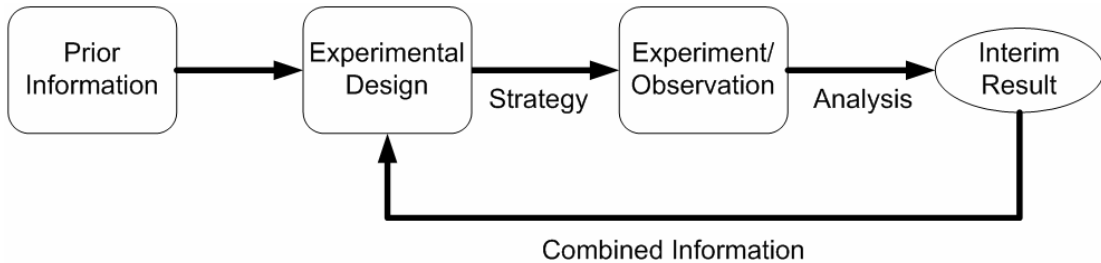


Figure 4.2.1 Adaptive Learning

The theories of experimental design have recognized that the use of partial knowledge can improve the design of experiments (Chaloner & Verinelli, 1995). The Bayesian experimental design is motivated by Bayesian decision theory and the fact that there is always information available prior to the experiment. The optimal design is made under uncertainty by enumerating the possible actions, and the possible outcomes, of which we are uncertain. Lindley introduced the idea of using

information as utility, while the information is quantified using information theory (Shannon, 1948). A utility $U(o, a)$ is assigned to action a if the outcome turns out to be o . The available information, I , implies the probability of observing outcome o , $p(o|I)$, then the expected utility associated with action a is

$$EU(a) = \sum_{i=1}^n p(o_i | I) \cdot U(o_i, a),$$
 where n is the number of all possible outcomes and

$$\sum_{i=1}^n p(o_i | I) = 1.$$
 The optimal action is the one that maximizes the expected utility. A

more formal way to express the framework can be found in (Chaloner & Verinelli, 1995; Lindley, 1972).

Inspired by the achievement of the Bayesian experimental design, and Bayesian adaptive exploration (Loredo, 2003), we believe similar procedure would help us in developing a new exploration strategy. Examples of such application are:

- The use of prior knowledge or expectations regarding the system's capabilities and behavior, and the source of such knowledge includes design documentations, such as requirements and specifications.
- The use of knowledge gained during the testing/ simulation itself, i.e., the results of test cases already performed, based on which the priorities during the remainder of the exploration can be dynamically adjusted.
- The use of knowledge regarding generic system vulnerabilities, such as

types of functions or operations that are more likely to introduce system failures.

These types of knowledge are therefore applied to actively and continuously decide on the priorities of running different types of test cases, to a large extent based on a sense of the extent of knowledge that can be gained by each of those test cases.

It is believed that the efficiency of DPRA procedures can be improved through the application of methods and techniques that duplicate or resemble this prioritizing behavior. The efficiency improvement would be achieved by making the exploration of the space of possible sequences more directed, and increasing the chance that the simulation of an event sequence is useful and provides insight into the behavior of the system.

4.3 Outline of a New DPRA Methodology

4.3.1 The Framework

Based on the view of DPRA simulations as an exploration process, it is believed that the DPRA can benefit from the incorporation of enhanced rules for the scheduling of sequence simulations. Here, scheduling is defined as the process of controlling the input to the simulation model, in order to stimulate the desired types of scenarios.

The new framework will replace the brute-force random and systematic exploration approaches with dynamic exploration rules modeled after the rules applied by an adaptive scheduler.

The enhancement would largely invoke two types of knowledge. First, the scheduling of sequences should take into account prior knowledge about the behavior of the system, which may include both system-specific information, such as the design of the system, as well as generally applicable information about the system or its elements. The experience of similar systems, near-miss incidents previously observed, common vulnerabilities of similar systems, are examples of this type of knowledge. Traditional Event Tree/ Fault Tree analysis normally try to capture such prior knowledge. Due to their highly abstract nature, ET/FT analysis is often inadequate to model complex dynamic behavior of the system. However such models and knowledge they embody could still serve as a guide of the simulation.

The simulation model includes some of the rules which dictate the system evolution. Such rules may be initially abstract, and it is possible to refine the model to represent a better understanding of the rules later. We shall bear in mind that the level of abstraction can evolve in time, when we obtain a better understanding the system, or at a later stage of the system development. Correspondingly, we should be able to have a more detailed model. The scheduling rules should also take this into account.

The second type of knowledge is what obtained from the generated event

sequences during the simulation. The results obtained from the simulated sequences should be applied to, adaptively, modify the focus of the exploration. In Figure4.3.1, the inclusion of the new sources of information is indicated by the highlighted arrows.

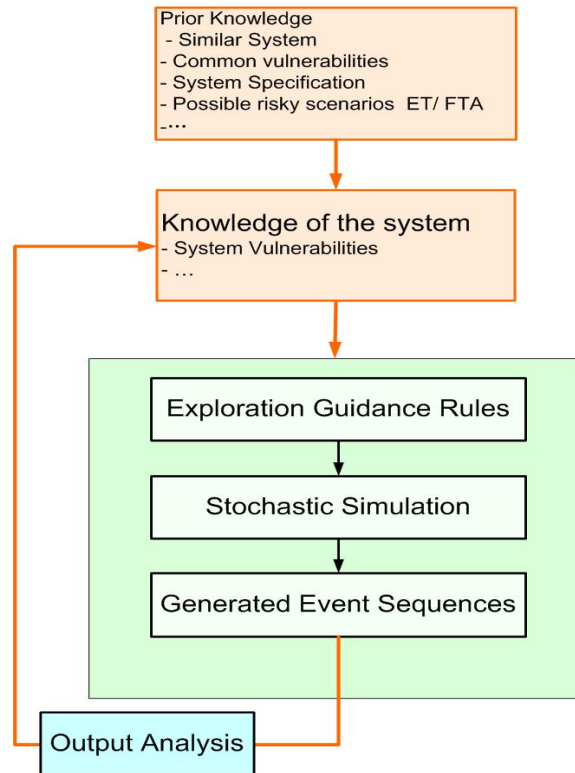


Figure4.3.1 Proposed enhanced usage of the information in the DPRA exploration.

Therefore the new framework emphasizes the capability of the simulation to be adaptive and self-adjusting, which is inspired by the analogy to human reasoning process.

4.3.2 Key Elements

The internal structure of the new framework consists of:

- A planner that generates a plan which is an interpretation of the knowledge of the system as an initial list of scenarios of interest. The plan serves as a map for exploration.
- A “scheduler” that manages the exploration process, including saving the system states, and restarting the simulation. The scheduler will guide the simulation toward the plan generated by planner.
- A simulation model of the system.
- Output analysis, which analyzes the event sequences generated by the simulation, and may update the plan if needed.

The knowledge of the system vulnerabilities can be expressed as a list of scenarios which may lead to undesirable end states. Recalling the definition of scenario, the scenario does not have to be complete event sequences. There is no requirement that the list of scenarios would cover all the event sequence space. In event tree analysis, the complete event sequences need to be laid out by the analyst, and it is essential to accurately cover all the event sequence space with the event trees. Fulfillment of this requirement of event tree relies on the expertise of the risk analysts, which is hard to verify, especially when dealing with new systems where we

do not have much relevant experience.

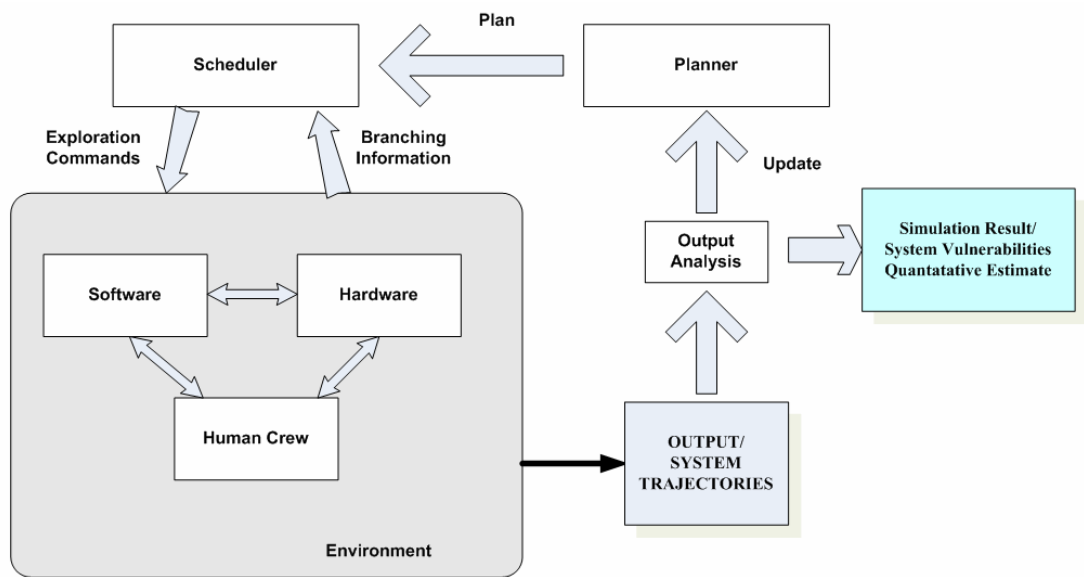


Figure 4.3.2 Framework of the New Model Based DPRA Platform

By contrast, in the proposed DPRA framework, the scenarios listed in the plan serve as a “guide” for the simulation. Due to the randomness of the simulation, it is believed that with a large number of guided simulation stories, all event sequences of interest would be touched. The “guide” is usually expected to be incomplete, even incorrect, information on which part of the event sequences has a higher priority to be explored.

The random elements of the simulation would be controlled by the scheduler. The objective of scheduler is to distribute the simulation effort among different scenarios. The scenario with high importance would be explored with higher priority, while all

other scenarios also have a chance to be simulated. Among the “important” scenarios, we want the scheduler to guide the simulation efforts evenly. Simulation focused on only one or few scenarios, leaving other important scenarios untouched, is undesirable.

Among the desired properties the scenario exploration should:

- Maintain sufficient coverage of important scenarios
- Guide simulation toward areas of greatest uncertainty
- Continuously adjust priorities based on simulated results
- Avoid test areas known to definitely lead to a specific end state.
- Simulation should be able to cover all the event sequence space.

4.4 Implementation of the Planner of DPRA Simulations

If we think of the DPRA as an exploration of event sequence space, the plan is the map to guide the exploration. In the map, the interesting scenarios are highlighted, and we want our simulation to explore such scenarios more often. We want to explore the whole map, and it is undesirable to miss any of the scenarios. As we are exploring the event sequence space, we may find out that the map is inaccurate in some place, or may miss some important scenarios. With these findings, it is possible for us to

update the map, and updating the map is one of the objectives of the exploration.

The plan is a list of scenarios toward which the risk analysts want to guide the simulation. Planner collects useful knowledge about the contributors to different classes of risk scenarios and generates the plan for the simulation. Planner gives guidance to the scheduler on how to reach the end-states of interest. It also receives some information from the simulator to update its knowledge. Finite State Machine (FSM), and Qualitative Differential Equations among other techniques have been used to generate the plans. The state transition diagram, if it is available, provides very useful information to generate a plan. The regular FSM presentation is extended so that the uncertain states and state transitions can be modeled as well. As stated earlier, the scenarios generated in the plan do not have to be accurate or complete. The simulation would update the knowledge of system, and in turn, may result in an updated plan, which is more accurate and more complete.

The planning methodology is being developed in a companion research effort, details of which are provided in (Mosleh et al., 2005).

4.5 Implementation of the Scheduler of DPRA Simulations

Aided by the plan the scheduler is responsible for guiding the simulation toward the more interesting scenarios. The scheduler will load the plan from the plan file, and store the plan in the scheduler. The scheduler will keep track of the simulation, and

guiding the simulation adaptively.

The simulation would propose the transitions (branches) to the scheduler whenever it comes to a branching point. The scheduler then retrieves the information of the proposed transitions, and decides which branch to explore. The exploration command is sent back to the simulation, and the simulation model would execute the command, and continue the simulation, until another branching point or end state is reached.

The scheduler aims at maintaining a fair distribution among the interesting scenarios, and at the same time, the scheduling is still random, and any transitions are possible, thus ensure that no scenarios would be ruled out. The details of the scheduler algorithm are in chapter 5.

The term “Scheduler” has been used in the implementations of DDET methodology, such as DYLAM (Cojazzi, 1996) and ADS (Hsueh & Mosleh, 1996), but the role of the scheduler is quite different in our framework. In DDET implementations the scheduler directs the simulation to perform the systematic traversal of all the possible branches, typically in a depth-first manner, and the scheduler does not make decisions of choosing branches. In our new framework, the scheduler not only is capable of performing the depth-first search as in the DDET, but also adaptively guides the simulation toward the scenarios of interest. The latter part is more important, and cannot be found in the “Scheduler” in DDET implementations.

4.6 Interactions Between Planner and Scheduler

4.6.1 Load Plan into Scheduler

At the beginning of each simulation, the scheduler will load the plan from the plan file generated by the planner. The plan file is text file, while list the scenarios line by line.

It is recognized before the simulation the knowledge of the system may be vague, abstract, incomplete and even inaccurate. The purpose of the simulation is to enrich the information of the system. Therefore, we cannot anticipate that the plan generated by the planner would be complete, detailed, or accurate.

The planer works with an abstract model of the system. The level of abstraction is usually not the same as the simulation model, as typically the plan captures knowledge at a higher (more abstract) level. For example, the scenarios generated in plan may include “engine failure”. In the simulation model, the engine is a complex sub-system, and there is no single event “engine failure”. The “engine failure” would be translated to one or more event sequences based on, for example, the fault tree type of model of the engine failure. The detailed scenarios are used by the scheduler to guide the simulation. The simulation model itself may be updated. We may get a more accurate and more detailed model of the component, and as a result, the event sequences corresponding to the same abstract scenario may change. This feature can

ease the burden of the planner. The plan generated from an abstract model does not have to be changed. Instead, we update the knowledge base of event sequences corresponding to each abstract scenario at different stage of modeling.

A database is maintained by the risk analysts to interpret the abstract scenarios generated by planner. When the scheduler loads the plan, if there are such abstract scenarios, the scheduler will query the database to get the detailed scenarios, and generate a detailed plan, which can be used to direct the simulation.

4.6.2 Update Plan Based on Simulation Result

As we have seen in Figure 4.3.2, the plan will be updated from time to time. There are several types of updating. The first type is automatic updating after simulating a specific number of event sequences. The planner will check the simulation results so far to determine how well the simulation is following the plan. If some scenarios have been underrepresented, the planner would automatically set the importance level of the specific scenarios to a higher level, which would in turn make the scenarios more favorable by the scheduler. The purpose of this adjustment is to maintain a exploration fairly distributed among different scenarios. This step may happen more than once in the simulation.

A second type of updating needs analysts' intervention. The result of simulation may disagree with the plan, for example, some scenarios have remained untouched.

The discrepancy is highlighted for further investigations.

Several different things may contribute to the discrepancy. It may be that the simulation model somewhat misrepresent the reality which make some scenarios impossible in the simulation. Or it may be that the plan is inaccurate, and needs to be modified. This is referred to as *spurious* scenarios in qualitative simulations (Berleant & Kuipers, 1997; Kuipers., 1986; Shults & Kuipers., 1997). A typical source of this situation is that some scenarios (paths) which appears in the state graphs are rendered impossible by the constraint of the physical reality. The qualitative reasoning part of the scheduler tries to identify such scenarios, and eliminate them from the plan. There is no guarantee that all spurious scenarios would be identified and eliminated (Say & Akin, 2003). Once there are such scenarios in the plan, no matter how hard the scheduler tried, no event sequences could be generated in these scenarios, since the simulation model is a representation of the physical reality. The output analysis may find such scenarios, and refine the plan accordingly.

Another reason could be an ill-designed sampling procedure. If the sampling distribution is heavily skewed, some scenario may never be sampled. In our implantation, we carefully design the sampling method to make sure that all the possible events have a chance to be sampled. The discussion of sampling is referred to branch point generating in our approach.

In any of the above cases, with the detailed discrepancy in the generated event

sequences and the planned scenarios, it should be easy to detect the error, and fix it.

5. Scheduler Algorithms

This section describes the methodology and algorithms of the scheduling of simulations. The objective is to develop the ability to adaptively schedule the simulation of sequences such that the simulation effort is guided toward scenarios of interest and is “fairly” distributed among the possible scenarios.

5.1 Problem Definition

We use the simulation to gain the knowledge of the evolution of the state of the system over time. There are several random elements in the system, which we cannot predict with 100% certainty. One kind of random events is that we cannot know for sure what will happen at a specific time. For example, the backup system may fail to start when there is a demand. We refer to this kind of random events as “demand-based” in this dissertation. The random events also can be time-distributed. They can occur within some time interval, but we do not know when. Sometimes the behavior of such events is characterized by their rate, or probability, of occurrence, which may be dependent on time as well as the state of the system. The occurrence of random events can influence the evolution of the system’s state.

The occurrence of random events is controlled from outside the simulation model. The responsible mechanism, referred to as the scheduler, guides the exploration of the sequence space by controlling the occurrence of individual random events according to some exploration strategy. The specification and implementation of the exploration strategy is the topic of this chapter.

Suppose that we have a list of interesting scenarios, which we want to investigate in detail. The objective of this study is to devise and implement a strategy for the exploration of the event sequence space that allows us to adaptively schedule the simulation of sequences such that the simulation is guided toward these scenarios and the effort is fairly distributed among them.

The scenarios are represented in the form of a tree structure. In the simplest case, these expressions specify the combination of single events, but more complex logic structures are also considered. The scenarios are not necessarily mutually exclusive.

The solution to the scheduling problem can be subdivided into two problems. First, the solution to the problem requires us to develop the ability to control the occurrence of individual events in such a manner that simulated sequences belong to a particular scenario. However, the repeated simulation of a single sequence belonging to the scenario of interest does not help in the assessment of the outcome of a scenario. The solution should therefore be able to ‘sample’ sequences randomly from the set of sequences that constitute the scenario.

Secondly, a “fairness” criterion must be formalized, and a corresponding rule for prioritization of the scenarios must be devised. The requirement that the scheduling takes place in an adaptive manner means that the fairness criterion should consider the results from sequences that have already been simulated. The scheduling mechanism should thus routinely revise its scheduling priorities based on the latest state of information.

5.2 Scheduler Overview

The scheduler is the procedure and software controlling the execution of a simulation model, and more specifically, deciding on the occurrence of non-deterministic events in the simulation model. The scheduler would accept a plan, which consists of a set of scenarios. Each scenario consists of an abstraction of a group of event sequences. Sequences are defined as specific realization of the simulated timelines. The objective of the scheduler is to generate sequences according with the plan and the defined importance levels.

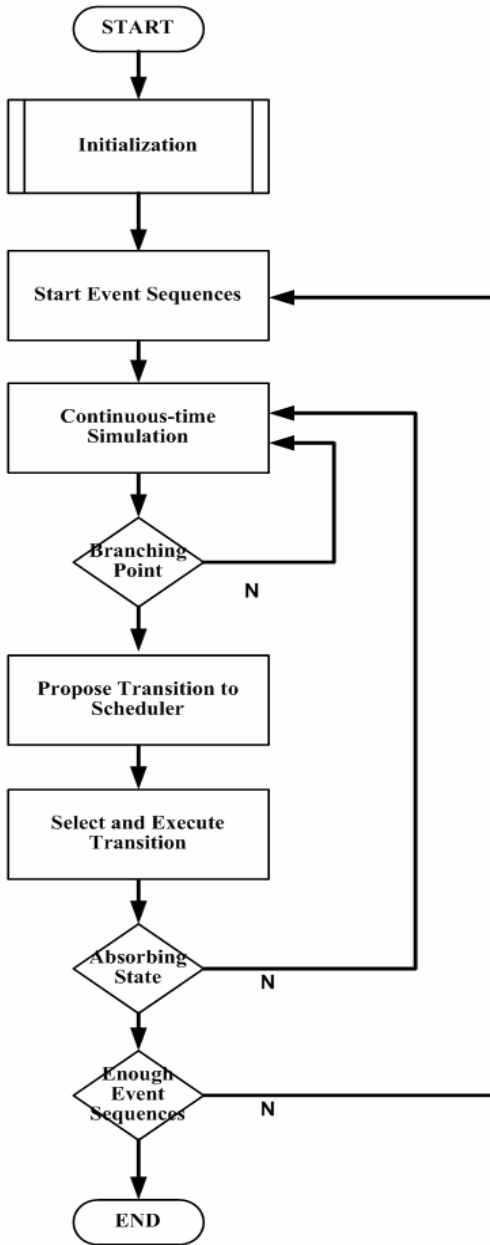


Figure 5.2.1 The Scheduler Framework

As the simulation starts, the scheduler and simulation model is initialized. In the

initialization, the first branching points conditions is calculated, and passed to the simulation model. Then the continuous-time (deterministic) simulation starts calculating the system trajectory based on the initial conditions. Once any one of the branching point conditions is reached, the simulation generates a branch point, which means that there are at least two branches possible. Each branch represents a possible stochastic event, for example, the occurrence and non-occurrence of a hardware failure. Another example is the human operator action, when there is a stimulus for the operator to take action, the operator may have several possible responses, each of which is represented as a branch. Branches are proposed events to the scheduler. The scheduler will check the current status of simulation, the plan, and previous simulation results. Based on the exploration rules, the scheduler will decide which branch to explore, and sends back the decision to simulation model. At the same time, the system state, criteria for next branching point and/ or the deterministic simulation model are all updated. This process is called “executing transition”. The simulation resumes, until it reaches next branching point or absorbing (end) state.

5.3 Representation of the Plan in Scheduler

The Event Sequence Space is represented by a tree structure. Each branch of the tree, which consists of a sequence of nodes, represents a scenario, which is a class of event sequences. This representation derives from the plan. It is the way we anticipate

the system to behave. We may be more interested in some branches than others, based what we have learned from previous experiences.

Each node represents a statement, of such types as

- Required event or events;
- Negation of some event or events;
- Events with time condition, e.g. happened in a specified time interval;
- Combination of several events.

The tree represents how the scenarios partially overlap. For instance, if two scenarios both define event A as a required event, the node specifying this will be shared by both scenarios.

The system evolution is depicted by the moving from one node to another. In order to keep track of the current state of the simulation, each node has a “status” variable, which is defined with respect to a single sequence, i.e., the status is reset at the start of every sequence. The status is used to track the progress of the simulation within the plan. So when the simulation reaches a certain stage in the plan that some certain nodes are ready to be explored, such nodes are marked as ‘arrived at’. At that time, when an event (or a combination of events) occurs that fulfills the requirement set by the node, the status of the node becomes ‘occurred’, and the immediate children nodes of that node becomes “arrived at”. At any stage of the simulation if

some nodes are no longer possible to explored, they are marked as “negated”. In this way, the scheduler is able to track the simulation in the plan.

To illustrate how this scheme works, let us look at the following example. Figure 5.3.1 demonstrates how the status of each node is changed during the progress. Each node is designated by a number. The nodes may have sub-nodes. The tree on the left side illustrates the current state of system evolution:

1. Node 1 is marked as “occurred”,
2. The other branches which are inconsistent with the current system state are “negated”; like node 2 and all the sub-nodes of node 2. By “inconsistent” we mean that those nodes can no longer be satisfied given current system state.
3. The immediate sub-node of node 1 is “arrived”
4. The nodes of the deeper levels are “not occurred”.

Note that we assume that nodes 1_1, 1_2, and 1_3 are associated with the event a, b, and c, respectively, which means, for example, if “event a” happens at this stage then the node 1_1 occurs.

Then, we assume that “event a” happens. The new system state is described in the tree on the right side of Figure 5.3.1. The occurrence of “event a” makes the node 1_1 status change to “occurred”. Consequently, the children nodes of 1_1 are “arrived”. The nodes 1_2, and 1_3 are no longer available and become “negated”. From the

illustration we see how the system moved one step forward.

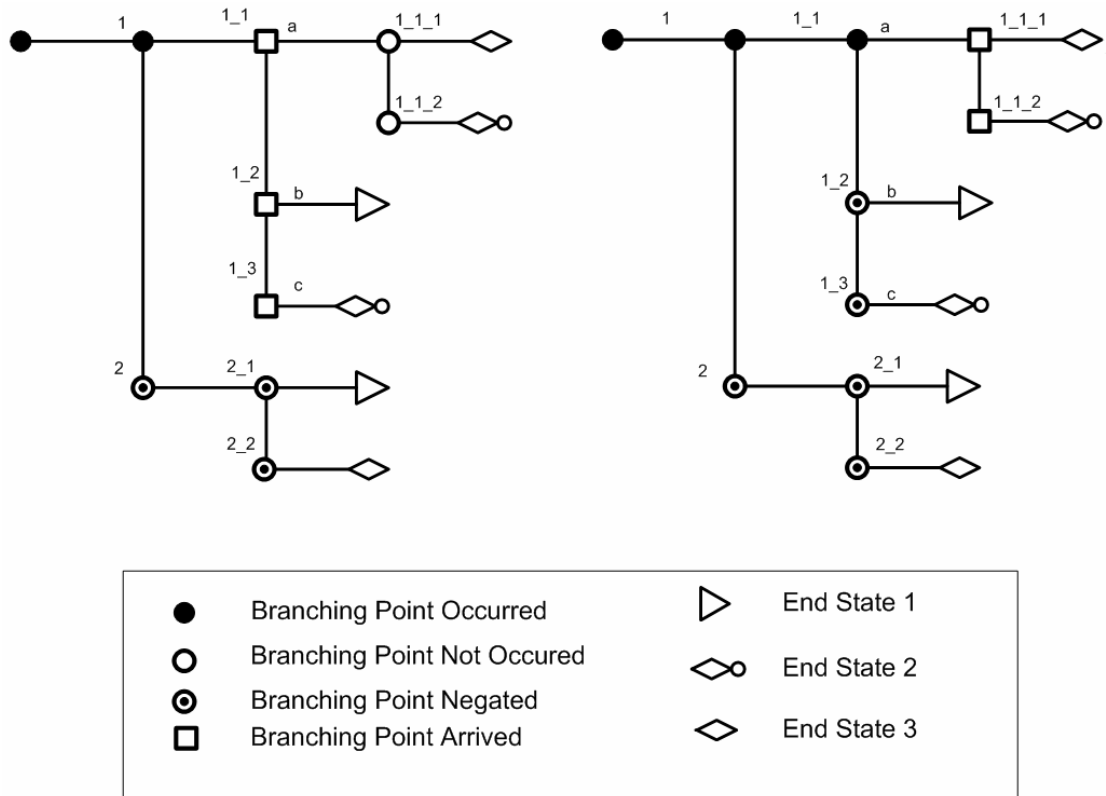


Figure 5.3.1 System Evolution

5.4 Branch Point Generation for Stochastic Events

We consider the rules for the generation of stochastic event sequence branch points in this section. The rules for branch point generation could be either probability-based or time-based.

5.4.1 Probability-based Branch Point Generation

This rule can be applied to events that occur with an intensity function $h(t,x)$, which may or may not be a function of time and the current state of the system.

The rule is applicable both to systematic search and random search event sequence space exploration schemes. In case of a systematic search, the simulation scheduler will, at each branch point, explore both branches corresponding to occurrence and non-occurrence of the event. In case of random exploration, the scheduler will (randomly) select one of the branches.

The rule divides up each simulated scenario in a number of subsequent time intervals, and generates one branch point in each of those intervals. The time intervals are chosen in such a way that, assuming that the event has not yet occurred, the conditional probability that the event takes place in a given interval is equal to ΔP . In mathematical terms,

$$\Pr(t < t'_k | t > t'_{k-1}) = \frac{\Pr(t'_{k-1} < t < t'_k)}{\Pr(t > t'_{k-1})} = \Delta P$$

where t'_{k-1} and t'_k are the bounds of the k -th time interval, t is the time of occurrence of the event, and

$$\Pr(t'_{k-1} < t < t'_k) = \Pr(t < t'_k) - \Pr(t < t'_{k-1})$$

The time of the k -th branch point, t_k , is chosen randomly within the k -th interval,

according to the system's time-to-occurrence distribution. Let u be a sample from the standard uniform distribution, then t_k is defined by the equation

$$u \cdot \Delta P = 1 - e^{-\int_{t'_{k-1}}^{t_k} h(\tau) \cdot d\tau}$$

which corresponds to the probability of occurrence of the event between t'_{k-1} and t_k .

The following is the algorithm that implements this rule. Let $H(t)$ denote the cumulative intensity at time t

$$H(t) = \int_{\tau=0}^t h(\tau) \cdot d\tau$$

Then the time t_k at which the k -th branch point is generated is given by

$$H(t_k) = H(t'_{k-1}) - \ln(1 - u \cdot \Delta P)$$

where

$$H(t'_k) = -k \cdot \ln(1 - \Delta P)$$

The last of these equations illustrates the fact that the choice of a constant conditional probability interval translates into a constant increment of the cumulative intensity function at the start of each interval.

$$1 - e^{-\int_{t'_{k-1}}^{t_k} h(\tau) \cdot d\tau} = \Delta P$$

Figure 5.4.1 illustrates the construction of the intervals for $\Delta P = 0.1$ and $h(t) = 0.01$. Since the intensity function $h(t)$ is constant, the intervals are of equal width.

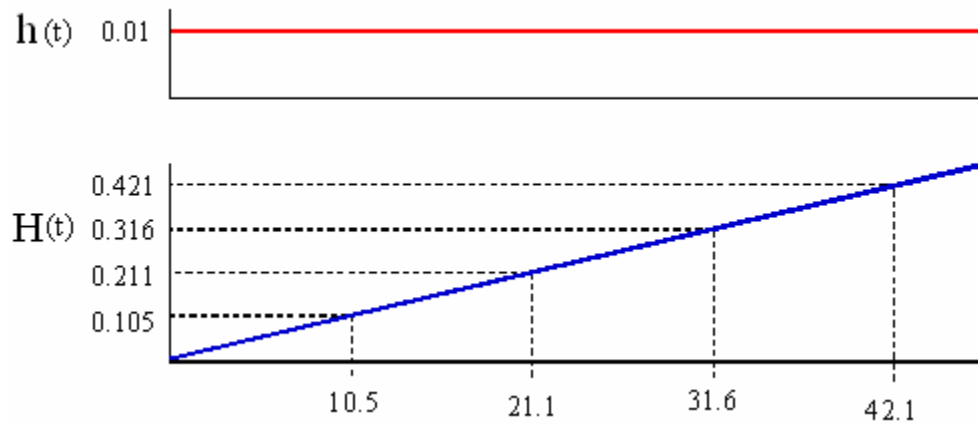


Figure 5.4.1: Construction of intervals.

The sampling of the branch point is illustrated in Figure 5.4.2. The horizontal axis shows the cumulative intensity function; the vertical axis represents the conditional probability of occurrence of the event. The example sample point $u \Delta P$ is converted into a value of the cumulative intensity at which the branch point should be generated. Note that for small values of ΔP , this conversion is approximately linear. Figure 5.4.3 shows an example of generated branch point times, randomly chosen within each interval.

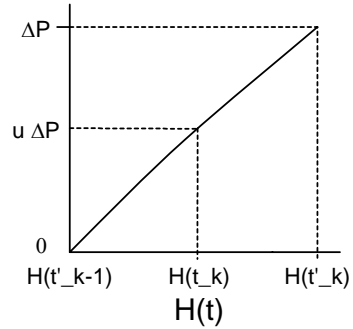


Figure 5.4.2 Conversion from $u \Delta P$ to $H(t_k)$.

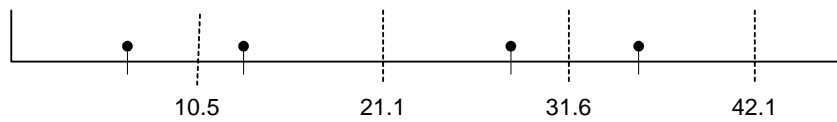


Figure 5.4.3 Example of generated branch point times within each interval

For certain time-to-occurrence distributions, such as exponential and Weibull, the value of t_k can be determined as a function of $H(t_k)$. In other cases, for instance when the intensity of occurrence depends on the physical conditions in the system, t_k may have to be determined through integration of $h(t,x)$. In this case, the boundary points t'_i will depend on the particulars of the scenario being simulation.

In general, when the intensity function is not constant, branch points will be spaced more densely when $h(t,x)$, or simply $h(t)$, assumes higher values.

5.4.2 Time-based branch generation

The rule can be applied to events that occur randomly in a specified time interval.

The time condition of the event is specified by the user/ modeler. The mission is divided into several critical time intervals, during which the event may happen. The rule will generate a random time point in each possible time interval.

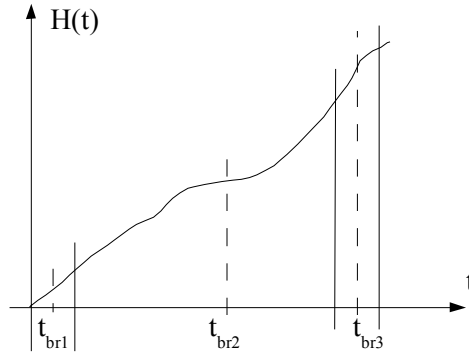


Figure 5.4.4 Illustration of the branching points

The branching time point t_{br} is uniformly distributed in the time interval (t_{ll}, t_{ul})

$$t_{br} = \alpha \cdot (t_{ul} - t_{ll}) + t_{ll}, \text{ where } \alpha \text{ is a uniform random number in } (0,1).$$

This branching generation scheme is useful when there are specific time intervals which are very critical to the mission, but are very short. Short in a sense that generating a branch point in that interval using probability based scheme is almost impossible.

5.4.3 Branching Point Generation for Demand Based Event

The stochastic behavior of a component may be described by the probability distribution function of time-to-failure. There is another class of failures. The

probabilistic branching stochastic process has a set of outcomes, each with of a probability of occurrence. The timing of the occurrence is not random; instead, the outcomes at that point of time are random. The failures of standby components or components starting on demand are examples of this class of stochastic behavior. Human actions may be considered this way as well if we ignore the delay in human reactions.

When system simulates this kind of behavior, possible outcomes are proposed to the scheduler. It is not limited by binary branches. When there are more than two possible outcomes, all possible outcomes are considered and proposed. The scheduler treats the proposed events in the same manner as the time-to-failure events. The branches generated represent different possible scenarios. There is no difference to scheduler whether the branch is generated by time-to-failure events or probabilistic branching events.

5.5 Scheduling Algorithm Based on Value

At a branch point, all the branches are proposed to scheduler. The value of a branch is defined as the measure of how much we want to simulate that particular branch. The scheduler will decide which branch to explore based on the values of all possible branches. The value of each branch is evaluated based on several factors. The section details the algorithm for calculating the value of branch.

5.5.1 Entropy as Measure of Information

One motivation for simulating the event sequences is to gain information, i.e., to reduce the uncertainty about the end state of similar sequences also belonging to that scenario.

If event sequences belonging to a particular scenario consistently result in a single end state, the motivation to spend more simulation time on the simulation of that scenario should decrease, as the expected added value of those simulations decreases. This applies regardless of whether the consistently encountered end state is a desirable or undesirable end state. We see, therefore, that we should favor the simulation of scenarios with a variable rather than a consistent outcome.

A second contribution to the uncertainty is the lack of knowledge about the scenario's outcome and an understanding of the system behavior, as well as an experience in the form of the observation of event sequences. Lacking either form of knowledge about the scenario, its outcome is going to be inherently uncertain, regardless of whether the system would behave in a consistent fashion were it observed.

Figure 5.5.1 illustrates possible states of uncertainty about the outcome of a scenario. For illustration purposes, we assume that event sequences end up in one of two end states, even though the presence of more than two end state types is assumed in the discussions in this dissertation. The horizontal axis in each chart represents the

probability p that a sequence belonging to the scenario results in the first of the end states. The vertical axis in each chart represents the likelihood, or belief, of p being the number. A Beta distribution with $\alpha=1$, $\beta=1$ is a uniform distribution in $(0,1)$. This is non informative, which means the likelihood of the p being any number in $(0,1)$ is the same. The Beta distribution with $\alpha=26$, $\beta=26$, yields a belief which the likelihood of the p being 0.5 is much higher than any other number. The Beta distribution with $\alpha=1$, $\beta= 51$ represents a belief that p being close to 1.

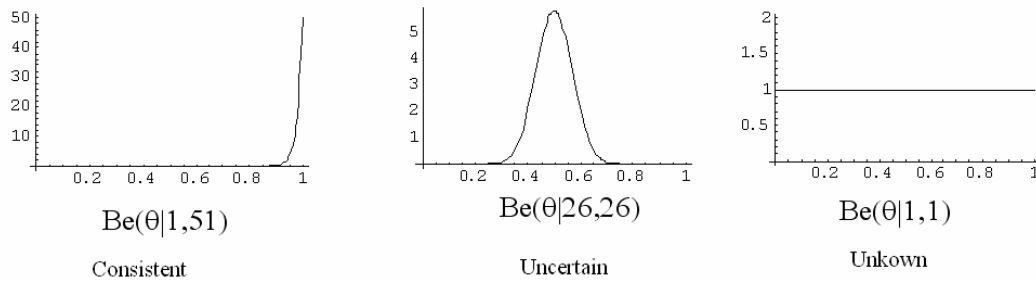


Figure 5.5.1 States of uncertainty about the outcome of a scenario.

The uncertainty or rather the amount of information, represented by such distributions can be measured using Shannon’s entropy measure (Lindley, 1956; Shannon, 1948). Given a probability distribution over the measure, the negative differential entropy measure is used, defined as

$$I(\pi(\theta)) = \int \pi(\theta) \cdot \ln \pi(\theta) \cdot d\theta$$

We note that the minus sign introduced by Shannon is not used here. In a

statistical sense, the maximum information is obtained when the probability distribution is concentrated on a single value, i.e. a δ function. The information is reduced when the PDF spreads. When Shannon introduced entropy, it is in the communication engineering, which is the opposite case faced by us.

We now apply this entropy to measure the state of information about the outcome of an event sequence belonging to a particular scenario S . We note that we will be considering the uncertainty about the end state of a *simulated* sequence, i.e., the probability distribution over the end states given that a sequence belonging to the scenario is simulated, rather than a probability distribution based on the natural frequencies of the event.

We initially consider that each sequence s in S ends in one of two end states. Let x be our degree of belief that end state E will be reached, and let our belief regarding this probability be described by a Beta distribution

$$\pi(x|\alpha, \beta) = \frac{x^{\alpha-1} \cdot (1-x)^{\beta-1}}{Be(\alpha, \beta)}$$

where $\alpha - 1$ and $\beta - 1$ respectively represent the number of times that end states 1 and 2 are observed in a total of $\alpha + \beta - 2$ sequences. Then the entropy measure is equal to

$$I(x|\alpha, \beta) = (\alpha - 1) \cdot (\psi(\alpha) - \psi(\alpha + \beta)) + (\beta - 1) \cdot (\psi(\beta) - \psi(\alpha + \beta)) - \ln Be(\alpha, \beta)$$

where, $\psi(z)$ is the digamma function,

$$\psi(z) = \frac{d}{dz} \ln \Gamma(z) = \frac{\Gamma'(z)}{\Gamma(z)}$$

which for integer arguments can be computed as (Abramowitz)

$$\psi(\alpha) = -\gamma + \sum_{m=1}^{\alpha-1} \frac{1}{m}$$

such that

$$\psi(\alpha) - \psi(\alpha + \beta) = \sum_{m=1}^{\alpha-1} \frac{1}{m} - \sum_{m=1}^{\alpha+\beta-1} \frac{1}{m} = - \sum_{m=\alpha}^{\alpha+\beta-1} \frac{1}{m}$$

For large values of z , the digamma function can be approximated efficiently by

$$\psi(z) \approx \ln(z) - \frac{z}{2}.$$

In case of a Dirichlet distribution,

$$\pi(\bar{\theta}) = \frac{1}{Z(\bar{u})} \prod_{i=1}^n \theta_i^{u_i-1}$$

$$Z(\bar{u}) = \frac{\prod_{i=1}^n \Gamma(u_i)}{\Gamma\left(\sum_{i=1}^n u_i\right)}$$

which applies when more than two end states are possible for each sequence, the entropy in this case is given by

$$I(\pi(\theta | \bar{u})) = -\ln Z(\bar{u}) + \sum_{i=1}^n (u_i - 1) \cdot \left[\psi(u_i) - \psi\left(\sum_{i=0}^n u_i\right) \right]$$

Figure 5.5.2 illustrates the information values for the Beta distribution. The figure illustrates that for the uniform distribution ($\text{Be}(\theta|1,1)$), the negative entropy is 0, representing a state of ignorance about the outcome of the scenario. The plot shows that as more sequences are simulated, the level of information about the outcome generally increases, and should in fact be expected to increase.

The negative entropy increases the fast along the axes of the surface plot. This indicates that the level of information about the outcome of an experiment increases fast when the outcome is **consistent**. However, an inconsistent outcome after a series of experiments with a consistent outcome can significantly reduce the information measure, representing a loss of confidence and increase of uncertainty, which can reasonably be expected in case of a surprise outcome. A surprise, or an outcome contradict the previous held belief may decrease the prior confidence.

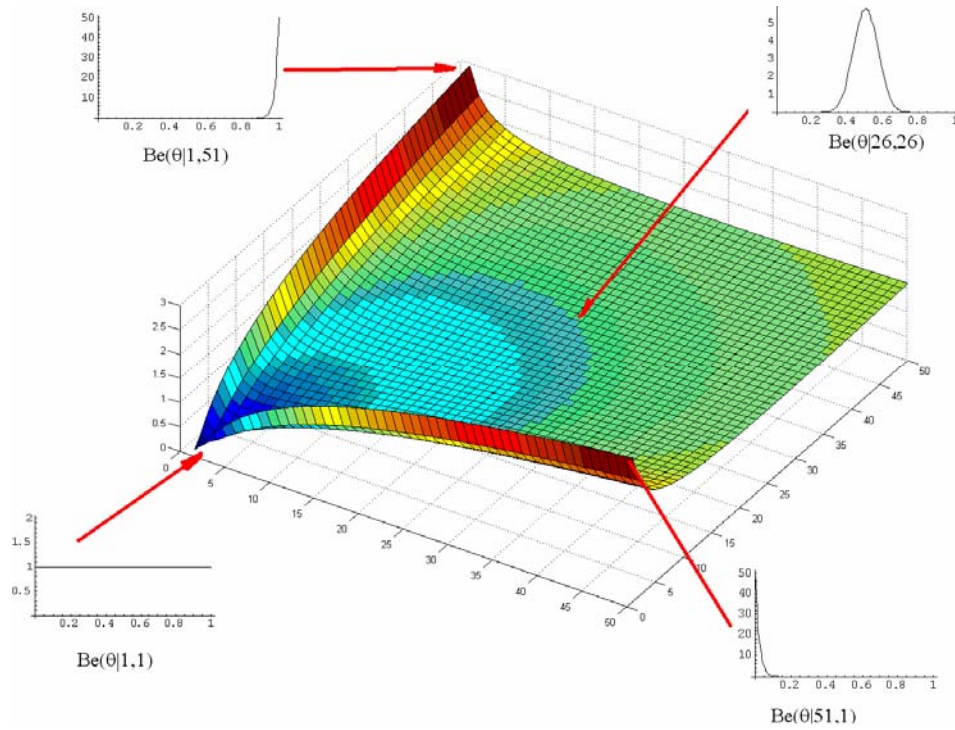


Figure 5.5.2 Information measure as a function of Beta distribution

5.5.2 Expected Entropy Gain Through Experiment

To calculate the value of a branch, we need to know the expected information we expect to gain if we follow that specific branch. Since we use entropy as the measure of uncertainty, it is natural to measure the expected information using entropy. The exploration of a branch can be viewed as a stochastic experiment, which has a limited number of possible outcomes.

Shannon introduced the idea that information is a statistical concept and proposed using entropy as measure of information (Shannon, 1948). Lindley applied these

ideas to measure the information in an experiment rather than in a message (Lindley, 1956). The amount of information provide by experiment is measured by comparing the knowledge before and after the experiment, while the measure of information is given by Shannon's entropy. The knowledge gained by an experiment can be described by a Bayesian inference model. The knowledge before the experiment is expressed by a prior distribution $\pi_0(\theta)$, where θ is the parameter we are interested in, and $\theta \in \Theta$. The experiment results in an observation x . Then, the knowledge after the experiment is $\pi(\theta | x)$, according to the Bayes theorem:

$$\pi(\theta | x) = \pi_0(\theta) \cdot p(x | \theta) / \int \pi_0(\theta) \cdot p(x | \theta) d\theta; \text{ where } p(x | \theta) \text{ is the likelihood function.}$$

The amount of information before the experiment, with respect to θ is

$$I_0 = \int \pi_0(\theta) \cdot \log[\pi_0(\theta)] d\theta$$

If we introduce the expectation operator E_θ , which denotes the expectation with respect to θ , the equation is rewritten as:

$$I_0 = E_\theta[\log[\pi_0(\theta)]].$$

We note that the minus sign introduced by Shannon is not used here. In a statistical sense, the maximum information is obtained when the probability distribution is concentrated on a single value, i.e. a δ function. The information is reduced when the PDF spreads. Shannon introduced entropy in the communication

engineering, which is the opposite case faced by us. The objective of communication is to transmit a message, x , which is received as message, y . The concentration on single value would allow no choice in his message, thus, no information transmitted. Therefore, the scales in these two cases are reversed.

After an experiment is performed and the value x observed, the amount of information is

$$I_1 = \int \pi(\theta | x) \cdot \log[\pi(\theta | x)] d\theta$$

While the prior knowledge $\pi_0(\theta)$, the amount of information provided by the experiment ε , when the observation is x , is defined as:

$$I(\varepsilon, \pi_0(\theta), x) = I_1 - I_0,$$

Further, the average amount of information provide by the experiment ε , is defined as:

$$I(\varepsilon, \pi_0(\theta)) = E_x[I_1(x) - I_0]$$

In the exploration problem here, the branch is corresponding to the experiment ε has n possible outcomes. We have the PDF of the outcomes $\bar{\pi}_0$, either from the prior knowledge, if we are at the start of the simulation, or the updated PDF with previous simulation results. The equation above still applies here.

To better understand the concept of expected information, we apply the algorithm

to a special case. We are performing an experiment of examining the probability p of getting a head from tossing a coin. The experiment has only two possible outcomes, head or tail $\{H, T\}$. Before the experiment we have no information of the probability, so our prior is a beta distribution between $(0,1)$ $\pi_0(p) = 1; p \in [0,1]$. The likelihood of getting result $\{n$ heads and m tails $\}$ is

$$L(E | p) = \binom{n+m}{n} p^n \cdot (1-p)^m, \text{ where } \binom{n+m}{n} = \frac{(n+m)!}{n!m!}$$

The posterior distribution would be
$$\pi(p | E) = \frac{\pi_0(p) \cdot L(E | p)}{\int_p \pi_0(p) \cdot L(E | p) dp}$$

Hence, we get,
$$\pi(p | E) = \frac{(m+n+1)!}{n!m!} p^n (1-p)^m$$

We recognize that this is a beta distribution with $\alpha = n + 1, \beta = m + 1$. The entropy of Beta distribution is showed in Figure 5.5.2. In a more general case, if our prior

information is expressed as a Beta distribution $\pi_0(p) = \frac{p^{\alpha_0-1} \cdot (1-p)^{\beta_0-1}}{B(\alpha_0, \beta_0)}$, where

$B(\alpha, \beta)$ is the Beta function. The posterior when we get result $\{E=n$ heads and m

tails $\}$ is
$$\pi(p) = \frac{p^{\alpha_0+n-1} \cdot (1-p)^{\beta_0+m-1}}{B(\alpha_0+n, \beta_0+m)}$$
, which is the Beta distribution with

$$\alpha = \alpha_0 + n + 1, \beta = \beta_0 + m + 1.$$

Then, we calculate the expected information gain from next experiment, starting

our prior knowledge expressed as a beta distribution with $\alpha = \alpha_0, \beta = \beta_0$. The result of the experiment would be either head or tail. We expect that we get a head with probability $L_H = \frac{\alpha_0}{\alpha_0 + \beta_0}$. The posterior distribution when result is a head is

$$\pi(p | H) = \frac{p^{\alpha_0} \cdot (1-p)^{\beta_0-1}}{B(\alpha_0+1, \beta_0)}.$$

Similarly, the posterior estimation when result is a tail is

$$\pi(p | T) = \frac{p^{\alpha_0-1} \cdot (1-p)^{\beta_0}}{B(\alpha_0, \beta_0+1)}.$$

The expected information gain is calculated based on the prior and posterior:

$$I(\varepsilon, \pi_0(p)) = E_x[I_1(x) - I_0] = L_H \cdot I_p(\pi(p | H)) + L_T \cdot I_p(\pi(p | T)) - I_p(\pi_0(p))$$

Figure 5.5.3 is plot of the expected information for $\alpha_0 \in (1,50); \beta_0 \in (1,50)$ (Note that on that plot, the information gain is on log scale.)

From figure 5.5.3 we can see that the information gain at the beginning of the experiment is high, and the information gain is decreasing. It is easy to understand that when there is little prior information, few or even one experiment result can gain us significant understanding, but when there are sufficient data already, single experiment is not likely to change the perceived information.

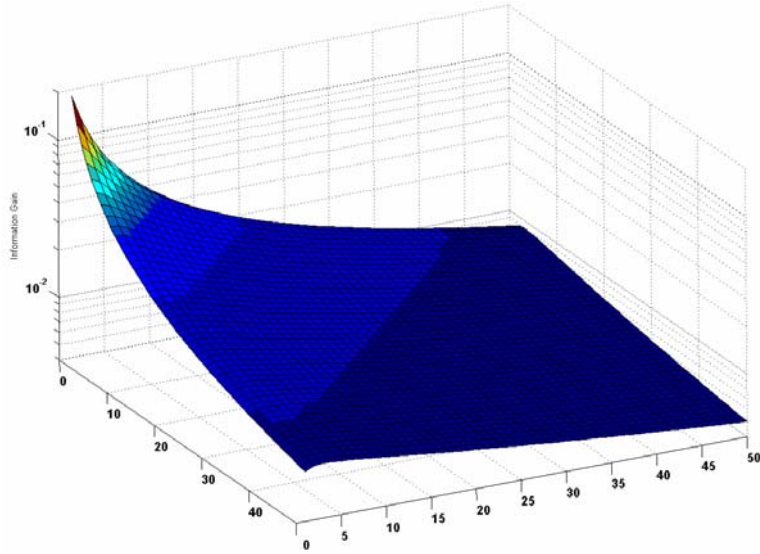


Figure 5.5.3 Expected Information Gain for Experiment with a Beta Prior

We list four representative cases in Table 5-1. The first row shows the prior information represented by a Beta distribution. The second row is the expected information of a single experiment starting from such prior information. The case of (1,1) represents non-informative prior, no experiments have been performed yet. It is expected that one single experiment could gain much information. In other cases with priors (6,6), (10,2) and (11,1), the prior information represents the information gain after ten experiments, where the results are {5 heads and 5 tails}, {9 heads, 1 tails} and {10 heads and no tails} respectively. We can see that (6,6) represent great uncertainty, and the information gain from this case is much higher than (11,1) where the experiment results are pretty consistent.

Table 5-1 Expected Information Gain of Beta Prior

Prior α, β	(1,1)	(6,6)	(10,2)	(11,1)
Expected Info	0.19	0.04	0.038	0.035

The case of multiple end states is studied in Table 5-2 Expected Information Gain. A non-informative prior is assumed. The posterior PDF is a Dirichlet distribution (see section 5.5.1). From the table we can find a clear pattern that the expected information gain decreases with number of sequences, and outcome consistency. The conclusion is an extension of the binary experiment case.

Table 5-2 Expected Information Gain for A Dirichlet Distribution

Previously Observed Outcomes, by end states				Expected Information Gain
ES-1	ES-2	ES-3	Es-4	
0	0	0	0	0.3030
0	0	0	4	0.1578
0	0	2	2	0.1626
2	2	0	0	0.1626
1	1	1	1	0.1684
0	0	0	9	0.0974
0	2	3	4	0.1050
0	20	30	40	0.0151

5.5.3 Principle of Evaluating the Value of Exploring a Scenario

As designed earlier value of each scenario serves as a measure of how much we want to simulate that specific scenario. The scheduler works by comparing the value of all possible branches. The higher the value, the more likely the scheduler will favor the branch. To generate the algorithm of evaluation, we summarize a set of principles:

- Value of a specific scenario should be consistent with how much **information** is expected to be gained by simulating/exploring that scenario.
- Value of a specific scenario should be consistent with the **importance level** based on prior information or engineering experience.
- One scenario with high value will make the value of whole group of scenarios high. A bunch of branches with low values, does not necessarily add up.
- Value should be coherent with how close it brings the system towards ending.

The expected information gain is one of the most important factors in the value of a scenario. Apart from the expected information gain, previous experience with same or similar systems is likely to give us a hint that some scenarios are more important than others, and this piece of information can also influence the decision on how to

explore the event space. This type of information would be provided as the “importance level” in the plan. The scenarios we are interested in may still have sub-branches, so to evaluate the value we will have to take account of all the sub-branches. If one of sub-branches has a high value, it will considerably increase our interest in that branch. But if all the sub-branches have very low values, they do not simply add up to a high value, i.e, increased interest in that specific branch.

5.5.4 Algorithm for Evaluating the Value of Proposed Event

The value of a proposed event is a function of the expected information gain, **impact factor** and **importance factor**. The information gain evaluation algorithm has been discussed in section 5.5.3. **Importance factor** is an engineering judgment of how much we want to explore or avoid the scenarios. It is subjective and relies on the expertise of the engineers and risk analysts. The importance of a scenario is considered a function of

- Learning value: ability to provide new insights
- Engineering criticality: significance from engineering perspective

The engineering criticality is a measure of the perception by engineers/analysts that particular scenarios are (not) of interest. If an event is expected to lead to severe end states, the importance level would be high. Other knowledge involved in the assessing of the importance factor includes knowledge of common vulnerabilities that

have been observed in similar systems. The risk analyst would assign an importance level to each scenario in the plan.

The impact factor is an object measure of how close the transition would bring the system towards an end state. It is a real number valued between 0 and 1. The algorithm of calculating the impact factor is designed in the scheduler.

The value of each event proposed at the branch point is evaluated by evaluating the value of any of the scenarios that are enabled by the proposed event. A value is assigned to each scenario originates from the branch point. The evaluation follows the steps:

- i. The value of a proposed transition event depends on the value of the scenario which it enables, and the evaluation of the scenario has been discussed in 5.5.3
- ii. Multiply the raw value from step ii measure with the importance factor.
- iii. We multiply the raw value deduced from step iii with the impact factor which is based on whether the transition associate with the branch takes the system closer to an end-state or not.
 - If the event satisfies a node, which moves the system forward, the impact factor is 1;
 - If the event makes the system impossible to follow any interesting

scenarios, the factor is 0;

- If the event partially satisfies a node, the factor is between 0 and 1.

5.5.5 Exploration of Branches

Whenever a branching point is reached, the further exploration options are proposed to the scheduler by the simulation model. The scheduler will decide which branch to explore, and send the exploration command back to the simulation model. The simulation model would execute the command.

In the scheduler, each option is like a branch. The scheduler will

- i. Retrieve the information of each option; and
- ii. Evaluate the value of each branch, as discussed in 5.5.4 and
- iii. Choose the option using a Russian Roulette style algorithm, and
- iv. Send the exploration command back to scheduler.

The scheduler will decide which branch to explore according to the value. The higher value implies higher likelihood of the scenario being simulated. If the scheduler simply choose the branch with highest value, it is likely that the simulation would be locked in the scenarios listed in the plan. As we have stated earlier, the plan is only a rough guide, and is not intended to be complete, or accurate. We do not want the simulation to be locked in the scenarios in the plan. The scheduler should be random to ensure that the scenarios which are not in the plan still have a chance to be

simulated. According to the algorithm, the probability of exploring a specific branch is proportional to the value of that branch. This is similar to Russian Roulette algorithm in (Marseguerra & Zio, 1993).

When there are n proposed branches, and the value of each branch is V_i , the value of the branches are normalized such that $\sum_{i=1}^n V_i = 1$, and the interval $(0,1)$ is divided into n subinterval accordingly. A uniform random number u is drawn from $(0,1)$ to determine the branch chosen, that is i^{th} branch is chosen, if u fall into i^{th} subinterval. The process is illustrated in Figure 5.5.4.

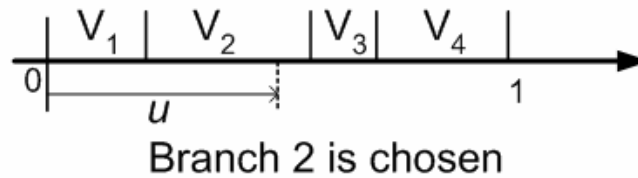


Figure 5.5.4 Choosing Branch Based on Value

5.6 Event Sequence Quantification

The scheduler works in such a way as to explore the scenario we are interested in more frequently. We force some stochastic events, such as hardware malfunction,

human errors, and software failures to happen more frequently, or in some cases, less frequently compared to their natural frequencies or probabilities. In order to get unbiased estimates of the quantities of interest, we take account of the modification introduced through accumulation of proper weights.

The natural probability that the event occurs during any given interval is by definition equal to ΔP . For demand-based event, the natural frequency of the event is P . The demand-based and time-distributed events can be treated in the same manner, the only difference is the notation of the natural frequency. In the following discussion, we use the notation ΔP to represent the natural frequency. This is only for convenience, and the procedure can be applied to both demand-based event, and time-distributed event.

Since an event at t_k is taken as representative for occurrence of the event anywhere during the interval, the probability of occurrence of the event will be taken as ΔP , and thus the probability of non-occurrence of the event as $1 - \Delta P$.

In case of a random search, the two branches originating from a branch point are assumed to be visited with probability Q (occurrence of the event), and $1 - Q$ (non-occurrence) respectively. Q can be used to control how often a particular branch is visited during the simulations, regardless of the actual probability of the branch.

In order to correct for a possible discrepancy between Q and ΔP , and thus also (1

– Q) and $(1 - \Delta P)$, a weight factor must be applied to event sequences whenever a branch point is encountered. If the probability of a branch is ΔP , and the probability that the branch is generated equals Q , the probabilistic weight of the event sequence following that particular branch should be adjusted (multiplied) by a factor

$$w = \frac{\Delta P}{Q}$$

In case of a systematic exploration, both branches are visited. In this case, the correction takes place by multiplying the branch weights by P or $(1 - \Delta P)$, depending on the branch that is followed. These values are found by setting Q to 1 in the above equation.

In the adaptive simulation, each time when a stochastic event is proposed to the scheduler, the statistical weight affected by the scheduling command is kept by the scheduler. The Q (probability that branch is explored) may vary each time. The algorithm stays the same. When the propose event is a probabilistic branching event, the actual probabilities (P_e) is associated with each possible outcomes, thus the weight is $w_i = \frac{P_e}{Q}$.

By applying this weighting scheme to event sequences, the probability of the corresponding scenarios can be computed. Let each generated event sequence belong to one or more scenarios. Then the sum of weights of event sequences belonging to a

given scenario is taken as the weight of the scenario, denoted by W_S

$$W_s = \sum_{i: S_i \in S} w_i .$$

If the scenarios are mutually exclusive, meaning that each event sequence belongs to exactly one scenario, the weights can be normalized into an estimated probability distribution over those scenarios

$$\hat{P}_s = \frac{W_s}{\sum_j W_j}$$

In case of a random event sequence generation, \hat{P}_s converges towards the true probability of the scenario as the number of simulated event sequences increases.

With each simulated event sequence, W_S increases *on average* by

$$\Delta W_s = \sum_i w_i \cdot q(s_i)$$

where

- $s_i, i = 1, \dots, n$ is the set of all event sequences that make up scenario S .
- w_i is the weight of sequence s_i , as defined above
- $q(s_i)$ is the frequency at which sequence s_i is generated.

To prove that this weighting scheme allows us to compute the appropriate values of event sequence frequencies, and thus scenario probabilities, we consider an event sequence s that contributes to a scenario S . The sequence may involve the occurrence

and non-occurrence of any number of events $A_i, i = 1, \dots, n$.

The expected (average) weight contribution of this event sequence to S is defined as

$$\bar{w}_s = q_s \cdot w_s$$

where q_s is the frequency at which s is generated, and w_s is the weight that would be assigned to s if it were generated. Furthermore, we define p_s to be the true frequency of s . We will show that

$$\bar{w}_s = p_s$$

For each event $A_i, i = 1, \dots, n$, s traverses zero or more intervals during which the event is not generated, possibly followed by an interval during which the event does take place.

Consider an interval I during which the event is simulated to not take place. By definition, the probability that this is the case equals $(1 - Q_I)$. Also by definition, the true probability that the event would not occur is $(1 - \Delta P_I)$. Therefore, the branch rule modifies the frequency at which event sequences that involve the non-occurrence of the event during I are generated according to

$$q_s' = \frac{1 - Q_I}{1 - \Delta P_I} \cdot q_s$$

The branch rule modifies the weight of such event sequences according to

$$w_s' = \frac{1 - \Delta P_I}{1 - Q_I} \cdot w_s$$

Therefore, the expected weight contributions of the event sequences remain constant

$$w_s' \cdot q_s' = w_s \cdot q_s$$

A similar argument can be made for intervals in which the event *is* simulated to occur. We find therefore that the average weight contribution \bar{w}_s of an event sequence is constant, and does not depend on the values of ΔP and Q .

Furthermore, we now that if for all intervals $\Delta P_I = Q_I$, the frequencies at which event sequences are generated equal their true frequencies, and that $w_s = 1$. We see therefore that

$$w_s \cdot q_s = 1 \cdot p_s$$

and thus the expected weight contribution of an event sequence to a scenario must equal the event sequence's frequency.

In case of a systematic exploration, all sequences are generated, and thus $q_s = 1$. The frequency of an event sequence is known immediately after it is generated. However, in order to obtain the scenario probabilities, all event sequences must be

simulated.

For an example of the weighting in case of a random search, see Figure 5.6.1. It shows four mutually exclusive scenarios that, for convenience purposes, correspond to the occurrence of the event during intervals 1, 2, 3, and 4. The ΔP for each scenario is chosen as 0.1. The Q for each scenario is chosen as 0.2. The figure lists the actual probabilities (P_s), the generation probabilities (Q_s), the event sequence weight factors (w), and the product of Q_s and w .

time \longrightarrow

	$\Delta P = 0.1, Q = 0.2$	$\Delta P = 0.1, Q = 0.2$	$\Delta P = 0.1, Q = 0.2$	$\Delta P = 0.1, Q = 0.2$
	S1	S2	S3	S4
P_s	0.100	0.090	0.081	0.073
Q_s	0.200	0.160	0.128	0.102
w	0.500	0.563	0.633	0.715
$W Q_s$	0.100	0.090	0.081	0.081

Figure 5.6.1: Example of scenario quantification.

We consider the weight factor for the scenario S_3 , in which failure takes place during the third interval. Due to the simplicity of the problem, we can compute the probability of this scenario to be

$$\Pr_p(S_3) = (1 - 0.1) \cdot (1 - 0.1) \cdot 0.1 = 0.081$$

The probability that a randomly generated event sequence belongs to the scenario is

$$\Pr_Q(S_3) = (1-0.2) \cdot (1-0.2) \cdot 0.2 = 0.128$$

The weight factor of any such event sequence is

$$w = \frac{(1-0.1) \cdot (1-0.1) \cdot 0.1}{(1-0.2) \cdot (1-0.2) \cdot 0.2} = 0.633$$

As the fraction of generated sequences belonging to S_3 approaches 0.128, the estimated probability \hat{P}_{S_3} approaches the scenario's true probability

$$0.128 \cdot 0.633 = 0.081$$

To summarize: let P_i be the probability associated with branch i originating from a given branch point. Let Q_i be the probability that the branch is explored. Then, the weight w of any event sequence originating from that branch must be multiplied by a

factor P_i / Q_i . The weight of a event sequence is $w = \prod \left(\frac{P_i}{Q_i} \right)$

In case of systematic exploration, $Q_i \equiv 1$, and the normalization is not necessary, since probabilities should by definition add up to 1.

5.7 Estimator of End State Probabilities

The simulation is guided toward the scenarios which are believed to provide more

information of the system vulnerability. This procedure is very similar to the importance sampling of Monte Carlo simulation. Each event sequence generated would have a statistical weight.

The estimator here is similar to the importance sampling estimator:

$$p^k = \frac{\sum_{i=1}^N L_i^k \cdot W_i}{\sum_{i=1}^N W_i},$$

where p^k is the probability of reach end state k, and the W_i is the statistical weight of simulated event sequence i .

L_i^k is *system state function*. $L_i^k = 1$, if the end state of i th event sequence is k, otherwise, $L_i^k = 0$. The *system state function* L_i is equivalent to the *system failure function* in (Campioni & Vestrucci, 2004), but the system state function here can express multiple end state, and is not limited by the <success, failure> binary logic.

The statistical weight W_i depends on the way that the event sequence is generated. The calculation has been discussed in 5.6.

5.8 Simple Test Case

A simple test case is constructed to test the efficiency of the new DPRA methodology and scheduling algorithm. This test case is assuming that the system

consists three items which follow the Weibull failure rate. The system is built with redundancy, i.e., if one of the items fails the system will still function as normal, but if two or all the three items fail the system will fail. The figure below shows the fault tree of the system.

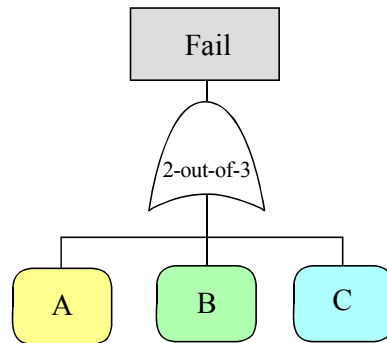


Figure 5.8.1 Fault Tree a Simple Test Case

There are two different end states that we are interested i.e. failed or Success. In this small system, we assume the components identically follows the Weibull failure rate, whose $\alpha = 5000$, and $\beta = 2$. The time span is 91 seconds. Thus, the reliability of this 2-out-of-3 system is 0.997, and the probability of system failure with the time span is $p = 0.003115$.

5.8.1 The plan.

The plan for this case is very simple. The combination of any two of the three components would lead to the system failure. If only one or none component fails,

the system will still function as designed.

5.8.2 End State Probability Estimates.

One important objective of the simulation is to estimate the probability of different scenarios accurately and quickly. The estimation is shown in the Figure 5.8.2. We can see the estimation converge to the real value. With the plan the simulation estimation converges much faster than crude Monte Carlo simulation. If there is no acceleration, the component failure would happen to would only about 10 times in 1000 event sequences generated. With such low simulation which may result in interested scenario, it is hard to get the accurate estimation.

The Figure 5.8.2 shows after the simulation generates 1000 sequences, the estimation is 0.00308, and the relative error is only 1.1%.

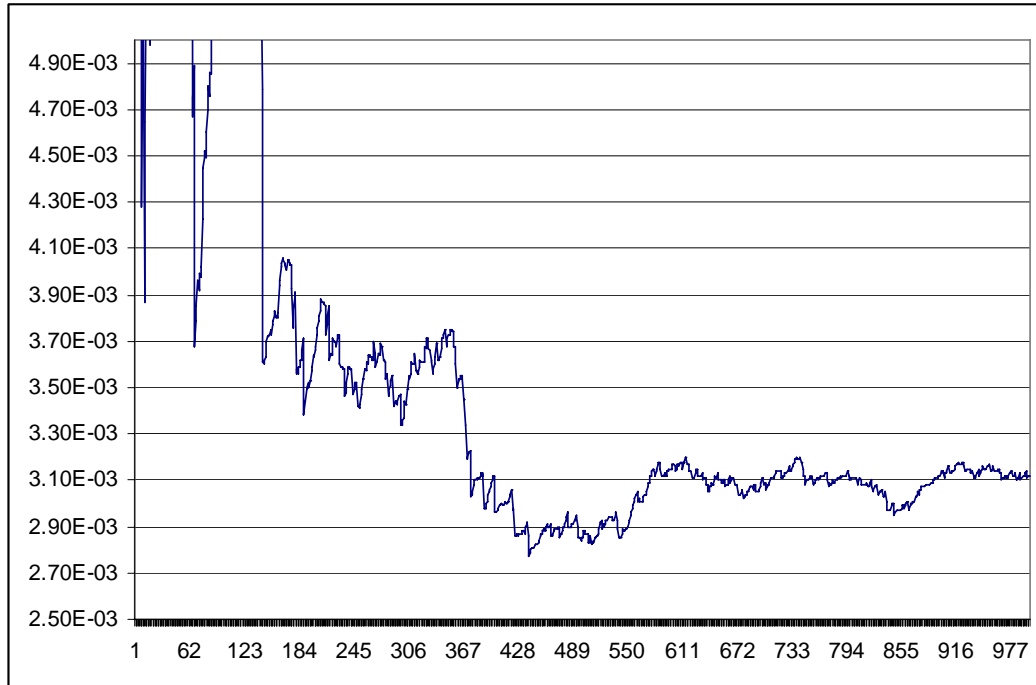


Figure 5.8.2 The Estimation of the Probability of System Failure

At the first part of simulation we can see the estimation "jumps", that is due to the fact that in the beginning we do not have many cases result in system failure, even only one sequence which results in system failure may change the estimation considerably. After about 100 event sequences, the scheduler adjusts the simulation more efficiently, more effort was devoted to the exploration of the interested scenarios, instead of repeating the scenario without any component failure over and over again.

If we simulate the system in a non-biased Monte Carlo manner, the estimator of the failure probability is:

$$\bar{h}_n = \frac{\sum_{i=1}^n h(\bar{x}_i)}{n},$$

where $h(\bar{x}_i) = 1$, when the system fails, otherwise, $h(\bar{x}_i) = 0$

The variance of the $h(\bar{x}_i)$ is $p(1-p)$, where p is the system failure probability. The

1σ (68% confidence) convergence envelop of the estimator is $\sqrt{\frac{p \cdot (1-p)}{n}}$. After 1000

simulation runs, the 68% confidence envelop of the failure probability estimator

would be $p \pm \sqrt{\frac{p \cdot (1-p)}{n}} = 3.1e-3 \pm 1.7e-2$, which implies a relative error of more

than 50%. The large variance is due to the fact the failure is rare under unbiased simulation, and it is expected to observe only 3 failures in 1000 simulation runs.

This example shows that the guided simulation improved the efficiency and accuracy considerably.

5.8.3 Distribution of Sequences

The table below shows the simulation result of a test case which set all the scenarios in the plan with the same importance level. With these setting, the scheduler will distribute the simulation effort according to the "value" of different scenarios, favoring the scenarios with higher value, which is more likely to increase out knowledge about the system.

From the table we can see, that the "two-component failure" scenario, which will lead to system failure was accelerated with a very high magnitude, while the "one-component failure" was also accelerated, but with a much lower magnitude. It is also clear the "no component failure" scenario, which happens with the highest probability was decelerated.

Table 5-3 Distribution of Event Sequences

	Number of Cases	Percentage of Occurrence	Probability of Occurrence	Acceleration Factor
Two Components Failure	486	48.6%	0.3%	162
One Component Failure	394	39.4%	9.2%	4.28
No Component Failure	120	12%	90.5%	0.133

Also, we notice that the acceleration factor for "two component failure" and "one component failure" is very different. This cannot be achieved by simply accelerating the component failure rate. This shows the capability of the scheduler to distribute the event sequences "fairly" among scenarios.

5.8.4 The Impact of Importance Factor

The decision of which scenario to explore is not only based on the value of scenario, but also the "importance" level. It is desirable to run the simulation with higher importance set some specific scenarios, which the system analyst would be interested.

If we change importance setting of "two-component failure" to "high" and "no component failure" to "low", we would see the distribution of event sequences summarized in the table below.

Table 5-4 Distribution of Event Sequences

	Number of Cases	Percentage of Occurrence	Probability of Occurrence	Acceleration Factor
Two Components Failure	572	57.20%	0.30%	190
One Component Failure	374	37.40%	9.20%	4.1
No Component Failure	54	5.4%	90.50%	0.0597

Compared to the results reported in Table 5-3, the scenarios with "high" importance level were accelerated with an even higher acceleration factor, while the low importance scenarios were decelerated more. This feature cannot be achieved by biased Monte Carlo which simply accelerates the component failures. In that type of Monte Carlo simulation, the event sequences distribution among scenarios is direct result of acceleration at component level. Typically, the acceleration of one component failure would result in the proportional acceleration in all related scenarios.

6. Introduction to SIMPRA

6.1 Overview

6.1.1 Framework of SIMPRA

Simulation-based Probabilistic Risk Analysis (SIMPRA) is a software package which implements the methodology proposed in Chapter 5. It is developed in Java and MATLAB® / Simulink general purpose simulation environment. SIMPRA is a general purpose PRA platform. It provides a DPRA library, with which the user can easily build DPRA simulation models in Simulink.

SIMPRA is a software package implementing the methodology we proposed in chapter 4 and 5. The key components of SIMPRA are:

- Planner
- Scheduler
- Simulation Model
- Output Analysis

The SIMPRA simulation model is built in MATLAB® / Simulink environment and the Scheduler and Planner is developed in Java. Matlab can import the Java class directly. During the course of the simulation, the simulation model would call

different methods of the scheduler class, to generate the branching points, and get exploration commands from scheduler. Scheduler would control the occurrence and timing of events at the branching points. The interactions between different blocks of SIMPRA are depicted in Figure 6.1.1.

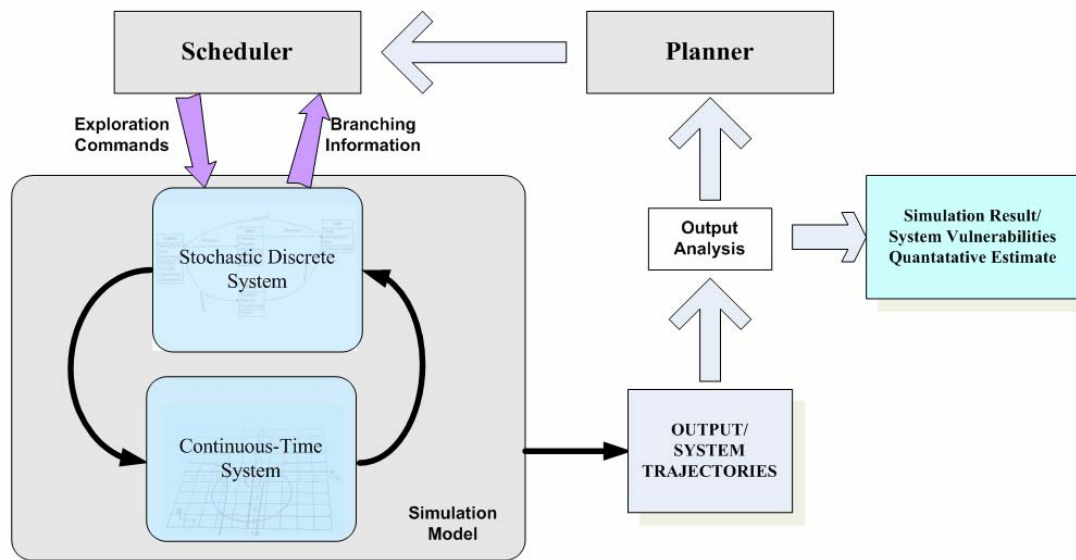


Figure 6.1.1 Framework of SIMPRA

6.1.2 Object-Oriented Paradigm

Object Oriented originally is a paradigm for writing programs. It has been widely accepted and supported in programming practice and it has been also applied in other areas. An object is an entity that holds both the descriptive attributes of the object as

well as defines its behavior. The simulation model consists of an integration of objects. The components of the system are defined by instantiating of the objects.

We will borrow some concepts of object oriented paradigm in our modeling procedure.

- **Abstraction**

An object is an abstraction of an entity in the real world. To deal with the complexity of the real world, we form abstraction of the things in it. In object-oriented paradigm, abstraction is the analysis that what a *class* knows (attributes) or does (methods). The abstraction includes all the attributes and methods of interest to the application, and the rest is ignored.

- **Encapsulation**

In the object-oriented world, the systems are modularized into classes, which, in turn, are modularized into methods and attributes. Encapsulation is the design how functionality is compartmentalized within system. The implication of encapsulation is that the implementation can be built in any way and then later changes of the implementation will not affect other parts of the system. The details of the implementation of an item is hidden from the users of that items.

- **Information Hiding**

To make the applications maintainable, the access to data attributes and some

methods is restricted. If one class wants information of another class, it should have to ask for it, not simply taking it. The purpose of information hiding is to make certain details inaccessible so that they should not affect other parts of a system.

- **Object-Oriented Simulation**

In an OO modeling simulation paradigm, objects are constructed to represent real-world entities that can interact with each other. The interactions are modeled as communications, where “messages” are exchanged between different objects. “Messages” here can represent the transfer of all kinds of entities, for example, information, materials, or energy. When an object receives a message, the response could be altering its internal state (i.e. the underlying behavior changes in a fundamental way), altering its important characteristics (attributes), and/or generating outbound messages to communicate its conditions to other objects in the model. The way in which the object responds to messages depends on the message it receives, its internal processes and on its internal state. One basic idea of object-oriented is encapsulation, which means the behavior of a component or subsystem entirely are enclosed within the confines of a self-contained object. The model of the entire system is created by combining and connecting the object models, and enabling the individual object models to communicate with one another in a way such that it faithfully duplicates the behavior in the real-world system.

The modeling of the system begins with identifying the objects in the system and

the relationships between objects. Each component is embodied in an **object** model. The **object** model represents a description of its behavior and its interaction with outside world. The object's information, events and actions are available to other objects only through specified interface. This feature, referred to as "encapsulation", is one of the underlying principles of object oriented paradigm. It helps to minimize the network when we are developing the model. Parts of the information is hidden, the interface of each module is designed is such a way that as little as possible about the inner working is revealed. As a result, the traffic between different parts of the work is minimized.

The inheritance is another key feature of object-oriented. It enables the analyst to declare an object whose common Attributes and methods are specified *once*, and extends and specialize those attributes and methods into specific cases. A number of objects are created and stored in the library. The objects in the library are reusable in different applications. The scale of reusability may differ. Some of the objects are universal, such as different failure modes, such as Weibull failure. These objects can be reused in any applications. Some objects may be application oriented. The engine of a space shuttle can also be an object, but only reusable in shuttle related applications.

One of major challenge faced by risk analyst is that the complex interactions in the system. With the graphical representation discussed in Appendix A, it is the

burden of the system analysts to identify all kinds of interactions and their possible consequences. By contrast, in the object oriented simulation model the interactions are depicted as the “messages” communicated between objects and the response to the messages. There is virtually no limitations of what kind of interactions can be modeled. All kinds of the interactions can be easily reproduced in the model. Another benefit is that this approach frees the modeler from the burden of defining all possible scenarios and/or generating the graph contains all the states of the system (as in state transition graph).

6.2 *Planner*

Plan is a map to guide the exploration. The Planner is a module of SIMPRA to generate such a map. Planner collects useful knowledge about the contributors to different classes of risk scenarios and generates the roadmap for the simulation.

In using the Planner, the first step is for the users to construct an abstract model of the system. The abstract model consists of a state transition diagram, a component tree, and a functionality tree. The component tree is used to establish relationships among various sub-systems and components involved in the system. A sub-component node can further have sub-components. The functionality tree is used to establish the relationships between the various sub-processes and functionalities associated with the system.

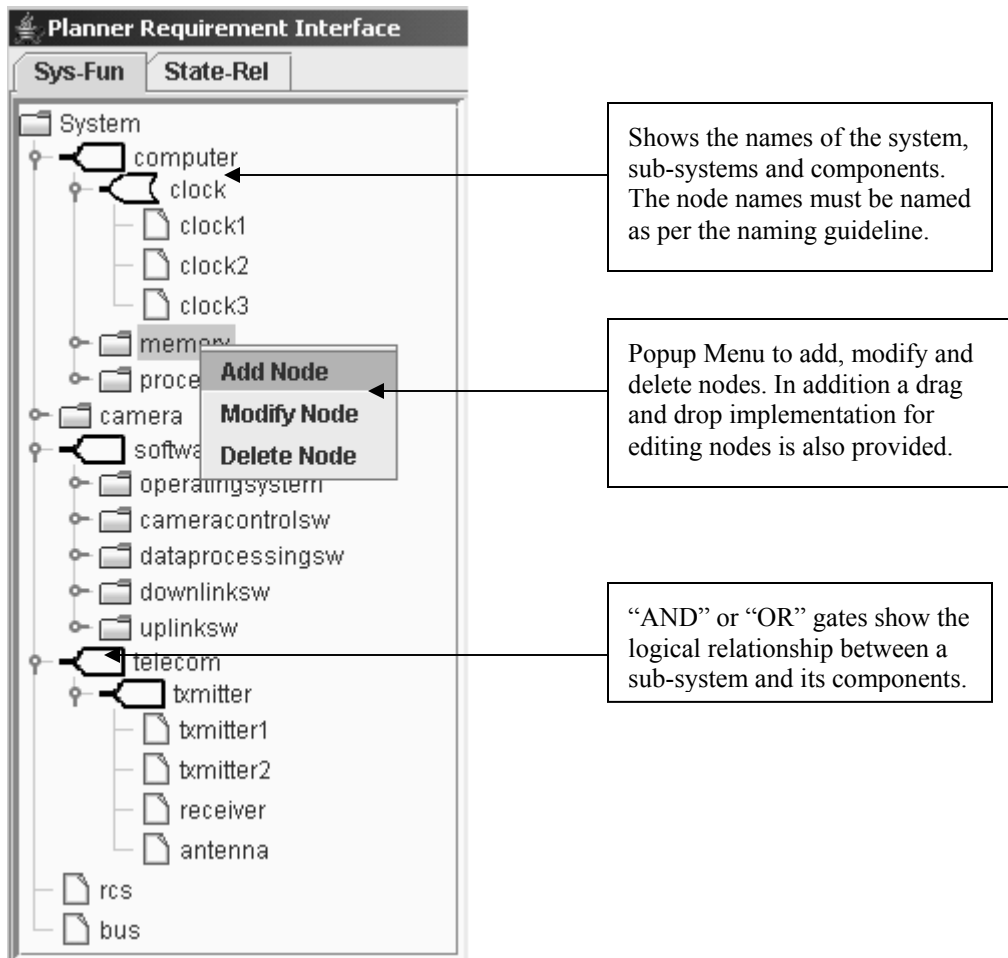


Figure 6.2.1 Component Tree

The component tree and functionality tree are used to define relationships between components and their associated processes (see Figure 6.2.1). The state transition diagram is used to draw the required state-relationship, associate them with the selected components and functionalities from the component-functionality matrix, and then generate a plan (see Figure 6.2.2).

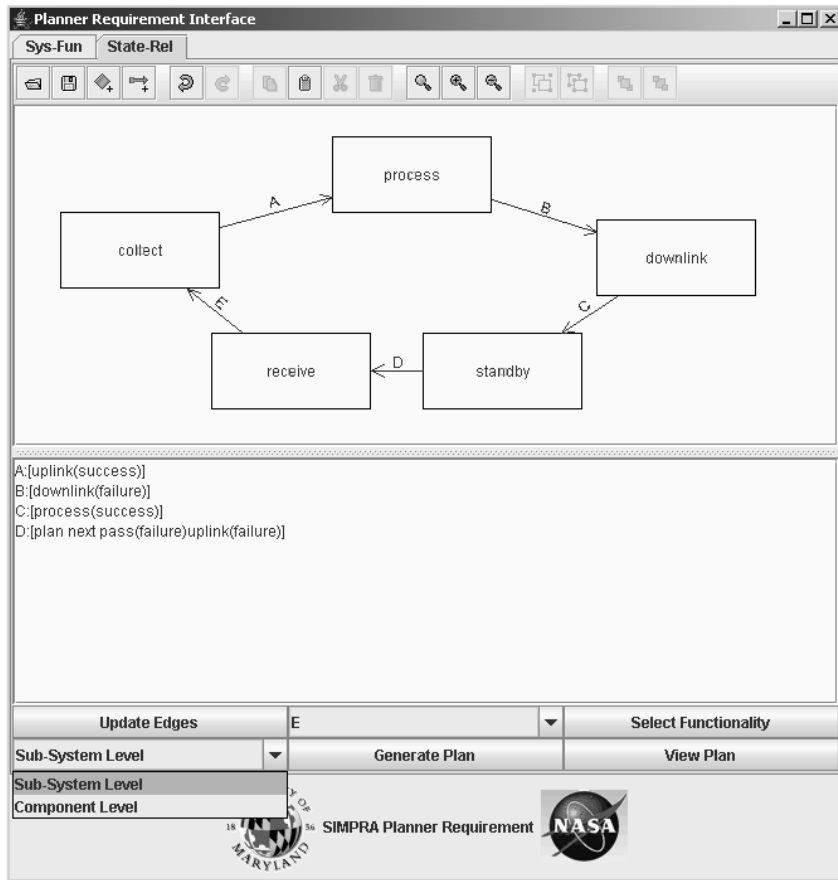


Figure 6.2.2 State Relationship Diagram Editor

The component tree and functionality tree are stored in a database file. The planner would query the database file while generating the plans.

The implementation of planner is a companion research of this dissertation, and the details of can be found in (Mosleh et al., 2005)

6.3 Scheduler

6.3.1 Functions of Scheduler

The scheduler is an implementation of the scheduling algorithms introduced in Chapter 5. The current implementation of the Scheduler consists of a set of Java classes. The Scheduler is used to control the simulation model which is implemented in Simulink, but the scheduler does not contain code that is specific to Simulink models. If it is implemented in Matlab, the simulation model can call the Java class directly by importing the archived jar file. There is no function for the Scheduler to call the simulation model directly.

Core classes in current implementation are in the ‘scheduler’ package:

- **Scheduler:** provides interfacing with the simulation model
- **MaxValueStrategy:** implementation of the scheduling principles as discussed in section 5.5 and the systematic traversal discussed in section 5.6.
- **ScenarioNode:** used to construct the tree structure which represents the ‘plan’, i.e., set of scenarios, the algorithm has been discussed in section 5.3.
- **ScenarioNodeLogic:** implementations of this interface represent the different types of logical constructs in the scenario, also discussed in section 5.3.

The following scheduler functions would be called during simulation:

- **Scheduler.getBranch():** Initialization; call the scheduler to get the branching information of stochastic events , the algorithm is discussed in the section 5.4.
- **Scheduler.proposeTransition():** When reaching a branch point, the simulation model calls the scheduler for exploration command and updating the branching information; the algorithm of scheduling has been discussed in section 5.5.
- **Scheduler.Notify():** Event notification; the algorithm has been discussed in section 5.7
- **Scheduler.NotifyEndState():** When simulation reaching an end state; the simulation model calls the scheduler to notify end state, and calculate the sequence weight and the algorithm has been discussed in section 5.7.

All the interfaces between simulation model and scheduler class are encapsulated in the DPRA library block. The users do not have to program the interface. They only need to specify the stochastic parameters in the simulation model.

6.3.2 Systematic Exploration

SIMPRA also supports the systematic exploration strategy as in the Discrete Dynamic Event Tree (DDET) methods. There are two types of systematic exploration supported in SIMPRA. The first type is the full-scale systematic exploration. The

second type is the systematic search for specific events. The types of systematic exploration are indicated in the plan.

Once a branch point is reached and branches are proposed to the Scheduler, the Scheduler would explore all proposed branches. The exploration of event sequences is managed in a depth-first manner, as in ADS. At the branch point, the current system state at the branch point and all the branches are stored in a database, and the first branch is executed and the simulation continues. When an end state is reached, scheduler goes one step back to the previous branch point. If at least one branch still remains to be explored, the scheduler would retrieve the state of the branch point to re-initialize the simulation, and explore the new branch. The simulation is restarted until another end state is reached. When all possible branches of that branch point have been explored, the scheduler brings the simulator one step backwards, until the dynamic tree exploration is completed.

As in ADS, there is a user-defined parameter P_{lim} . If the probability of branch gets lower the P_{lim} , that branch is no longer simulated. The probabilities of such event sequences are collected as “truncated” in the scheduler. We want to keep sum of the truncated probabilities low; otherwise it may introduce significant error in the estimator.

Another difference between systematic search supported by SIMPRA and the ADS type DDET is that the branch point is generated randomly as we have discussed in

5.4, as opposed to only at the predefined points as in DYLAM or ADS.

If we are performing a full scale systematic search, the event sequences are generated and explored in a systematic way. Users define the number of event sequences they want to generate. The number of event sequences generated by the exploration of one round is limited by the number branch point generated during the simulation, and may be lower than the user defined number. The systematic search will go on for another round until the number of event sequences is greater than that designated by the user.

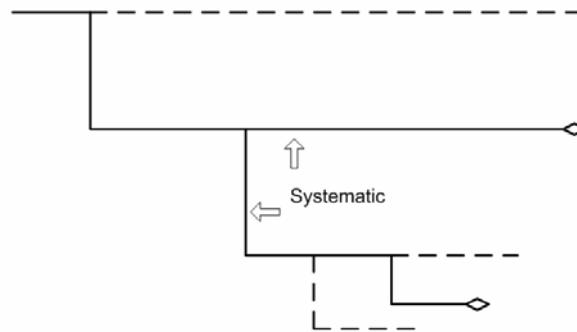


Figure 6.3.1 Partial systematic search

Figure 6.3.1 illustrates an example of partial systematic search. The dashed lines represent the branches not being explored. One node has been explored systematically, which means, all possible branches originated from that node are explored. The nodes before or following this node are still explored randomly. If the partial type of systematic search is performed, only the branches proposed by the designated events would be proposed systematically. The other branches would still be explored randomly. The event sequences generated by a systematic search would

carry a statistical weight as the discussed in 5.6.

6.4 Structure of the Simulation Model

6.4.1 Simulation Model

SIMPRA relies on an executable model of the system, which emulates the system behavior. Given the operational profile and adequate input, we assume that this model would reproduce the behavior of the real system under the specific circumstances.

The model is an abstraction of the real system. The abstract level may change during the course of the risk analysis. When we have better understanding of the system, or at later stage of system development, we may replace the existing model with a more detailed model.

In a typical DPRA problem, the system under investigation consists of discrete component state and continuous process variables. The evolution of continuous process variables are governed by the deterministic physical and logical laws. Physical processes are described using mathematical expressions of such laws. The components have discrete states, which represent a set of operational modes or configurations. The component state may change from one state to another. This change may be internally determined by the system logic and/or physical laws, or described by statistical or probabilistic laws. The occurrence of discrete state transition is defined as an event. The change of component state may in turn influence

the mathematical equations describing their behavior in each state. Examples of stochastic events include component malfunction with time-to-failure following a Weibull distribution. One example of deterministic event is the fuse in circuit melting when the electrical current reaches a certain level.

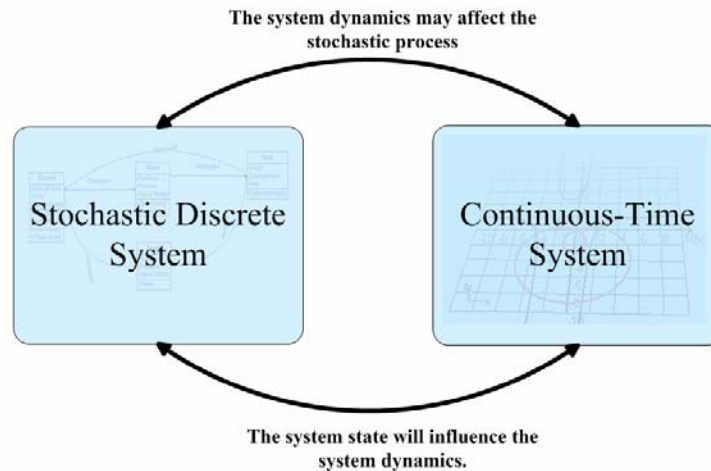


Figure 6.4.1 The Interaction between the discrete model and Continuous model.

The model is a combination of both deterministic and stochastic models (Figure 6.4.1). The stochastic process dynamically interact with the continuous-time, deterministic processes. The interactions between them consist of at least:

- The state of the discrete system determines the boundary conditions of the continuous-time process;
- The continuous-time process, e.g. the system dynamics, such as pressure or temperature, may affect the stochastic process, e.g. failure rate;

- The continuous-time process may generate events, which in turn change the discrete state of the system.

The simulation model consists of hardware, software and human crew. Modeling choices for hardware systems are well established in most cases.

The evolution of the model is traced by solving the continuous-time system in the intervals between the discrete events generated by the discrete system. Whenever the time for a scheduled event is reached, the continuous-time simulation is stopped, and the corresponding event is executed. In some cases, the continuous-time system may generate an event, e.g. one of the variables crosses a given threshold.

The scheduler only directs the behavior of the stochastic model. There is no direct interaction between the scheduler and continuous-time deterministic model.

6.4.2 Interactions between Planner, Scheduler and Simulation Model

At the beginning of each simulation, the scheduler will load the plan from the plan file generated by the planner. The plan file is a text file, where the scenarios of interest are listed line by line.

As we have discussed in 4.6, the Planner may work at higher (more abstract) level. The scenarios generated in the plan may include high level events which are not directly represented in the simulation model. Such high level events would be translated to one or more event sequences based on, e.g., the fault tree type of model

of the engine failure. The detailed scenarios are used by the scheduler to guide the simulation. A database is maintained by the risk analysts to interpret the abstract scenarios generated by planner. When the scheduler loads the plan, if there are such abstract scenarios, the scheduler will query the database to get the detailed scenarios, and generate a detailed plan, which can be used to direct the simulation.

Also we have discussed in 4.7 that the plan is updated from time to time during the simulation. In the SIMPRA Navigator the user would specify the number of event sequences of one updating interval, and number of updating rounds. The planner will check the simulation results at the end each updating interval to determine how well the simulation is following the plan. If some scenarios have been underrepresented, the planner would automatically set the importance level of the specific scenarios to a higher level, which would in turn make the scenarios more favorable by the scheduler. The purpose of this adjustment is to maintain the exploration fairly distributed among different scenarios.

6.5 Simulation Model Building

MATLAB® is a computer language for technical computing. Simulink, which is a toolbox extension of MATLAB, is a software package for modeling, simulating, and analyzing dynamic systems. An icon-driven interface is used to construct a block diagram representation of a process. The process is composed of an input, the system,

and an output. A comprehensive block library of sinks, sources, linear and nonlinear components, and connectors is provided. The modeling of the time-dependent mathematical relationships among the system's inputs, states, and outputs is constructed in the form of block diagram, with blocks representing functional elements, and lines representing signals between those blocks.

6.5.1 The Library to Build the Simulation Model:

The SIMPRA provide a library for the analyst to build the DPRA model. The basic elements required to build the DPRA are provided in the library as blocks. Users can simply click and drag the blocks into the model they are building. The blocks include different failure modes, interface with scheduler, and system logic. The users can build such library blocks by themselves, or modify from the existing library blocks. Table 6-1 lists some of the elements available in SIMRPA library.

Table 6-1 Examples of the Elements of SIMPRA Library

Logic Gate	AND, OR, k-OUT-OF-n, if
Dynamic Fault Tree Gate	Functional Dependency, Spare, Priority-AND
Failure Rate	Weibull Failure Rate, Exponential Failure Rate
Failure Modes	Time-distributed, demand-base, Repairable.
Specialized for SIMPRA	Notify, End-State.

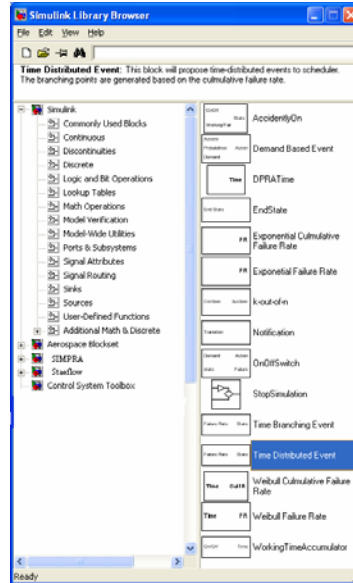


Figure 6.5.1 SIMPRA Library

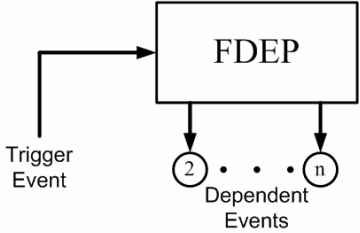
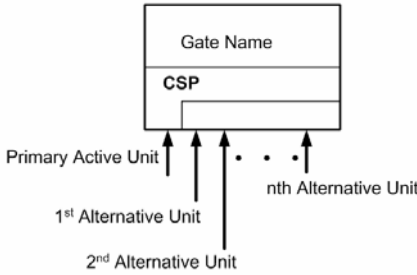
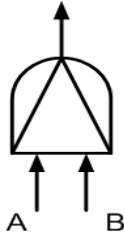
The interface between the simulation model and the scheduler is encapsulated in the library blocks. The user cannot see the interface and there is no need for the users to program the interface at all. Only the parameters of the components need to be specified.

Dynamic Fault Tree Library

Dynamic Fault tree Library is part of the DPRA library. Several dynamic fault tree gates are implemented. A major disadvantage of traditional fault-tree analysis is that it is incapable of capturing sequence dependencies in the system and still allowing an analytic solution. Several new dynamic fault tree gates were designed to model the sequence dependencies (Dugan, Bavuso, & Boyd, 1992). With these gates,

the fault tree is converted to a Markov chain, instead of the usual fault-tree solution methods. The gates provide a compact way to represent certain sequence dependencies. In a simulation environment, as we have discussed, there is no limitation for modeling the dependency. All possible sequence timing dependencies can be modeled explicitly. Reproducing the dynamic gates in the simulation environment gives the users an easy and compact way to represent some frequently seen types of dependencies, just as in the dynamic fault tree. The capability of modeling dependencies and other dynamic features of SIMPRA are far beyond the dynamic fault tree gates we list here. The Markov assumption in the Dynamic Fault Tree is also lifted here. The gates in SIMPRA represent the same logic of the dynamic tree gates, but they do not require Markov assumptions. Unlike the fault tree, which is failure oriented, in SIMPRA the state of the component is typically success oriented. In fault tree, including dynamic fault tree, when we say the output of a gate is true implies failure events happen. While in SIMPRA, a positive number implies that the component is working, and zero implies failure. The user can define specific numbers to represent different degraded state. So in SIMPRA environment using the dynamic tree gates may be a little confusing. The other Boolean logic gates should also receive special care to make sure the logic is correct.

Table 6-2 Dynamic Fault Tree

Gate	Illustration	Description
Functional-Dependency Gates		<p>The occurrence of some trigger event causes other dependent components to become inaccessible or unusable. The dependent component events (failure) are forced when trigger event occurs.</p>
Spare Gate		<p>Spares are components which replace the primary unit, if the primary unit fails. The output is true if all the input events happen.</p>
Priority-And Gate		<p>The output of the gate is true if both of the following conditions are satisfied:</p> <ul style="list-style-type: none"> - Both A and B have occurred, - A occurs before B.

i. Functional-Dependency Gates

In SIMPRA, a functional dependency gate has two inputs, the trigger event and dependent event. They represent the working/failure state of components. If the first input change from 1 to 0, which implies component failure, it is equivalent to “trigger event occurs” in dynamic fault tree. The output is the state of the dependent

components. If trigger event occurs, the dependent components fail. The dependent components may have other failure or degradation modes. The input of dependent events may be multi-dimensional, and represent many components.

ii. Spare Gate

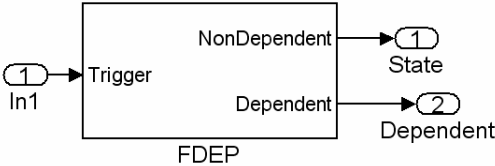
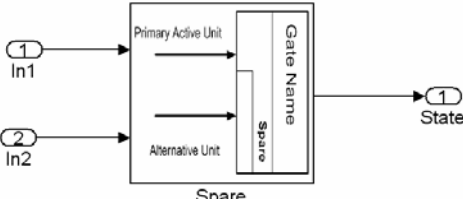
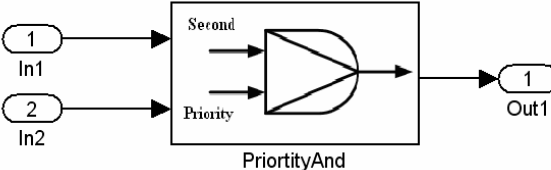
In SIMPRA, a spare gate has two inputs, the primary component and spare component. They represent the working/failure state of components. Initially the primary input is 1, meaning that the primary unit is working. The output is also 1. If the first input change from 1 to 0, it implies that primary unit fails. If the spare unit successfully replaces the primary unit, the output is the state of the spare unit, otherwise the output is 0. The dynamic fault tree has three distinctive spare gates: hot, warm, and cold. Different types of spare will be translated into quite different Markov chains. In the simulation environment, the component failure behavior is the characteristic of the component, whether it is before or after the activation. In SIMPRA, the spare gate only represents the system configuration. There is no need for separate gates of different spare types. The spare units can be cold, warm or hot. The only different between these spare types is that they have different failure/degradation behavior before it is activated. The system logic will not be influenced by this, and it is represented by the same spare gate.

iii. Priority-And Gate

In SIMPRA, a Priority-And gate has two inputs, the priority component and

second component. They represent the working/failure state of components. Initially both inputs are 1, meaning that the units are working. The output is also 1. If input changes from 1 to 0, it implies that the corresponding unit fails. The output stands for the system state. Only when both units fail and fail the exact order, the output will change to 0 at the failure of the second component.

Table 6-3 Dynamic Fault Tree Implemented in SIMPRA

Gate	Illustration
Functional-Dependency Gates	
Spare Gate	
Priority-And Gate	

6.5.2 Running the Simulation:

Figure 6.5.2 shows the SIMPRA Navigator. The names of parameters are self-explaining. Users need to specify the parameters, number of event sequences, name of the plan file, etc. Users can initiate the planner by clicking the “Generate Plan” button. User can input the component tree, functionality tree and the state diagram as we have discussed in 6.2.

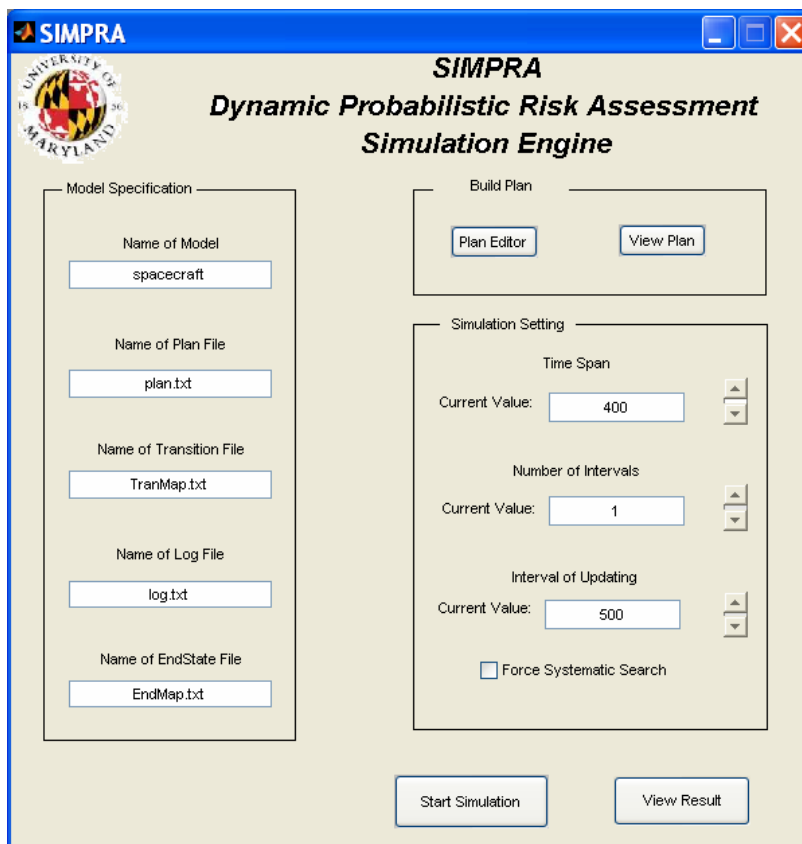


Figure 6.5.2 SIMPRA Navigator

Clicking “Start Simulation” button would start the simulation. The simulation will read the plan file and initialize the scheduler. A window will pop up at the beginning the simulation (see Figure 6.5.3). It displays the simulation results dynamically. The left upper window shows the estimate of end state probabilities. Right upper window shows the details of generated event sequences. The lower part shows the how event sequences distributed among the scenarios in the plan. We can see the expected information gain and number of sequences already generated in each scenario.

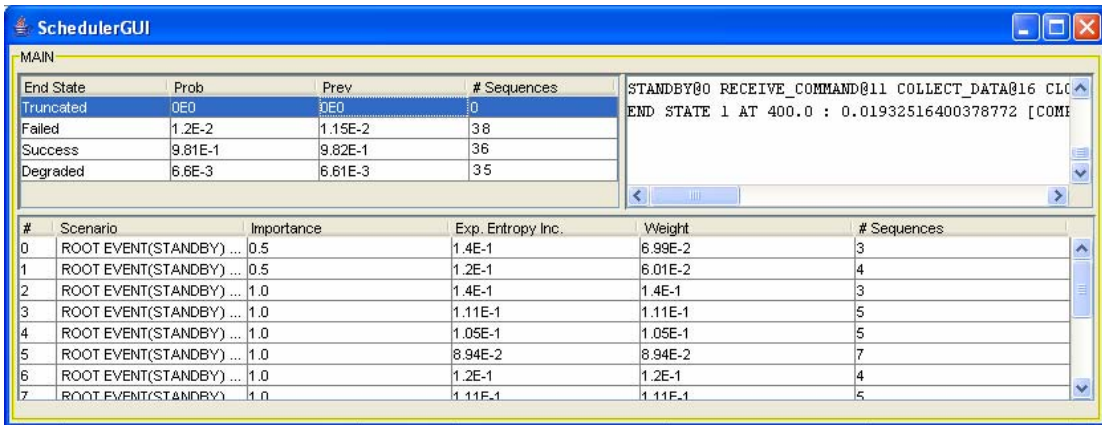


Figure 6.5.3 Runtime GUI

The simulation result is stored in a text file, for later investigation. The probability estimation plot and event sequences details are dynamical updated in a pop window.

When simulation finishes, we can see the result. The End State Display GUI displays estimate of probability of the end states and the event sequences (see Figure

6.5.4). In addition to the above displaying capabilities the Display has a built in filtering capability, which can be used to filter a sequence, or a set of sequences, e.g. if the user is interested in displaying only sequences which have the expression “AOG”. The user types in “AOG” on the text field provided and only sequences which contain the term “AOG” are displayed.

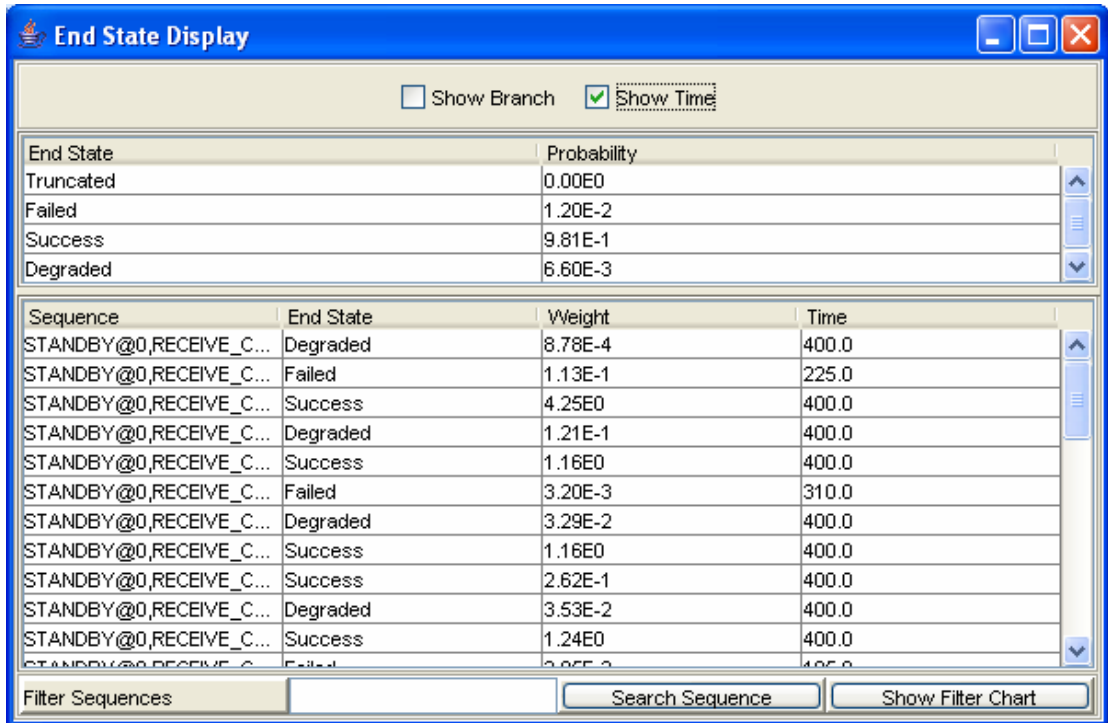


Figure 6.5.4 End State Display

6.5.3 Hardware Component Failure Modeling

The hardware model is able to simulate the behavior the hardware, including random

events, especially the transitions of system state, such as hardware failure.

When we are building the simulation model, it is assumed that the hardware components have a finite number of states, e.g. working, failed, degraded.

Furthermore, we assume that the transition from one state to another is instantaneous.

To implement the hardware failure in the model, we use the branching rules described in the previous sections.

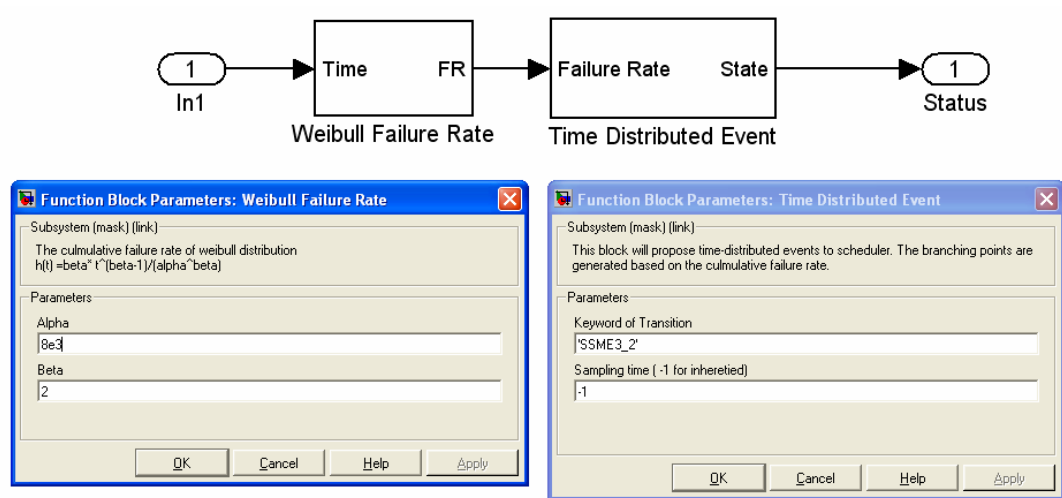


Figure 6.5.5 A Typical Hardware runtime failure block

In the figure above, the “Time” port is the input port of the module. The “Weibull Failure Rate” and “Time Distributed Event” are blocks of the SIMPRA library. The “Weibull Failure Rate” is an example of the failure rate calculation block, which is responsible for calculating the failure rate the components. Such blocks may require the input, such as time and working load to get the failure rate. The parameters of the

failure distribution will be input by the users. The “Time Distributed Event” block is a branch generation block, which generates branch point according the rules we have discussed in Chapter 5. Whenever a branching point is reached, this block will communicate with the “scheduler”. The output of this block is the current state of the component, designated by an integer.

6.5.4 Event Notification

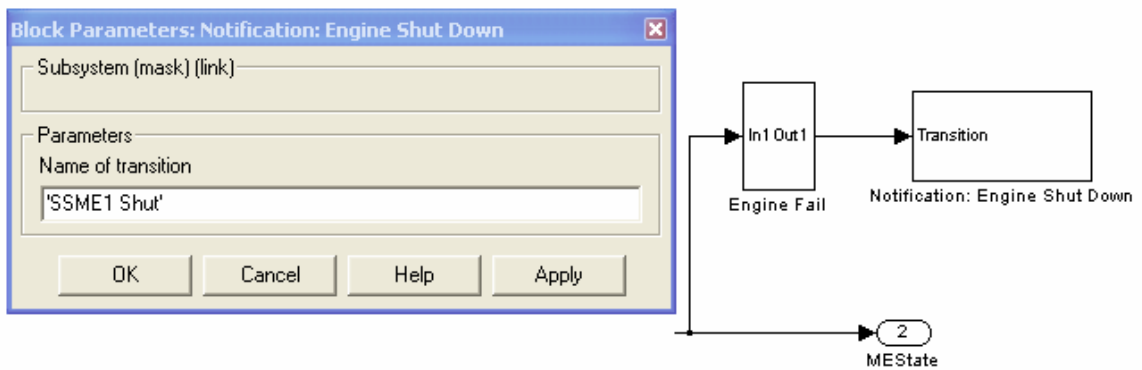


Figure 6.5.6 Event Notification Block

Event state notification blocks monitor the occurrence of events that can not be directly controlled, e.g., physical conditions, indirect failure of systems. This kind of events may play an important role in the system’s dynamic evolution. For example, if the pressure reaches a threshold, the wall of the tank may rupture. In such situations, the plan would specify such events, and scheduler would thus monitor the occurrences. In the simulation model, “Event State Notification” blocks notify the scheduler whenever such events take place.

6.5.5 System State Block

The system consists of a number of components, which may interact with each other. The hardware interacts with the other modules of the system, such as human and software modules. The output of the state variable would be fed into other blocks or modules of the model. The state of the components will influence the behavior of corresponding components, such as thrust provided by main engine, which in turn may result in the transition of the state of the system.

The State Logic Blocks determine the system or sub-system states by analyzing the components states, using the logic operation gates, provided in standard Simulink library or the SIMPRA block-set library, or user defined s-functions. The logic operation gates provided in the Simulink standard library includes the Boolean operations: AND gate, OR gate, NOT gate and etc. The SIMPRA block-set library provides the k-out-of-n gate, which is more generic, and the Dynamic Fault Tree gates, such as Priority-And, Spare, and Functional-Dependency Gates.

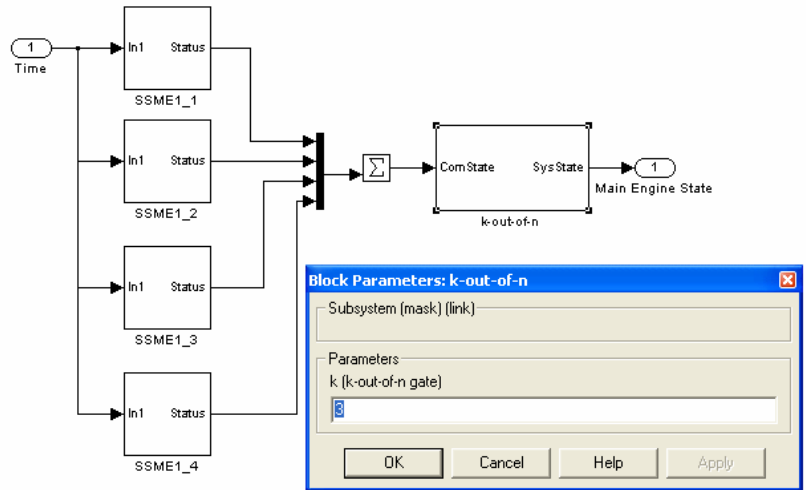


Figure 6.5.7 Example of Hardware State Logic

Note that although the gates in SIMPRA look like the fault tree gates, the logic is slightly different than those in the fault trees. Fault trees are failure oriented. Our simulation model is not limited to failure only. In a fault tree gate, if the output is true, that means component failure. In SIMPRA, often the opposite is the case. Typically, 0 implies failure, and 1 implies working. The state of the SIMPRA block is not limited by binary states. The components can have multiple states defined by users.

If the state logic requires more complex logic operations, such as time condition, a user-defined s-function would be more appropriate. S-functions can be written in different computer languages, including MATLAB®, C, C++, Ada, or FORTRAN. S-functions can interact with Simulink equation solvers, in a very similar way that takes

place between the solvers and built-in Simulink blocks. User can implement algorithms in an S-function. A customized user-interface can be obtained by writing an S-function and placing its name in an S-Function block (available in the User-Defined Functions block library).

6.5.6 End State Notification

End state notifications cause the simulation of a sequence to stop when a predefined condition is reached in the system. The simulated event sequence would be stored for later study.

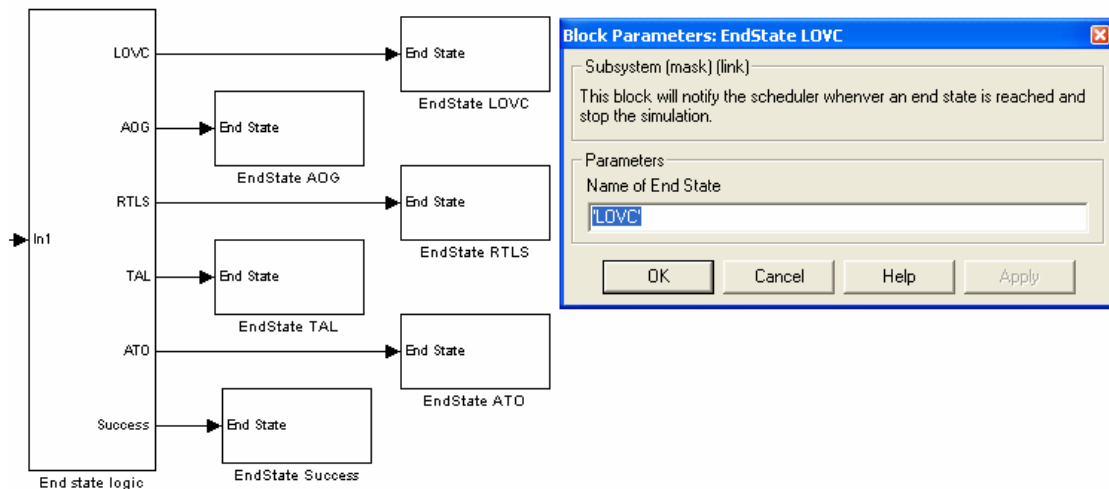


Figure 6.5.8 End State Block

The “End State Logic” block is an instance of “System State Logic” block. It may use the logic operation gates and/or user-defined s-functions to determine whether an end

state is reached or not and which end state is reached. When an end state is reached, the simulation is stopped, and the scheduler is notified. The scheduler will record the event sequence of this round of simulation, and start the simulation again.

6.5.7 Human Behavior Modeling

The objective of the human model is to simulate human behavior and the interaction between human and other parts of the system. The human model receives system information as inputs, and outputs the human action.

The current human model is based on previous IDAC, ADS-IDAC model (Mosleh & Chang, 2004). IDAC is a model of human error. IDAC uses a representation of human behavior in information processing (I) problem solving/decision making (D), and performing tasks (A), in order to develop an explanation of the likely response of a crew (C).

The human model consists of two modules: the information processing behavior module and the knowledge representation module. The information processing behavior module simulates the information processing of the human crew. It simulates that a given information processing strategy which action the human will take and estimates the associated probabilities. An often used approach of cognition modeling is by means of Performance Influencing Factors (PIFs) or Performance Shaping Factors (Hollnagel, 1998). Human action is influenced by psychological

factors like stress, level of attention and time constraints. The human model estimates the probability of possible human actions, based on the states and values of the PIFs.

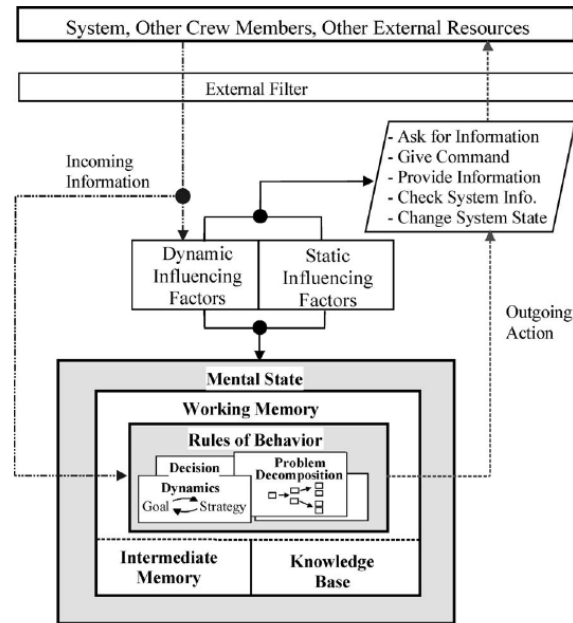


Figure 6.5.9 High level view of the IDAC response model (Mosleh & Chang, 2004)

The knowledge representation module simulates the human knowledge base (i.e., memory). The individual gathers cues about the system’s status and searches his/her memory to establish which action is most appropriate for the situation. However, the way the memory is searched depends on the problem solving strategy adopted.

The memory is classified into long term memory and short term memory. The knowledge base (long-term memory) is where all the knowledge and experiences gathered during an individual’s life are stored. For example, an operator’s long term

memory may contain the understanding of the functional characteristics of the system and its underlying physical processes; guidelines on how to respond to accidents, and expected response of system to perturbations, learned through training and operating experience (Mosleh & Chang, 2004).

The quantity and quality of information accumulated in the long-term memory reflects the level of experience and training of an individual. The knowledge base has a great impact on how an individual will deal with threatening encounters. The working memory (also called short-term memory), as the name suggests, decays quickly. In addition, the capacity is very limited.

In many human reliability models, human error probability (HEPs) are calculated as a function of PIFs, and one common assumption made is that the PIFs are independent. For example, in one method, the HEP is assumed to be a function of the weighed sum of the effects of the PIFs, by combining them in the linear formulation of equation. Such assumption does not reflect the reality. For example, stress cannot be independent of task complexity or information load. Another negative aspect of these techniques is that they don't provide any explicit description of how the PIFs influence human performance.

The Bayesian Belief Network (BBN) approach is an alternative, and is used in SIMPRA human models. It uses the PIFs to estimate the likelihood that a specific cognitive behavior is going to take place in certain situations, as opposed to the

calculation of HEPs. All the PIFs are arranged in a Bayesian Belief Network (BBN), where each node represents one PIF, and the connections between them represent their affects in each other. Figure 6.5.10 shows an example of how BBN model the relationship among PIFs.

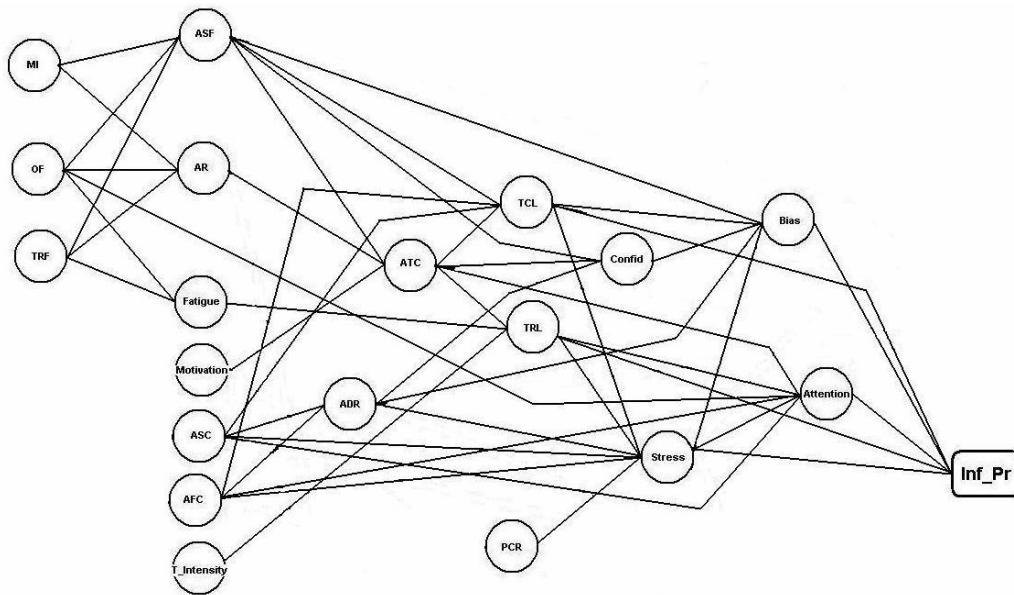


Figure 6.5.10 The BBN framework

For details of human modeling, see (Mosleh et al., 2005)

6.5.8 Software Modeling

A significant number of system failures can be attributed to software malfunction. It is thus imperative to take account of the software behavior and model the impact of software on system evolution to correctly represent the risk scenarios. Research has been conducted to integrate software contribution into traditional PRA framework

(Li, Li, Ghose, & Smidts, 2003; Li, Li, & Smidts, 2004, 2005). Past effort have focused on the software testing (Li, 2004). In this research, we put the software in the context of system dynamic evolution.

In the DPRA environment, we first construct an executable software model to simulate the software behaviors. Different methodologies exist for software modeling, including finite state charts (Harel, 1987), UML (Fowler & Scott, 1999), pattern concept (Gamma, Helm, Johnson, & Vlissides, 1995). A broad categorization divides these and other methods into those that are based on the data flow inside software, and represent the software through decomposition of system into dataflow diagrams that captures the successive transformations of system input into system output, and those that model the procedural stages of the software, represented in the form of states and transitions between these states, leading to some kind of finite state chart. Finite State Machine (FSM) is chosen to build the software behavior model.

The analysts' task is to build an executable software model and identify possible software related initiating events. The simulation environment will explore the scenario space based on the system model. The software risk and vulnerabilities will be identified using the simulation results. The analyst no longer needs to study the fault propagation and enumerate all the possible accident sequences.

The software behavior model is a combination of a deterministic model and stochastic model. The deterministic model is used to simulate the behavior of the

software, as well as the interaction between the software and the other parts of the system. The stochastic model represents the uncertain behavior of the software. The software related failure modes can be identified in a similar way as in the traditional PRA framework. The selected failure modes will be super imposed on the executable behavior model as stochastic events, and are controlled by the simulation scheduler during simulation based on the predefined rules to explore the risk scenarios space following the selected initiating events.

7. Application I – Hold Up Tank

7.1 Introduction

Holdup tanks are widely used in different engineered systems, and actually the control of liquid level in the tank is one of the oldest control problems. Variations of holdup tank problem have been widely discussed in the dynamic PRA literatures. Aldemir used a hypothetical holdup tank problem as an example for his dynamic approach based on Markov chain to analyze process control systems dynamics (Aldemir, 1987). In (Deoss & Siu, 1989) the same problem was studied using DYMCAM (Dynamic Monte Carlo Availability Model). Later Siu studied the problem to demonstrate different dynamic PRA methods (Siu, 1994). Cojazzi applied DYLAM to study similar tank control risk analysis (Cojazzi, 1996). (Dutuit et al., 1997) use Petri nets to study a similar problem with Markov assumptions.

7.1.1 Outline of the Holdup Tank

Figure 8.1.1 shows the layout of the simple tank system. The tank holds liquid chemicals, and liquid level is regulated by the actions of the control loops. At **time** = t_0 , the initial level of the tank is L_0 . Under nominal conditions Pump1 will pump in

the same amount of liquid as that flow out through the valve, and therefore the level is maintained constant. If a failure of the components happens, the level of tank changes, and the control system may intervene. The events when the tank level (L) rises above a certain level (overflow), or falls below a certain level (dry-out), are considered to be a system failure. The time-dependent probabilities of “overflow” and “dry-out” are quantities of interest.

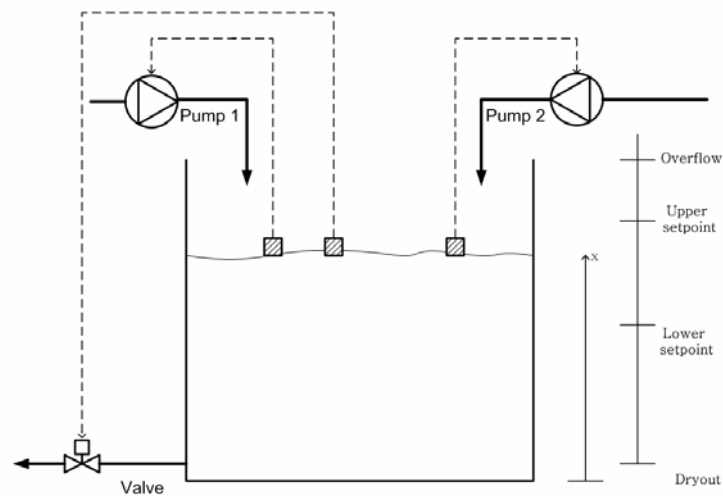


Figure 7.1.1 Holdup Tank System Layout

In all the following studies we have made these assumptions:

- 1) The pumps and the valve have separate level sensors.
- 2) The control units of pumps and valve are activated by the level signal.

The response is instantaneous, and the time delay is negligible.

- 3) Failures are not repairable.
- 4) The units are either On or Off.
- 5) The flow rate of the pumps and valve is constant when they are on.

7.1.2 Dynamic Feature of the Holdup Tank Problem

In the holdup tank problem, timing and order of failures are critical to system safety. The accident initiated by the failure of the pump would be quite different from the one initiated by the failure of the valve. The system control relies on the process variables, so it would be oversimplifying the problem if we neglect the process variables in the analysis.

Siu first analyzed the tank problem with traditional event tree methods (Siu, 1994). This analysis showed that the event tree analysis can display the correct failure logic of dynamic systems, but by ignoring the process variable, it cannot determine the distribution of time to an undesirable state. The event tree is basically a Boolean logic, so the only way event tree can take the process variable into account is by discretizing the process variables ranges. When we need detail process variables or when the number of variables increases, the event tree may grow unmanageable. Furthermore, without a physical model the event tree analysis has to involve subjective judgment of the interaction between variables. As a result the assessment of the probability of arriving the end states may be inaccurate.

Cojazzi has reported that with long time constant, there would be a significant difference between the results obtained from static fault tree methods and DYLAM.

In the following sections we will build several different simulation models with additional assumptions. The simulation model is based the SIMPRA DPRA platform. The user can set up the parameters in the model, which include the flow rates of pumps and valve, the capacity of tank, the set points of the control system and other parameters depicting the failure characteristics. A Weibull failure model is used of the pumps and the valve to simulate the failure modes. We can set parameters and slightly modify the model to model different systems in the following case studies.

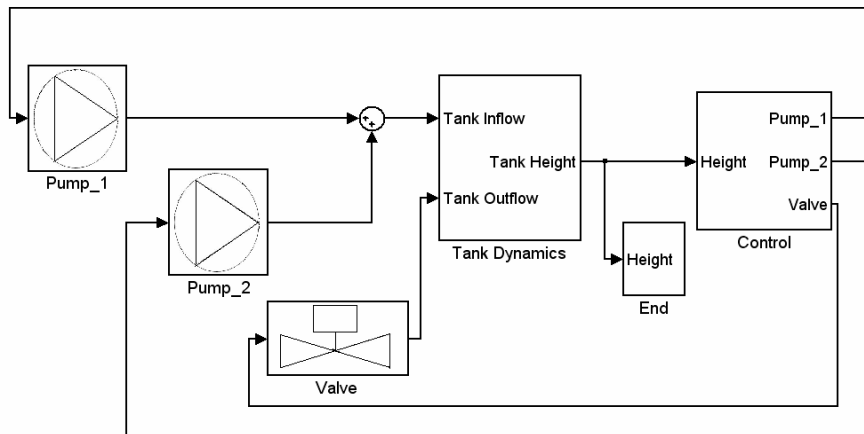


Figure 7.1.2 Layout of the simulation model of Holdup Tank

7.2 Case I

7.2.1 Problem Statement

This case is a reproduction of *case F* in (Aldemir, 1987) and case 1 in (Cojazzi, 1996). Additional assumptions in this case are:

- 1) System set points and corresponding control laws shown in Table 3-1.
- 2) Flow rate of valve and Pump1 is 0.06 m/h, and for Pump2, 0.03m/h.
- 3) All failure rates are constant, in time. The failure rate for valve $\lambda_1 = 1/320h$, pump1 is $\lambda_2 = 1/219h$, and pump2, $\lambda_3 = 1/175h$.
- 4) If the failure occurs, the unit works on the contrary to the control signal.
- 5) If the liquid level is greater than **3**, the system fails by “overflow”, and if the liquid level is less than **-3**, the system fails by “dry-out”.

Table 7.1 Control Laws as a Function of Liquid Level

Level	Valve	Pump1	Pump2
$L \leq 1$	Open	On	On
$-1 \leq L \leq 1$	Open	On	Off
$L \geq -1$	Close	Off	Off

7.2.2 Analysis in Previous Work

In this case Pump 2 is not sufficient to restore the system to nominal state if the valve “fails on” when Pump 1 “fails off” (dry-out), or when the Pump 1 “fail on” when valve “fails off” (overflow). A further assumption was made in (Aldemir, 1987) that no failure occurs following a component failure until the system enters a new control region. Figure 8.2.1 illustrates two accident scenarios under this assumption. .

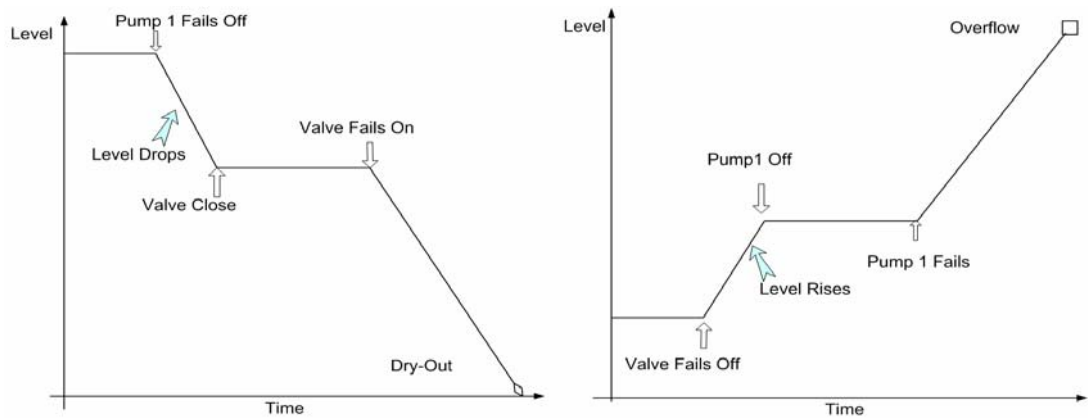


Figure 7.2.1 A Typical History of Tank Level Evolution

The two accident scenarios depicted in the figure are triggered by Pump1 failure first or valve failure first:

- i. The water level drops when Pump 1 fails off, while the valve is still open. No other component failures occur until the valve closes when the set point is reached. After that, if the valve fails (fails on), the tank will dry out.

- ii. The water level rises when the valve fails off, while Pump1 is still on. No other component failures occur until Pump1 is turned off when the set point is reached. After that, if the Pump1 fails (fails on), the tank will overflow.

If we lift the assumption that no failure occurs following a component failure until the system enters a new control region, there are several new scenarios:

- i. Given Pump1 fails off, if valve fails when the liquid level is still dropping (fails off), then the system is still under control. After that if Pump2 fails, the tank overflows.
- ii. Given Pump1 fails off first, if Pump2 fails (fails on) when the liquid level is still dropping, it will take longer for the liquid level to reach the set point to shut the valve. If the valve fails before the liquid level reaches the set point (fails off), then the tank overflows.
- iii. Given valve fails off first, if pump2 fails (fails on) when the liquid level is still dropping, it will take less time for the liquid level to reach the setting point to shut the valve. If the valve fails before the liquid level reaches the set point (fails off), then the system still under control. After that if pump2 fails, the tank overflow.
- iv. Given valve fails off first, if Pump2 fails when the liquid level is still

rising (fails off), then the system still remains under control. After that if Pump2 fails, the tank overflows.

We can see that if we remove the additional assumptions, even if Pump 1 fails before valve fails, the tank may overflow instead of drying-out. If the valve fails off before Pump 1 or 2 fails, it is impossible for the tank to dry-out. The time window for any component to fail between one component failure and reaching another control region is relatively short (about 20 hours) compared to the MTTF of the components involved. Neglecting such scenarios would not lead to significant error.

With this assumption, an approximated solution is:

$$probability(dryout_before_t') = \int_0^{t'} f_{valve}(t) \cdot R_{pump1}(t) dt$$

$$probability(dryout_before_t') = \int_0^{t'} f_{pump1}(t) \cdot R_{valve}(t) dt$$

Note that the equation implies that when the cut-set occurs, the top event occurs simultaneously. In this case, we must consider sequence. “Pump1 failure followed by valve failure” is the cut-set leading to dry-out, while “valve failure followed by pump 1 failure” is the cut-set leading to overflow. We know, however, there is a time delay for the top event to happen. If we are calculating asymptotic values of CDF for very long mission times, we can neglect the time delay. In this case:

$$F_{dryout}(\infty) = \int_0^{\infty} \lambda_2 \cdot e^{-\lambda_2 t} \cdot (1 - e^{-\lambda_1 t}) dt = \lambda_1 / (\lambda_1 + \lambda_2) = 0.41 \quad (7.1)$$

$$F_{overflow}(\infty) = \int_0^{\infty} \lambda_1 \cdot e^{-\lambda_1 t} \cdot (1 - e^{-\lambda_2 t}) dt = \lambda_2 / (\lambda_1 + \lambda_2) = 0.59 \quad (7.2)$$

Equations 7.1 and 7.2 are used in (Aldemir, 1987) and (Cojazzi, 1996).

Figure 8.2.2 shows the result reported in (Aldemir, 1987) and (Cojazzi, 1996).

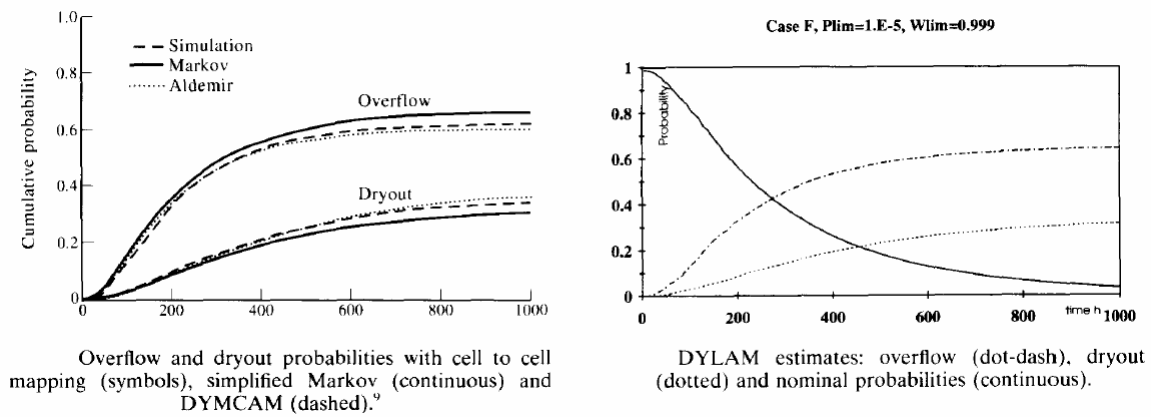


Figure 7.2.2 Others Work

7.2.3 Simulation with SIMPRA

A SIMPRA simulation model built with the same parameters used by others.

Because the mission time is much longer than the MTTF, it is assumed that at least two components fail during the mission. We run the simulation with a simple plan which numerates all possible event sequences:

- Pump 1 failure, valve failure

- Valve failure, pump1 failure
- Pump 1 failure, Pump 2 failure,.
- Valve failure, Pump 2 failure,
- Pump 2 failure, Pump 1 failure,
- Pump 2 failure, Valve failure,
- Pump 1 failure, valve failure Pump 2 failure.
- Valve failure, pump1 failure Pump 2 failure
- Pump 1 failure, Pump 2 failure, valve failure.
- Valve failure, Pump 2 failure, pump1 failure
- Pump 2 failure, Pump 1 failure, valve failure.
- Pump 2 failure, Valve failure, pump1 failure

The cumulative distribution functions of the overflow and dry-out are obtained through SIMPRA. They are shown in Figure 8.2.3. The result agrees with the result from DLYAM.

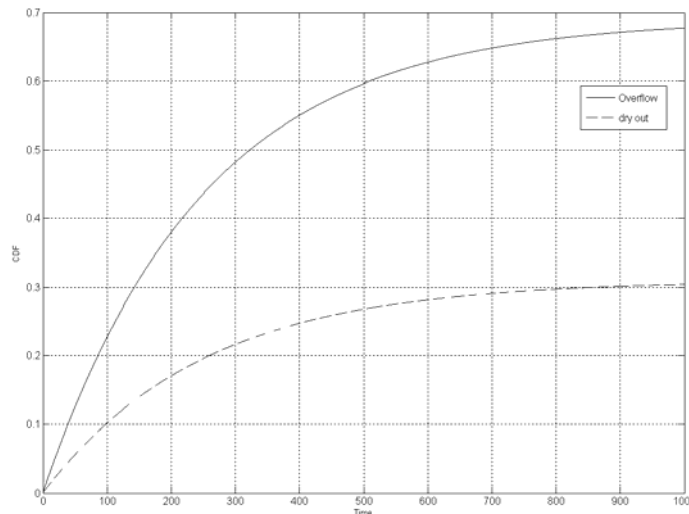


Figure 7.2.3 Probability estimate from SIMPRA

7.2.4 Scenario Analysis

The simulation estimate of overflow probability is higher than the result reported in (Aldemir, 1987). So in this section, we analyze solution provide in (Aldemir, 1987) and the simulation result in details.

The equations 7.1 and 7.2 imply that:

- i. Dry-out is equivalent to the scenarios that Pump 1 fails before valve fails.
- ii. Overflow is equivalent to the scenarios that valve fails before the Pump 1 fails.

An unstated assumption is that whether or when Pump 2 fails does not change the end state of the system. This assumption is shown to be false through the SIMPRA

simulation, and in fact the solutions in equations 7.1 and 7.2 are not accurate. During the simulation, we have seen scenarios contradictory to this assumption. Figure 7.2.5 is a pictorial summary of some scenarios we have observed in the simulation.

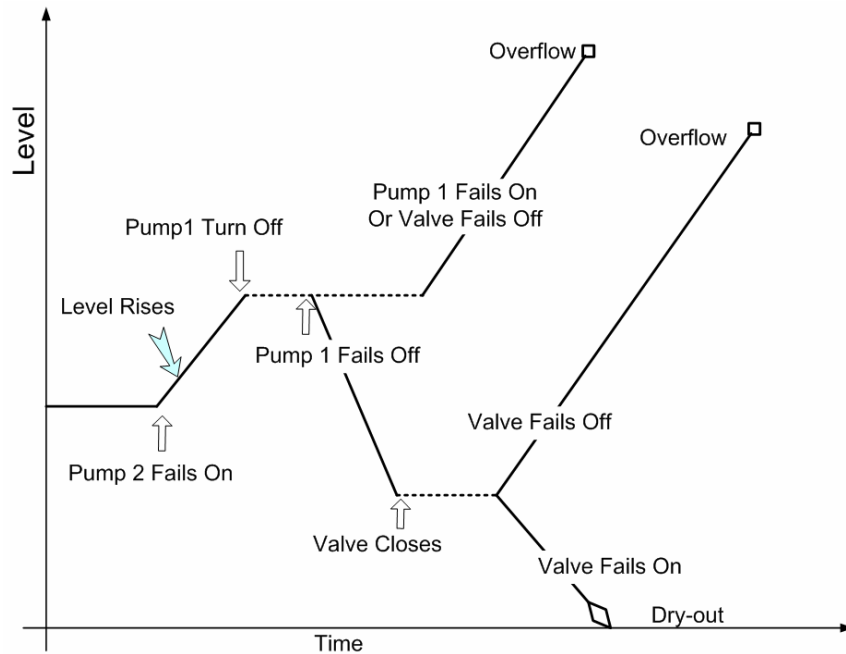


Figure 7.2.4 Accident Scenarios Triggered by Pump2 Failure

If we look into the scenarios more carefully, even with the assumption that no failure occurs following a component failure until the system enters a new control region, there are several scenarios neglected by Aledmir (Aledmir, 1987).

To summarize:

- 1) If Pump2 fails first (fail-on), Pump 1 will oscillate around the set point. The tank may overflow or dry-out

- a. Pump 1 fails before valve
 - i. If Pump 1 fails on, the tank will overflow definitely.
 - ii. If Pump 1 fails off, liquid level drops, the valve oscillates around the set point.
 - If valve fails on, tank will dry-out, and
 - If valve fails off, tank will overflow.
 - b. Valve fails before Pump1 (valve fails off), the tank will overflow definitely.
- 2) If Pump 1 fails first (fail-off), the liquid level drops, till the valve close at the set point.
- a. If the valve fails off before reaching the set point, then Pump 2 failure would lead to overflow.
 - b. If valve fails on, the tank will dry-out definitely.
 - c. If Pump 2 fails on, the valve oscillates around the set point.
 - If valve fails on, tank will dry-out, and
 - If valve fails off, tank will overflow.
- 3) If the valve fails first (fails close), tank overflows

We calculate the asymptotic probability of the overflow and dry-out by summing

the probabilities of all these scenarios.

$$F_{dryout}(\infty) = 0.31$$

$$F_{overflow}(\infty) = 0.69$$

This analytical solution is very close the result reported in (Cojazzi, 1996) and simulation result obtained by SIMPRA in 7.2.3.

7.3 Case II

7.3.1 Problem Statement

In previous cases, all the failures are considered to be randomly distributed in time. In the present case, we consider failures on demand also. Possible unit states are nominal on, off, failing to switch (either turn on or off), runtime failure, accidentally turned on. Table 6-3 provide the list of stochastic failure models.

Table 6-4 Parameters Used in Case II

	Fail-off (Weibull)	Fail-on (exponential)	Fail to Switch
Pump1	$\alpha=500, \beta=2$	$\lambda = 0.001$	P = 0.001
Pump2	$\alpha=500, \beta=2$	$\lambda = 0.001$	P = 0.001
Valve	$\alpha=500, \beta=2$	$\lambda = 0.001$	P = 0.001

7.3.2 Scenario Analysis

This case was studied by Siu (Siu, 1994). An ESD is given in (Siu, 1994)

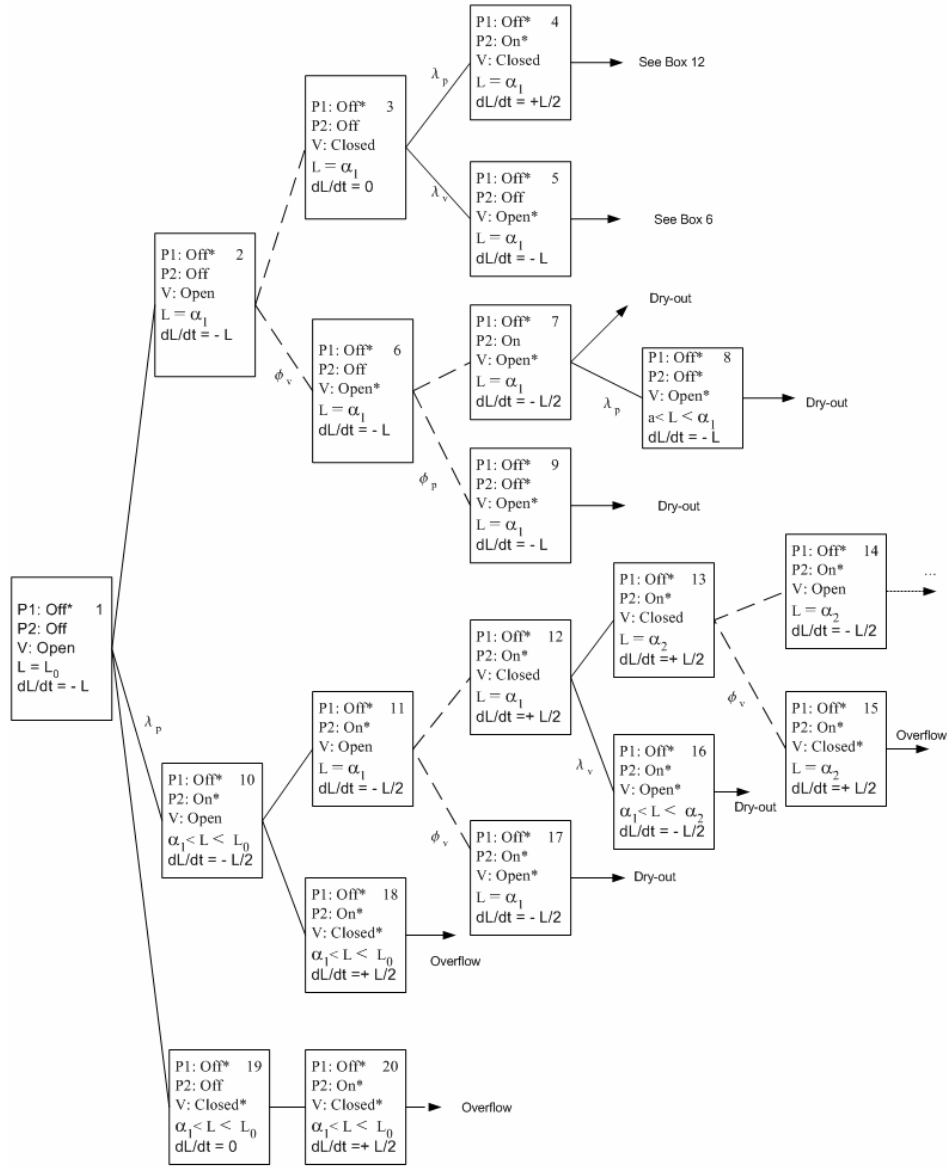


Figure 7.3.1 ESD of the holdup tank, adapted from (Siu, 1994)

The plan for SIMPRA is generated based on the ESD developed in (Siu, 1994). The simulation generates many scenarios outside the ESDs by Siu. One scenario which happens frequently outside the plan is depicted in the following figure.

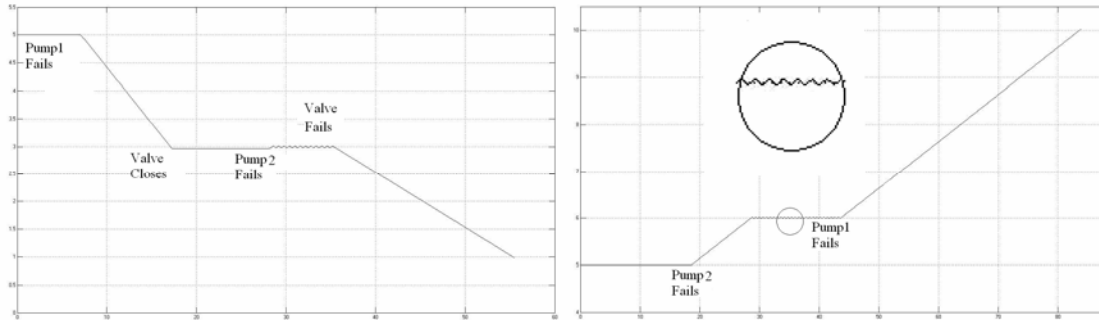


Figure 7.3.2 Scenarios of Case II

One of failure scenario (left) goes like this:

1. Pump 1 failed at 7 sec, the valve is still open. (shown by the downhill slope)
2. When the liquid level reaches 3 meters, which is the open/close set point of the valve, and the valve successfully closes. The liquid level becomes stable.
3. Pump 2 is accidentally turned on at 28 sec, the system becomes unstable, Pump 2 is frequently turned on and off, and valve is frequently opened and closed.
4. The valve fails 35.4 sec.

5. Tank dries-out.

This scenario is very similar to the one described by Siu as ESD 1-10-11-17. The ESDs developed by Siu did mention the possibility of system becoming unstable. The unstable system almost would definitely lead to component failure, because of the frequent turn on/off operation. If this is ignored the resulting system reliability estimate would be too optimistic. The real system design would surely avoid such unstable situation; one easy way is to use relay control. These event sequences resulted from our simulation show that the accidents scenarios we are studying are more complex than those analyzed in (Siu, 1994), and system behavior is significantly influenced by the control design.

Similar failure scenario is also described in Figure 7.3.2 (right).

The detailed accident event sequence goes like this:

1. Pump 2 is accidentally turned on at 18.5 sec. (shown by the uphill slope)
2. When the liquid level reach 6, which is the higher turn off set point of Pump1, the system becomes **unstable**. Pump 1 is frequently turned on and off.
3. Pump 1 failed to be turned off at 43.7 sec.
4. Tank overflows.

By using relay in the control system, the stability of the system would be

enhanced considerably. In Figure 8.3.4 the control relay is 0.5m. The valve /pump switch on/off about every 10 minutes, instead of switching on/off constantly.

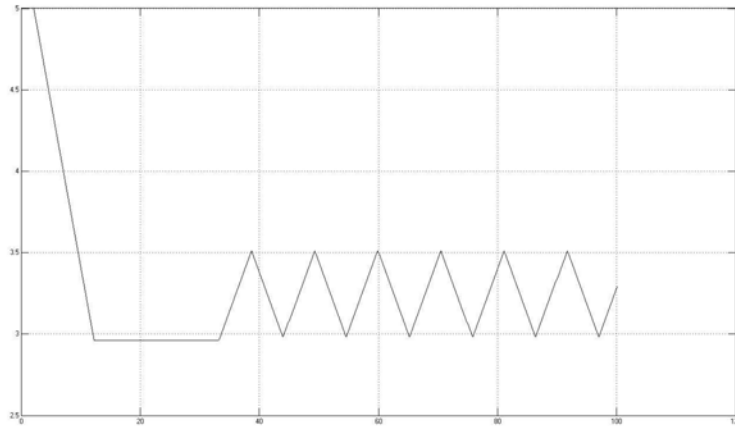


Figure 7.3.3 Accident Scenario Under Different Control Laws

These scenarios have not been discussed in (Siu, 1994). If we take a closer look at the ESDs developed by Siu, the repetitive turning on/off of the valve or pump2 is not expressed explicitly. A quasi-stable state is implicitly assumed. One explanation may be that any real control systems would consider the stability all the time, and are more sophisticated than the one we discussed here. Thus, the quasi-stable assumption is legitimate. Even so, we have to admit the important role played by the control laws. Apart from the reliability of the components, a good control law would improve the system safety/availability considerably. The control systems always involve complex feedback in the system and sophisticated algorithm, which is hard to be analyze qualitatively in traditional PRA, but this can be analyze quantitatively in the DPRA

framework. We can determine the effect of different control laws on the system safety and availability, as we have seen in the example.

7.3.3 Simulation with SIMPRA

The ESDs of (Siu 1994) were modified to account for the repetitive events such as turn on/off the pumps or valve. With modified plans, 500 histories were simulated by SIMPRA. Figure 7.3.4 shows how the generated event sequences are distributed among the plans. As we can see the amount of event sequences generated distribute relative fairly among the plans. The true probabilities of some of the plans are extremely low. For example, the theoretical probability of scenario 5 is below $5e-4$. Because there are several competing failure modes, and these scenarios require that the pumps and the valve to work properly for some cycles, before generating a failure, simply increasing the likelihood of single component failure, (i.e., traditional probability biasing) may miss this scenario.

The plan only includes the scenarios triggered by pump1 failure (Table 7-5). Sometimes the risk analysts may want to analyze some specific accident scenarios, and are not interested in the probabilistic estimate very much. They may list only the scenarios they want to simulate in the plan. In this example, out of 500 event sequences, about 466 event sequences are generated in the plan. About 7% of event sequences are generated outside the plan.

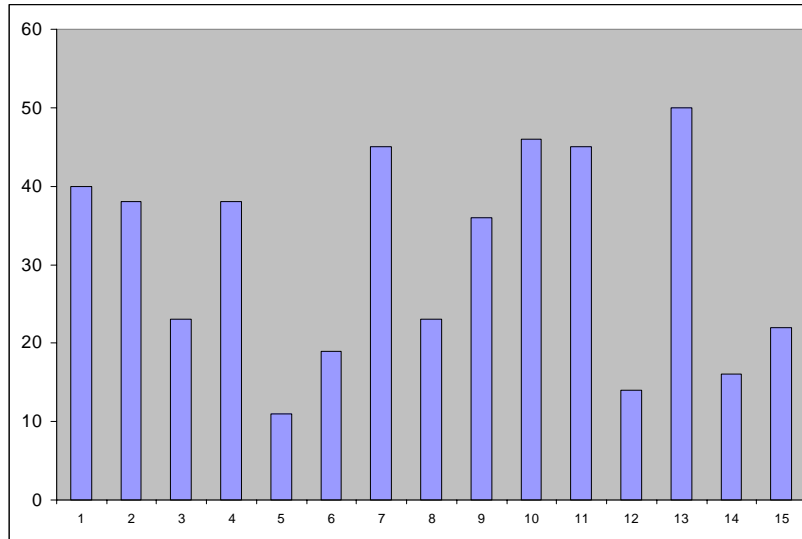


Figure 7.3.4 Allocation of Event Sequences Among Plans

Table 6-5 Plan for Case II

1	k=2;Pump1Switch_s;ValveSwitch_s Pump1_F_Off ValveSwitch_s Pump2_F_On ValveSwitch_f END_0
2	k=2;Pump1Switch_s;ValveSwitch_s Pump1_F_Off ValveSwitch_s Pump2_F_On REPEAT;ValveSwitch_s END_0
3	k=2;Pump1Switch_s;ValveSwitch_s Pump1_F_Off ValveSwitch_s Pump2_F_On REPEAT;ValveSwitch_s Valve_F_Off END_0
4	k=2;Pump1Switch_s;ValveSwitch_s Pump1_F_Off ValveSwitch_s Pump2_F_On REPEAT;ValveSwitch_s ValveSwitch_f END_0
5	k=2;Pump1Switch_s;ValveSwitch_s Pump1_F_Off ValveSwitch_s Valve_F_On Pump2Switch_s Pump2_F_Off END_0
6	k=2;Pump1Switch_s;ValveSwitch_s Pump1_F_Off ValveSwitch_s Valve_F_On Pump2Switch_f END_0
7	k=2;Pump1Switch_s;ValveSwitch_s Pump1_F_Off ValveSwitch_f Pump2Switch_s NEGATE;AFTER;ALL END_0
8	k=2;Pump1Switch_s;ValveSwitch_s Pump1_F_Off ValveSwitch_f Pump2Switch_s Pump2_F_Off END_0
9	k=2;Pump1Switch_s;ValveSwitch_s Pump1_F_Off ValveSwitch_f Pump2Switch_f END_0
10	k=2;Pump1Switch_s;ValveSwitch_s Pump1_F_Off Pump2_F_On REPEAT;ValveSwitch_s END_0

```

11 k=2;Pump1Switch_s;ValveSwitch_s|Pump1_F_Off|Pump2_F_On|
    REPEAT;ValveSwitch_s|ValveSwitch_f|END_0
12 k=2;Pump1Switch_s;ValveSwitch_s|Pump1_F_Off|Pump2_F_On|
    REPEAT;ValveSwitch_s|Valve_F_Off|END_0
13 k=2;Pump1Switch_s;ValveSwitch_s|Pump1_F_Off|Pump2_F_On|V
    alveSwitch_f|END_0
14 k=2;Pump1Switch_s;ValveSwitch_s|Pump1_F_Off|Pump2_F_On|V
    alve_F_Off|END_0
15 k=2;Pump1Switch_s;ValveSwitch_s|Pump1_F_Off|Valve_F_Off|Pu
    mp2_F_On|END_0

```

7.4 Comparison between SIMPRA results and other approaches:

The holdup tank is a relatively simple example. In Case 1 and 2, there are only a few dynamic elements. The analysis compares the numerical results obtained from SIMPRA with the analytical results and other dynamic reliability methods. The result from SIMPRA and other methods compare satisfactorily.

In Case 3, the model is more realistic and the accident scenarios are much more complex. A similar model has been studied by Siu. We start our simulation with a plan generated from his ESDs. Our simulation results show that the ESDs are simplified, and the real event sequences are more complex.

ESDs rely on the risk analysts to develop the accident scenario. This is true for ET/FT- based methods. ET/FT is a Boolean logic representation of the system. The mapping of scenarios into logic representations depends on engineering analysis,

determination of the interactions between different parts, identification of the consequence of a random event, and determination of the severity of the consequences associated with scenarios. Behind every logic model, there is another body of modeling efforts.

In the simulation environment the underlying modeling effort is more explicit. The accident scenarios are developed through simulation. In other word, the system expresses itself through simulation.

In DYLAM, the system is updates at fixed time step. This limits the capability of the DYLAM. It is very usual to find out that the system develops fairly slowly in one stage but changing rapidly in another stage. We can see this in the holdup tank example. In nominal conditions, the system is steady, but if some component fails, the system evolution is very fast. In such cases using one fixed time step in the whole procedure is not an appropriate choice. In SIMPRA, we have a discrete model coupled with a continuous-time model. The time-step of the continuous-time is adaptive.

8. Application II - Satellite Telecommunication Example

8.1 Introduction

In this chapter, a telecommunication system is analyzed in with SIMPRA. The system is still under design. An abstract model is built to simulate the behavior the system. In this example, a model that is only using logical and temporal relations in its input functions is presented. This example is about a space craft that receives commands from a ground station to collect data while orbiting a planet and sends the data back for analysis. A comprehensive model of the ground station and the spacecraft is built with the SIMPRA software.

In general, eight states are determined for the spacecraft: one initial state, five transitional states and two failure states. It is assumed that space craft spends a fixed amount of time in each state before moving to the next state except for

- 1) The standby state: The duration of time in that state is related to the commands received from the ground station and

- 2) The "Failed" state. This is an absorbing state, and the system will remain in this state till the end of simulation.

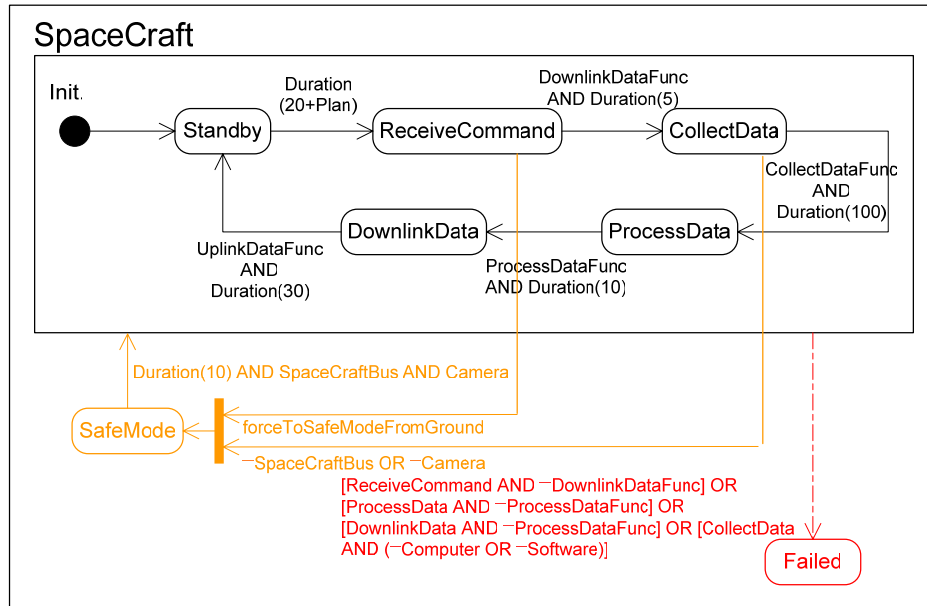


Figure 8.1.1 State Transition Graph

In this example, the transition input function includes a temporal function which indicates the duration of stay in the first state and a logic that describes the actions that need to be taken before transitions become activated. The Planner and the Scheduler work at different levels of abstraction. In the Planner, the process is to send data to the ground station, if all the components involved in the process are working, then the process is going to be successful. This assumption is just for planner. The detailed simulation will simulate the data generated from camera and simulate the downlink data and analyze the quality of the data sent to the ground station and then decide if the process was successful, failure, or degraded.

The Table 8-1 shows what subsystems and components are involved with which

functionalities. The table, for example, shows that “DownLink Data” has the following relations:

- High Level: Computer AND Telecomm AND SpaceCraftBus AND RCS AND Software
- Low Level: Clock AND Processor AND Transmitter AND Antenna AND HKSensors AND Pointing AND Software

Table 8-1 Component Functionality Matrix

Components		Downlink Data	Collect Data	Process Data	Uplink Data
Camera			×		
Computer		×	×	×	×
AND	Clock	×	×		×
	Memory		×	×	
	Processor	×	×	×	×
Telecomm		×	×	×	×
AND	Transmitter	×			
	Receiver				×
	Antenna	×			×
SpaceCraftBus		×	×	×	×
	HKSensors	×	×	×	×
RCS		×	×		×
	Pointing	×	×		×
Software		×	×	×	×

Table 8-2 Plan For Telecom System

Plan 1	STANDBY RECEIVE_COMMAND ANTENNA_F L END_0 L
Plan 2	STANDBY RECEIVE_COMMAND HOUSEKEEPING_F L END_0 L
Plan 3	STANDBY RECEIVE_COMMAND ANTENNA_D END_0
Plan 4	STANDBY RECEIVE_COMMAND HOUSEKEEPING_D END_0
Plan 5	STANDBY RECEIVE_COMMAND COLLECT_DATA CAMERA_F END_0
Plan 6	STANDBY RECEIVE_COMMAND COLLECT_DATA CAMERA_D END_0
Plan 7	STANDBY RECEIVE_COMMAND COLLECT_DATA HOUSEKEEPING_F END_0
Plan 8	STANDBY RECEIVE_COMMAND COLLECT_DATA HOUSEKEEPING_D END_0
Plan 9	STANDBY RECEIVE_COMMAND COLLECT_DATA PROCESS_DATA HOUSEKEEPING_F END_0
Plan 10	STANDBY RECEIVE_COMMAND COLLECT_DATA PROCESS_DATA HOUSEKEEPING_D END_0
Plan 11	STANDBY RECEIVE_COMMAND COLLECT_DATA PROCESS_DATA DOWNLINK_DATA ANTENNA_F END_0
Plan 12	STANDBY RECEIVE_COMMAND COLLECT_DATA PROCESS_DATA DOWNLINK_DATA HOUSEKEEPING_F END_0
Plan 13	STANDBY RECEIVE_COMMAND COLLECT_DATA PROCESS_DATA DOWNLINK_DATA ANTENNA_D END_0
Plan 14	STANDBY RECEIVE_COMMAND COLLECT_DATA PROCESS_DATA DOWNLINK_DATA HOUSEKEEPING_D END_0

The modeling process for the components is simple. The assumption is that all of the components have only two states: “Work” and “Fail” (without any repair). Logic of the functional relations between the components and subsystems are provided by the users so when it comes to low level modeling, those relations are already provided. A library of subsystems and components as well as a library of functionalities with importance measures can be kept as a knowledge base for planning and simulation of different systems. To generate a list of scenarios that might cause the system to fail, a model of the system should be generated. Then by using the library of functionalities, transitions between the states of the system can be defined and a plan for failing the system can automatically be generated. Table 8-2 lists the plan that is used to run the simulation. Based on the knowledge, the Planer generates plans to take system from one state to another.

8.2 Scheduler/Simulator side

The satellite telecommunication example simulates the satellite communicating with the ground station. The software model for the system includes the software on spacecraft and software on ground. The software on the spacecraft is responsible for receiving command, collecting data, processing data, downlink data. The software on the ground station is responsible for generating new a plan, uplinking command and

receiving data from the spacecraft.

Different failure scenarios have been planned by the Planner. The failure may involve a software or hardware malfunction. If the ground station detects that the data transmitted from spacecraft is erroneous, a command would be issued that require the spacecraft to switch to safe mode. By switching to safe mode, it is possible that the spacecraft could recover. If the spacecraft fails to recover, thus no data are transmitted to ground station, and this is considered to be a system failure. If the spacecraft successfully recover, some data would still be transmitted to ground station, even though maybe not as much as previously required, and this is considered a degraded system.

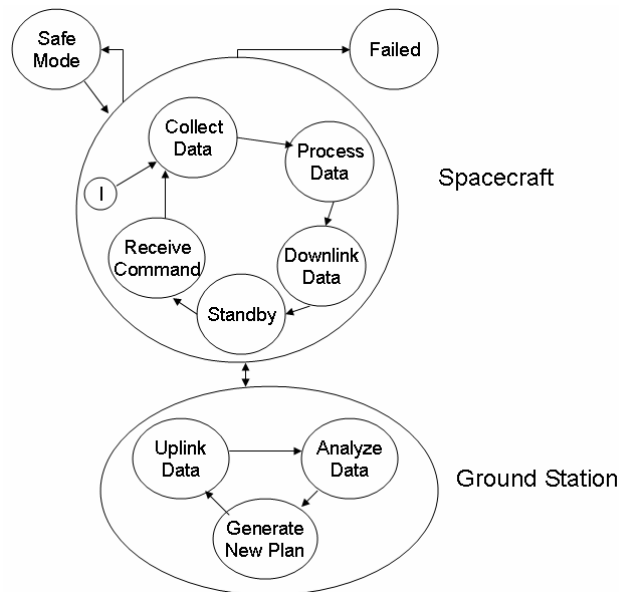


Figure 8.2.1 State Diagram

8.3 Result Analysis

8.3.1 End State Probability Estimation

There are three different end states: Failed, Degraded and Success. We run the simulation to generate 500 event sequences, and the estimation converges after about 400 event sequences. In this application, the system of interest is still under design. The simulation model is very abstract. This application shows the capability of SIMPRA to perform risk analysis at the design stage of the system. The risk analysts may gain some insights of the failure scenarios. It is helpful for them to design safety/reliability requirements and make improvements to the system safety and reliability before it is too late. It is predictable that as the design refines, and detailed information becomes available, the simulation model could be refined.

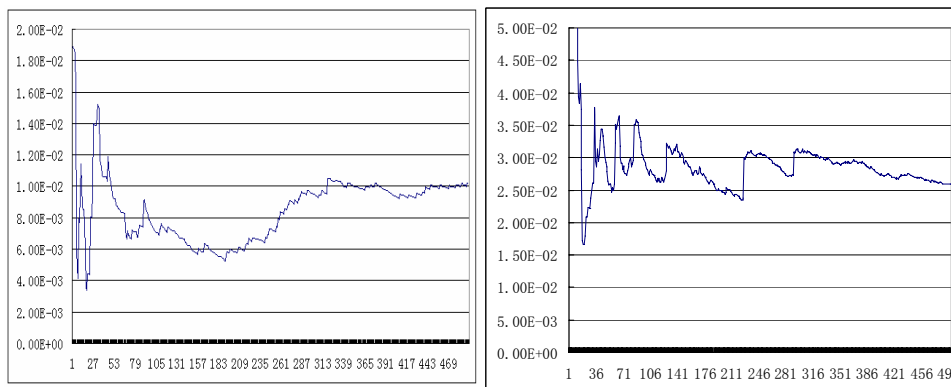


Figure 8.3.1 Estimation of Probability of Degraded (left) and Failed (right) System

8.3.2 Allocation of samples over the planed scenarios

In the plan (see table. 8-2), there are 14 different scenarios. The figure shows the Number of sequences in each scenario.

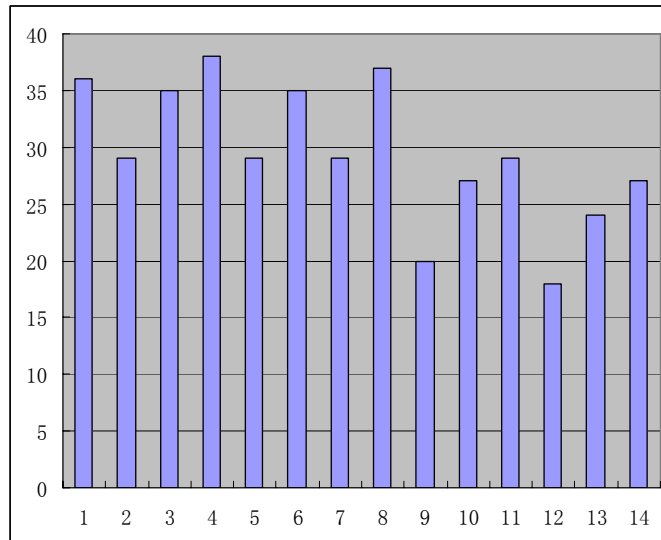


Figure 8.3.2 An example of the allocation of event sequences among plan.

It is clear that the simulation efforts were evenly distributed within the 14 scenarios. There are 413 event sequences out of 500 in the plan. There are 87 event sequences end outside plan, which include the system success scenario.

9. Application II - Space Shuttle Ascent Phase

This section describes the application of SIMPRA DPRA platform to a simplified model of the Space Shuttle mission in its ascent phase. In the application, we show the basic modeling procedure and some of the capability of SIMPRA. A simulation model of Space Shuttle was implemented, with simplified flight dynamics as well as hardware, software and crew functions and failure events.

9.1 Summary of the Shuttle Ascent Phase:

In the Space Shuttle missions, the thrust required to reach orbit is provided by the three Space Shuttle main Engines (SSMEs) and the two Solid Rocket Boosters (SRBs). About 2 minutes after ignition, the two SRBs are separated from the External Tank. The SSMEs continue working for about 8 minutes. They shut down just before the orbiter is inserted into orbit. The external tank is then separated from the orbiter. (NASA, 2000)

During launch and ascent, the guidance, navigation and control (GN&C) software controls the three SSMEs and SRBs and other hardware to maintain the designed trajectory.

If there is a failure that affects vehicle performance, abort may become necessary. The objective of Space Shuttle launch abort is to safely recover the flight crew, the orbiter and its payload.

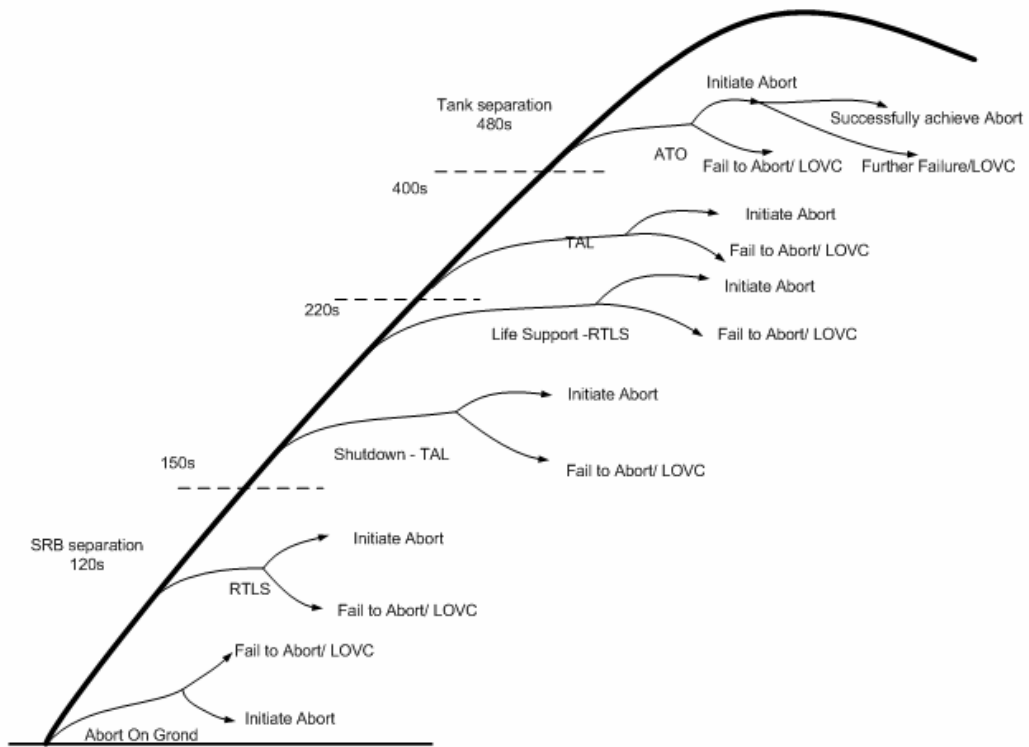


Figure 9.1.1 Figure Space Shuttle Intact Abort.

There are four intact abort modes

- Abort to Orbit: The ATO mode is designed to allow the vehicle to achieve a

temporary orbit that is lower than the nominal orbit.

- Abort Once Around: The AOA is designed to allow the vehicle to fly once around the Earth and make a normal entry and landing.
- Transatlantic Landing: The TAL mode is designed to permit an intact landing on the other side of the Atlantic Ocean.
- Return to Launch Site: The RTLS mode involves flying downrange to dissipate propellant and then turning around under power to return directly to a landing at or near the launch site.

The types of events that lead to initiation of a mission abort can be classified as failures of critical shuttle systems, such as life support system, that make the successful completion of nominal mission impossible, and the shutdown or degradation of engines, leading to a loss of thrust.

An abort mode is selected depending on the cause and timing of the failure which causes the abort. The safest one is preferable. In cases where performance loss is the only factor, the preferred modes would be ATO, AOA, TAL and RTLS, in that order. The mode chosen is the highest one as long as it can be completed with the remaining vehicle performance. In the case of some support system failures, the preferred mode might be the one that will end the mission most quickly. In these cases, TAL or RTLS might be preferable to AOA or ATO.

A simplified model of the shuttle ascent procedure was used in our simulation. In our model, a 6-degree-of-freedom dynamic model is used to calculate the simplified flight dynamics, such as altitude, velocity. More details can be found later in this chapter.

The model incorporates several failure modes:

- SSME benign shutdown. The SSME may shutdown prematurely, but no major damage to other components of the Shuttle. The thrust provided by remaining engines may be inadequate for the shuttle to reach the desired orbit. This may make the abort procedure necessary.
- SSME catastrophic failure. The engine fails and cause damage to other component. This will lead to LOC/ LOV. The abort procedure is not an option in this failure mode.
- Software failure, which will cause loss of thrust. This may also require the crew to abort.
- Other failures, such as life support system failure. Life support system failure may require early termination of flight.

9.2 Building The Simulation Model

The simulation model was implemented using the library block provided by SIMPRA. The highest layer of the model is shown in figure 9.2.1.

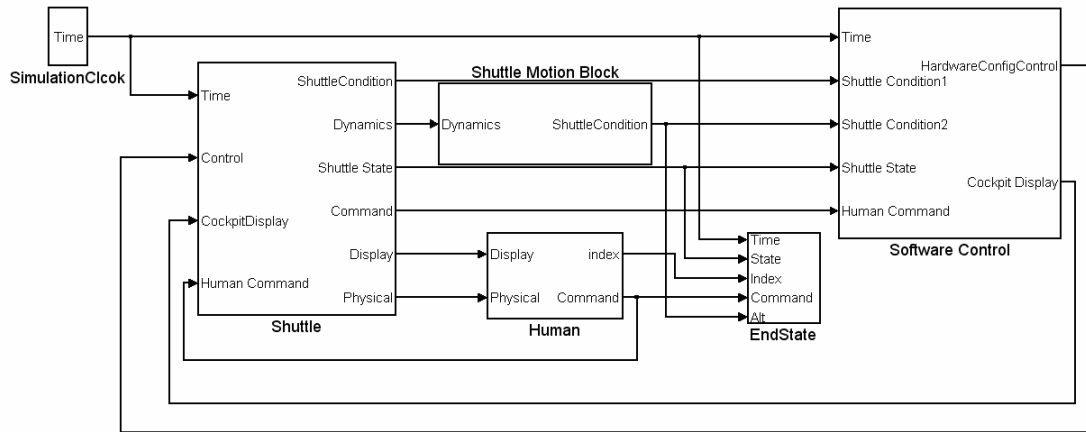


Figure 9.2.1 Space Shuttle Example Simulation Model

Hardware Module will simulate the hardware performance. Thrust and other dynamics data are input to the Motion Block, where the system trajectory is calculated. Human and Software Modules will control the system behavior, such as ignition, abort, SSME separation, etc.

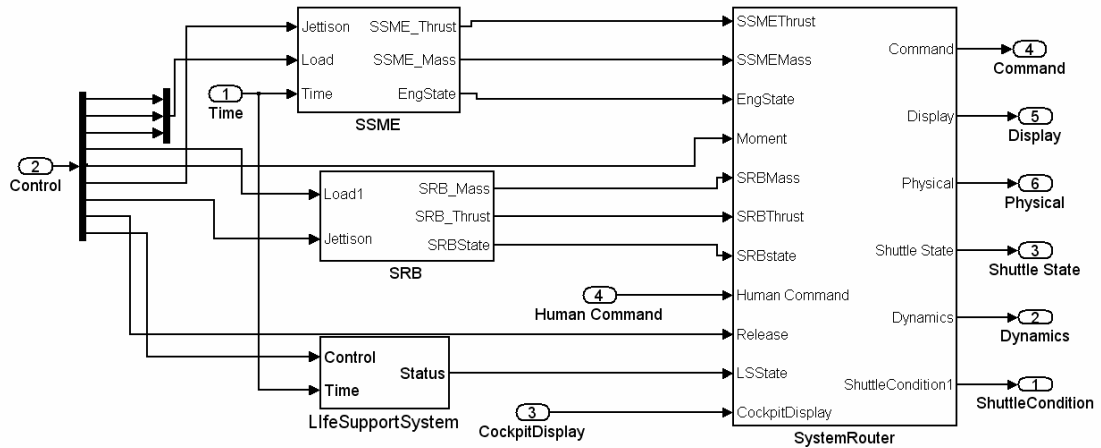


Figure 9.2.2 Hardware Simulation Module.

Hardware Module covers main engines, solid rocket boosters, life support system, etc. The inputs are control commands, Shuttle state and time. The module calculates the thrust, fuel consumption, Shuttle mass and other hardware behavior. Three SSMEs and two SRBs provide the thrust. Life Support System is responsible for maintaining the Shuttle cabin environment. The hardware is controlled by the control software and crew.

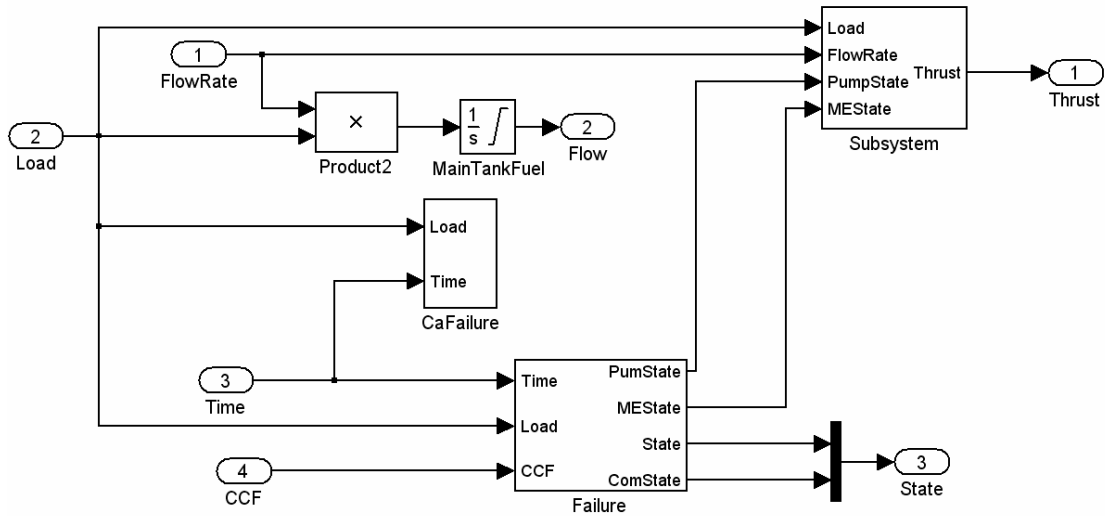


Figure 9.2.3 SSME module

The SSME and SRB modules calculate the thrust they provides controlled by software. The load of the engine may vary from 60% to 110%. The SRB and ET will be separated when the fuel is consumed. SRB and SSME modules also calculate the mass, which is needed to calculate flight dynamics.

The hardware dynamics is fed into the Shuttle Motion Block. Other input data of the Motion Block include environment data, such as gravity. A 6-degree-freedom model is used to calculate the shuttle dynamics.

The nominal trajectory is showed in Figure 9.2.4. This graph is also part of the simulation model as one of the physical variables.

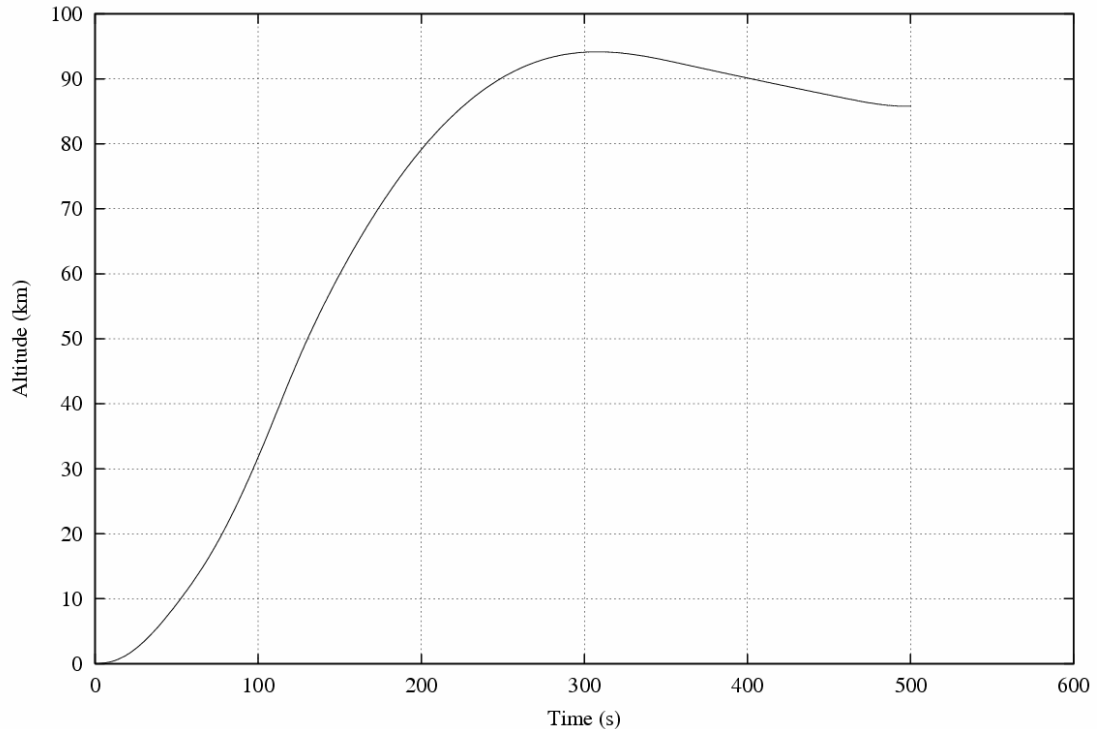


Figure 9.2.4 Height vs. Time for Space Shuttle

Failures of the components are modeled using the library block of the SIMPRA. The occurrence and timing of failures are controlled by the Scheduler.

The failure behavior module is embedded in the component models, which will generate branching points. The **Weibull Failure Rate** block calculates the failure rate $\lambda(t)$, as a function of time and system state. This is a built-in library of SIMPRA, and used can click and drag the block to the model, and specify the parameters. The “Time Distributed Event” is another built-in SIMPRA library block. It generates the branching points of time-distributed failures according to a set of rules.

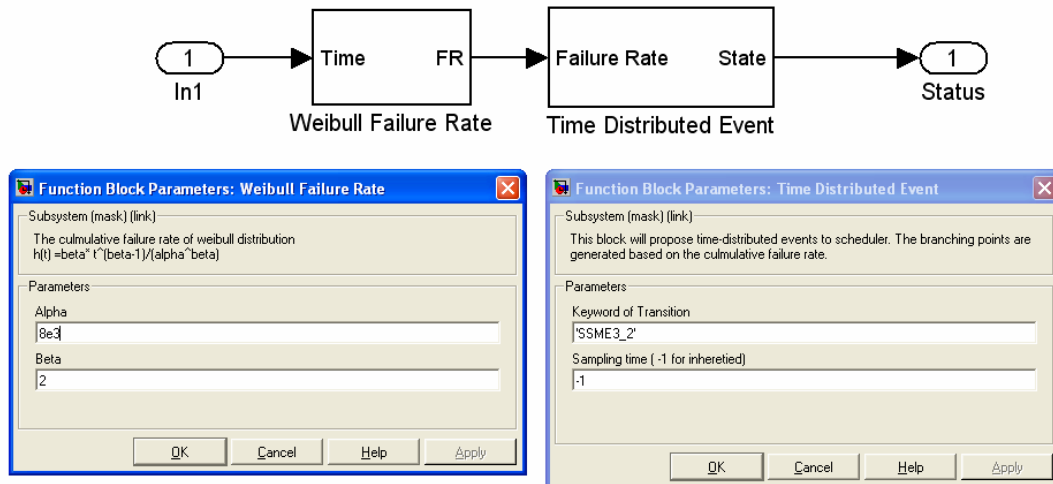


Figure 9.2.5 Runtime Failure Subsystem (inside each component module)

In the simulation model, different failure modes of SSME are considered: the main engine may either experience a benign shutdown or a catastrophic failure. By “catastrophic failure” we mean that if the main engines fail this way, the Shuttle and the crew will be lost. The benign shutdown may be triggered by different component failures or degradation.

Common cause failures are also considered. One example of CCFs is that a sub-component shared by different components. Figure 9.2.6 is the fault tree of this example. We can see the component SSME_CCF is shared by all three SSMEs. This type CCF sometimes is referred to as “shared equipment dependency” (Mosleh, Rasmuson, & Marshall, 1998).

Another example is caused by the generic environment shared by different

components. For example one of the components of SSME, the failure rate depends on the temperature, $f(t) = g(temp, t)$. The identical components can be found in all three SSMEs, and the temperature is almost the same. This type CCF sometimes is referred to as “extrinsic environment dependency” (Mosleh et al., 1998).

Such CCF is not easy to model in the traditional fault tree analysis. In the SIMPRA modeling environment all these are easy to be taken account of.

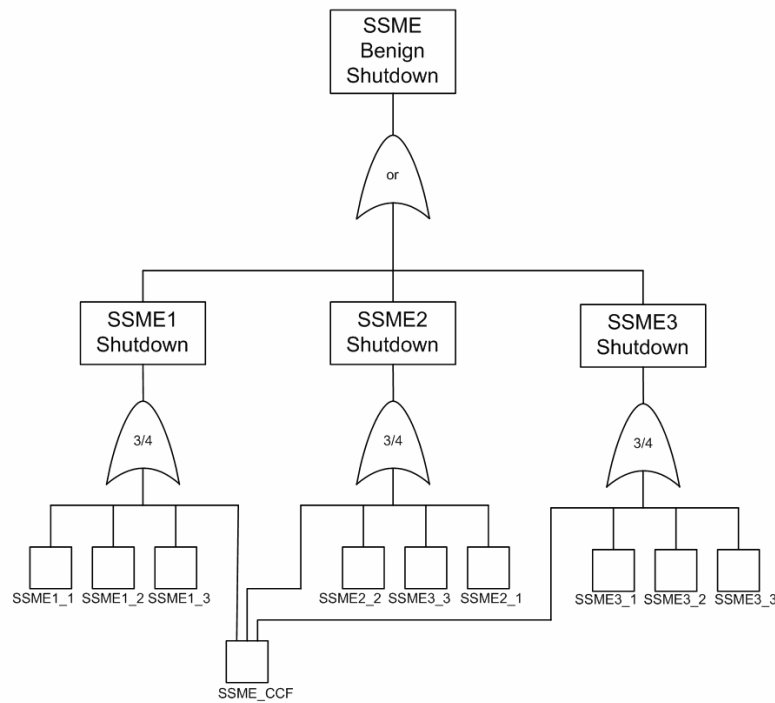


Figure 9.2.6 Fault Tree of SSME Shut Down

9.2.1 Software Model

Software model reads the flight dynamics and hardware state, and controls the Shuttle ascent procedure, such as ignition, thrust of SRBs/ SSMEs, and ET/SRB separation. The software is modeled by finite state machine, and implemented by Stateflow®. The software may fail randomly. The stochastic model represents the uncertain behavior of the software. The stochastic failure behavior is controlled by the simulation Scheduler during simulation based on predefined rules

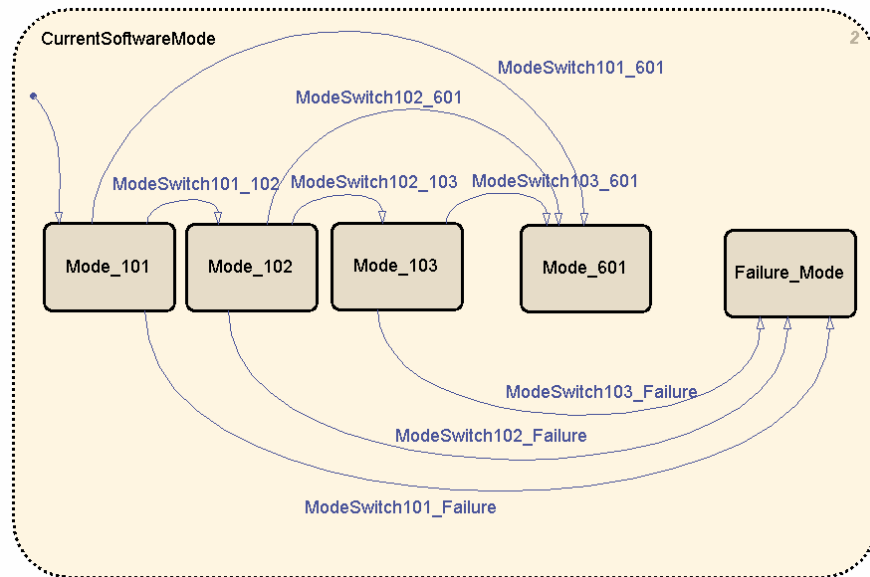


Figure 9.2.7 Stateflow® model of Shuttle GNC Software

The nominal operation of the software has several states: Mode 101, Mode 102, Mode 103 and Mode 601, which represent different phases of the mission. During the mission, the software would switch from one mode to another when the mission

advanced to next phase. At different phase of the mission the GNC software performs different functions. For example, during first-stage ascent, the software operates in major mode 102. The GNC software controls the SRBs and SSMEs. At the end of the first stage, the SRBs are separated from the space shuttle orbiter, and software automatically shifts to major mode 103. In the major mode 103 (second stage), software calculates the required main engine steering commands to achieve preflight-defined MECO conditions. During the mission, the software would switch from one mode to another when the mission advanced to next phase. Different software modes perform different functions. The software may fail to switch to new mode. Also the software may fail during the mission. Failure here means that the output of the GNC software is wrong, which is critical to the shuttle safety.

There are indications (alarms) to the crew when such failure happens. In such situation, the crew would have to abort the mission, and initiate the abort procedure. These aspects involved interaction between the software Module and Crew Behavior Module of the simulation model.

9.2.2 Crew Behavior Model

If the hardware malfunctions, such as engine premature shutdown, the crew will check the Shuttle state and decide whether to abort. There are indicators and alarms to alert the crew of component failures. If there are hardware failures, the crew may feel

the unusual condition of the shuttle, even without indicators or alarms. If the crew decides to abort the mission, they should choose the correct abort mode according the rules shown in Table 8-1, and initiate the corresponding abort procedure. It is assumed that human crew may make wrong decision. Also considered is that the probability of a successful abort depends on the time when the abort is performed and the type of abort.

An IDAC-based crew behavior model (Mosleh & Chang, 2004) is developed to simulate the human crew action. The main idea behind the cognitive behavior model is to use PIFs to estimate the probability that a specific behavior is going to take place in a given situation. The cognitive model uses a Bayesian Belief Network (BBN) to model the cause-effect relationship among the PIFs and how the PIFs influence information processing behavior and decision making during an event.

Crew knowledge and experiences are stored in the IDAC model of human knowledge base (long term memory). An operator's long term memory may contain the functional and physical characteristics of the system and its underlying processes; general guidelines on how to respond to accidents, knowledge of available options and memorized procedures; and expected response of system to perturbations, learned through training and operating experience (Mosleh & Chang, 2004). In preset simulation example the rules and procedures for mission abort are stored in the knowledge base.

The number and quality of information accumulated in the long-term memory reflects the level of experience of an individual, and have a great impact on how an individual will deal with threaten encountered.

Table 9-1 Mission Abort Procedures Rules

	LSS Failure	SSME Failure	Software Failure
MET = 0 seconds (shuttle on ground)	Abort on ground	Abort on ground	Abort on ground
MET < 150 seconds	RTLS	RTLS	RTLS
150 < MET < 220 s	RTLS	TAL	TAL
220 < MET < 400 s	TAL	TAL	TAL
MET > 400 seconds	ATO	ATO	ATO

In the simulation, all indicators of flight dynamics and alarms for component failures are fed into the crew model. Another type of system feedback modeled in the crew model is abnormal vibrations caused by engine failure. This is a signal independent of the indicator and alarms. In real system, much more information would be displayed in the cockpit, and the mental burden of the astronauts would be greater than what is simulated here. In the simulation, it is assumed that the crew may neglect the alarms, and fail to detect the need to abort in time, or when they decide to abort, they may choose an incorrect wrong abort model.

Human error is a contributor to many accidents, especially when we take account of the fact that the shuttle crew works in a very disturbing and stressful environment. The objective of the human model is to simulate the human behavior under such conditions.

When there is a need to initiate abort procedure, the crew has to decide which abort mode to initiate. The astronaut would search the knowledge base to find the right abort mode. Because the astronaut is highly stressed, and the timing is important, the astronaut may apply a limited search strategy. This means that the astronaut would search only part of the knowledge base. The time for finding the answer is shorter, but the likelihood of get a wrong answer is higher. Note that limited search strategy does not necessarily produce a wrong answer and the reaction time when applying this strategy is generally shorter.

9.3 Analysis Results:

9.3.1 Exploration Methods

Different exploration methods are applied to the Space Shuttle model. End state probabilities were estimated using different methods:

- Planned Exploration:
- Unplanned Exploration:

The plan used for the planned exploration is generated by the SIMPRA planner with the analysis of the state transition diagram of the Shuttle launch procedure. Hardware failure, software malfunctions and human behavior are all considered in the plan. The difference in timing of same type of failures may lead to different end state, thus they may belong to different scenarios in the plan.

Table 9-2 Plan for Shuttle Model

Plan 1	Catostrophic End_0
Plan 2	time(400;500)SSME ABORT End_0
Plan 3	time(220;400)SSME ABORT End_0
Plan 4	time(150;220)SSME ABORT End_1
Plan 5	time(6;150)SSME ABORT End_0
Plan 6	time(400;500)LSS ABORT End_0
Plan 7	time(220;400)LSS ABORT End_0
Plan 8	time(150;220)LSS ABORT End_0
Plan 9	time(6;150)LSS ABORT End_0
Plan 10	TIME(0;6);SW101_2 END_0
Plan 11	TIME(0;6);SSME ABORT END_0
Plan 12	TIME(0;6);LSS ABORT END_0
Plan 13	TIME(6;120);SW102 ABORT END_0
Plan 14	Negate; ALL

The plan in table 9-2 is a high level abstract plan. For example SSME represent the SSME benign shut down. The shuttle has three identical SSMEs, each of which is a very complex component. In the simulation each SSME consists of many

components and failure modes. A fault tree analysis can interpret the “SSME shut down” as a cut-set, or a combination of stochastic failure events. This piece of information is stored in a database file. When scheduler loads the plan, the scheduler would query the database file and generate the detailed plan to guide the simulation.

With the unplanned exploration strategy, the failure rates of components are accelerated to achieve better estimation. The idea is very similar to the strategy used by many practitioners of Monte Carlo simulation in PRA.

9.3.2 Event Sequences

The simulation generates a large number of event sequences. If no failures occur during the mission time, the mission is achieved, and the end state is Success. We are more interested in looking at the failure scenarios to find out what lead to accidents.

Table 9.2 records one event sequence generated by the simulation.

Table 9.2 A Recorded Event Sequence

{ SSME3_1@93 , SSME3_3@146 , RTLS@149.1 , E-3@150 , }
[ES-3 , 2.6E-4 , 150.0]
Correct abortRTLS@150

The first line is the actual event sequence. We see consecutively two sub-components of SSME3 fail, which lead to the main engine failure. This results in a

benign engine shutdown. The engine fails at 146 seconds, and the crew initiates abort procedure at 149.1 second.

The second line shows the end state, statistical weight of this sequence and the end state time. ES_3 indicates that the end state is a successful abort. We find the statistical weight is pretty low, which implies that this is a rare scenario.

The third line indicates that the crew response is correct.

Table 9-3 Another Recorded Event Sequence

{ SSME3_1@63 , SSME2_3@96 , LSS@120, ATO@124.3 , E-1@125 , }
[ES-1, 4.1E-4 , 124.3]
Wrong abortATO@125

Another example of recorded event sequences is given in table 9-2. The first line is the actual event sequence. We see consecutively two sub-components of main engine SSME2 and SSME3 fail. This does not lead to the main engine failure due to the redundancy. The Life Support fails at 120 seconds, and the crew initiates abort procedure at 124.3 second. We may notice that the crew has initiated the wrong abort procedure, which is recoded in the third line. The wrong abort procedure leads to the loss of vehicle and crew (LOVC) end state. This is recoded as ES_1 in the second line. Again, the statistical weight is pretty low, which implies that this is a rare

scenario.

9.3.3 End State Probabilities Estimation

The simulation results of the three different exploration methods are summarized in Table 8-4. The theoretical result is obtained through a Monte Carlo simulation with 10,000 histories.

Table 9-4 End State Probability Estimation

	Theoretical	Planned 500 samples	Not Planned 500 samples
LOVC	6.12E-3	6.3E-3	5.40E-3
Abort	6.55E-3	6.46E-3	7.1E-3
Success	0.987	0.987	0.988

The Planned exploration gets the estimation with a low relative error. But the estimation generated by “not planned exploration” was rather poor.

Figure 9.3.1 shows the convergence of the estimate of probability of LOVC. It shows that the estimate converge to the correct value roughly after 600 event sequences.

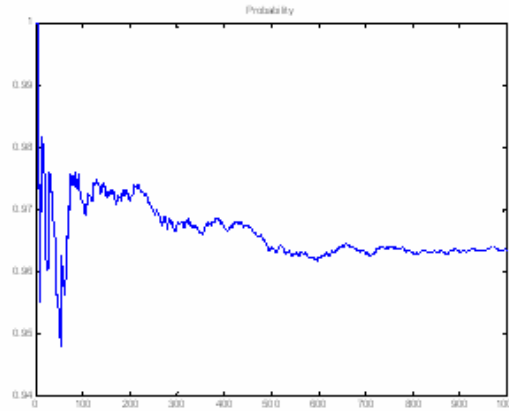


Figure 9.3.1 Estimation of the Probability of LOVC

9.3.4 Allocation of Event Sequences

The role of the plan is to specify the scenarios of interest, as a guide to the scheduler.

The scheduler will favor the scenarios listed in the plan. If there is no plan, the scheduler will distribute the simulation effort randomly.

Table 8.5 listed different scenarios which are of interest to the system analyst. These scenarios are rare due to the high reliability of the system. The scheduler guides the simulation to explore these scenarios with high efficiency. Not much effort is wasted in repeating event sequences where there is no component failures and result in system success.

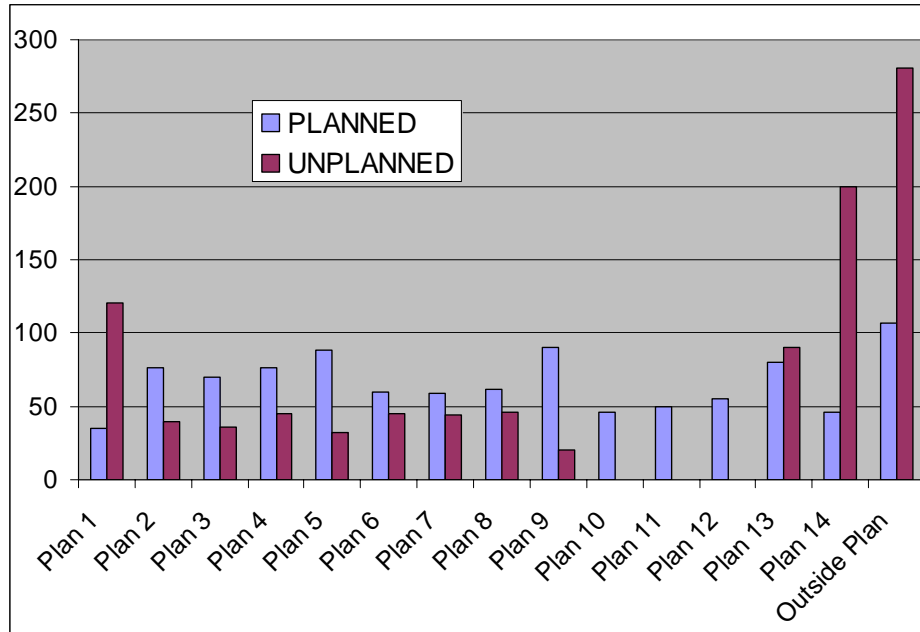


Figure 9.3.2 Allocation of Event Sequences

The scheduler constantly guides the simulation towards scenarios in the plan. The expected information gain is calculated at the end of the event sequence. In this example all the scenarios have the same “importance level”, so the most important contributor to the value of each branch is the expected information gain. The Scheduler maintains the balance among scenarios by favoring branches with higher values. The expected information gain does not only depends on the number sequences already generated, but also depends on how consistently the end states have been reached. For example, the catastrophic failure of SSME would always lead to end state LOVC. The outcome of this event is certain. The expected information gain from this scenario is low, and we do not want spend too much effort on simulate

this scenario. The scheduler does avoid the scenario (plan 1), as we can tell from figure 9.3.2. “Plan 1” is the scenario least simulated.

If there is no plan, the scheduler decides on the exploration randomly. Many of the event sequences ended in the scenarios outside the plan. Some of the scenarios were left untouched.

The result shows that the scheduler with the plan successfully guides the simulation and distributes the simulation effort fairly.

9.4 Conclusion

Even though the simulation model is only hypothetical and simplified, the interactions between the different parts in the system are fairly complex. For example, the human action depends on the PIFs and the information they receive. The PIFs would be influenced by the history, current system status and environment. The information the crew receive may be affected by the hardware and/or software failure. The system is very sensitive to the timing and order of the random events. These dynamic features cannot easily captured by traditional ET/FT methods. With this model we show the capacity of SIMPRA to model hardware, software and human. SIMPRA provides a platform to integrate different parts of the system in a single simulation model.

Due to the high reliability of the system, we have to simulate the rare events with

limited computing cost. This is the major difficulty faced by Monte Carlo simulation. The SIMPRA has an efficient algorithm to guide the simulation towards the scenarios generate more information, and avoid spending much computing power on the event sequences that do not provide much insight into the system vulnerability.

10. Summary and Future Research

10.1 Summary of Research Results

10.1.1 Overview

In this dissertation, we introduced a new Dynamic Probabilistic Risk Assessment methodology. A general purpose DPRA platform implementing the methodology is developed at University of Maryland.

In this dissertation, we interpret the DPRA problem as an exploration of the event sequences space. The DDET and Monte Carlo simulation methods represent two different exploration strategies: systematic and random exploration. Instead of focusing on obtaining a numerical result, we approach the problem of exploring the unknown space efficiently. Some of the terminology used throughout the dissertation is explained in this chapter. These terms may be loosely defined and used in DPRA literatures. We have clearly defined the terminology and it has been used in consistence with the definitions, throughout this dissertation. We have particularly stressed the difference between the terms scenarios and event sequences.

We propose a DPRA methodology, which employs a new exploration strategy. In

this framework, the knowledge of the system is explicitly used to guide the simulation to achieve higher efficiency. A planner is responsible for generating plans as a high level map to guide simulation. A scheduler is responsible for guiding the simulation toward the scenarios, which may generate more information about the system. In scheduler a tree structure is used to represent the scenarios we are interested in. During the simulation, the scheduler checks the status of the simulation and guides the simulation toward the scenarios of interest. The scheduler is also required to maintain a balance between the different scenarios. When the scheduler decides the path in which to guide the simulation, it compares the values of all possible options. The scheduler always favors the branches with higher values; however, it is a random process, and the one with the highest value is most likely to be picked, but not always. The guided simulation will generate a lot of event sequences and our knowledge of the system is updated. A new round of simulation may be required with the updated plan.

. The SIM-PRA is a generic-purpose risk assessment software package which implements the guided simulation methodology. It provides a rich library of DPRA elements, with the help of which the users can easily design their own DPRA simulation models. Some ideas of an object-oriented paradigm are used in the implementation. The users only need to specify the failure characteristics of the system. The scheduler will automatically guide the simulation based on the

information.

We apply our methodology to solve the DPRA problems of three different models. The first one is the holdup tank problem, which has been discussed frequently in the DPRA literatures. It is a relatively simple one. We solve the problem using the assumptions made by other researchers, and compare our results with theirs. We also provide a numerical and asymptotic solution. Our results agree with both the numerical solution and the works of other researchers. After lifting some of the assumptions we have made in this case, the situation becomes more complex. The accident scenarios have been discussed in other literatures. Our simulation demonstrates that the accident scenarios are highly dependent on the system dynamics; the event sequences need to be analyzed with great care.

SIMPRA is applied to a satellite telecommunication system which is still under design. An abstract model is built with SIMPRA. After analyzing the state transition graph and the component functionality relations, a plan for the simulation is generated. A number of accident event sequences were simulated, and an estimate of the end state probabilities is achieved.

Another example is a hypothetical model of a space shuttle ascent phase. In the model, we observe complex system interactions, between human, software, and hardware. Apart from the hardware model, we built a human behavior model and a software model based on the finite state machine. All parts of the system play

important roles in the evolution of the system. The system is tightly coupled. The SIM-PRA very efficiently produces a model to depict the system and generate the risk analysis. With these two examples, we have shown the capability of our methodology and its difference from previous works.

10.1.2 Comparison with Others' Work

Inspired by the analogy to human reasoning and learning, we propose a new framework for dynamic PRA. In this new framework prior knowledge is actively used and the simulation is guided adaptively by the Scheduler. The traditional PRA methods, such as state transition diagram, and fault tree analysis, are still actively applied in the new framework. Such methods are not used to develop and solve the numerical or mathematical model directly, but to generate a plan which would be high level map to guide the simulation. The simulation is carefully guided by the scheduler to maximize the information gain from the simulation. The simulation is guided toward scenarios which has higher potential to provide insight into the system vulnerabilities.

The hybrid type of DDET and Monte Carlo is desired by many authors (Labeau et al., 2000). This research is one of the attempts. The new exploration strategy used in our approach is not purely random as in Monte Carlo simulation. Nor is it predefined as in DDET approaches. The exploration strategy takes relevant prior information

into account to design the simulation rules. The systematic exploration strategy is also supported in our framework. We can perform a DDET analysis just as the one performed by ADS. Also we can perform a “partial” DDET analysis, where only the designated events are explored systematically, while the remaining parts are explored with the new exploration strategy we have proposed.

We have shown the capability of this methodology with the applications. When dealing complex dynamical system, it detects system vulnerabilities and estimates the probabilities of end states with high efficiency and accuracy.

10.2 Future Research

The new framework proposed in this dissertation still leaves many areas open for future researches.

10.2.1 Planner

Planner is the software to gather the prior information before the simulation and generate the plan to guide the simulation. It also updates the plans using the simulated results. In the current Planner the prior information mainly comes from an abstract model of the system under study. SIMPRA is able to construct such models and generate the plans from the models.

Apart from the abstract model of the system, there are many other sources of

information, e.g., experience with the similar systems, general risk knowledge. The qualitative reasoning is another powerful tool to refine the plan. How to integrate all these information together constitutes a major challenge.

10.2.2 Multi-level Modeling

By definition, the simulation model is an abstraction of the real-life system. The abstraction level is somewhat arbitrary. We want to want model to be as simple as possible, but at the same time we do not want to miss any important details of the system. We even can say that the model which best represent the system, is the system itself. There is always trade-off between the details/precision and computational cost.

For risk analysis purpose, we have to investigate the system behavior under different circumstances. We believe that the most detailed model is not required under all circumstances. There are circumstances under which the system evolution is more stable, and a simplified model would be sufficient for the analysis purpose. If we were able to develop an algorithm to determine which level of abstraction would suffice the need, and adjust the simulation accordingly, we could predict a considerable reduction in computation time without significant loss of precision.

The key issue is to identify the area where the risk-critical evolution is sensitive to the dynamics, and a detailed precise model is required in such areas. Also we should

have the simulation model built in different levels, and we know the error range of each level. Thus we can find the appropriate level to satisfy our simulation requirement. This procedure should be done adaptively and automatically by the Scheduler. The past simulations could provide important information on whether the precession requirements are met, and thus whether to switch to a more detailed or more simplified level under the same or similar circumstances.

The idea has been applied a small system. Application to larger scale systems is still under development.

10.2.3 Human Modeling and Software Modeling

Human modeling and software modeling play a very important role in the system risk analysis. The modeling of hardware is better understood compared to the one of human and software. In highly coupled human-software-hardware systems the human and software must be integrated into the PRA work.

The modeling of human or software behavior is still an open problem for future research. The human model and software model used in SIMPRA applications represent the state-of-art techniques. With the improvements in human and software modeling, the integrated DPRA methodology introduced here would become even more powerful.

Appendices

Appendix A. Graphical Representation in DPRA

Graphs provide an intuitive representation of the system logic. To perform probabilistic risk analysis, one widely accepted way is to represent the system by Fault tree analysis, and to represent the scenarios by Event Trees. In recent years, efforts are made to grasp the dynamic characteristics and/or dynamic scenarios, which are not explicitly expressed in classical FT/ET framework.

1. DFT

Due to the wide acceptance of the FTA, it is natural to extend the Fault Tree Analysis to capture the dynamic characteristics of the systems.

Dugan et al introduced several dynamic Fault-Tree gates (Dugan1992). Static fault tree is poor to deal with sequence dependency, fault and error recovery, use of spare.

Four new gates are introduced, Functional dependency gate, Cold spare gate, Priority-AND gate, and Sequence-enforcing gate.

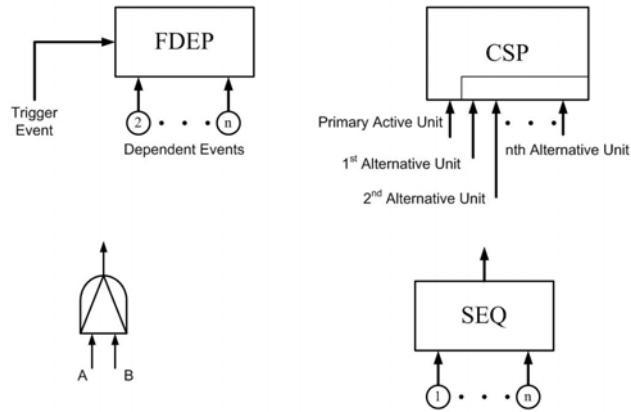


Figure A.1 Special Gates of Dynamic Fault Tree

The dynamic fault tree can be automatically converted into a Markov model. When the dimension of system grows, this conversion may generate a huge state space. Efforts have been devoted to automatically separate the Fault-Tree into sub-trees, which may be either static or dynamic, and the sub-trees would be solved using different algorithms. Combinational methods, such as BDD would be used if the sub-tree is static, and Markov-chain models would be used to solve the dynamic sub-tree. Software packages, such as DITree and later Galileo are developed at the University of Virginia to solve the Dynamic Fault tree model. (Dugan, Sullivan, & Coppit, 2000) The model is automatically decomposed into independent sub-trees (Dugan, 2000). To avoid the complexity of Markov model, several different approaches have been proposed to solve the Dynamic Fault Tree (Amari, Dill, & Howald, 2003).

This type of DFT is widely used to solve phased-mission problems. The phased-mission problem, which is characterized by the fact that the system structure, failure and recovery processes, or success criteria can change with each phase, has been investigated for more than two decades. Dugan proposed a unified framework to define the separate phases using fault trees. (Dugan, 1991) The system is evaluated by constructing and solving the resulting Markov model. The models for each phase are combined into one model. Later on, more efforts were devoted to solve the PMS more efficiently, several different algorithms were proposed: PMS-BDD, (Zang 1999) GPMS-CPR (Xing & Dugan, 2002)(Xing2002), Modular approach, Ou2004. Most methods are either combinational models or Markov-chain based models.

Cepin et al proposed a different kind of Dynamic Fault Tree to address the time requirements in safety systems (Cepin & Mavko, 2002). House events, which could be turned on or off at discrete time points, were introduced to the classic fault tree to deal with the time requirements. A house event matrix represents the house events being switched on and off through discrete time point.

The Dynamic Fault Trees are supported in several commercial fault tree software packages, such as Relex, Sapphire, and FaultTree+. Both types of DFT extend the functionality of the FTA to include some dynamic features, but neither of them is able to deal with the full spectrum of dynamic characteristics. They rely on the Markov assumption of the model. Complex dynamic interactions, such as the interaction

between component behavior and the process parameters are not captured by the DFT methodology. They rely on the Markov assumption of the model.

2. GO-Flow

GO-FLOW was originally a success-oriented implementation of diagraph based modeling methodology derived from the earlier GO methodology (Matsuoka & Kobayashi, 1988). The GO-FLOW chart represents the system configuration and functions as well as the failure. A number of operators representing different failure modes, logic operators and signal generators are used to construct the chart. Signals, which represent physical variables or time or any information, propagate through the system of interest. The state of the system or components at any point in time is determined algebraically through the logic gates and other operators. The NOT logic gate and procedures to address common cause failures were later introduced into the methodology. Later improvements make GO-FLOW able to treat the logic-loop and maintenance activity (Matsuoka, 2004).

The generation of GO-FLOW chart and the follow-up computation has been computerized. Matsuoka claimed it can be used to model phased mission more compactly compared with ET/FT approach. It is shown that a complex phased mission can be analyzed with a single GO-FLOW model and the availability function was obtained in one single run.

However, all the possible combinations should be laid out explicitly in GO-

FLOW charts, so it is difficult to represent some types of system configuration, such as k-out-of-n system. Hierarchical charts are not available either, thus the problem of combinational explosion is significant while dealing with large systems. Some important information routinely provided by ET/FT, such as minimal cuts sets and importance measures is not easily obtained in GO-FLOW model (Siu 1994). Also, please note that GO-FLOW methodology can only be applied to treat constant failure rate or repair rate.

3. Petri Nets

A Petri Net is a directed graph. There are two types of nodes: places and transitions, and arcs that connect them. The abstract objects (tokens, drawn as bold face dots) move in the nodes. Each place (denoted as circles) can store a non-negative number of tokens. Transitions (denoted as rectangles) model activities changing the state of the system. Transitions are allowed when the prediction for the activities is fulfilled (enabled). A transition can fire and remove one or more tokens from the input place and deposit the tokens to the output place. For more details about Petri Nets, there are several textbooks or monograph available. (Bause & Kritzinger, 2002; Schneeweiss, 1999, 2001)

Petri Nets are widely used in such fields as operational research, computer network, software modeling, and flexible manufacturing systems. There were proposed for reliability modeling in 1980s (Movaghar 1984). In (Schneeweiss 2001),

dynamic reliability/maintenance models are built with Petri Nets. (Dutuit 1997) used the SPN to model two test cases and evaluate the safety characteristics using Monte Carlo techniques. Volovoi et al. introduced aging token as means to model aging (Volovoi, 2004), and claims that aging tokens improve the flexibility and clarity of dependability (reliability and availability) modeling of aging systems. Hybrid Petri Nets (Chatelet et al., 1998; Kermisch C., 2004) model impact of continuous process variables on the firing of transitions based on the theory of probabilistic dynamics (Labeau et al., 2000).

The variations of the Petri Nets formalisms and the lack a of unified standard may lead to confusion and misunderstanding. The stochastic Petri Nets are hard to be verified.

4. Event Sequence Diagram

ESD is a graphical representation of the success or failure scenarios. It shows the path from an initiating event to the system end. ESDs can be used to document the accidents and help the engineers to understand the accidents scenarios. In nuclear industry, Pickard (Pickard, 1983) used ESDs to help construct event trees.

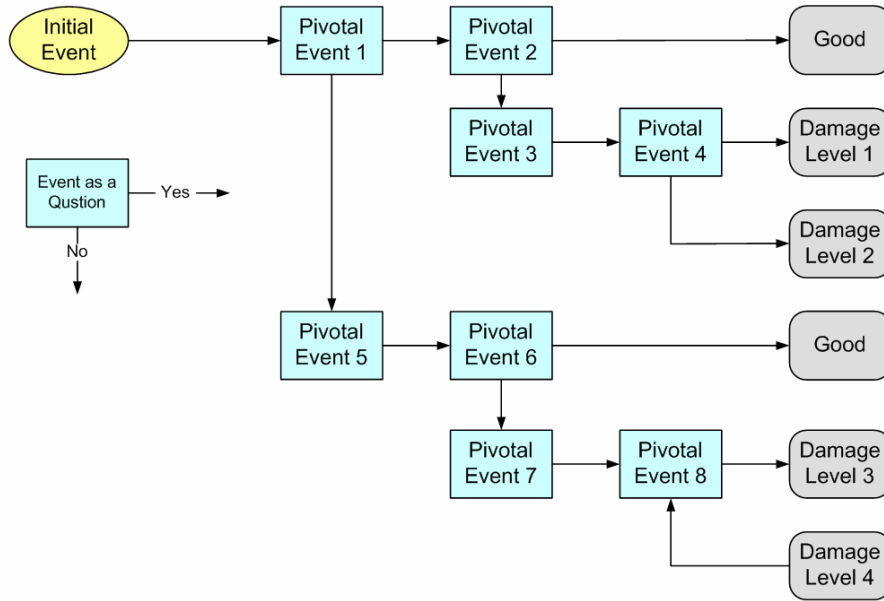


Figure A.2 Concept of Event Sequence Diagram

ESDs are extensively used in space industry to identify possible accident scenarios. ((Swaminathan & Smidts, 1999 a, 1999 b) extended the ESD framework to handle the temporal logic and process dynamics. The mathematical formulization to set up the Markov or semi-Markov state transition equations is also provided.

QRAS® (Groen, Smidts, Mosleh, & Swaminathan, 2002) is a software package, which provides an interactive tool to build ESDs, but the software is primarily a classic (static) PRA tool.

ESD approach relies on the accurate description of sequences, which cannot be performed automatically. The quality depends on the analysts. (Swaminathan & Smidts, 1999) attempts to identify the missing scenarios by MC methods.

5. Dynamic Flow-graph Methodology

Garrett et al. introduced DFM to model software-driven embedded systems (Garrett, Guarro, & Apostolakis, 1995). Later DFM has been used in nuclear, space and other industries to analyze control systems.

The system models are developed in the light of the cause-effect relationships between physical variables and timing characteristics. The DFM model is analyzed by tracing backward the sequences of events through the model, i.e. deductively from effect to cause to determine how the systems reach certain state. The result is timed fault-trees, which take the form of the logic combinations of static trees relating system parameters at different time. DFM based hazard analysis can be used to identify system hazards, including the previous unknown failure modes, and thus guide the hazard mitigation efforts. The use of multi-value logic is advantageous compared to the binary nature of Fault Trees.

Labeau pointed out that the discretization of physical variables may produce large multi-dimensional matrices and even discretization errors (Labeau et al., 2000). The DFM is not capable of representing the stochastic characteristics.

Appendix B: Application of Dynamic Fault Tree Simulation

A small system is constructed in SIMPRA. It consists two components and a Priority-

AND gate as shown in Figure B.1.

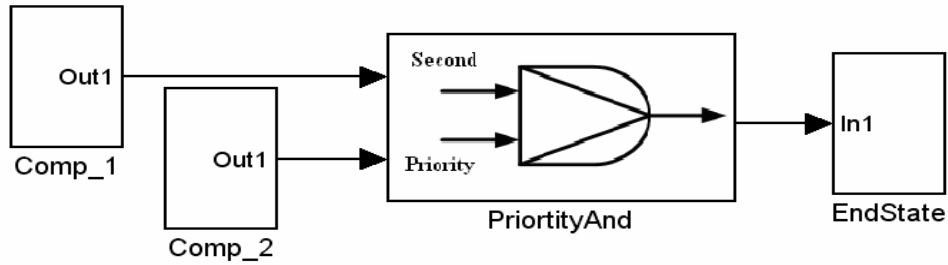


Figure B.1 Application of Dynamic Fault Tree

The two components follow a Weibull failure rate where $\alpha_1=400, \beta_1=2$ and $\alpha_2=500, \beta_2=2$. The analytical solution for this system is easy to get. Given the mission time is 500s, we can calculate the system reliability analytically.

$$R(500) = \int_0^{500} f_1(t) \cdot [F_2(500) - F_2(t)] dt = 0.773$$

We run the simulation for 500 event sequences. The estimated $R(500)' = 0.781$. The relative error is 1%. The convergence of the estimator is shown figure B.2. We can see that the simulation can predict the reliability of the system with satisfactory precision.

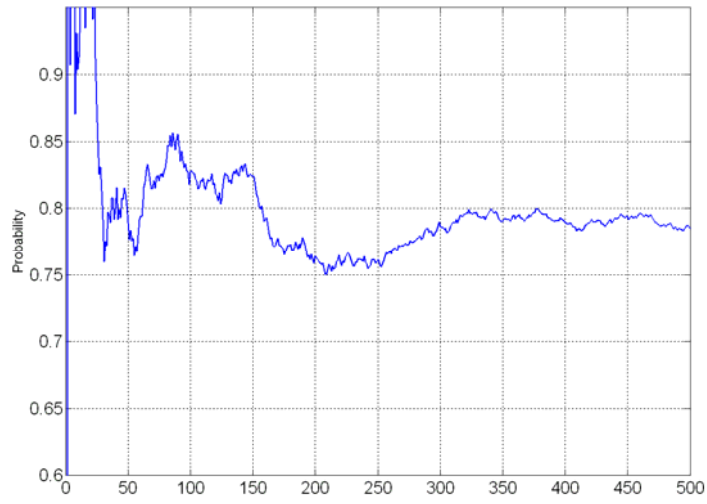


Figure B.2 Reliability Estimate

Bibliography

- Acosta, C., & Siu, N. (1993). Dynamic event trees in accident sequence analysis: application to steam generator tube rupture. *Reliability Engineering and System Safety*, 41, 135-154.
- Aldemir, T. (1987). Computer-Assisted Markov Failure Modeling Of Process-Control Systems. *IEEE Transactions On Reliability*, 36(1), 133-149.
- Aldemir, T., & Zio, E. (1998). *New Domain of Application: Discussion Group II*. Paper presented at the Fifth International Workshop on Dynamic Reliability: Future Directions.
- Amari, S., Dill, G., & Howald, E. (2003). A new approach to solve dynamic fault trees. In *Annual Reliability And Maintainability Symposium, 2003 Proceedings* (pp. 374-379).
- Amendola, A. (1988). Accident Sequence Dynamic Simulation Versus Event Trees. *Reliability Engineering & System Safety*, 22(1-4), 3-25.
- Amendola, A., & Reina, G. (1981). Event Sequences And Consequence Spectrum - A Methodology For Probabilistic Transient Analysis. *Nuclear Science And Engineering*, 77(3), 297-315.
- Bause, F., & Kritzinger, P. S. (2002). *Stochastic Petri Nets - An Introduction to the Theory* (2nd edition ed.): Vieweg Verlag.
- Berleant, D., & Kuipers, B. (1997). Qualitative and quantitative simulation: bridging the gap. *Artificial Intelligence*, 95(2), 215-255.
- Bucklew, J. A. (2004). *Introduction to Rare Event Simulation*: Springer.
- Cacciabue, P. C., Carpignano, A., & Vivalda, C. (1992). Expanding The Scope Of Dylam Methodology To Study The Dynamic Reliability Of Complex-Systems - The Case Of Chemical And Volume Control In Nuclear-Power-Plants. *Reliability Engineering & System Safety*, 36(2), 127-136.
- Cacciabue, P. C., & Cojazzi, G. (1994). A Human-Factors Methodology For Safety Assessment Based On The Dylam Approach. *Reliability Engineering & System Safety*, 45(1-2), 127-138.
- Campioni, L., Scardovelli, R., & Vestrucci, P. (2005). Biased Monte Carlo optimization: the basic approach. *Reliability Engineering and System Safety*, 87, 387-394.
- Campioni, L., & Vestrucci, P. (2004). Monte Carlo importance sampling optimization for system reliability applications. *Annals Of Nuclear Energy*, 31(9), 1005-

- Carson, J. S. (2004). *Introduction to Modeling and Simulation*. Paper presented at the Winter Simulation Conference.
- Cepin, M., & Mavko, B. (2002). A dynamic fault tree. *Reliability Engineering & System Safety*, 75(1), 83-91.
- Chaloner, K., & Verinelli, I. (1995). Bayesian Experimental Design: A Review. *Statistical Science*, 10(3), 273-304.
- Chang, Y.-H. (1999). *Cognitive Modeling and Dynamic Probabilistic Simulation of Operating Crew Response to Complex System Accident (ADS-IDACrew)*. University Of Maryland.
- Chatelet, E., Chabot, J. L., & Dutuit, Y. (1998). *Event Representation in Dynamic Reliability Analysis Using Stochastic Petri Nets*. Paper presented at the Fifth International Workshop on Dynamic Reliability: Future Directions.
- Chatelet, E., Zio, E., & Pasquet, S. (1998). *The Use of Neural Networks in Reliability Analysis of Dynamic Systems: An Overview*. Paper presented at the Fifth International Workshop on Dynamic Reliability: Future Directions.
- Cojazzi, G. (1996). The DYLAM approach for the dynamic reliability analysis of systems. *Reliability Engineering & System Safety*, 52(3), 279-296.
- Cukier, M. (1991). *Determination of the exit time from the safety domain of the state space during an accidental transient*. Unpublished Thesis, Université Libre de Bruxelles.
- Dang, V. (1998). *Frameworks for Dynamic Risk Assessment and their Implications for Operator Modeling*. Paper presented at the Fifth International Workshop on Dynamic Reliability: Future Directions.
- Deoss, D. L., & Siu, N. (1989). *A Simulation Model for Dynamic System Availability Analysis*: Massachusetts Institute of Technology.
- Devooght, J. (1998). *Dynamic Reliability: The Challenges Ahead*. Paper presented at the Fifth International Workshop on Dynamic Reliability: Future Directions, Greenbelt, Maryland, U.S.A.
- Devooght, J., & Smidts, C. (1992). Probabilistic Reactor Dynamics.1. The Theory Of Continuous Event Trees. *Nuclear Science And Engineering*, 111(3), 229-240.
- Devooght, J., & Smidts, C. (1992). Probabilistic Reactor Dynamics.3. A Framework For Time-Dependent Interaction Between Operator And Reactor During A Transient Involving Human Error. *Nuclear Science And Engineering*, 112(2), 101-113.
- Devooght, J., & Smidts, C. (1996). Probabilistic dynamics as a tool for dynamic PSA. *Reliability Engineering & System Safety*, 52(3), 185-196.

- Dubi, A. (1998). Analytic approach & Monte Carlo methods for realistic systems analysis. *Mathematics And Computers In Simulation*, 47(2-5), 243-269.
- Dubi, A., & Gerstl, S. A. W. (1980). Application Of Biasing Techniques To The Contribution Monte-Carlo Method. *Nuclear Science And Engineering*, 76(2), 198-217.
- Dugan, J. B. (1991). Automated-Analysis Of Phased-Mission Reliability. *Ieee Transactions On Reliability*, 40(1), 45-&.
- Dugan, J. B. (2000). Galileo: A tool for dynamic fault tree analysis. In *Computer Performance Evaluation, Proceedings* (Vol. 1786, pp. 328-331).
- Dugan, J. B., Bavuso, S. J., & Boyd, M. A. (1992). Dynamic Fault-Tree Models For Fault-Tolerant Computer-Systems. *Ieee Transactions On Reliability*, 41(3), 363-377.
- Dugan, J. B., Sullivan, K. J., & Coppit, D. (2000). Developing a low-cost high-quality software tool for dynamic fault-tree analysis. *Ieee Transactions On Reliability*, 49(1), 49-59.
- Dutuit, Y., Chatelet, E., Signoret, J. P., & Thomas, P. (1997). Dependability modelling and evaluation by using stochastic Petri nets: Application to two test cases. *Reliability Engineering & System Safety*, 55(2), 117-124.
- Fowler, M., & Scott, K. (1999). *UML Distilled: A Brief Guide to the Standard Object* (Second Edition ed.): Addison Wesley.
- Fragola, J. R. (1995). *Probabilistic Risk Assessment of the Space Shuttle* (No. SAIC doc. no. SAICNY95-02-05). New York.
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of reusable object orientated software* (1st edition ed.): Addison-Wesley Professional.
- Garrett, C. J., Guarro, S. B., & Apostolakis, G. E. (1995). The Dynamic Flowgraph Methodology For Assessing The Dependability Of Embedded Software Systems. *Ieee Transactions On Systems Man And Cybernetics*, 25(5), 824-840.
- Groen, F. J., Smidts, C. S., Mosleh, A., & Swaminathan, S. (2002, Jan, 28-31). *QRAS - the Quantitative Risk Assessment System*. Paper presented at the Reliability and Maintainability Symposium, 2002. Proceedings. Annual.
- Harel, D. (1987). Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8, 231-274.
- Houtermans, M., Apostolakis, G., Brombacher, A., & Karydas, D. (2000). Programmable electronic system design & verification utilizing DFM. In *Computer Safety, Reliability And Security, Proceedings* (Vol. 1943, pp. 275-285).

- Houtermans, M., Apostolakis, G., Brombacher, A., & Karydas, D. (2002). The dynamic flowgraph methodology as a safety analysis tool: programmable electronic system design and verification. *Safety Science*, 40(9), 813-833.
- Hsueh, K. S., & Mosleh, A. (1996). The development and application of the accident dynamic simulator for dynamic probabilistic risk assessment of nuclear power plants. *Reliability Engineering & System Safety*, 52(3), 297-314.
- Izquierdo, J., & Labeau, P. E. (2004). *The Stimulus-Driven Theory of Probabilistic Dynamics as a Framework for Probabilistic Safety Assessment*. Paper presented at the Proceedings of the PSAM 7 – ESREL 04 conference.
- Izquierdo, J. M., Melendez, E., & Devooght, J. (1996). Relationship between probabilistic dynamics and event trees. *Reliability Engineering & System Safety*, 52(3), 197-209.
- Kaplan, S., & Garrick, B. J. (1981). On the Quantitative Definition of Risk. *Risk Analysis*, 1, 11-27.
- Kaufman, L. M., Bhide, S., & Johnson, B. W. (2000). Modeling of common-mode failures in digital embedded systems. In *Annual Reliability And Maintainability Symposium - 2000 Proceedings* (pp. 350-357).
- Kermisch C., L. P. E., Lardeux E., Chabot J.L. (2004, June 13-18). *Implementation of hybrid Petri nets - Lessons learned from their application to a SMR unit*. Paper presented at the Proceeding of the 7th International Conference on Probabilistic Safety assessment and Management, PSAM-7, Berlin, Germany.
- Kuipers., B. J. (1986). Qualitative simulation. *Artificial Intelligence*, 29, 289-338.
- Labeau, P.-E., & Zio, E. (2001). *Biasing schemes in component-based and system-based Monte Carlo algorithms in system engineering*. Paper presented at the Esrel 2001, Torino, Italy.
- Labeau, P. E. (1995). A Method Of Benchmarking For 2-State Problems Of Probabilistic Dynamics. *Nuclear Science And Engineering*, 119(3), 212-217.
- Labeau, P. E. (1996). A Monte Carlo estimation of the marginal distributions in a problem of probabilistic dynamics. *Reliability Engineering & System Safety*, 52(1), 65-75.
- Labeau, P. E. (1996). Probabilistic dynamics: Estimation of generalized unreliability through efficient Monte Carlo simulation. *Annals Of Nuclear Energy*, 23(17), 1355-1369.
- Labeau, P. E. (1998). A survey on Monte Carlo estimation of small failure risks in dynamic reliability. *Aeu-International Journal Of Electronics And Communications*, 52(3), 205-211.
- Labeau, P. E., Smidts, C., & Swaminathan, S. (2000). Dynamic reliability: towards an

- integrated platform for probabilistic risk assessment. *Reliability Engineering & System Safety*, 68(3), 219-254.
- Labeau, P. E., & Zio, E. (1998). The cell-to-boundary method in the frame of memorization-based Monte Carlo algorithms. A new computational improvement in dynamic reliability. *Mathematics And Computers In Simulation*, 47(2-5), 347-360.
- Labeau, P. E., & Zio, E. (2002). Procedures of Monte Carlo transport simulation for applications in system engineering. *Reliability Engineering & System Safety*, 77(3), 217-228.
- Li, B. (2004). *Integrating Software into PRA (Probabilistic Risk Assessment)*. University of Maryland, College Park.
- Li, B., Li, M., Ghose, S., & Smidts, C. (2003, Nov 17-20). *Integrating Software Into PRA*. Paper presented at the Proceedings of ISSRE, Denver, Colorado.
- Li, B., Li, M., & Smidts, C. (2004, June 14 - 18). *Integrating Software into PRA: A Test-Based Approach*. Paper presented at the Proceeding of the 7th International Conference on Probabilistic Safety assessment and Management, PSAM-7, Berlin, Germany.
- Li, B., Li, M., & Smidts, C. (2005). Integrating Software into PRA: A Test-Based Approach. *Journal of Risk Analysis*, in press.
- Lindley, D. V. (1956). On the Measure of Information Provided by an Experiment. *Annals of Statistics*, 27(4), 986-1005.
- Lindley, D. V. (1972). *Bayesian Statistics, A Review*: Soc for Industrial & Applied Math.
- Loredo, T. J. (2003). *Bayesian Adaptive Exploration*. Paper presented at the Bayesian Inference And Maximum Entropy Methods In Science And Engineering:23rd International Workshop.
- Malhotra, M., & Trivedi, K. S. (1995). Dependability Modeling Using Petri-Nets. *Ieee Transactions On Reliability*, 44(3), 428-440.
- Marseguerra, M., Masini, R., Zio, E., & Cojazzi, G. (2003). Variance decomposition-based sensitivity analysis via neural networks. *Reliability Engineering & System Safety*, 79(2), 229-238.
- Marseguerra, M., & Zio, E. (1993). Nonlinear Monte Carlo Reliability Analysis With Biasing Towards Top Event. *Reliability Engineering & System Safety*, 40(1), 31-42.
- Marseguerra, M., & Zio, E. (1993). Nonlinear Monte Carlo reliability analysis with biasing towards top event. *Reliability Engineering and System Safety*, 40, 31-42.

- Marseguerra, M., & Zio, E. (1995). The Cell-to-boundary method in Monte Carlo based dynamic PSA. *Reliability Engineering and System Safety*, 48, 199-204.
- Marseguerra, M., & Zio, E. (1996). Monte Carlo approach to PSA for dynamic process systems. *Reliability Engineering & System Safety*, 52(3), 227-241.
- Marseguerra, M., & Zio, E. (2000). Monte Carlo biasing in reliability calculations with deterministic repair times. *Annals Of Nuclear Energy*, 27(7), 639-648.
- Marseguerra, M., Zio, E., & Cadini, F. (2002). Biased Monte Carlo unavailability analysis for systems with time-dependent failure rates. 76(1), 11-17.
- Marseguerra, M., Zio, E., Devooght, J., & Labeau, P. E. (1998). A concept paper on dynamic reliability via Monte Carlo simulation. *Mathematics And Computers In Simulation*, 47(2-5), 371-382.
- Mason, R. L., Gunst, R. F., & Hess, J. L. (2003). *Statistical Design and Analysis of Experiments: With Applications to Engineering and Science* (Second edition ed.).
- Matsuoka, T. (2004). *Improvement of the GO-FLOW Methodology*. Paper presented at the Proceedings of the PSAM 7 ESREL '04 conference.
- Matsuoka, T., & Kobayashi, M. (1988). GO-FLOW: A New Reliability Analysis Methodology. *Nuclear Science and Engineering*, 98, 64-78.
- Mosleh, A., & Bier, V. (1992). On Decomposition and Aggregation Error in Estimation: Some Basic Principles and Examples. *Risk Analysis*, 12, 203-214.
- Mosleh, A., & Chang, Y. H. (2004). Model-based human reliability analysis: prospects and requirements. *Reliability Engineering & System Safety*, 83(2), 241-253.
- Mosleh, A., Groen, F., Hu, Y., Pirest, T., Zhu, D., & Nejad, H. (2005). *Simulation-Based Probabilistic Risk Analysis; Report of Research Activities*.
- Mosleh, A., Rasmuson, D. M., & Marshall, F. M. (1998). Guidedlines on Modeling Common-Cause Failures in Probabilistic Risk Assessment. In NUREG/CR-5485 (Ed.).
- NASA. (2000). Shuttle Reference Manual
<http://science.ksc.nasa.gov/shuttle/technology/sts-newsref/stsref-toc.html>.
- Nivolianitou, Z., Amendola, A., & Reina, G. (1986). Reliability-Analysis Of Chemical Processes By The Dylam Approach. *Reliability Engineering & System Safety*, 14(3), 163-182.
- NRC. (1975). Reactor Safty Study: an Assessment of Accident Risks in US Commercial Nuclear Power Plants. In N. R. Commission (Ed.).
- Pickard. (1983). *Seabrook station probabilistic safety assessment PLG-0300, Prepared for Public Service Company of New Hampshire and Yankee Atomic*

- Electric Company*. Newport Beach, CA: Lowe and Garrick Inc.
- Rubinstein, R. Y., & Melamed, B. (1998). *Modern Simulation and Modeling*: Wiley Interscience.
- Say, A. C. C., & Akin, H. L. (2003). Sound and complete qualitative simulation is impossible. *Artificial Intelligence*, 149(2), 251 - 266.
- Schneeweiss, W. G. (1999). *Petri Nets for Reliability Modeling*: Hagen/Germany: LiLoLe Publishing.
- Schneeweiss, W. G. (2001). Tutorial: Petri Nets as a Graphical Description Medium for Many Reliability Scenarios. *IEEE TRANSACTIONS ON RELIABILITY*, VOL. 50, NO. 2, 50(2), 159-164.
- Shannon, C. (1948). Mathematical theory of communication. *The Bell Labs Technical Journal*, 27, 379--457.
- Shults, B., & Kuipers, B. (1997). Proving properties of continuous systems: qualitative simulation and temporal logic. *Artificial Intelligence Journal*, 92, 91-129.
- Siu, N. (1994). Risk Assessment For Dynamic-Systems - An Overview. *Reliability Engineering & System Safety*, 43(1), 43-73.
- Smidts, C. (1994). Probabilistic Dynamics - A Comparison Between Continuous Event Trees And A Discrete-Event Tree Model. *Reliability Engineering & System Safety*, 44(2), 189-206.
- Smidts, C., & Devooght, J. (1992). Probabilistic reactor dynamics. II. A Monte Carlo study of a fast reactor transient. *Nuclear Science and Engineering*, 111, 241-256.
- Stamatelatos, M., Apostolakis, G., Dezfuli, H., Everline, C., Guarro, S., Moieni, P., et al. (2002). Probabilistic Risk Assessment Procedures Guide for NASA Managers and Practitioners. In NASA (Ed.).
- Swaminathan, S., & Smidts, C. (1999). The event sequence diagram framework for dynamic probabilistic risk assessment. *Reliability Engineering & System Safety*, 63(1), 73-90.
- Swaminathan, S., & Smidts, C. (1999). Identification of missing scenarios in ESDs using probabilistic dynamics. *Reliability Engineering & System Safety*, 66(3), 275-279.
- Swaminathan, S., & Smidts, C. (1999). The mathematical formulation for the event sequence diagram framework. *Reliability Engineering & System Safety*, 65(2), 103-118.
- Tombuyses, B. (1999). Reduction of the Markovian system by the influence graph method: error bound and reliability computation. *Reliability Engineering &*

System Safety, 63(1), 1-11.

- Tombuyses, B., DeLuca, P. R., & Smidts, C. (1998). Backward Monte Carlo for probabilistic dynamics. *Mathematics And Computers In Simulation*, 47(2-5), 493-505.
- Vernez, D., Buchs, D., & Pierrehumbert, G. (2003). Perspectives in the use of coloured Petri nets for risk analysis and accident modelling. *Safety Science*, 41(5), 445-463.
- Volovoi, V. (2004). Modeling of system reliability Petri nets with aging tokens. *Reliability Engineering & System Safety*, 84(2), 149-161.
- Xing, L. D., & Dugan, J. B. (2002). Analysis of generalized phased-mission system reliability, performance, and sensitivity. *Ieee Transactions On Reliability*, 51(2), 199-211.