ABSTRACT

Title:	AUTOMATIC GENERATION OF GENERALIZED EVENT SEQUENCE DIAGRAMS FOR GUIDING SIMULATION BASED DYNAMIC PROBABILISTIC RISK ASSESSMENT OF COMPLEX SYSTEMS
	Seyed Hamed Nejad-Hosseinian, Ph.D., 2007
Directed By:	Professor Ali Mosleh, Department of Mechanical Engineering

Dynamic probabilistic risk assessment (DPRA) is a systematic and comprehensive methodology that has been used and refined over the past two decades to evaluate the risks associated with complex systems such as nuclear power plants, space missions, chemical plants, and military systems. A critical step in DPRA is generating risk scenarios which are used to enumerate and assess the probability of different outcomes. The classical approach to generating risk scenarios is not, however, sufficient to deal with the complexity of the above-mentioned systems.

The primary contribution of this dissertation is in offering a new method for capturing different types of engineering knowledge and using them to automatically generate risk scenarios, presented in the form of generalized event sequence diagrams, for dynamic systems. This new method, as well as several important applications, is described in detail. The most important application is within a new framework for DPRA in which the risk simulation environment is guided to explore more interesting scenarios such as low-probability/high-consequence scenarios. Another application considered is the use of the method to enhance the process of risk-based design.

AUTOMATIC GENERATION OF GENERALIZED EVENT SEQUENCE DIAGRAMS FOR GUIDING SIMULATION BASED DYNAMIC PROBABILISTIC RISK ASSESSMENT OF COMPLEX SYSTEMS

By

Seyed Hamed Nejad-Hosseinian

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
2007

Dedication

For Sobhan, who plans on defending many dissertations,

and Matine, who will fill the world with her sweetness,

…with all my love.

Acknowledgements

A number of people and institutions supported the work presented in this dissertation. First and foremost my gratitude goes to the chair of my committee, Dr. Ali Mosleh without whom I may never have embarked upon or completed the Ph.D. degree. He challenged me throughout the process with his expectations of excellence that pushed me to keep improving my work up to the last draft. Dr. Carol Smidts has encouraged and supported my ideas and efforts since hiring me as a research assistant and overseeing the completion of my Master's thesis. Dr. Mohammad Modarres and Dr. Eyad Abed have been truly inspirational teachers. I felt like every time I walked out of one of their classes, I had something new to think about for the next few days. I was thrilled and grateful that they agreed to be on my committee. Finally, Dr. Jeffrey Herrmann was kind enough to wade through a very early draft of my dissertation and provide feedback whose incorporation, I believe, greatly improved the clarity and readability of the final product.

I would like to thank Dr. Frank Groen, Dr. Dongfeng Zhu, Dr. Yunwei Hu, and Mahendra Ramani, my friends and partners on the SimPRA project.

I would also like to thank the ELORET Corporation and NASA Ames Research Center for supporting this effort financially and giving me the opportunity to continue my work on it post-graduation.

Thanks must also go to Ehsaneh Sadr whose support, encouragement, and, above all, patience throughout the process can rightly only be described as saintly.

This work has been supported by a grant from NASA Ames Research Center, Engineering for Complex Systems.

Table of Contents

List of Figures

List of Tables

1 Introduction

Dynamic probabilistic risk assessment (DPRA) is a systematic and comprehensive methodology that has been used and refined over the past two decades to evaluate the risks associated with complex systems such as nuclear power plants, space missions, chemical plants, and military systems. A critical step in DPRA is generating risk scenarios which are used to enumerate and assess the probability of different outcomes. The classical approach to generating risk scenarios is not, however, sufficient to deal with the complexity of the above-mentioned systems.

The primary contribution of this dissertation is in offering a new method for capturing different types of engineering knowledge which are used to automatically generate dynamic risk scenarios that are presented as generalized event sequence diagrams. These scenarios can be used in at least three ways. The risk analyst can use them to study the behavior of the system. A simulation environment may utilize the risk scenarios to guide the simulation toward more interesting scenarios. And, the system designers can incorporate their analysis into the process of risk-based design.

The organization of this study is such that each chapter provides greater detail about a concept introduced in the previous chapter. In chapter 2, dynamic probabilistic risk assessment is discussed in detail and the advantages of simulation methods of DPRA are emphasized. In chapter 3, SimPRA, the improved DPRA simulation technique enhanced by automated risk scenario generation is introduced and the three main components of it including the *simulator*, *scheduler*, and *planner* are discussed. In chapter 4, SimPRA's advantages over other simulation techniques are explained. In chapter 5, the contribution of the planner as a new method for incorporating engineering knowledge into the automatic generation of dynamic risk scenarios is explained. In chapter 6, the contribution of this thesis is assessed by presenting an evaluation of the SimPRA planner at three different levels—within the context of the SimPRA approach as a package, as an element within the SimPRA approach, and as a stand-alone method for risk scenario generation. Chapter 7 highlights the strength of this method by demonstrating its use to the field of risk-based design. Chapter 8 summarizes the contributions of this work and recommends a path forward.

2 PRA/DPRA and the role of scenarios in risk assessment

2.1 Probabilistic Risk Assessment (PRA)

Probabilistic risk assessment (PRA), also known as Quantitative Risk Assessment and Probabilistic Safety Assessment (PSA), has been defined as "a systematic and comprehensive methodology to evaluate risks associated with every life-cycle aspect of…complex engineered" systems (such as nuclear power plants, airliners, space missions, chemical plants and military equipment) "from concept definition, through design, construction and operation, and up to removal from service" (Stamatelatos, 2000). PRA was first used by the aerospace industry around the time of the Apollo space program (Stamatelatos, 2000). The first full-scale application of PRA methods was WASH-1400, The Reactor Safety Study (NRC 1975). WASH-1400 considered the course of events which might arise during a serious accident at a light water reactor and estimated the radiological consequences of these events as well as the probability of their occurrence using a fault tree/event tree approach. Another major adaptation of PRA methodology occurred after an extensive review of NASA safety policies following the Challenger accident in 1986. At that time, NASA instituted a number of programs for quantitative risk analysis such as risk assessment of the Space Shuttle program (Fragola 1995). The Office of Safety and Mission Assurance at NASA headquarters has published several handbooks to enhance the PRA expertise at NASA (Stamatelatos et al. 2002).

In the process of assessing the risk associated with a system, PRA answers three major questions, which can be represented as the set of triplets $\{< s_i, f_i, p_i >\}$: "scenarios – frequencies – consequences" (Kaplan and Garrick 1981):

1. What can go wrong? *[Scenarios]*
2. How likely is each scenario? *[Probability]*
3. How severe is the consequence of each scenario? *[Consequence]*

Scenarios with the same consequence are usually grouped together to provide the risk factor for each consequence.

The classical PRA approach involves the construction of separate models describing the system behavior and structure by the risk analysts. Models are typically built in the form of fault trees and event trees (FT/ET) 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 granularity 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 and Bier 1992). The knowledge required to solve the FT/ET is basic probability calculation. Commercial software packages are readily available to construct FT/ETs and conduct the necessary computations.

The accuracy of PRA results will diminish when the necessary probabilistic failure information is scarce or entirely unavailable for use as input to the quantification process. In such cases, PRA results are more suitable for relative risk comparisons or risk rankings rather than absolute risk evaluations. Such comparisons and rankings are extremely useful for the improvement of systems designs and concepts in a resource scarce environment (Stamatelatos, 2000).

Naturally, when probabilistic failure information is more readily available, PRA efforts have been found to have a number of additional benefits for the implementing organization. A number of these benefits are listed in Stamatelatos 2000 and include:

- New insights into design flaws and cost-effective ways to eliminate them in design prior to construction and operation;
- An in-depth understanding of the normal and abnormal operation of complex systems and facilities;
- A better understanding of approaches to reducing operation and maintenance costs while meeting or exceeding safety requirements;
- Acquisition of technical bases to request and receive exemptions from unnecessarily conservative regulatory requirements.

The importance of PRA techniques have come to be recognized by a number of regulatory agencies that have integrated them directly into their regulatory frameworks. In situations where risk management is critical to mission success, PRA methods are increasingly playing an important role in informing the decisions of management and/or regulatory agencies. In 1988 the NRC, for example, instituted the requirement that each nuclear power plant in the United States perform an Independent Plant Evaluation (IPE) to identify and quantify plant vulnerabilities to hardware failures and human faults in design and operation (NRC 1995).

2.2 Classical PRA methodological limitations

The PRA methodology has been successfully applied in a number of different projects. It has been recognized, however, that it is difficult to characterize some complex dynamical systems by solely applying such techniques as event tree/fault tree analyses. Classical techniques face challenges in every aspect of the problem when applied to complex and dynamic systems that are often comprised of hybrid systems of hardware, software, and human components among which many interactions occur. Probability estimation and consequence determination processes are affected by the influence of the dynamics of the system on the failure rates and failure mechanisms of the components of the system. Most importantly, scenario generation confronts many impediments, one being the problem of combinatorial explosion.

As mentioned earlier, scenarios play an important role in probabilistic risk assessment. One of the major questions that PRA tries to answer is what can go wrong or, more precisely, what are the risk scenarios. The quantification of risk is also dependant upon the risk scenarios defined by the analyst. For the purposes of risk assessment, it is necessary that the set of scenarios be complete, that scenarios be disjointed and that the number of scenarios be finite (Kaplan and Garrick 1981). One of the biggest challenges that PRA methods face when applied to a dynamic complex system is capturing a complete set of disjointed scenarios.

PRA methods such as event trees and fault trees are implementations of logic. A major limitation of the Boolean logic-based models is their inability to specify the timing of events or even the order in which events occur. It is also difficult to model the dependency of the 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 a task is far from trivial.

In the fault tree analysis, an often-used but 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 elements, the sojourn time or response time must be explicitly taken into account. This is illustrated in Cojazzi 1996.

Apart from the issue of time-dependent analyses, leaving the system dynamics out of the picture is in some cases a significant oversimplification (Devooght 1998). The event tree analysis can display the correct failure logic of dynamic systems, but due to its ignoring the role of process variables explicitly, it cannot determine the distribution of time to an undesirable state. An event tree is basically a pictorial representation of Boolean logic, so the only way it can take the process variable into account is by discretizing the process variables' ranges. When we need detailed 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 a subjective judgment of the interaction between variables. As a result, the assessment of the probability of achieving 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.

2.3 Dynamic PRA

Acknowledging such difficulties, a set of new methodologies were developed under the name "Dynamic Reliability" or "Dynamic PRA (DPRA)". Because of the range of disciplines working on this problem, it is sometimes hard to recognize the different terms being used to refer to similar proposed methodologies, nevertheless, it is accepted that the methods that try to handle systems with the following characteristics for PRA purposes are DPRA methods (Aldemir and 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 component failure behavior and human operator actions depend 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.

Dynamic PRA methods are supplemental tools to PRA techniques that are capable of handling interactions among components and process variables explicitly. In principle, they constitute a more realistic modeling of systems for the purposes of reliability, risk and safety analysis. With the DPRA tools in hand, system risk analysts can more clearly understand the limits of classical approaches, and determine when the dynamic methods are needed and applicable. Since the problem with classical PRA lies in the simplifications and compromises which need to be made by the analyst, the burden of proof of correctness of the methodology lies on the analyst (Labeau, Smidts et al. 2000). However, DPRA techniques provide a framework for explicitly capturing the influence of time and process dynamics on scenarios and therefore, the burden of proof of correctness is significantly shifted from the analyst to the methodology.

DPRA methods have evolved significantly over the past few decades. In 1981, Amendola and Reina 1981 explored the possibility of global treatment of dynamic PRA. A few years later, the DYLAM and ADS implementations were applied to treat DPRA problems in nuclear power plants and other areas (Nivolianitou, Amendola et al. 1986; see also: Cojazzi 1996; Hsueh and Mosleh 1996; Chang 1999). More recently, a more general mathematical framework was introduced for probabilistic dynamics (Devooght and Smidts 1992). Probabilistic dynamics theory interprets the DPRA problems as problems equivalent to transport problems to be solved by, for example, Monte Carlo simulation. For a broader overview of dynamic PRA methodologies, several review papers of DPRA are available (Siu 1994; Labeau, Smidts et al. 2000).

2.4 Review of different DPRA approaches with an emphasis on modeling/scenario generation

DPRA methods can be divided into two basic categories depending upon whether or not they originated from attempts to enhance classical PRA modeling techniques. Those DPRA methods that originated from classical PRA methods include Dynamic Fault Tree (DFT) and Extended Event Sequence Diagram (ESD) methods. Those that did not originate from classical PRA methods include GO-

FLOW, Dynamic Flowgraph, and Simulation techniques. Below, each of these methods is described in greater detail.

2.4.1 Dynamic Fault Tree

Static fault trees are poorly suited for dealing with real-world PRA problems that may include sequence dependency, fault and error recovery, and the presence of redundancies, among other things. Due to the wide acceptance and use of static Fault Tree Analyses, however, one natural approach was to try to modify and enhance them in ways that would allow them to capture the dynamic characteristics of systems while preserving the familiar FTA format and capabilities. The Dynamic Fault Tree (DFT) approach proposed by Dugan et al. (Dugan 1991) does this by adding several new gates such as a functional dependency gate, a cold spare gate, a priority-AND gate, and a sequence-enforcing gate to the regular fault tree so that it can handle the redundancy and sequencing of the events.

Markov Chain (MC) models and Binary Decision Diagrams (BDD) are used to solve a DFT. The process of converting a DFT into a Markov model has been automated. When the dimension of a system grows, however, this conversion may generate a huge state space. As a result, 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 (Dugan 2000; Amari, Dill et al. 2003). Combinational methods, such as BDD would be used if the sub-tree is static, and Markov-chain models would be used to solve dynamic sub-trees.

Software packages, such as DITree and later Galileo have been developed at the University of Virginia to solve the Dynamic Fault tree model (Dugan 2000;

Dugan, Sullivan et al. 2000). The Dynamic Fault Trees are supported in several commercial fault tree software packages, such as Relex, Sapphire, and FaultTree+.

Cepin et al. proposed a different kind of Dynamic Fault Tree to address the time requirements in safety systems (Cepin and 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.

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 both rely on the Markov assumption of the model. Therefore, complex dynamic interactions, such as the interaction between component behavior and the process parameters are not easily captured.

2.4.2 Extended ESD

An Event Sequence Diagram (ESD) is a graphical representation of the success or failure scenarios. An ESD shows the path from an initiating event to the end-states. ESD can be used to document accidents and help engineers understand the accidents scenarios. In the nuclear industry, Pickard, Lowe and Garrick, Inc. (Pickard 1983) used ESD to help construct event trees. ESD is used extensively in the space industry to identify possible accident scenarios.

Swaminathan and Smidts extended the static ESD framework to capture many dynamic phenomena like time conditions, physical conditions, competing events, synchronizations, concurrent independent processes, mutually exclusive processes, and cyclic scenarios (Swaminathan and Smidts 1999). The mathematical formulization to set up the Markov or semi-Markov state transition equations is also provided (Swaminathan and Smidts 1999). QRAS® (Groen, Smidts et al. 2002) is a software package, which provides an interactive tool to build ESDs, but the software is primarily a classic (static) PRA tool.

The ESD approach relies on the accurate description of sequences, which cannot be performed automatically. The quality of the ESD depends, therefore, on the analyst's attempts to identify the missing scenarios by Monte Carlo methods (Swaminathan and Smidts 1999).

2.4.3 GO-FLOW

GO-FLOW (Matsuoka and Kobayashi 1988) is a success-oriented system analysis technique. The GO-FLOW chart consists of signal lines and operators that represent the system configuration and functions as well as possible failures. Nearly a dozen different types of operators are used to construct the charts that represent the functioning or failure of physical equipment, logical gates and a signal generator. 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. Additional improvements allow GO-FLOW to treat the logic-loop and maintenance activity (Matsuoka 2004). The generation of a GO-FLOW chart as well as the calculation procedures has been computerized.

GO-FLOW charts are very useful insofar as they correspond to the physical layout of a system, are easy to construct and easy to validate. Matsuoka (Matsuoka 2004) claimed GO-FLOW can model a phased mission more compactly than the FT/ET approach.

Despite these advantages, GO-FLOW has some limitations. GO-FLOW can only handle binary systems in phased mission situations and any deviations from nominal behavior are treated as failures. Furthermore, although GO-FLOW does not track the evolution of each individual risk scenario, all the possible combinations of events must be laid out explicitly in GO-FLOW charts. This makes the representation of some types of system configuration, such as k-out-of-n systems very difficult (Labeau, Smidts et al. 2000). Hierarchical charts are not available either, thus the problem of combinational explosion is significant when dealing with large systems. Another problem is that some important information that is routinely provided by FT/ET, such as minimal cuts sets and importance measures is not easily obtained in the GO-FLOW model (Siu 1994). Finally, the GO-FLOW methodology can only be applied to treat constant failure and/or repair rates.

2.4.4 Dynamic Flow-graph

Garrett et al. introduced the Dynamic Flow-graph Model (DFM) to model software-driven embedded systems (Garrett, Guarro et al. 1995). More recently, DFM has been used in the nuclear, space and other industries to analyze control systems (Houtermans and Apostolakis 2002).

DFM has two fundamental goals: 1) to identify how certain postulated events may occur in a system; and 2) to identify an appropriate testing strategy based on an analysis of the system's functional behavior. To achieve these goals, the system models are developed in light of the cause-effect relationships between physical variables and timing characteristics. The DFM model is analyzed by a backward tracing of the sequences of events through the model, i.e. deductively moving from effect to cause to determine how the system reaches certain states. The result is timed fault-trees, which take the form of the logic combinations of static trees relating system parameters at different times. The minimal cut sets of the fault trees can be used to identify and eliminate system faults resulting from unanticipated combinations of potential failure modes arising from these combinations of system conditions.

DFM based hazard analysis can be used to identify system hazards, including the previously unknown failure modes, and thereby guide hazard mitigation efforts. The use of multi-value logic is advantageous as compared with the binary nature of fault trees.

The biggest draw back of DFM is the combinatorial explosion problem. Labeau pointed out that the discretization of physical variables may produce large multi-dimensional matrices and even discretization errors (Labeau, Smidts et al. 2000). DFM is also not capable of quantifying the probability of scenarios. Efforts to add quantification capabilities to DFM that would enable it to represent stochastic characteristics are under way (Oliva 2006).

2.4.5 Simulation methods

Simulation methods have been extensively used for the analysis of the behavior of dynamic systems and comprise a major portion of the scholarly literature on DPRA. There are two primary strategies for simulation approaches to DPRA: the Discrete Dynamic Event Tree (DDET) strategy, and the Continuous Event Tree strategy (in its more general form, CET is known as the Monte Carlo method).

Discrete Dynamic Event Tree (DDET) methods systematically explore a large number of scenarios by generating, at certain points in time (usually fixed time steps), branch points whose branches represent distinct courses of events, thus leading to distinct sequences of events. All possible branches of the system evolution which form all possible directions that the scenarios of the system might take are simulated systematically.

Continuous Event Tree (CET) simulations do not involve the discretization of the event sequence space. The event sequences are randomly generated by randomly deciding on the occurrence and timing of events. Biasing techniques are typically applied in the DPRA approaches based on CET simulation (Labeau and Zio 2001).

2.4.5.1 <u>Discrete Dynamic Event Tree simulation methods</u>

Discrete Dynamic Event Tree (DDET) simulation methods 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 (Nivolianitou, Amendola et al. 1986; Cojazzi 1996; Hsueh and Mosleh 1996). 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, DDET will investigate all possible scenarios. As such, DDET is a straightforward extension of the classical event trees with the binary logic restriction removed.

One issue with DDET simulation methods is that the systematic branching may easily lead to such a huge number of sequences that the management of the output event tree becomes awkward. Different measures have been suggested for dealing with this problem. The length of the time step, for example, may be increased although this may be at the expense of the accuracy of the analysis. Another approach is to introduce cut-off criterion in some implementations so that the branches that meet that criterion (for example, branches with a probability lower than a specific value) would be discarded. Finally, Amendola suggested that when the number of failures in a sequence exceeds a user-defined value (which is problem-dependent), further evolution along this sequence would be stopped (Amendola 1988).

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

2.4.5.2 Continuous Event Trees (Monte Carlo) simulation methods

While DDETs require the events to occur at predefined discrete times only, CET approaches allow events to happen at any time. This avoids the combinatorial explosion problem that DDETs face. Monte Carlo methods are insensitive to the complexity and dimension of the system. Any modeling assumption such as the non-fixed failure rate assumption, random delays, interaction between components and process dynamics, etc can be included. Indeed, Dubi claimed that Monte Carlo is the only practical approach to solving realistic systems (Dubi 1998).

Discrete Event Simulation is a special case of the CET approach based on the concepts of state, events, activities, and processes (Carson 2004). In discrete event simulation, the operation of a system is represented as a chronological sequence of events. Each event occurs at an instant in time and marks a change of state in the system (Robinson 2004). When an event occurs, it may trigger new events, activities, and processes. Between events, the system is considered to be deterministic. Like the Monte Carlo approach, this is an implicit state-transition methodology, where there is no need to enumerate the possible system states, and the possible transition rates. Extensions to treat continuous process variables are needed in most risk analysis problems (Dang 1998). According to Siu, discrete event simulation is better equipped to dealing with arbitrary complex systems than Markov analysis, DDET, or analog Monte Carlo approaches (Siu 1994).

The analogy between Monte Carlo simulation for PRA and the transport problem is often drawn (Devooght and Smidts 1992; Smidts and Devooght 1992; Dubi 1998). In the Monte Carlo simulation framework, a system is defined as a collection of components and a given state of the system is described by at least one real valued "system function" as the function of its state vector and possibly other relevant parameters. The system function is defined on phase space $(\mathbf{S},t)$ where the state vector of the system is defined as $\mathbf{S}=(S_1,S_2,...,S_n)$ and S_i is the state of component *i*. The phase space vector $(\mathbf{S},t)$ indicates that the system entered the state vector **S** 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: $K(\overline{S}',t'\rightarrow\overline{S},t)=T(\overline{S}',t'\rightarrow t)\times C(t;\overline{S}'\rightarrow\overline{S})$ The free flight kernel $T(\overline{S}',t'\rightarrow t)$ is defined as the probability density that a system which entered state S' at time t' will have a next event at time $t>t'$. The collision kernel, which is also called event kernel, $C(t,\overline{S}'\rightarrow\overline{S})$ is defined as the probability that upon an event at *t* in state $\overline{S}'$ the system will change its state into $\overline{S}$.

Let $\{\overline{S}'\}$ be the set of all state vector $\overline{S}'$ from which it is possible to transfer to $\overline{S}$ in a single event. The event density $\psi(\overline{S},t)=P(\overline{S}_0)\cdot\delta(t)+\sum_{\{\overline{S}'\}}\int_{\tau}\psi(\overline{S}',t')\cdot K(\overline{S}',t'\rightarrow\overline{S},t)\cdot d\tau$.

If we consider a system with n components, each of which has k_i different states, the dimension of the phase space would be $\prod_{i=1}^{n}k_i$. And 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 practically the only feasible solution.

Please note here that the state vector is defined as a 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 has been addressed by a number of scholars. (Smidts and Devooght 1992; Labeau and Zio 1998; Tombuyses, DeLuca et al. 1998).

Despite all their advantages, CET simulation methods (such as MC and DES) have some limitations when applied to DPRA problems. Due to the high reliability of most systems, the number of simulation runs required to explore the high-risk scenarios may become extremely large and impractical. Much scholarship 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 scenarios generated while ensuring that everything happens within a reasonable time frame.

Importance sampling tries to address these problems by improving sampling efficiency with the help of advanced biasing techniques that allow statistically superior results to be obtained with fewer runs of simulations. The principle behind it is that some variables are more important in terms of their impact on the outcomes of the scenarios and should, therefore, be sampled more frequently. However, more frequent sampling of a chosen variable introduces bias that must be corrected. This is done by weighting the simulation outputs.

A number of scholars have proposed and/or tested different variations on importance sampling methods. Marseguerra et al. introduced a new biasing technique that seeks to improve computational efficiency by driving the system toward a cut set configuration of more interesting but highly improbable events (Marseguerra and Zio 1993). Labeau and Zio 2002 compared the system-based indirect Monte Carlo method with the component-based direct MC method and found that component-based sampling is more efficient. Finally, Marseguerra, Zio et al. 2002 concluded that sampling from a discrete uniform distribution generates better results than sampling from an exponential failure rate, at least in the case of smaller systems. A more general discussion of variance reduction techniques and rare event simulation can be found in Rubinstein and Melamed 1998, and Bucklew 2004.

2.4.5.3 <u>SimPRA</u>

The SimPRA framework which is the foundation of the methods developed in this dissertation can be seen as a combination of CET and DDET approaches. In fact SimPRA is a CET approach that is guided by a planner that, like DDET approaches, takes advantage of the classical event sequence diagram and event tree approaches with the binary logic restriction removed.

In SimPRA, events are simulated in continuous time. When the simulation reaches a branch point, the direction that it has to take is decided based not only on a random selection of the events according to the probability of branches, but also by the importance measure of the events and the expected entropy gain from simulating each branch. The SimPRA framework is explained in more detail in the following chapter.

3 SimPRA framework overview

The SimPRA framework is comprised of three major components: a *simulator* that generates the detailed scenarios and identifies the ultimate fate of each one, a *scheduler* which controls the timing and occurrence of the events and a *planner* which is responsible for guiding the simulation through high level scenario generation. Figure 1 illustrates the main elements of the SimPRA framework as well as the interactions among them.

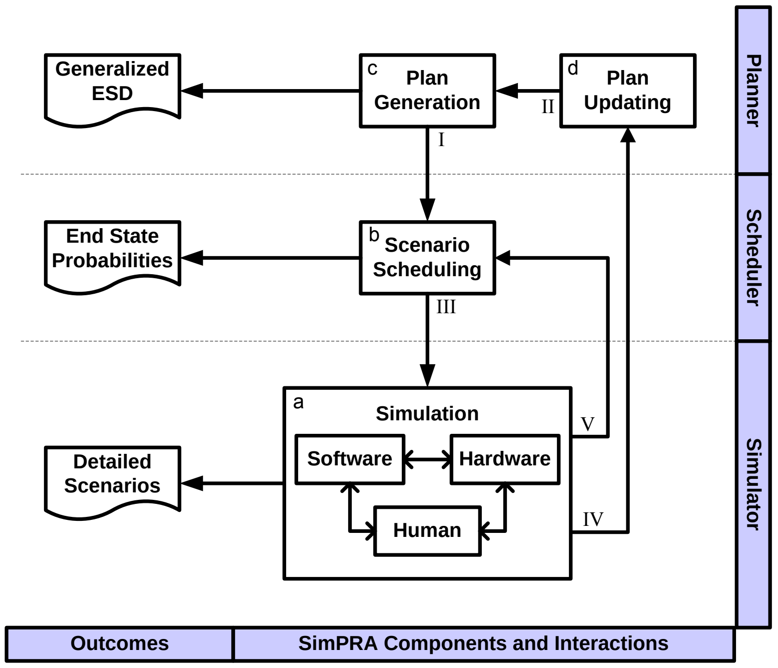


Figure 1: SimPRA framework components and the interactions among them

In the SimPRA framework, high level knowledge about the system's behavior and vulnerabilities is actively employed through a plan—or group of *high level* scenarios—to guide the simulator to generate those *detailed* scenarios that are most likely to lead to interesting situations. These scenarios and their consequences, as identified by the simulator, are recorded and grouped for later study. The planner may ‘learn’ from the simulation results, and update the high level scenarios to better guide the scheduler in future runs of the simulation. The rest of this chapter discusses SimPRA components and the interactions among them in more detail.

3.1 Simulator

The simulator is responsible for generating detailed behavior of the system as intended by the system designers. Simulation can consist of both discrete and continuous elements that simulate the behavior of software, hardware or human components of a real system. It is even possible to let the real elements play a part in the simulation by providing an interface between them and the rest of the simulation program.

The granularity or the level of detail in a simulation model depends on many factors including the risk assessment goals, availability of data, computational power and the simulation speed.

There are two types of stochastic events in the simulator: time-based events and demand-based events. Time-based events describe the probability distribution function of the time to occurrence of the event. Demand-based events, on the other hand, present the probability of occurrence of each set of outcomes of the event. For these events, the timing of the occurrence is not random; instead, the outcomes at that point of time are random.

To link the simulator with the scheduler, the simulation model needs to include some model elements that are able to notify the scheduler of the branch points and look for guidance from it. Practically, this translates into using some elements from a tool box in the simulation model that are predefined to provide the communication. Most of these elements will be the elements with stochastic behavior that are guided by the scheduler.

Specifically, any simulation model in the Simulink® environment with embedded SimPRA tool-box components can be used by SimPRA for risk assessment (Figure 2). However, to make a simple model that only simulates the reliability behavior of the software, hardware, and human elements of simulation without presenting other unnecessary aspects of these elements, several modeling techniques are developed and suggested for the SimPRA environment. Please refer to Hu, et al. 2004, Mosleh, et al. 2004, and Zhu, et al. 2006 for more details about these modeling techniques.

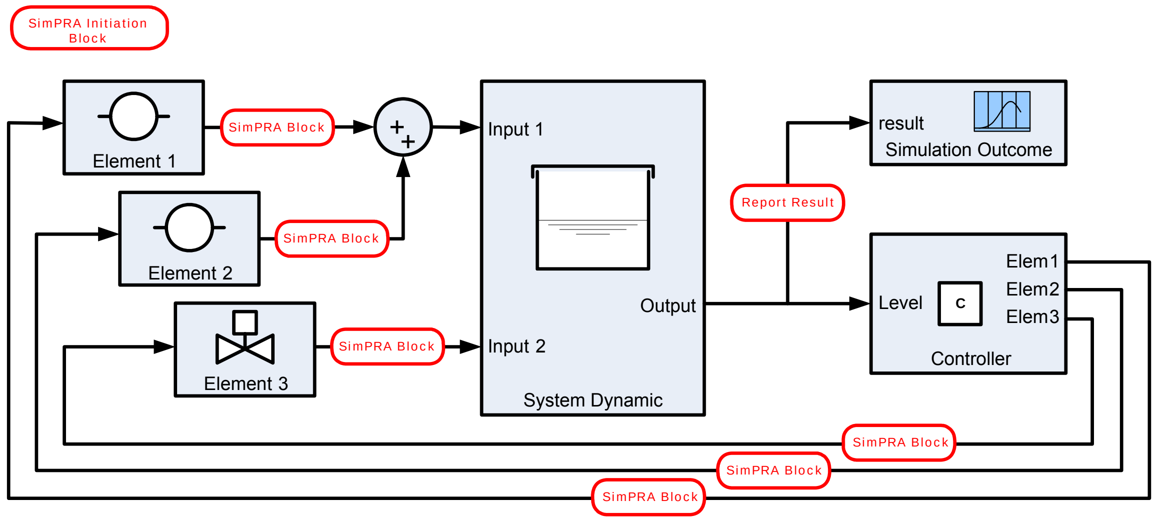


Figure 2: A typical simulation model with SimPRA blocks added for risk assessment

In the case that the multi-level objects are defined in hardware, software, or human components, a separate knowledge base needs to be constructed for each sub-component. The prior knowledge related with level selection is stored in the knowledge base, including time-factor related information (the relationship between the high-level scheduler simulation time requirement and the simulation level of detail for this component) and the relationship between pre-defined component conditions and the simulation level of detail.

Data provided to the simulator (Figure 1: link III) contains the simulation commands that are executed in the branch points. These commands include: a) Checking some conditions in the simulation model; b) Setting some simulation variables that are directly provided by the scheduler; and c) Running queries on the simulation model to decide the availability of options to set for the continuation of the simulation and perhaps doing a random selection among the possible options.

The simulator notifies the scheduler when branching points are generated. It also provides the results of the condition tests on the simulation elements when requested by the scheduler (Figure 1: link V). The simulator also keeps a record of all of the state changes in every simulation run with the time of these changes. Figure 3 shows the algorithm behind the simulator component.

```
Set the simulation parameters
For each run of the simulation {
   Initiate event sequences
   While (not reached an end state or cut off time) {
      Perform continuous/discrete-time simulation
      If (Reached a branching point) {
         Inform the scheduler of the possible branches with their
               probabilities
         Wait for the scheduler to select a branch and the level of detail
               for multi-level components by giving appropriate
               commands
         Execute the scheduler commands
      }
   }
   Inform the scheduler of reaching an end state or cut-off time
}
End the simulation
Report the results
```

Figure 3: Overview of SimPRA simulator algorithm

3.2 Scheduler

The scheduler manages the simulation process, by, among other things: saving system states, deciding the branch selection, adjusting the simulation levels of detail, and restarting the simulation. The scheduler guides the simulation toward the plan generated by the planner.

The scheduler keeps track of the simulation and guides it adaptively. The simulator proposes the transitions to the scheduler whenever the simulator reaches a branching point. The scheduler then retrieves the value of the proposed transitions and decides which branch is to be explored. The exploration command is sent to the simulator for execution and the simulation continues until the next branching point or an end-state is reached. Two kinds of knowledge—prior knowledge and knowledge obtained from previous runs of the simulation—are used to guide the scenario exploration.

The engineering knowledge is represented in the plan. In the SimPRA framework, the high level scenarios generated by the planner serve as a guide for the simulation (Figure 1: link I). The importance level of events based on prior information or engineering experience is integrated into the plan. The knowledge obtained during simulation is measured using the expected entropy gain, which evaluates how much information is expected to be gained by simulating / exploring that scenario. 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 provided by an experiment is measured by comparing the knowledge before and after the experiment, while the measure of information is given by Shannon's entropy. The details of the entropy-based exploration strategy are discussed in Hu et al., 2004. It is important to mention that although the guide is not expected to be complete or even correct, due to the randomness of the simulation, a sufficiently large number of guided simulation runs should generate all event sequences of interest.

In addition to branch selection, the scheduler also handles the adjustment of the simulation levels in the case where multi-level objects are defined in the simulator. The planner is loaded into the scheduler at the beginning of the simulation. The simulation level of detail is adjusted adaptively based on the information in the plan, the information in the knowledge base of different components, and the previous simulation results.

A System Level Knowledge Base (SLKB) is established in the scheduler to represent the compatible combinations of the level of detail for different sub-components. The type of level control nodes includes:

1. Direct level control: the direct value for the level of simulation
2. Time Factor: the required time factor
3. Undefined: the level control is undefined.

The scheduler is established to handle multi-level objects in the simulation model if a multi-level plan is loaded. The scheduler checks the plan for level control commands and, based on the type of the command, it will take one of the following approaches:

1. If Level Control Node (LCN) contains the direct control information, the scheduler reads the level information from LCN and sets the simulation level of detail for all multi-level objects accordingly.
2. If LCN contains time factor requirements, the scheduler sends the time factor requirements to the sub-components. The sub-component queries its own knowledge-base to get the level control information based on the time factor requirement and sends it back to the high-level scheduler. The scheduler sets the simulation level of detail for all multi-level objects based on the combination.
3. If LCN is undefined, the scheduler sends this information to the sub-component. The sub-component decides the simulation level based on the information in its own knowledge base and sends it back to the high-level scheduler; the scheduler sets the simulation level of detail for all multi-level objects based on the combination.

After the start of the simulation, the scheduler controls the simulation level of detail during each simulation run. The simulation continues until it reaches an LCN. The simulation level of detail for multi-level objects only changes when the simulation reaches an LCN. Figure 4 shows the algorithm behind the scheduler component.

It is necessary to mention that the term scheduler has been used in other simulation-based probabilistic risk assessment frameworks, such as DYLAM (Cojazzi 1996) and ADS (Hsueh and Mosleh 1996) as well. However, 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. In the SimPRA framework, the scheduler is not only capable of performing the depth-first search as in the DDET, but is also able to adaptively guide the simulation toward the scenarios of interest by actively choosing branches. The scheduler's role is crucial since it has to be able to cover all the event sequence space, and it also has to maintain sufficient coverage of all of the planned scenarios and guide the simulation toward areas of greatest uncertainty. For the details of the scheduler algorithm please refer to Hu et al., 2004; Zhu et al., 2006

```
Load the plan
Event tree ← Generate a tree of the plan data
Initialize multi-level objects
For each round of simulations {
  Reset the scenario values
  For each run of simulation {
    Weight of this simulation run = 1
    Wait for the simulator to send a message{
       If (end state message) {
          Add the weight of this simulation run to the end state weight
          Normalize weights to calculate the probabilities of end states
          End this simulation run
       }
       If (branching point message) {
          Find the node on the event tree
          Calculate the value of each possible branch
          Choose a branch based on the calculated values
          Calculate the weights needed to bias toward the branch
          Randomly choose a branch considering the weights
          Calculate the total weight of simulation run to this point
          If (weight is too high meaning reaching an absorption point){
            End this simulation run
          }
          Check if (a level control node is reached in the event tree){
             Get level control information
             if (the new level is different with the current level){
                Adjust the simulation level of detail
             }
          }
       }
    }
  }
  Calculate level control measure for each multi-level objects
  Update plan based on the calculation results
}
```

Figure 4: Overview of SimPRA scheduler algorithm

3.3 Planner

As mentioned earlier, the planner is responsible for guiding the simulation by providing the high-level scenarios to the scheduler. These scenarios might be incomplete or even in some cases incorrect. The planner itself, after a number of simulation runs, learns the detailed simulation results (Figure 1: link IV) and comes up with suggestions for the system analysts on ways to improve the plan model (Figure 1: link II) to generate unseen scenarios or eliminate impossible ones.

Modeling the system for plan generation is done in three steps: First the hierarchy of the system is defined in the form of a structure tree. Then, state transitions are defined for each element of the system hierarchy, and finally, the details of the transitions that link higher level elements to lower level ones are defined.

The primary contribution of this dissertation is its proposal and development of the planner portion of the SimPRA approach which, on a stand-alone basis, can be seen as an automated tool for the generation of risk scenarios presented in the form of generalized event sequence diagrams. The theory and operation of the planner are discussed in greater detail in chapter 5 after chapter 4 reviews the primary advantages of the SimPRA framework over alternative simulation-based methods for risk assessment.

4 Problem statement and contributions

The advantages of the SimPRA framework over other simulation-based techniques used for risk assessment can be grouped into the following five categories. The contributions of this thesis are comprised of the capabilities provided by the planner element of the SimPRA environment.

4.1 Handling a large number of scenarios without compromising completeness

The ability to handle a large number of scenarios without compromising completeness is one of the most challenging problems that simulation-based risk assessment methodologies have encountered. As mentioned in Chapter 2, DDET simulation methods are faced with the combinatorial explosion problem while CET simulation methods are unable to ensure the completeness of risk scenarios.

All the different combinations of events happening in a large number of discrete time steps define the universe of event sequence possibilities in DDET simulation methods. Assuming there are n independent combinations of events and t time steps, a crude estimate of the size of the event sequences will be n^t. A small system with five components that are either up or down (working or failed) observed in only 10 time steps will generate $(2^5)^{10}$ event sequences. Obviously, however, some of these event sequences are either physically impossible or they are so low in probability compared to others that simulating them is not worth the effort.

The classical way of dealing with this problem in DDET simulation methods has been to define a certain number of criteria that will halt the progress of a scenario when the simulation encounters them:

- Cut-off criteria: The probability of a scenario is continuously monitored during the simulation run and when that probability falls lower than a certain level, that scenario will be considered to be an extremely rare scenario and will no longer be pursued.
- Absorption point criteria: A set of rules are defined before running the simulation that would define the conditions under which the scenario will be deemed uninteresting. Examples are scenarios containing specific combinations of events or scenarios reaching specific pinch points.

A problem with these approaches is that a lot of precious simulation time may be wasted before a cut-off or absorption point is finally attained that flags a given scenario as extremely rare or uninteresting. Also, since DDET simulation methods employ search algorithms to find possible scenarios, searching in a large universe with a lot of uninteresting possibilities is a big waste of resources and can even overflow system resources before results are achieved.

For CET methods, the primary challenge is in defining a *complete* set of risk scenarios. CET methods generally rely on the natural probability of scenarios. Since most systems are designed to be highly reliable, the natural probability of high-consequence scenarios is likely to be very low. As such, a prohibitively large number of simulation runs may be required to ensure that those low-probability high-consequence scenarios are sufficiently explored. Recalling that risk is a function of consequences as well as probabilities, the generation of low-probability high-consequence scenarios is at least as important as the generation of high-probability low-consequence scenarios and the limitations of CET methods in this regard is significant.

The typical way of dealing with this problem has been the use of Importance Sampling. Importance Sampling artificially pushes the simulation to sample important or risk-relevant variables more frequently so that high-consequence scenarios are more likely to be explored. Simulation outputs are then weighted in order to try to correct for any bias that may have been introduced.

The SimPRA approach is a hybrid of the DDET and CET approaches that seeks to avoid the combinatorial explosion problem while simultaneously optimizing the completeness of the set of risk scenarios. Similar to the DDET approach, SimPRA begins by discretizing the world of possibilities into high-level events. System behavioral knowledge is then used to reduce the world of combinatorial scenarios to a pool of physically, logically, and temporally possible high-level scenarios called a plan. The simulation is run in a CET-type format that uses the plan for biasing (Importance Sampling) purposes. Entropy rules with user-defined importance measures of events are used to select the branch points on an event tree generated from the plan. Multi-level scheduling is also used to decide on the level of detail the events should have and to ensure that the simulation time is spent wisely on each detailed scenario generated by the simulator. Cut-off points and absorption points are also used to increase productivity.

4.2 Doing risk assessment without compromising the complexity of the problem

Most risk assessment tools use or provide a specific modeling framework that suits the theory behind their approach. The use of these modeling frameworks is not without cost:

1. The user must remodel the problem to fit the new framework which can be very time consuming. He or she also will need some risk assessment expertise.
2. The complexities in the model are often reduced to comply with the tacit and explicit assumptions behind the risk assessment methodology. This complexity reduction makes the system behavior model and risk assessment results increasingly unrealistic and hence decreases the accuracy or precision of the results.

The model used in SimPRA for risk assessment can be the same model that the designers have developed in the simulation environment to show how their system works. The simulation model might combine discrete and continuous events as the model for risk assessment. The only limitations on the simulation are the general computational limitations that a simulation environment might have and there is nothing imposed on it that is specific to the risk assessment approach taken.

4.3 Providing tools for the system modelers to do risk assessment easily and quickly

As mentioned earlier, SimPRA uses general simulation capabilities for risk assessment. SimPRA software comes with a Matlab/Simulink® toolkit which helps to implant stochastic and deterministic elements in the simulation model and connect these elements to the SimPRA scheduler in order to guide the simulation for risk assessment purposes. The simulation model itself can be any model generated in the Simulink® environment. This provides a simple and quick environment for risk assessment, since the simulation model can come from the designers and not the risk analysts.

4.4 Making the communication of risk scenarios compatible with what risk analysts are generally familiar with

Automatically generated high level scenarios are grouped and presented in a generalized form of event sequence diagrams (ESD). This is a very useful feature for the risk analysts to verify the outcome of the planner and decide whether the high level scenarios generated by the planner are sound and complete. This will also be good for communicating the risk scenarios with system designers and verifying the high level scenarios before running the simulation.

4.5 Providing an environment that will progressively improve efficiency over time

SimPRA provides a dynamic environment in which the risk assessment technique matures in the process of simulation. An updating algorithm provides feedback and suggestions to the analysts as to how the planner's knowledge may be updated so that it will be able to generate observed scenarios that were not originally in the plan and stop generating high-level scenarios that do not correspond to any observed detailed scenarios generated by the simulator. After several refinements of the plan, plan-updating might stop giving new suggestions. This would be an ideal situation in which the plan is a reflection of all of the possible high-level scenarios generated by the simulator and none of the scenarios in the plan is left unexecuted.

5 Planner

Automated planning is a part of computer science literature that concerns the realization of strategies or action sequences for execution by intelligent agents, autonomous robots, unmanned vehicles, etc. A plan is defined as a representation of future behavior, usually a set of actions with temporal and other constraints that is executed by some agent or agents (Wilson 2001). Planning is an abstract, explicit deliberation process that chooses and organizes actions by anticipating their expected outcome. This deliberation aims at achieving as best as possible some pre-stated objectives (Ghallab, et al. 2004).

Both the binary and multi-state planning processes introduced here assume that:

1. The model is finite, meaning it has a finite set of states.
2. The system is fully observable, meaning that the planner has complete knowledge about the state of the system.
3. The system is deterministic, that is, if an action or event is applicable to the system, it brings the system to a single other state.
4. The system is static, meaning it remains in the same state until another action or event is applied (Ghallab, et al. 2004).

It is important to mention that these assumptions only apply to the planner's planning model and not to SimPRA's simulation model. More specifically, assumptions 3 and 4 are only used for generating a plan from the planner's model of the system. The planner makes these assumptions to simplify the model and give a road map to the scheduler for guiding the simulation. However, events in the simulator can be probabilistic and fully dynamic. The possible wrong or missing assumptions are fixed or added to the planner's model during the updating process.

The planning process is done off-line meaning that the planner alone knows the initial states and final (goal) states and does not track the state of the system in real time for generating the plan, although on-line planning capabilities are also present and can be used for the future work.

5.1 Comparison between SimPRA planner and other automated planning algorithms

In this chapter, relevant areas of the Automated Planning field—which is a part of the Artificial Intelligence literature in computer science—are very briefly reviewed and the place of the SimPRA planner among other automated planning approaches is clarified. It is important to mention that since the SimPRA planner is designed to serve the risk assessment field and be used for risk scenario generation purposes, it does not follow a generic modeling language (such as first or second order logic languages) like other general purpose planners. The SimPRA planner modeling language can thus be categorized as a Domain Specific Language and SimPRA planner should be categorized as a Domain Specific planner. The following comparison is, however, still valid because the purpose of the comparison is not to compare the models but the planning process itself. To reduce the confusion between the domains, the planning processes suggested in the literature are explained as broadly as possible. However, it is almost impossible to avoid the terminology of the automated planning field which is more intuitive for explaining these planning approaches.

5.1.1 STRIPS

STRIPS is probably the most well-known classical planning approach (Fikes and Nilsson 1971) and usually used as the benchmark problem for planning algorithms. Using the set theoretic planning representation, a STRIPS instance is composed of:

- An initial system configuration $s_0 \in S$ where s_0 is the initial configuration and S is the set of all enumerable sets of configurations of the system ($S \subseteq 2^L$ assuming binary states for L system elements—comparable to L propositions);
- Goal states $g \subseteq L$ – states which the planner is trying to reach;
- Operators O where each operator is itself a triple $o = < name(o), precond(o), effects(o) >$. These three elements, in order, specify the name of the operator, pre-conditions for the operator and the effects of the operator (comparable with functionalities in the SimPRA modeling language);

A plan for such a planning instance is a sequence of operators (functionalities) that can be executed from the initial state and that leads to a goal state. Transitions between states are modeled by a transition function γ, which is a function mapping states into new states that result from the execution of actions and events.

Figure 5 shows the STRIPS planning algorithm. Simply stated, STRIPS tries to find a sequence of actions that can take the system from state s_0 to state g by identifying the sequence of actions whose preconditions can be satisfied by the effects of earlier actions. This is achieved by going backward from the goal state g and finding actions whose effects would satisfy g. When an action is found, the state that would satisfy its precondition is set as the new goal (or sub-goal) and the process will continue until state s_0 is reached. Reversing the order of the actions will now take the system from s_0 to g.

```
STRIPS ($O, s_0, g$)
    $\pi \leftarrow$ the empty plan
    loop
        if $s_0$ satisfies $g$ then return $\pi$
        $A \leftarrow \{a \mid a$ is an operator in $0$, and $a$ is relevant for $g\}$
        if $A = \varnothing$ then return failure
        non-deterministically choose any action $a \in A$
        $\pi' \leftarrow$ STRIPS($O, s_0, precond(a)$ )
        if $\pi' =$ failure then return failure
        ;; if we get here, then $\pi'$ achieves $precond(a)$ from $s_0$
        $s \leftarrow \gamma(s_0, \pi')$
        ;; $s$ now satisfies $precond(a)$
        $s_0 \leftarrow \gamma'(s\ , a)$
        $\pi \leftarrow \pi.\pi'.a$
```

Figure 5: STRIPS planning algorithm extracted and modified from Ghallab et al. 2004

According to Ghallab, et al. 2004 the limitations on STRIPS are:

1. In each attempt to identify a sub-goal, the only sub-goals that are considered are the preconditions of the last previous operator that was added to the plan. This reduces the branching factor substantially but makes STRIPS incomplete.
2. If the current state satisfies all of an operator's preconditions, STRIPS will commit to executing the operator and will not backtrack over this commitment. This prunes off a large portion of the search space but, again, makes STRIPS incomplete.

Other than incompleteness—which is a huge problem for the application of the planner in the SimPRA approach—modeling real world problems for solving with STRIPS is not trivial. The STRIPS language is very abstract and unable to capture the wide range of expert knowledge that can help the planning process become more effective and accurate.

5.1.2 Hierarchical Task Network (HTN)

Hierarchical Task Network (HTN) planning is probably the most popular planning technique currently used in real world applications. HTN is similar to STRIPS in the deterministic assumption made for actions. It differs from STRIPS in the way the problem is modeled, goals are defined, and consequently, the planning process is done.

In HTN, the goal of the planning is not just to reach a number of goals but to find a way to reach the goals by performing a number of *tasks*. Tasks are a group of operators and other tasks that define a sequence of actions that are put together to serve a specific purpose in the planning domain. Tasks can be either primitive or non-primitive. Primitive tasks are only composed of the planning operators while non-primitive tasks are composed of both operators and other primitive or non-primitive tasks. During the planning process, tasks are recursively decomposed to lower level tasks, primitive tasks and finally to simple operators while the preconditions of the tasks and operators are constantly checked and the new states generated by the execution of actions and events are traced and accounted for. Figure 6 shows the algorithm for an abstract-HTN planner.

Using set theoretic presentation, an HTN domain is a pair $D =< O, M >$ and an HTN planning problem is a four-tuple $P =< s_0, w, O, M >$ where:

- s is the (initial) configuration.
- $w =< U, C >$ is the task network where U is a set of task nodes and C is a set of Constraints and t_u is the task of U.
- O is the set of operators.
- M is a set of HTN methods where each method:

  $m =< name(m), task(m), w(m) >$.

```
Abstract-HTN($s$, $U,C,O,Ml$)
    if $(U, C)$ can be shown to have no solution
        then return failure
    else if $U$ is primitive then
        if $(U, C)$ has a solution then
            nondeterministically let $\pi$ be any such solution
            return $\pi$
        else return failure
    else
        choose a nonprimitive task node $u \in U$
        $active \leftarrow \{m \in M$ I $t_m$ can follow $t_u\}$
        if $active \neq$ ø then
            nondeterministically choose any $m \in active$
            $(U', C') \leftarrow$ decompose $(U, C)$
            return Abstract-HTN($s$, $U',C'$, 0, $M$)
                else return failure
```

Figure 6: HTN planning algorithm extracted and modified from Ghallab et al. 2004

Simply said, the HTN planner tries to identify a sequence of tasks that, when decomposed to primitive tasks and actions and executed, would change the state of the system from the initial state to the goal state while ensuring that the preconditions of each task and action is satisfied by the execution of the previous tasks and actions. The decomposition of tasks to subtasks and actions is predefined by the task network.

As mentioned before, Hierarchical planning is much more intuitive compared to the STRIPS language. Albus and Meystel 2001 argue that Hierarchical planning is similar to the way that the human brain performs planning. HTN provides a platform in which the planning process is done by first considering the longer-term goals in the highest levels of the task network and then breaking it into simpler tasks and actions to achieve shorter-term goals until the entire task is complete.

5.1.3 SimPRA Planning Method

The SimPRA planning method uses the hierarchical task network approach with a number of simple modifications:

- First, In HTN planning, to execute a task it is decomposed into another predefined group of sub-tasks which in turn are decomposed into predefined primitive tasks and actions and so on. In the SimPRA planner, however, each task (or in SimPRA terminology, each function) defines a new sub-goal (i.e. reaching a subsystem state) which after a simple planning process will provide the sub-tasks (or, in SimPRA terminology, events).
- Second, the SimPRA planning algorithm seeks to find all of the solutions to the planning problem unlike STRIPS and HTN, which seek only one solution.
- Third, for reasoning purposes, SimPRA takes advantage of other sources of information (like qualitative reasoning) which are not necessarily a part of the planning process but can help making the planning process more

efficient by pruning the branches that are not interesting to the user of the system.

As mentioned earlier, it is important to remember that although in this chapter the SimPRA planner is being compared with general purpose planners like STRIPS and HTN, the SimPRA planner is designed to best serve the *risk analysis domain* and thus the *terminology* used in the rest of the thesis is more consistent with risk analysis concepts. Also the *modeling process* that is suggested to capture the planner's model is tailored to meet the risk assessment field's needs.

In the SimPRA approach, in contrast with the HTN approach which has a solid predefined task network, the task network itself is dynamically generated during the planning process. This approach provides the planner with an opportunity to perform real time planning by taking into account the state of system elements at the moment of planning. It also provides the users an opportunity to focus on one system component at a time during the modeling process and not worry too much about the whole structure of the network.

Having compared the SimPRA planner with other general purpose planners, the next section describes SimPRA's planning approach in greater detail.

5.2 SimPRA binary vs. multi-state planner

The SimPRA planner component proposed in this dissertation comes in two versions: A *binary planner* and a *multi-state planner*. The difference between these two versions is not only in the model of the system that they use, but also in the planning process and assumptions behind it.

The binary planner assumes that all of the subsystems and components have only two states, up and down. It also considers the failure and success of functionalities as the failure and success of the sub-components that provide those functionalities (this will be discussed in greater detail below). The planning problem for the binary planner is almost a classical one. The goal is to reach a set of system states and the plan is all possible sequences of actions and events that take the system from the initial state to the goal states.

In the multi-state planner, subsystems and components can have more than two states. The functionality definitions have more substance to them as well. They can be defined by a range of states of the sub-components providing them. They can also have time and landmark testing preconditions and duration or landmark settings as their side effects. Qualitative reasoning is used to enhance the definition of the planning goals and to generate more realistic plans for complex and dynamic systems. The planning problem in the multi-state planner is different from classical planning problems in that: 1) the definition of goals entails more than just reaching a set of system states; goals are defined through a qualitative reasoning process; 2) The plan is not just a sequence of actions or events as it also includes some time and landmark constraints; and 3) Time can be explicit in this modeling process—actions and events can have a duration and the occurrence of them can be conditioned on time and landmarks as well. In the planning process, some of the lines of plan that do not comply with the goal statement are pruned and quickly taken out of the process.

Among the most important capabilities of both the binary and multi-state planners is the capture of different types of engineering knowledge with which risk scenarios can be automatically generated. Table 1 shows which data structures are used by the planners to capture twelve different types of engineering knowledge. Each of these data structures are discussed in greater detail in the following two sections of this chapter.

Table 1: Types of engineering knowledge captured by planner

Type of Engineering Knowledge	**Captured by**	
	Binary planner	**Multi-state planner**
System elements and hierarchy	Structure Tree	Structure Tree
Elements' states and operational modes	Assumed binary (work or fail)	Structure Tree
Functionalities/ Activities/Events provided/Acted upon by elements	Functionalities for System level only	Functionality Tree
The relationship between functionalities and sub-functionalities/Activities and events	-	Functionality Tree
The allocation (assignment) of functionalities among components	Mapping between Functional and Structural Trees	Mapping between Functional and Structural Trees
The interplay between functionalities and states of the system	State Transition Diagram	State Transition Diagrams
The interplay between functionalities and states of the subsystems/ components	Assumed only one transition from work to fail state	State Transition Graphs
The relationship between the functionality of the system with the state of the subsystems and components	Mapping between Functional and Structural Trees	Transition Rules
Time dependencies	-	Transition Rules
Conditionality of the functionalities on the state of the other elements	-	Transition Rules
Importance of the elements to risk assessment	-	Transition Rules
Boundary conditions	Deducted from the Mapping between Functional and Structural Trees	Qualitative Reasoning Influence Diagram

To aid in the explanation of both of the planning approaches, an earth observation satellite (EOS) system is used as an example. This example was introduced by a Jet Propulsion Laboratory (JPL) research team as a way to compare the risk assessment approaches of several NASA and university research groups during a workshop held on the topic.

The example EOS system has two major parts, a ground station and a satellite. To simplify the problem, the ground station is assumed to work without failure and is controlled automatically by the simulation software so that only the satellite part is modeled for risk assessment. The satellite system goes through several operational modes in cycle and in each mode it uses some of its subsystems and components to perform the needed tasks. The major operational modes are:

- *Receive-command (uplink-data):* Various commands are sent to the satellite for observation and housekeeping purposes to determine the timing and locations for observations and to maintain the satellite in a healthy state. The satellite's computer, communication subsystem (including receiver and antenna), BUS, RCS and software are engaged in the functionalities of this mode.
- *Collect-data:* The satellite acts on the received commands to collect the planned data and store it in the memory. The satellite's computer (memory and processor), BUS, RCS and software are engaged in the functionalities of this mode.
- *Process-data:* The raw data collected by the observation tools and housekeeping sensors go through several processes such as quality control and compression for future use. The satellite's computer, BUS and software are engaged in the functionalities of this mode.

- *Downlink-data:* The range where the satellite and the ground station can communicate is limited. The data collected outside of that range will be transmitted to the ground when the satellite is in the range where it can communicate with the ground. The Satellite's computer, communication subsystem (including transmitter and antenna), BUS, RCS and software are engaged in the functionalities of this mode.
- *Standby:* The satellite waits for the ground station to process the transmitted data and provide it with a new plan for data collection and housekeeping purposes. The satellite's computer (clock and processor) and software are engaged in the functionalities of this mode.
- *Safe-mode:* When there is a minor hardware or software problem, the satellite goes into this mode for automatic repairs. If the satellite can recover, it continues in the same mode it was operating in before the failure. For the purposes of simplification, the assumption in this example is that the satellite will only go to the safe-mode if there is a problem in the collect-data mode. The satellite's computer (clock and processor), communication subsystem (including transmitter and antenna) and software are engaged in the functionalities of this mode. The computer's clock failure will get the system in this mode and if there is a redundant clock available, it will recover, otherwise the system will go to the fail mode.
- *Fail:* The satellite goes to the fail mode when a major subsystem or component has failed and the satellite can not recover from this failure. Again, to simplify the problem, it is assumed that the system goes into fail

mode from collect-data and safe-mode. If the computer's memory or processor, or if BUS, RCS, or software fail, the system will end up in this state.

5.3 Binary planner

As mentioned earlier, the binary planner assumes that all of the subsystems and components have only two states, work and failure (or up and down). It also considers the failure and success of functionalities as the failure and success of the sub-components that provide those functionalities. The planning problem is defined as finding all sequences of actions and events that take the system from the initial state to the goal states.

The inputs for the binary planner include: states, state transitions (transition name, sub-goals and constraints), goal, and system hierarchy. Using the set theoretic presentation, a SimPRA domain is a pair $D = < T, \gamma >$ and a SimPRA planning problem is a four-tuple $P = < s_0, g, T, \gamma >$ where:

- s_0 is the (initial) state and g is the goal state.
- $T = < W, E >$ is the hierarchical network which is composed of Events E and functionalities W where in each functionality $w = < F, Decomp(F) >$, F is the functionality name and *Decomp(F)* is a function that decomposes F to events E (practically returns the cut-sets for F)
- $s' = \gamma(t, s)$ where $t \in T$ and $\gamma(t, s)$ is the transformation function that transforms the state of the system from s to s'.

Figure 7 presents the set theoretic presentation of the SimPRA binary plan algorithm. At the start of the planning process *Sol*[] and Π[] are empty. In this algorithm, if we consider $Sol[] = (t_1, t_2, ..., t_n)$ then $Sol[] \oplus t^* = (t_1, t_2, ..., t_n, t^*)$ and $t_{last}(Sol[]) = t_n$. Also, the $execute(\pi)$ function will return 'True' when the functionalities can be decomposed to events. An example of a case in which $execute(\pi)$ might return 'false' would be if the decomposition logic requires that a component be in the success state but the same component has already been required to fail (no repair assumption). In the binary planner, the decomposition process is very simple and consists of replacing the functionality with the logic that links it to the success or failure of the components. To have a complete plan, all the combinations of decomposition logics must be considered.

```
SimPRA-Binary-Plan ($s_0, g, T, \gamma, Sol[], \Pi[]$)
  if $s_0 = g$
    then return $Sol[]$
  else
      $Sol[] = Sol[] \cap \{ \Pi[] \oplus t \mid t \in T, \gamma(t_{last}(\Pi[]), s_0) = g \}$
      $P[] = \{ p \oplus t \mid t \in T, p \in \Pi, \gamma(t_{last}(p), s_0) = s, s \neq g \}$
      $\Pi[] = \{ \pi \mid \pi \in P, execute(\pi) \equiv True \}$
      if $\Pi[] = \emptyset$ then
          return $Sol[]$
      else
          Return SimPRA-Binary-Planner ($s_0, g, T, \gamma, Sol[], \Pi[]$)
```

Figure 7: SimPRA binary plan algorithm

The modeling process suggested for the SimPRA binary planner starts with a top down, step-by-step approach to soliciting knowledge from the user. When the model is ready, the planning algorithm is applied to generate the scenarios. Using detailed scenarios generated by the simulator after a number of simulation runs, the generated model can later be refined through the updating process. All of these steps are explained in more detail in the following three sections.

5.3.1 Modeling

The first step in the modeling process for the binary planner is for the user to define the hierarchical structure of the system that includes the relationship between the system, subsystems and components. In the second step, the functionalities that are involved in the process are defined and mapped to the subsystems and components that enable them. In the last step, the user defines the state transitions for the system. These transitions show the states of the system and how the system evolves over time. Each of these steps is explained in more detail below.

STEP ONE: ***Defining structural hierarchy***

Structural hierarchy refers to the subsystems and components that the system is built of. The 'AND' relation shows the distinct elements while 'OR' is used to present the redundancies. Figure 8 presents the structural hierarchy of the example EOS system. The main subsystems are computer, camera, communications, BUS, RCS and software. The computer subsystem has memory, processor and clock (with redundancy) as components. The camera system has no other components. Finally, the communications subsystem has transmitter (with redundancy), receiver and antenna as its components.

STEP TWO: ***Defining the functionalities***

Functionalities are defined by the state of the components that are needed to perform the tasks. In the binary version of the SimPRA planner, the assumption is that components have binary states; they either work or fail. Since we are interested in the functionalities that change the state of the system, component failures are only seen in this context.

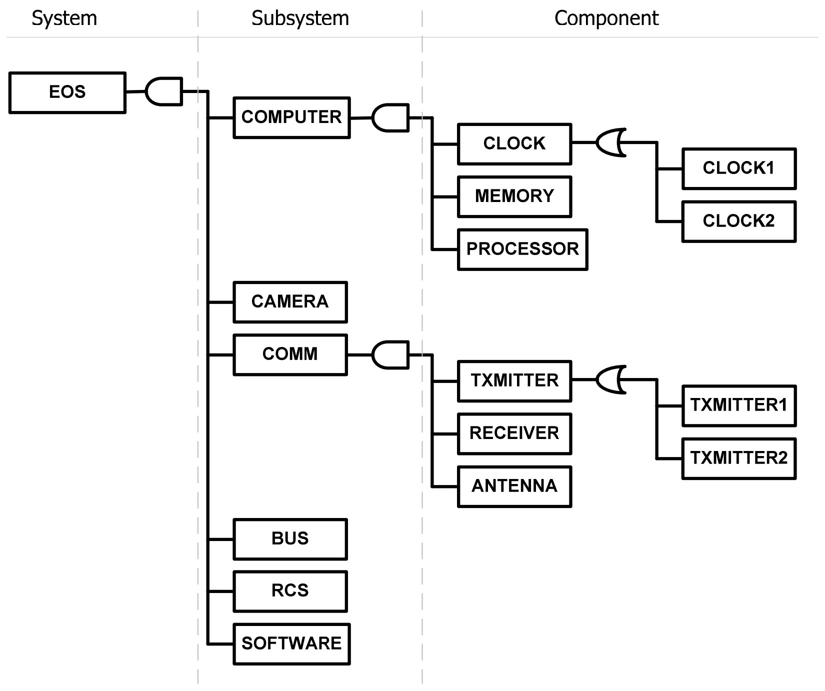


Figure 8: Structural hierarchy of example EOS system

Christensen and Lind 1996 have suggested a similar approach to modeling the functional requirements of a complex system. They present the successful use of a Master Plan Logic Diagram in mapping between functionalities and components for capturing the functionalities of an autonomous submarine.

Modarres 1996 also suggested a framework for the functional modeling of complex physical systems composed of human, software and hardware elements. In his framework, a complex system is described by five different hierarchies: structural hierarchy (system organization in space), functional hierarchy, behavioral hierarchy, goal/conditional hierarchy, and event hierarchy (the last four refer to system organization in time). Among these 5 hierarchies, however, the functional hierarchy is viewed as especially important. This is because aspects of the functional hierarchy can be defined by other hierarchies and, more importantly, through use of the relationship class concept the functional hierarchy can be used to build other hierarchies. One of the most important outcomes of this framework is the generation of a Goal Tree Success Tree (GTST) model of the system that presents the logic of the success of the system considering the current state of the system. GTST changes as the state of the system changes to reflect changes in the goals or in the environment.

The approach suggested in this dissertation is, in concept, very similar to the Modarres approach. The behavior of the system is defined by changes in the state of the system and its elements in the hierarchy. Changes in the states are triggered (or from the planner's point of view are initiated) by changes in the functionalities and scenarios are generated by putting these changes in order to reach the goal states. The only main difference in the approaches is that in GTST, when the goal is defined, lack of any functionality that is deemed necessary for the success of the goal is considered a prohibitory event for reaching that goal and thus a failure initiator. This assumption is kept in the SimPRA binary planner, but is then relaxed for the multi-state planner.

Figure 9 presents the relationship between components and functionalities. According to Figure 9, the STDB function, for example, uses COMPUTER (PROCESSOR and CLOCK which has a redundant part) and SOFTWARE in its process.

If a component is shown to be used by a function, the subsystem that the given component is a part of will be selected as well. For example, if CLOCK1 is selected, then CLOCK and COMPUTER will automatically be selected as well. The logic behind this is that it is not possible to have a component of a subsystem be involved in a function without the subsystem itself having a role in it.

The logical relationship between the selected components and functionality comes from the structural hierarchy of the system. This logic is defined separately at the subsystem level and the component level (discussed later).

Functionality / Component	Uplink	CollectCrit	CollectSafe	Process	Downlink	STDB	Repair
COMPUTER	X	X	X	X	X	X	X
CLOCK	X		X	X	X	X	X
CLOCK1	X		X	X	X	X	X
CLOCK2	X			X	X	X	X
MEMORY	X	X		X	X		
PROCESSOR	X	X		X	X	X	X
CAMERA		X					
COMM	X				X		X
TXMITTER					X		X
TXMITTER1					X		X
TXMITTER2					X		X
RECEIVER	X						
ANTENNA	X				X		
BUS	X	X		X	X		
RCS	X	X			X		
SOFTWARE	X	X		X	X	X	X

Figure 9: Component-Functionality matrix of example EOS system

Another important use of this table is to define the complexity of the components/subsystems. One very simple criteria of complexity would be the coupling between functionalities and components/subsystems. Using components to do multiple tasks makes the design of the system very complicated. Complexity criteria such as coupling are used to rank the generated scenarios for the scheduler to prioritize the more complex scenarios for simulation.

STEP THREE: ***Defining system level state transition***

Explicit knowledge about the system behavior is obtained by Finite-State Machines (FSM). The Finite-State Machine is depicted by a directed graph (digraph) and consists of a set of states S, an initial state S_0, a subset A of S which is the set of accepting-states or end-states, a finite set of input symbols T and a transition function $\gamma : S \times T \rightarrow S$ that maps input symbols and current states to a next state.

The reason for choosing Finite State Machines over Event Sequence Diagrams for capturing the system behavior is that FSM is a stronger language for capturing the behavior of dynamic systems than ESD. FSM captures both states and transitions which in a sense defines not only the rules of changing the behavior, but also the conditions needed for these changes (states).

FSMs are used in computability theory and in some practical applications such as digital logic design and graphical user interface design (Börger and Stärk 2003; Horrocks 1999).

The FSM that is used for modeling is a Mealy machine which assumes that actions are associated with transitions (Börger and Stärk 2003). There is also the possibility of defining more than one transition for a given input symbol and state (nondeterministic finite state machine). For our modeling purposes, if the state machine is always starting from one specific state, then the user has the option to define an initial state. There are three steps in modeling the system FSM:

I. Defining the system level states:

Modeling starts with the study of the system's mission to identify the modes of success and failure within the system. Each failure and success mode is given a name and will be considered as an end-state. There are usually several modes through which the system progresses before reaching the end-states. These modes will be considered as the transitional states.

Going back to our EOS system, the Satellite will be in the standby mode (Standby) until it is accessible from the ground station. At this time, the Satellite will start looking for signals that are sent from the ground station. When it receives the commands (Receive-command), it will start collecting data (Collect-data) as specified by the commands received, for example taking pictures from a specific location. The data will then be processed (Process-data) and sent to the ground station (Downlink-data). If there is a critical failure, the system goes directly to the fail (Fail) mode, otherwise it will go to the safe mode (Safe-mode) for simple repairs. If the repairs are successful, the system will continue the work, otherwise it will enter Fail state.

II. Defining the transition diagram:

The visual representation of the transition of the system between states is the Transition Diagram. Figure 10 illustrates the example EOS system's transition diagram. In this figure, states are shown as circles and the functionalities that enable the transitions among them are depicted by rectangles. The loop on the left is the normal process that the system goes through to collect the requested data and send it back to earth. On the right, two failure modes are shown. In this diagram, Standby is the initial state and Fail state is a sink, meaning that there is no way to recover the system once it enters the Fail state.

III. Defining the transition rules:

In order for a transition to become available, some input function should be provided. Since we are using FSM for planning, the input function is the set of actions that need to be taken to change the state of the system. The system will not change its state unless one of the actions that can take it to another state is activated and has been successful. For example, Figure 10 shows that for the system to go from PROCESS_DATA to DOWNLINK_DATA, Downlink should be successful (Downlink_Success). If the Downlink process is activated but is not successful, the system will still remain in the PROCESS_DATA state.

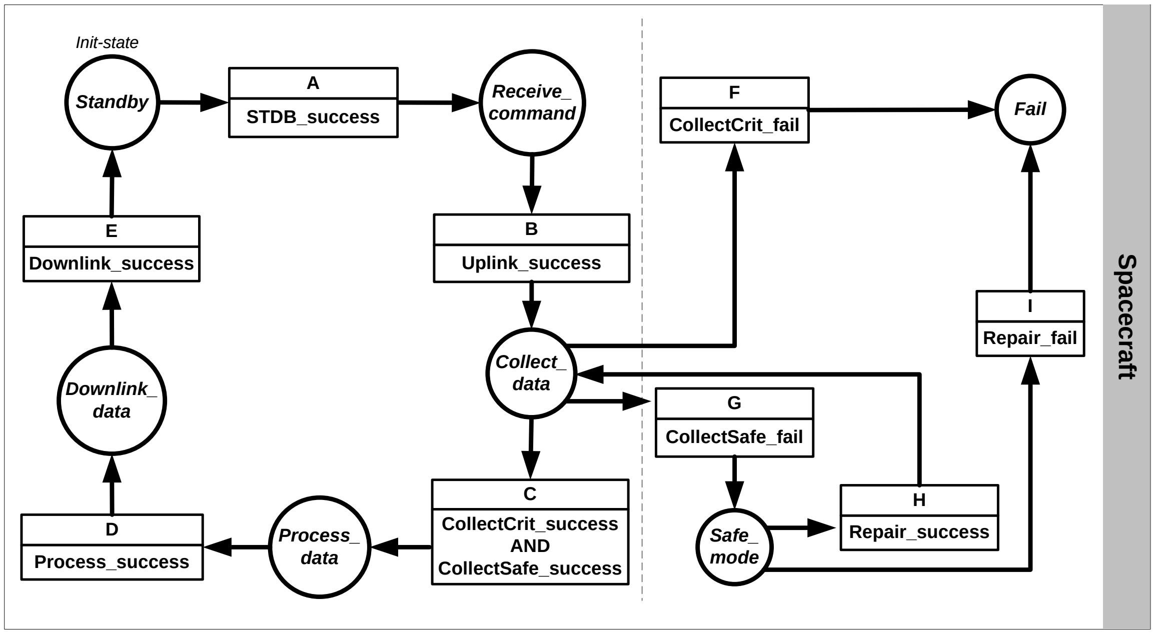


Figure 10: State transition diagram of example EOS system

Input functions should be sentences of Boolean logic built from atomic sentences by means of truth-functional connectives alone. Constants (known as propositional names or simple statements) are Boolean function names that refer to the success or failure of functionalities. For example, “CollectCrit_Success AND CollectSafe_Success” is considered a valid statement.

5.3.2 Plan generation algorithm

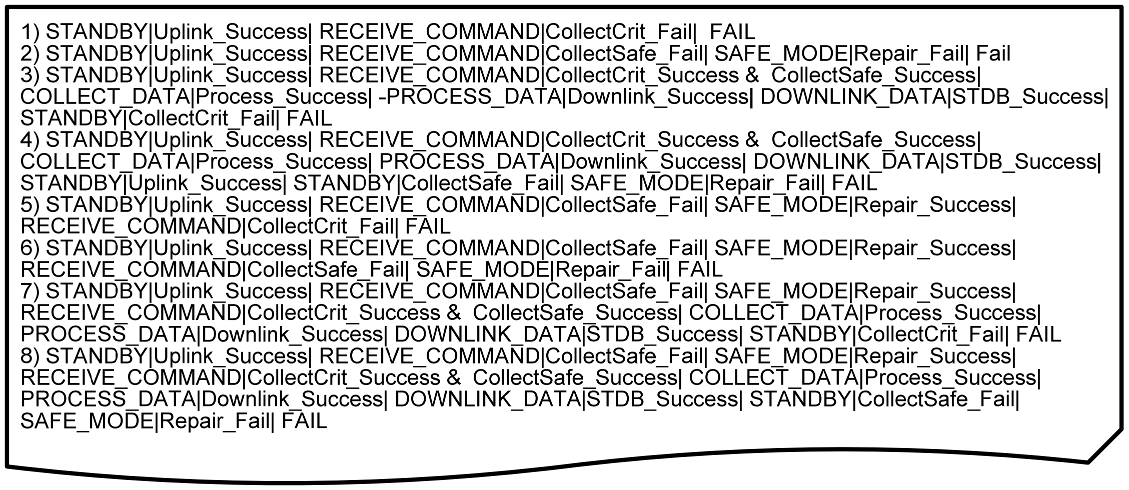
The plan is generated at two different levels: the subsystem level and the component level. This will affect the extent of control that the scheduler might have over the simulation. Plan generation can also be broken down into several steps:

Step 1) The user defines the initial state and end-state(s) for generating the plan. The user also decides how many times an event might be repeated in a scenario. This is necessary to prevent the planner from getting trapped in infinite loops. Figure 10 presents an example of this. As illustrated, it is theoretically possible for the system to enter and exit safe-mode a near infinite number of times. This is not realistic, however. Although it is true that a scenario with 100 safe-mode visits, for example, is different from a scenario with 101 safe-mode visits, limitations on the number of scenarios that can be executed within a reasonable amount of time dictate that the user should limit the number of revisits to a reasonable number.

Step 2) All possible paths from the initial-state to end-state(s) will be considered and the functionalities that make the transitions possible are listed. For example, all of the scenarios for going from the Standby state to the Fail state with only one revisit would be the ones that are listed in Figure 11.

States and functionalities are separated from one another by a bar sign. When an AND is used, it means that an event should be completed and the system should be in the new state before a new event can be started.

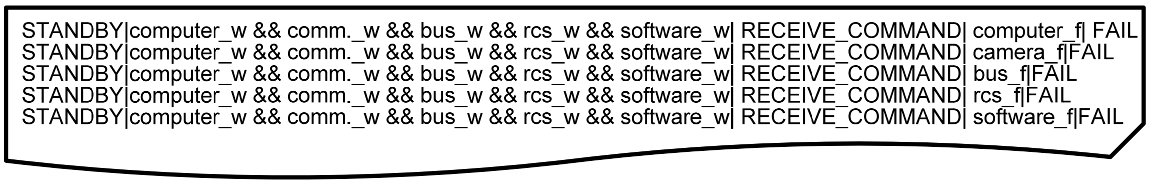
Step 3) Functionalities are replaced with their logical equivalents in terms of subsystem/component states based on the level of planning. For example, with the help of Figure 8 and Figure 9, the logic for the success of STDB functionality at the subsystem level is determined to be COMPUTER_W AND SOFTWARE_W and at the component level is (CLOCK1_W AND PROCESSOR_W AND SOFTWARE_W) OR (CLOCK2_W AND PROCESSOR_W AND SOFTWARE_W).



```
1) STANDBY|Uplink_Success| RECEIVE_COMMAND|CollectCrit_Fail|  FAIL
2) STANDBY|Uplink_Success| RECEIVE_COMMAND|CollectSafe_Fail| SAFE_MODE|Repair_Fail| Fail
3) STANDBY|Uplink_Success| RECEIVE_COMMAND|CollectCrit_Success &  CollectSafe_Success|
COLLECT_DATA|Process_Success| -PROCESS_DATA|Downlink_Success| DOWNLINK_DATA|STDB_Success|
STANDBY|CollectCrit_Fail| FAIL
4) STANDBY|Uplink_Success| RECEIVE_COMMAND|CollectCrit_Success &  CollectSafe_Success|
COLLECT_DATA|Process_Success| PROCESS_DATA|Downlink_Success| DOWNLINK_DATA|STDB_Success|
STANDBY|Uplink_Success| STANDBY|CollectSafe_Fail| SAFE_MODE|Repair_Fail| FAIL
5) STANDBY|Uplink_Success| RECEIVE_COMMAND|CollectSafe_Fail| SAFE_MODE|Repair_Success|
RECEIVE_COMMAND|CollectCrit_Fail| FAIL
6) STANDBY|Uplink_Success| RECEIVE_COMMAND|CollectSafe_Fail| SAFE_MODE|Repair_Success|
RECEIVE_COMMAND|CollectSafe_Fail| SAFE_MODE|Repair_Fail| FAIL
7) STANDBY|Uplink_Success| RECEIVE_COMMAND|CollectSafe_Fail| SAFE_MODE|Repair_Success|
RECEIVE_COMMAND|CollectCrit_Success &  CollectSafe_Success| COLLECT_DATA|Process_Success|
PROCESS_DATA|Downlink_Success| DOWNLINK_DATA|STDB_Success| STANDBY|CollectCrit_Fail| FAIL
8) STANDBY|Uplink_Success| RECEIVE_COMMAND|CollectSafe_Fail| SAFE_MODE|Repair_Success|
RECEIVE_COMMAND|CollectCrit_Success &  CollectSafe_Success| COLLECT_DATA|Process_Success|
PROCESS_DATA|Downlink_Success| DOWNLINK_DATA|STDB_Success| STANDBY|CollectSafe_Fail|
SAFE_MODE|Repair_Fail| FAIL
```

Figure 11: Possible paths from STANDBY state to FAIL state of the example EOS system

Although SOFTWARE is only defined at the subsystem level, to maintain the consistency of the plan, it is repeated at the component level as well. The first line of the above plan (Standby|Uplink_Success| Receive-command| CollectCrit_Fail|Fail) will generate the scenarios at the subsystem level as presented in Figure 12.



```
STANDBY|computer_w && comm._w && bus_w && rcs_w && software_w| RECEIVE_COMMAND| computer_f| FAIL
STANDBY|computer_w && comm._w && bus_w && rcs_w && software_w| RECEIVE_COMMAND| camera_f|FAIL
STANDBY|computer_w && comm._w && bus_w && rcs_w && software_w| RECEIVE_COMMAND| bus_f|FAIL
STANDBY|computer_w && comm._w && bus_w && rcs_w && software_w| RECEIVE_COMMAND| rcs_f|FAIL
STANDBY|computer_w && comm._w && bus_w && rcs_w && software_w| RECEIVE_COMMAND| software_f|FAIL
```

Figure 12: Partial plan based on one path from STANDBY state to FAIL state

Scenarios that are generated by the planner are very rough probable paths toward end-states. The scheduler uses this high level guidance to guide the simulation and generate distinct scenarios. The major difference between the scenarios generated by the planner and the scenarios generated by the scheduler/simulator is in the timing and level of detail. In the planner's scenario, for example, we might see a command such as: "RECEIVE_COMMAND|SOFTWARE_F" which means that there should be a Software failure when the system is in the RECEIVE_COMMAND state. This one command can be used by the scheduler to generate a number of scenarios that differ as to: 1) the exact time of the failure within the time interval that the system is in the RECEIVE_COMMAND state; and 2) the type and importance of the software failure that is induced.

5.3.3 Comparison and conclusion

Risk scenarios in a traditional event tree analysis are all the paths from the initiating event to end-states. Some of the events might have a fault tree, showing how those events can be caused by some other primary events. Replacing these events with the set of minimal cut-sets of those events in each scenario will give all combinations of primary events that can generate that scenario.

For example, the event tree in Figure 13 shows that the loss of Autopilot and loss of Engine can cause Loss of Airplane. The fault trees show how each of the 'Loss of Autopilot' and 'Loss of Engine' events can be generated by the primary events. For example, Loss of Autopilot can be generated by one primary event such as 'Power System _Fail'. 'Loss of Engine' needs two primary events to happen which are 'Main Engine _Fail' and 'Spare Engine _Fail'. Putting the primary events together, the first scenario is generated.

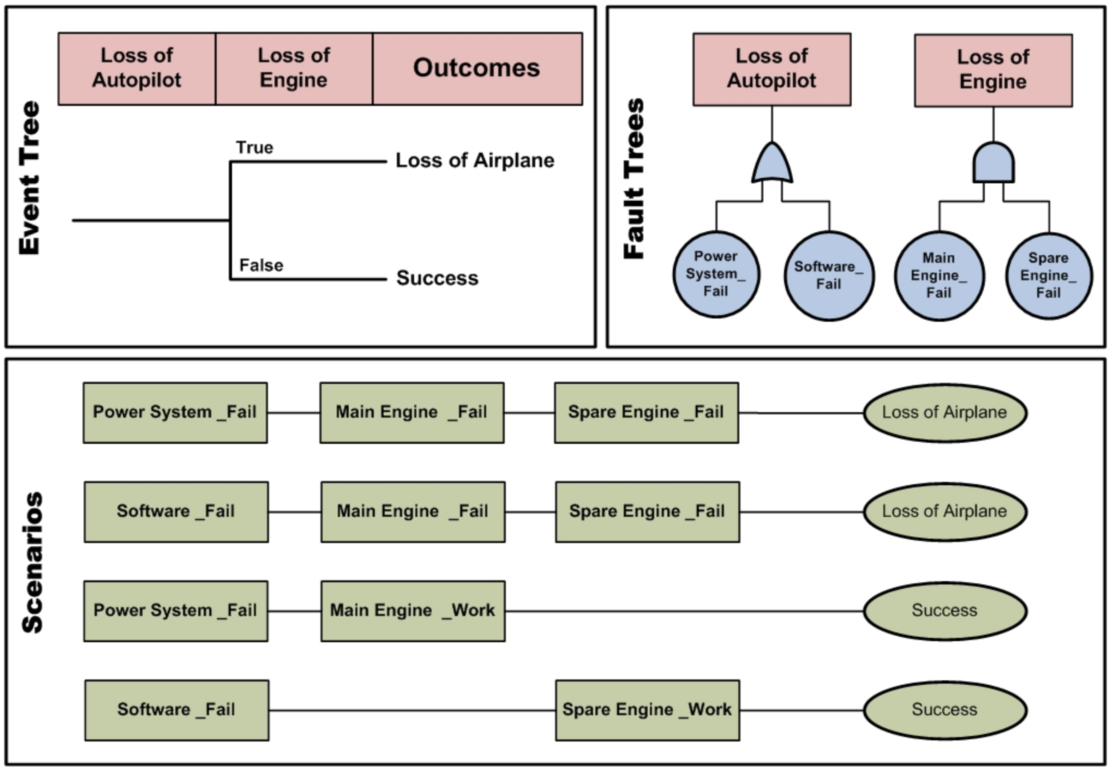
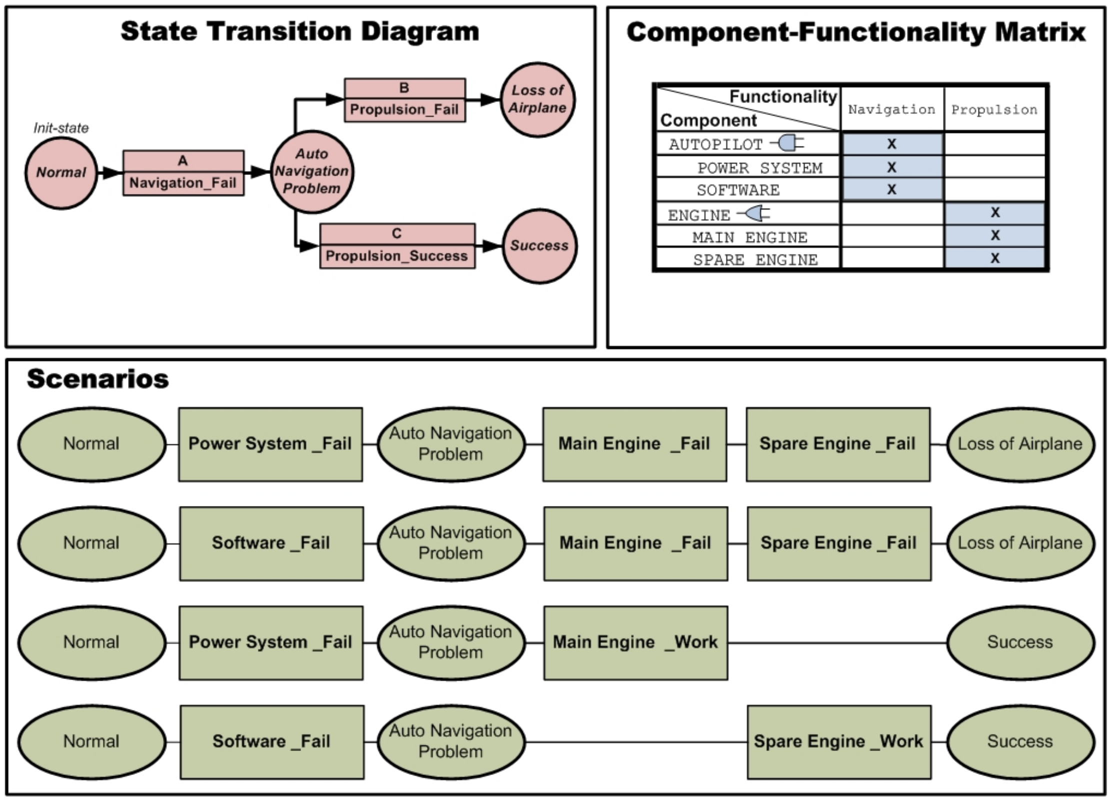


Figure 13: Scenario generation in traditional FT/ET approach



Functionality / Component	Navigation	Propulsion
AUTOPILOT	X	
POWER SYSTEM	X	
SOFTWARE	X	
ENGINE		X
MAIN ENGINE		X
SPARE ENGINE		X

Figure 14: Scenario generation in SimPRA binary planner approach

For the sake of comparison between the approaches, Figure 14 shows the same problem modeled with the new approach. The state transition diagram shows that the airplane goes from the ‘Normal’ operational state to the ‘Auto Navigation Problem’ state when the Navigation functionality fails. We know from the Component-Functionality Matrix that the Navigation functionality is provided by the Autopilot subsystem which itself consists of ‘Power system’ and ‘Software’. So the navigation failure can be caused by the failure of either of these components. The same logic can be followed to the end of a path in the state transition diagram to generate the scenarios.

The new approach produces the same results but from a different perspective. Instead of thinking about the sequence of events and what triggers them, one can think about the system’s hierarchy and its behavior in the form of state transitions.

5.4 Multi-state planner

For the multi-state planner, some of the assumptions used in binary planning are relaxed. As mentioned earlier, the binary planner assumes that all of the subsystems and components have only two states, up or down. In multi-state planning, however, all of the subsystems and components can have more than two states. This brings more flexibility to the modeling and planning environment, but it also requires more effort from the system modeler to determine the state transitions and what enables these transitions based on the state of any lower level elements that exist. Time is also explicitly presented in this version of the planner. State transitions can have a duration or set landmarks to indicate the progress of time. Transitions can be conditioned on time calculated from the start of a scenario based on the duration of transitions coming before them. They also can be conditioned on the presence or absence of landmarks set by other transitions.

Another major difference between the two versions of the planner is that in the binary planner, the system failure model is simply built upon the failure and success of functionalities providing the transitions between the system states, so that there is no ambiguity about what is considered to be a failure and what is considered as a success. In the multi-state plan, however, there is no exact definition for success and failure and the emphasis is on the system behavior in general. To provide a tool to specify the risk scenarios among all of the possible scenarios, a qualitative reasoning tool is provided that works as a filter for risk scenarios. The user can specify any combination of states of the system elements that he or she deems 'interesting' in the form of qualitative reasoning logic—called a QR influence diagram in this dissertation—and use it to define the goal of the planning process. This helps the system analyst to focus solely on the system behavior when she is defining the system hierarchy and state transitions, and then focus solely on defining risky behaviors when the risk logic tree is constructed with the qualitative reasoning tool.

The theory behind the multi-state planning process is explained using the same steps used to explain the binary process. Below, the modeling steps are explained first. This is followed by an explanation of the planning and updating processes. The reader may also refer to Appendix A: Software implementation of the SimPRA multi-state planner for a step-by-step implementation of the multi-state planning process in the JAVA programming language.

5.4.1 Defining risk scenarios in the multi-state planner

According to the Kaplan-Garrick definition of risk, Risk *R* is defined as (Kaplan and Garrick 1981):

$R = F\langle S_i, L_i, X_i\rangle\}$

where S_i denotes the *i*th risk scenario, L_i denotes the likelihood of that scenario and X_i denotes the damage vector or consequences of that scenario. For the purposes of risk assessment, it is necessary that the set of scenarios be complete, that individual scenarios be disjointed and that the number of scenarios be finite (Kaplan and Garrick 1981). To achieve this goal we use a systematic approach to modeling that considers the hierarchy of a typical system.

By defining the states of each element of the system such that they partition the space of all possible behaviors of that element, different combinations of the states of the elements will generate all theoretically possible snap-shots of the overall system at a certain point in time. By considering all possible sequences yielded by different combinations of the snap-shots at time A with those at time B, C, D, etc. along with the actions required to move from one snap-shot to another, all possible scenarios of the overall system are generated. The number of scenarios generated may be extremely large but will nonetheless be finite. Furthermore, the acquisition of system behavior knowledge will restrict the eligible pool of scenarios even further by indicating that some combinations of states/snapshots are physically impossible while others may be impossible because of the history of the system, etc. As such, the number of scenarios that must be considered is expected to drop significantly.

Returning to our satellite example, Figure 15 shows the states of the hardware subsystems and components. At any given point in time, each of the subsystems or components is in one of its states. Based upon the engineering knowledge acquisition stage, however, we may know that some combinations are physically impossible.

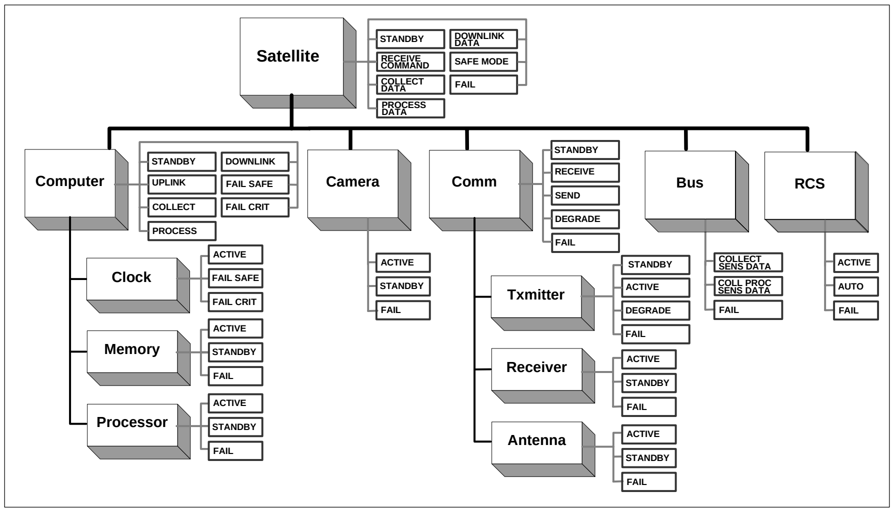


Figure 15: System hierarchy of the example satellite system

Figure 16 further develops the satellite example by presenting four different snapshots of the system. Figure 16A, B, and C show the states of the satellite system and subsystems/components while sending data to the ground station, waiting for the specified time at which it can start receiving commands, and after having experienced a critical clock failure respectively. Figure 16D shows the physically impossible combination of the overall system being in the “PROCESS DATA” state while the computer processor is in the “FAIL CRIT” state.

Sequencing these snapshots in time without repeating any given snapshot will give a number of different scenarios (for example, ABCD, BD, DAC). Based upon the system behavior model, we know, however, that not all of these scenarios are legitimate. For example we know that the satellite is inaccessible for repairs when the overall system is in the failed state—snapshot C. As such, there is no way for the system to exit snapshot C. Therefore, any scenario in which C is not the last snapshot of the sequence (ABCD, CAB, DCA, etc.) is impossible and will be discarded. As another example, any sequence including snapshot D will also be discarded since, as indicated above, this snapshot is itself physically impossible. It is important to mention that although scenarios consist of the sequence of snapshots combined with the actions required to move from one snapshot to the next, this example shows that the sequence of snapshots alone can be enough to indicate that a given scenario is unrealistic and should be eliminated.

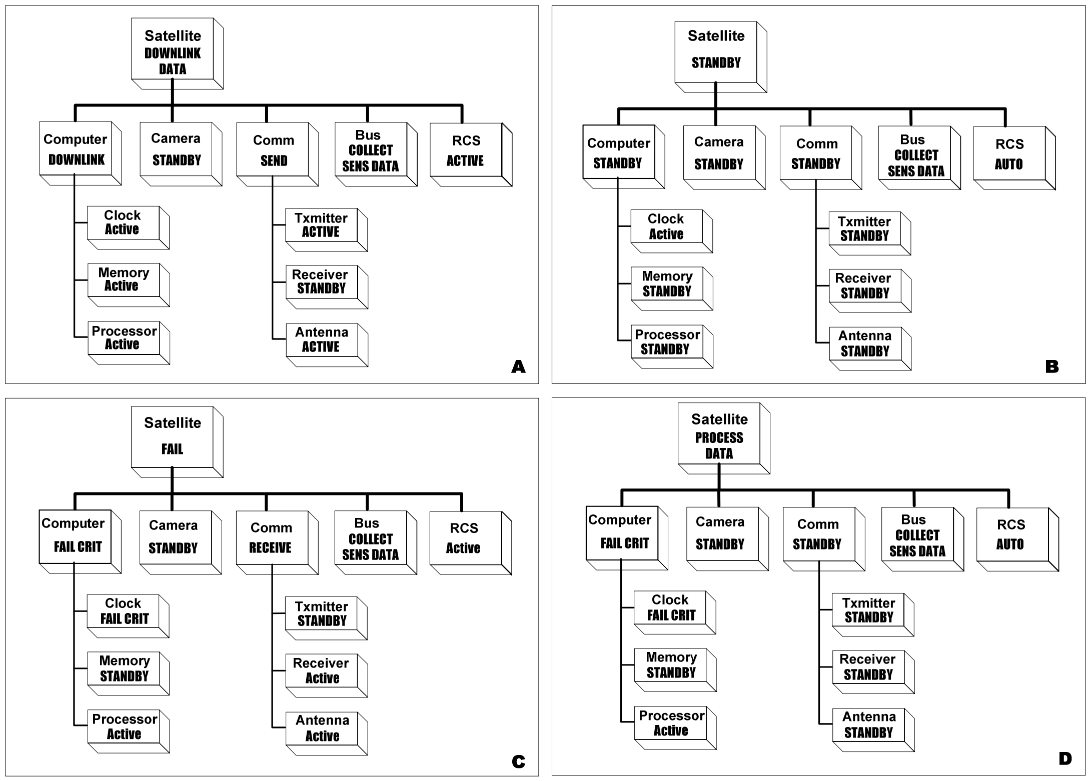


Figure 16: Four snapshots of the example system states

The scenario generation approach introduced here takes an active role in generating scenarios by assuming a complete knowledge about the relationship among the system elements and the behavior of each element, presented as states and functionalities that provide transitions between them. Since completeness of knowledge is a big assumption that might be far from reality, an updating algorithm introduced later, tries to come up with suggestions as to ways one can complete or repair the knowledge used for generating the scenarios when observations show that a scenario is missing from the plan or a scenario in the plan is impossible to generate.

5.4.2 Modeling: Behavior modeling

Behavior modeling for multi-state planning is designed in such a way as to follow the system design. It starts with defining the functionalities in a hierarchical manner. Then, the system's high level structure is defined which will correspond to the previously defined functionalities. Each structure element will be responsible for providing a group of functionalities. Next, the exact mapping between functionalities and structures are defined. Each structure on the structure tree will provide one or more functionalities that are at the same level of hierarchy as that structure. Then, the state transition diagram of each structure tree element is defined based on the states and functionalities mapped to that element. The last step is defining the transitions and what enables them by the states of the sub-elements and the conditions that might block them. These steps are explained in more detail below.

5.4.2.1 Functionality Tree

The first step in designing a system is almost always defining the functional requirements. Functional requirements specify the functionalities that the system is required to provide and how the system behaves to provide them. The functionality tree, therefore, shows how the system's functional requirements are provided by the subsystems and components and how these functionalities are divided among them. To enhance the functionality tree and make it more suitable for presenting the behavior of the system, actions and events that enable these functionalities are also added to the functionality tree. To sum up, the functionality tree elements are defined here as:

1. *Functionalities* are the system and subsystems' functional requirements. Functionalities are provided by subsystems and hence should be broken down to events or replaced by actions.
2. *Actions* are defined as the functionalities that are provided by the subsystems themselves and which, unlike other functionalities of a subsystem, do not break down to smaller elements.
3. *Events* are functionalities provided by components of the system. Like actions, they are basic elements of the functionality tree and are indivisible.

The functionality tree of the example EOS system is shown in Figure 17.

5.4.2.2 Structure tree

A structure tree shows the hierarchy of the system elements. Conceptually, a structure tree comes after the analysis of system functionalities and the clarification of how the functionalities are provided by different parts of the system. The system is usually supported by a number of components and subsystems. Subsystems themselves are comprised of additional groups of subsystems and components and so on. Figure 18 and Figure 19 show the structural tree and the states of the elements for the example EOS system.

The structure tree is expanded to include knowledge about the states of the system and its elements. The state of a system can, in general, be thought of as an optimal ensemble of system parameters which characterize it independent of its surroundings or its history. When the system is in a state, its history of getting to that state is irrelevant to the future of its behavior. The process of partitioning a system in to different states is more of an art than a science. However, the general guideline is to try to keep the number of states at a minimum without sacrificing the plan's ability to guide the system in an effective way.

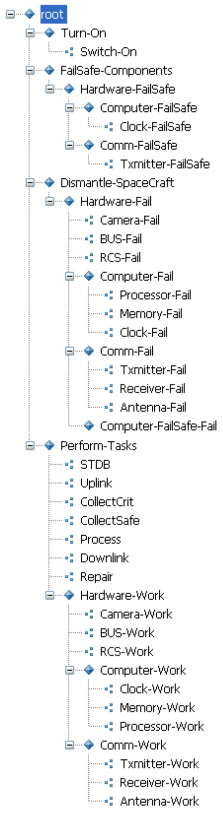


Figure 17: Functionality tree of the example EOS system

There are several small assumptions that are made in multi-state planning that need to be mentioned. The first assumption is that every element of the system has an initial state. This initial state can, in fact, be imaginary and only a starting point for connecting to the real initial states. The second assumption is that the system level states include all the end-states. So, the fate of the scenarios is practically defined by the state that the system ends up being in when the simulation is run fully to the end.

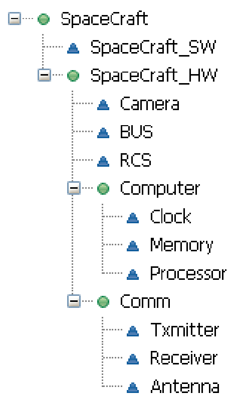


Figure 18: Structural tree of the example EOS system

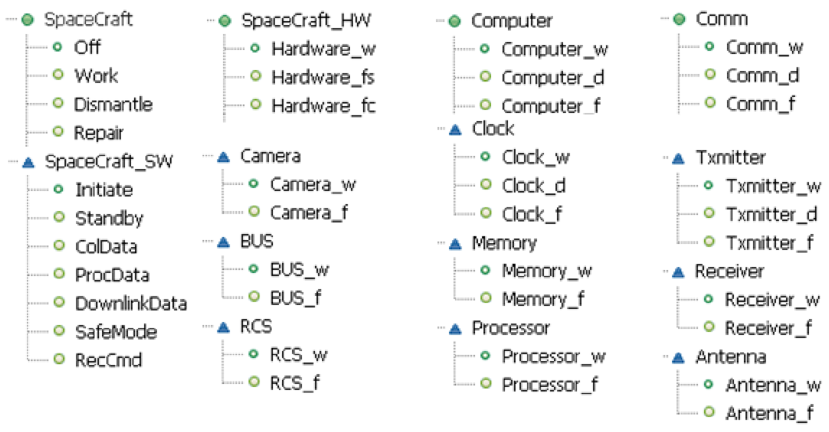
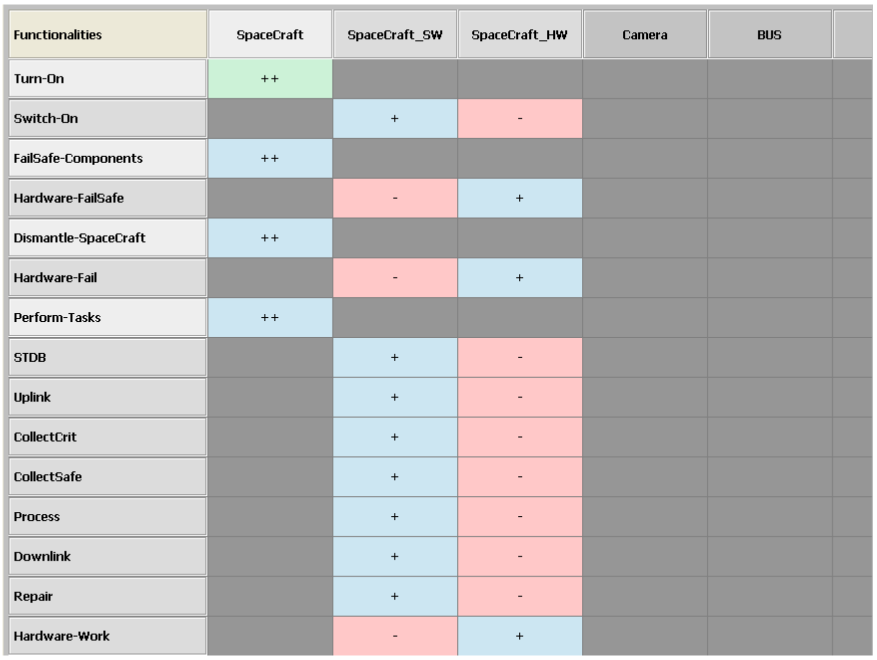


Figure 19: States of the elements for the example EOS system

5.4.2.3 *Mapping between functional and structural trees*

The mapping between the functionalities and structures specifies which component or components are providing a specific functionality. As mentioned earlier, events only map to components while actions and functionalities map to subsystems. This mapping provides a guideline for the system designer to make sure that all aspects of system behavior are modeled in response to the requested functionalities.



Functionalities	SpaceCraft	SpaceCraft_SW	SpaceCraft_HW	Camera	BUS	
Turn-On	++					
Switch-On		+	-			
FailSafe-Components	++					
Hardware-FailSafe		-	+			
Dismantle-SpaceCraft	++					
Hardware-Fail		-	+			
Perform-Tasks	++					
STDB		+	-			
Uplink		+	-			
CollectCrit		+	-			
CollectSafe		+	-			
Process		+	-			
Downlink		+	-			
Repair		+	-			
Hardware-Work		-	+			

Figure 20: A partial mapping between functionalities and structures for the example EOS system

Generally speaking, a good design is one in which each functionality is provided by only one physical element. A design in which every physical element of the design provides one and only one functionality is called an uncoupled design (Suh 2005). More coupling in the design can cause more complication, thus this map can also work as an indicator for exploring possible sources of complexity in the design.

Figure 20 shows a partial mapping between functionalities and structures of the example EOS system. A '+' or '++' sign show that there is a mapping between the functionality in the row and component in the column while a '-' sign is an indicator of no relationship between the functionality and component. A gray cell in the cross section means that because of the difference between the levels of component in its hierarchy with the level of functionality in its own hierarchy, there is no chance of a mapping between them.

5.4.2.4 State transition diagrams

In this step, information about the state transition graphs is obtained from the system modeler. Transition graphs show how each structure element provides the expected functionalities through the changes or lack of them in different functional states.

Considering state transition graphs, it is possible to define three types of states: *source* states are the states from which transitions only exit; *sink* states are states which transitions only enter and *transient* states are states that transitions can both enter or exit.

The sink states of the *system*'s state transition diagram *only*, are considered as the end-states. All of the scenarios will end up in one of these end-states. The state transition diagram should be a connected graph.

Figure 21 shows an example of a state transition diagram. States are shown as the blue vertices and transitions are depicted as directed lines connecting them. The names of the transitions are provided next to the transition lines. It is possible to have several transition lines connecting two states. It is also possible to see a transition name being used for several transitions. This means that there is the same underlying mechanism for providing those transitions.

5.4.2.5 Transition rules

Each of the transitions in the state transition diagram can be conditional on time, a landmark, or the state of another component of the system. If the condition is not met, it means that the state of that particular element cannot be changed by the transition that requires that condition.

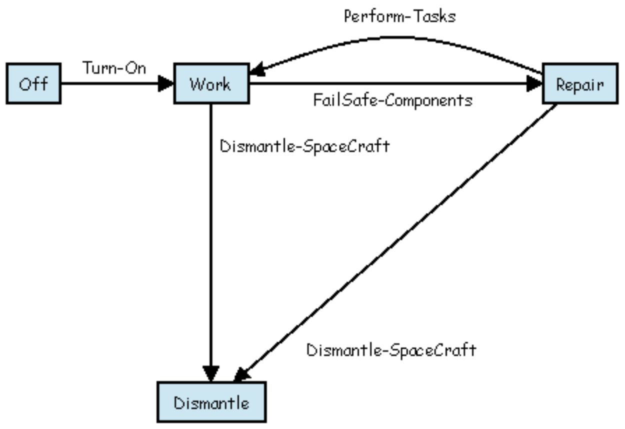


Figure 21: State transition diagram of EOS system

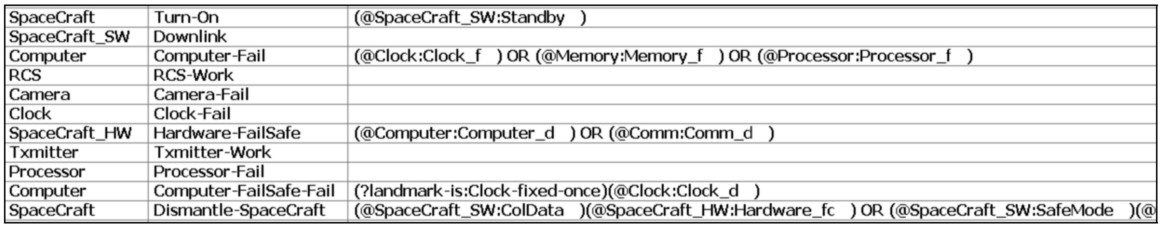
Each transition can also have duration or set a landmark. This helps the modeler to condition the transitions on the clock of the system or by the scenario's achievements to a certain point in the scenario.

However, the most important thing is to define how the states of the system and subsystems are changed according to the states of their sub elements. This is where the hierarchical planning is shaped. The planner uses this knowledge to understand how to change the state of sub elements to get the system and subsystems to a particular state. These transition rules are summarized in Table 2.

Figure 22 shows the details of some of the transitions used in the example EOS system. The first column in Figure 22 provides the name of the element, the second column gives the name of a transition for that element and the last column provides the details.

Table 2: Transition rules

Duration:	Each transition can have a specified duration. When the planner reaches this rule, it adds the value of the duration time to the time of the scenario. It's good practice to assign duration times for lower (rather than higher) level elements of the system.
Landmark:	Each transition can set a landmark. When the planner reaches this rule, it adds the landmark mentioned in the command to the scenario.
Time condition:	When the planner reaches a time condition, it compares the time of the scenario (which is the addition of the duration times in the scenario up to that point) with the time specified in the condition.
Landmark condition	When the planner evaluates a landmark condition, it searches in the scenario to see if the specified landmark has been set.
State condition	The planner checks the state of the element referred to in the condition to see if the condition holds or not.
Sub-goals:	This transition rule is only valid for the system and subsystem elements. The sub-elements of the system or subsystem and the state which can cause this transition are defined in this step. The planner considers these changes in the states to expand the plan to the lower level elements. The order of these transition rules are considered to be trivial to the process, however, the planner considers these transition rules in the order they appear. If the order can make a difference, another transition with the same transition rules but different order of elements can be added to the graph. The importance of each sub-goal is also set at this point.
Level control:	This is just a note for the scheduler to indicate whether there is a need to switch the level of detail during the simulation (see 3.2 for details)



SpaceCraft	Turn-On	(@SpaceCraft_SW:Standby)
SpaceCraft_SW	Downlink	
Computer	Computer-Fail	(@Clock:Clock_f) OR (@Memory:Memory_f) OR (@Processor:Processor_f)
RCS	RCS-Work	
Camera	Camera-Fail	
Clock	Clock-Fail	
SpaceCraft_HW	Hardware-FailSafe	(@Computer:Computer_d) OR (@Comm:Comm_d)
Txmitter	Txmitter-Work	
Processor	Processor-Fail	
Computer	Computer-FailSafe-Fail	(?landmark-is:Clock-fixed-once)(@Clock:Clock_d)
SpaceCraft	Dismantle-SpaceCraft	(@SpaceCraft_SW:ColData)(@SpaceCraft_HW:Hardware_fc) OR (@SpaceCraft_SW:SafeMode)(@

Figure 22: A number of transition rules for the EOS example

5.4.3 *Qualitative reasoning for developing risk scenarios*

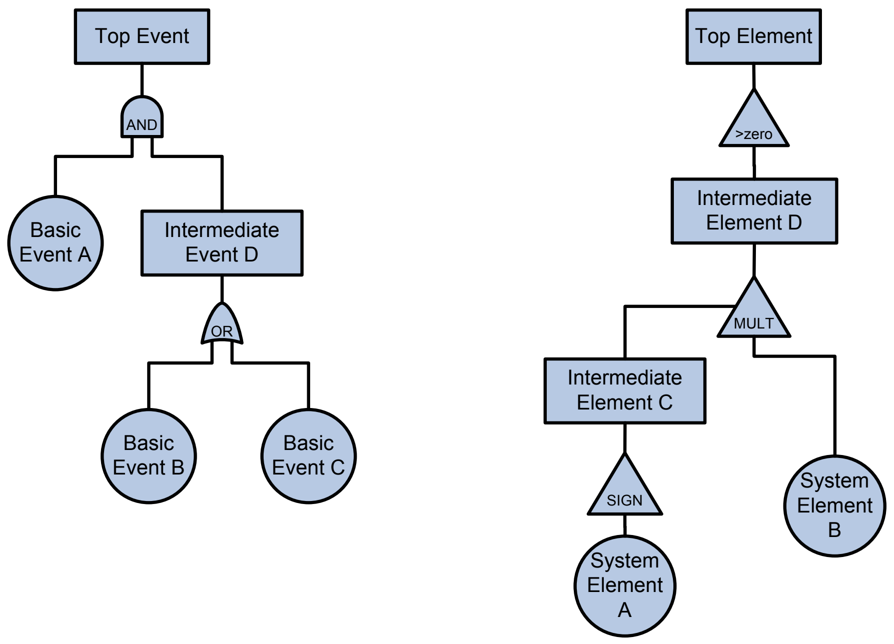
Using the data provided so far, the planner can generate all of the scenarios that can take the system from its initial state to the end-states. Some of these scenarios indicate the normal behavior of the system while others show how the system can deviate from its normal behavior and what might follow as a consequence. Risk scenarios are the ones that show how the system deviates from the normal behavior and these are the ones that are more interesting for the purposes of risk analysis.

To distinguish the risk scenarios from the pool of system behavior scenarios in a multi-state environment, a tree structure, similar to a fault tree is introduced here. Qualitative reasoning is used in 2 ways:

1. As an absorption point: If a scenario under development reaches a predefined group of system states, the scenario will be considered not interesting and will be abandoned from further development. This process is used when the scenarios are under construction and helps to increase the efficiency of the scenario development process.
2. As a filter: Among the pool of developed scenarios, a) scenarios that during their development put the system in a specific group of states or b) scenarios that contain a group of events in them, are selected and the rest are considered not interesting. This is a post scenario development process and chooses the scenarios with a group of initiating events or vulnerable states among them from the rest of the scenarios.

The modeling and reasoning process is very easy and straightforward. Like a fault tree with a top event, intermediate events, basic events and logic gates, a QR influence diagram has 4 main components: a top element, QR- elements, system elements and QR-Gates. Like fault trees in which events are connected to one other by logic gates, elements in a QR influence diagram are connected by QR-Gates.

However, unlike logic-gates that need at least 2 events to connect to another event, QR-Gates can connect even one element to another element.



Basic Event A: True, False
Basic Event B: True, False
Basic Event C: True, False
Intermediate Event D: True, False
Top Event: True, False

System Element A: minf, low, zero, high, pinf
System Element B: zero, 0-1, one
Intermediate element C: neg, zero, pos
Intermediate element D: neg, zero, pos
Top Event: True, False

Figure 23: Comparison between fault tree on the left and QR influence diagram on the right

The reasoning process for a QR influence diagram starts from the basic elements. Since the system is considered to be fully observable and static, the state of every element is known at any time during the development of scenarios. Considering the lowest level QR-Gates on the QR influence diagram, since all their input elements are system elements, it is possible to find the state of their output element. This process will then continue with the next level of gates until the fate of the top element is determined.

For example, let’s consider the QR influence diagram in Figure 23 and assume that System Element A is in *low* state and System Element B is in *zero* state. The SIGN-Gate says that Intermediate Element C is in *neg* state since *low* is on the left side of the System Element A’s *zero* state. Now considering the MULT-Gate, the Intermediate Element D can be solved once the state of both of its inputs is known. Since Intermediate Element C is in *neg* state and System Element B is in *zero* state, Intermediate Element D will be in *zero* state as well. Finally, the state of the top element can be found to be *False* because Intermediate Element C is on the left side of the *zero* state.

The list of QR-Gates can be very large. Below, 18 gates are listed and the logic behind them is described. More gates can be added to specific domains for domain-specific modeling needs.

1. **`M+`** (Positive Monotonic): Two variables are positively related by some unknown monotonic function when one of them changes value in some direction if and only if the other changes in the same direction
2. **`M-`** (Negative Monotonic): Two variables are negatively related by some unknown monotonic function when one of them changes value in some direction if and only if the other changes in the opposite direction
3. **`S+`** (Positive Saturation): Like M+ but when one of the variables reaches a value, it won’t go higher than the value for the other variable.
4. **`S-`** (Negative Saturation): Like M+ but when one of the variables reaches a value, it won’t go lower than that value for the other variable.

5. **<?** (Smaller than): A logical gate which is true if the state of the input variable is before the question-mark (?) state and false otherwise.
6. **<=?** (Smaller than): A logical gate which is true if the state of the input variable is not after the question-mark (?) state and false otherwise.
7. **>?** (Smaller than): A logical gate which is true if the state of the input variable is after the question-mark (?) state and false otherwise.
8. **>=?** (Smaller than): A logical gate which is true if the state of the input variable is not before the question-mark (?) state and false otherwise.
9. **=?** (Smaller than): A logical gate which is true if the state of the input variable is the same as the question-mark (?) state and false otherwise.
10. **!=?** (Smaller than): A logical gate which is true if the state of the input variable is not the same as the question-mark (?) state and false otherwise.
11. **AND** (And)**:** A logical gate which is true if all the input variables are true
12. **OR** (Or): A logical gate which is true if at least one of the input variables is true.
13. **SIGN**: Looks for zero in the output states. States below zero will be mapped to negative state (neg) and states above it will be mapped to positive state (pos).
14. **MULT** (Multiplication): Makes a new variable such that its values are the results of multiplication of the values of the input variables

The QR-Gates listed above assign one state to the output element. These gates might be considered to be accurate gates. However, some QR-Gates (listed below as numbers 15 through 18) might only determine the possibility of the range of the output values. These gates might be very useful for modeling purposes and show the best prediction for the risk scenario models; however, it is important to mention that using these gates might introduce erroneous scenarios into the plan. This might be totally fine for the first rounds of simulation since the updating process will later find these scenarios and help the designer to take these scenarios out of the model.

15. **INT** (Integral): by considering the rate of change in the input's value, it provides a range of possible values for the output variable.
16. **ADD** (Addition): Makes a new variable such that its values are the result of adding the values of the other variables
17. **D/DT+** (Positive Derivative): Direction and the rate of change in the values of a variable are positively correlated to the value of the other variable.
18. **D/DT-** (Negative Derivative): Direction and the rate of change in the values of a variable are negatively correlated to the value of the other variable.

Gate name	Number of Inputs	Input Size	Output Size	Fixed Input States	Fixed Output States	Output type
M+	1	m	n=m	-	-	Single value
M-	1	m	n=m	-	-	Single value
S+	1	m	n<m	-	-	Single value
S-	1	m	n<m	-	-	Single value
<?	1	m	2	-	{true, false}	Single value
<=?	1	m	2	-	{true, false}	Single value
=?	1	m	2	-	{true, false}	Single value
!=?	1	m	2	-	{true, false}	Single value
>?	1	m	2	-	{true, false}	Single value
>=?	1	m	2	-	{true, false}	Single value
AND	n	2	2	n{true, false}	{true, false}	Single value
OR	n	2	2	n{true, false}	{true, false}	Single value
SIGN	1	m	3	{…, zero, …}	{neg, zero, pos}	Single value
INT	2	m, 3	n=m	-,{neg,zero,pos}	-	Range of values
ADD	2	m, m	n=m	-	-	Range of values
MULT	2	m, 3	n=m	{…, zero, …}, {…, zero, …}	{neg, zero, pos}	Range of values
D/DT+	1	3	3	{neg, zero, pos}	{neg, zero, pos}	Range of values
D/DT-	1	3	3	{neg, zero, pos}	{neg, zero, pos}	Range of values

Table 3: QR-Gates and their input/output variables

Table 3 shows the above mentioned QR-Gates and the possible input/output elements. A '-' indicates that there is no restriction on the format of the states.

5.4.4 Generating risk scenarios

Risk scenarios are generated by the planner following the same basic concept presented theoretically in 5.4.1. The planner begins by defining the first snapshot of the system which consists of the initial state of the system, subsystems and components. Next, one of the immediate possible states of the system that can be reached from the system's initial state will be considered. If all of the conditions for reaching this state are met, then the new state as well as the actions that can take the system from the initial state to this state will be recorded. If the conditions for reaching this state are not met, the planner will consider the next immediate possible state of the system. This process continues until all of the immediate possible states branching from the initial state of the system have been examined. The same process is repeated with the consideration of the next immediate reachable states from the state the system is at. The planner repeats the process until the scenario falls into a state for which there are no immediately following states (end-states) or until the conditions for continuing from that state are not available. In cases where the system's FSM falls into a loop, the planner will limit the number of the revisits of a state so that it can generate a finite number of qualitatively distinct scenarios.

After choosing the next state of the system, the planner will define the states of the subsystems that need to change to make the transition to that next state possible based on the details of the transition that takes the system to the new state. This process will continue until all of the new states of the subsystems and components are defined and the actions that take them to the new states are determined. For the subsystems and components whose states are not changed, the state would be the same as it was before.

An example will make this process more clear. Figure 24.A shows the initial states of the system, subsystems and components of our satellite example. Let's assume the only available state for the system to go to from this initial state is the Receive Command state. According to Figure 24.B, for the system to be in this state, RCS must be ACTIVE, Computer must be in the UPLINK state and Communications must be in the RECEIVE mode. Computer's UPLINK mode needs the processor to become ACTIVE while the Communications change to the RECEIVE mode requires that the Receiver and Antenna be in the ACTIVE mode. Finally, for the system to make all these transitions the following actions have to be taken: `RCS_Active; Proc_A; Rec_A; Ant_A`

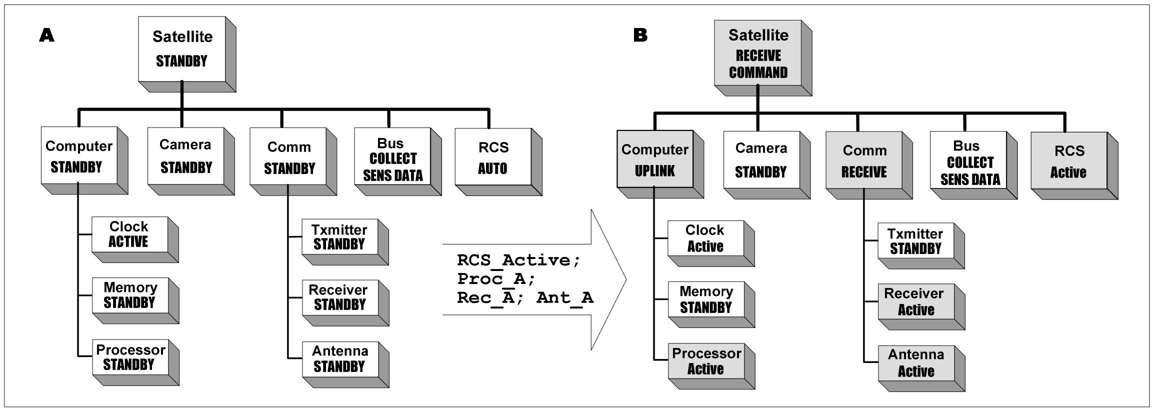


Figure 24: Snapshots of the example satellite system in A) Inital state and B) Next available state

The planner follows a similar process of identifying the changes required to move from the RECEIVE COMMAND state to the next available states, (which in this case would include FAIL, SAFE_MODE or COLLECT_DATA). Of course, since the satellite system is working in a looped process, there must be a limit to the number of revisits to each state of the system to guarantee the generation of a finite number of qualitatively distinct scenarios.

The planning process is done from top to bottom recursively. The planning goal that is taking the system from the initial-state to all (or sometimes a group of) end-states is considered first. All of the transition paths that take the system to its goals are determined. These paths will include all of the states and transitions that change these states to finally reach the goal state(s). The reason that state changes are included is the delay in the system response to the transitions and events during the simulation; the transitions and events that follow should only be called upon when the local states are reached. In other words, the presence of states works as a benchmark for the start of the next transition or event. Figure 25 depicts a plan generated from a generic state transition graph.

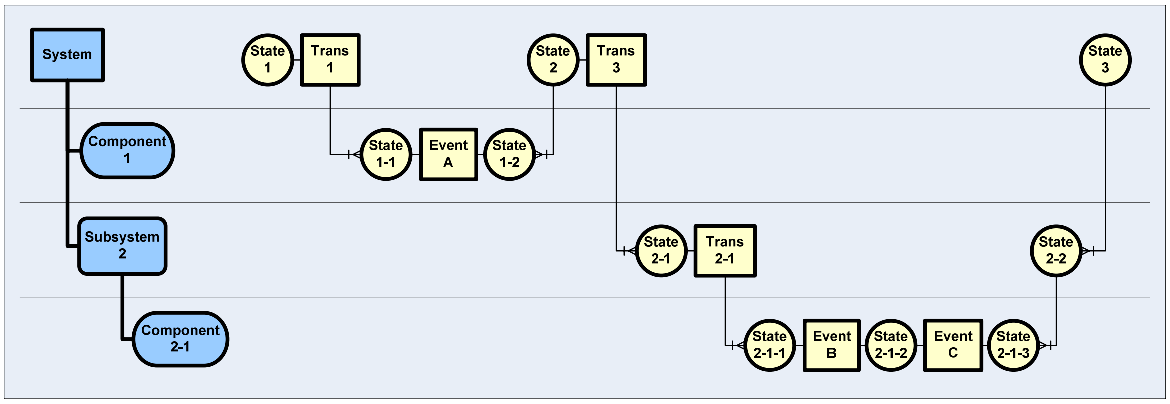


Figure 25: A high level scenario generated from the system hierarchy, state transitions and transition details

Obviously, for a transition to be considered in the plan, all of the preconditions of that transition need to be satisfied. Transitions that indicate the need for a multi-level planning process or the need to satisfy a sub-goal trigger the recursive planning process which in return will give all of the transitions and events that satisfy the sub-goals in the lower levels of the system.

Using set-theoretic presentation, inputs to the multi-state planner include the system hierarchy and the states of its elements, state transitions (transition name, sub-goals and constraints), qualitative reasoning logic and the goal state of the system. A SimPRA multi-state domain is a pair $D =< T, \gamma >$and a SimPRA planning problem is a four-tuple $P =< S, g, T, \gamma >$where:

- S is the system's hierarchical structure with (initial)states of the elements and *g* is the goal state for the system.
- $T =< W, E >$ is the hierarchical network which is composed of events *E* and functionalities *W* where in each functionality $w =< f, TR < C, SG_{el}[] >>$, *f* is the functionality name and TR is the transition rule. A transition rule consists of constraints *C* and sub-goals SG—for sub-elements *el*—in order.
- $s' = \gamma(Elem, t, s)$ where $t \in T$ and $\gamma()$ is the transformation function that transforms the state of the Element *Elem* from *s* to *s'* in response to transition *t.*

Figure 26 provides the planner's algorithm. At the start of the planning process, the root of the structure hierarchy, which is the system itself, is passed to the Element-Plan() function to generate a plan just for that level (as detailed in Figure 27).

```
SimPRA-Multi-State-Plan ( $S, g, T, \gamma$ )
    Elem = root(S)  'returns the System
    $\Pi[]$ = Element-Plan ( $Elem, initState(Elem), g, T, \gamma, ø, ø$ )
    if $\Pi[] = ø$
        return ø
    else
        if Elem has sub-elements
            $Sol[] = \{ p \mid c = subElem(Elem), p = decomp(c, \Pi[]) \}$
    return QR(Sol[])
```

Figure 26: High level planning process algorithm for SimPRA multi-state planner

```
Element-Plan ( $Elem, s, g, T, \gamma, Sol[], \Pi[]$ )
    s = g
        then return $Sol[]$
    else
        $Sol[] = Sol[] \cap \{ \Pi[] \oplus t \mid t \in T, \gamma\ (Elem, t_{last}(\Pi[]), s_0) = g \}$
        $P[] = \{ p \oplus t \mid t \in T, p \in \Pi, \gamma(Elem, t_{last}(p), s_0) = s, s \neq g, revisit(s) < n \}$
        $\Pi[] = \{ \pi \mid \pi \in P, execute(\pi) \equiv True \}$
        if $\Pi[] = ø$ then
                return $Sol[]$
        else
                return Element-Plan ( $s_0, g, T, \gamma, Sol[], \Pi[]$ )
```

Figure 27: The algorithm for Element-Plan function

```
decomp ( $c, \Pi[]$ )' c is the Element and $\Pi[]$ is the plan that is not decomposed
        select each line $\pi \in \Pi[]$
            for each $t \in \pi$
                if there is a $TR < c, SG_{el}[] >$ where $sg \in SG_{el}[]$ is a sub-goal for c
                        $execute(\pi)$ up to sg to get s the latest state of element c
                        replace sub-goal with Element-Plan ( $c, s, g, T, \gamma, ø, ø$ )
        if Elem has sub-elements
            $\Pi[] = \{ p \mid el = subElem(c), p = decomp(el, \Pi[]) \}$
        return $\Pi[]$
```

Figure 28: The algorithm for decomp function

As in binary planner, considering $Sol[] = (t_1, t_2, ..., t_n)$ then $Sol[] \oplus t^* = (t_1, t_2, ..., t_n, t^*)$ and $t_{last}(Sol[]) = t_n$. The result of that planning is a set of all the paths from the initial state to the goal state of the system in the format of ordered functionalities. Functionalities have transition rules that include constraints and sub-goals. Constraints are checked when executed during the Element-Plan() function. Sub-goals define a planning process for the sub-elements of the system. Planning for sub-goals is handled by the decomp() function (algorithm presented in Figure 28). The final plan is returned after the plan has gone through the qualitative reasoning process (explained in section 5.4.3) for filtering reasons.

5.4.5 Generalized ESD generation algorithm

High level scenarios that are generated in the form of sequences of events, changes in the states, and conditions are grouped into the format of Generalized Event Sequence Diagrams (ESD). An Event Sequence Diagram (ESD) is basically a flowchart of events where each path leading to an end-state is a scenario (Stamatelatos, 2002). The Generalized ESDs generated by the SimPRA planner incorporate condition elements and state changes in addition to events. Figure 29 presents one such GESD with events and functionalities that are represented by rectangles, condition elements that are represented by diamonds, and state changes that are represented by ovals.

The ESD generation algorithm uses the high level scenarios it receives from the planner to generate a tree data structure of events. Figure 30 presents the ESD generation algorithm which uses the high level scenarios it receives from the planner to generate the tree structure of the generalized ESD.

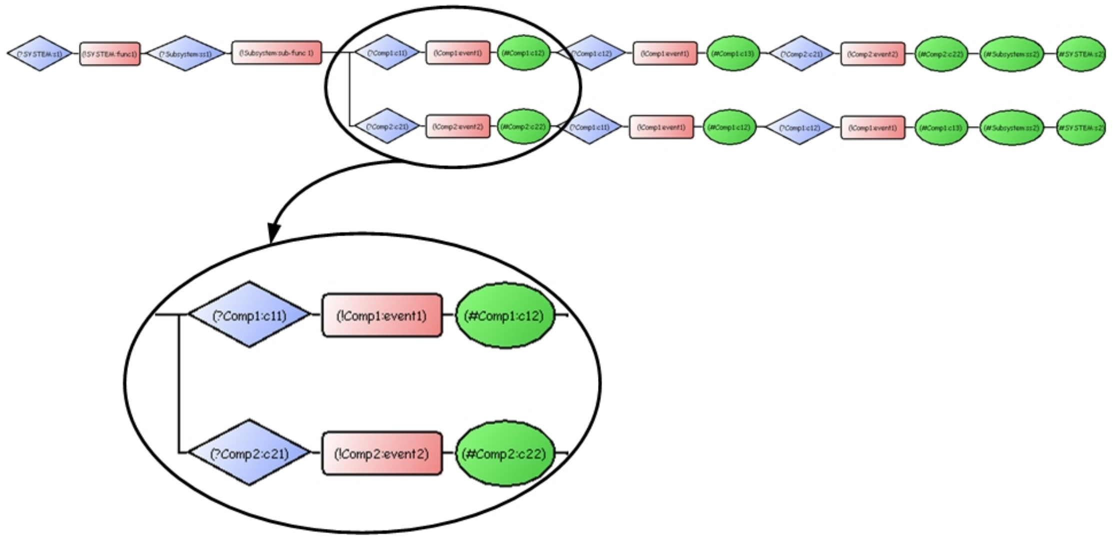


Figure 29: A generalized ESD example

To explain this algorithm better, let's consider the following sequences of elements (events, conditions, and/or state changes):

- A, B, D
- A, B, E, F, G
- A, C, D, E
- A, B, C, E

According to the algorithm, L= 5 which is the length of the 2nd sequence. For each 1 to L location, the unique elements are defined:

1. {A}
2. {B, C}
3. {C,D,E}
4. {E,F}
5. {G}

Now the tree will be built in L layers from the top, adding one layer at a time (see Figure 31):

1st layer: A

Adding the 2nd layer, B and C will be added to A

Adding the 3rd layer, C and E will be added to branch B only, while one D is added to B and one to C.

Adding the 4th layer, E is added to C and D (on the 4th row of Figure 31), while F is only added to E.

Adding the 5th layer, G is only added to F.

```
L← number of events and conditions in the longest scenario
From 1st to Lth location in each scenario {
        Find the list of all unique events or conditions in the nth location of all scenarios if exists
        Add each of them to a leaf on the tree that shares the same sequence of events in their path
}
Merge the branches that have only one child with their parent node
```

Figure 30: Generalized ESD generation algorithm

The following graph shows the generated tree for the above example. The next step would be to change the visual representation of the elements in the graph to reflect differences between events, conditions, and state changes.

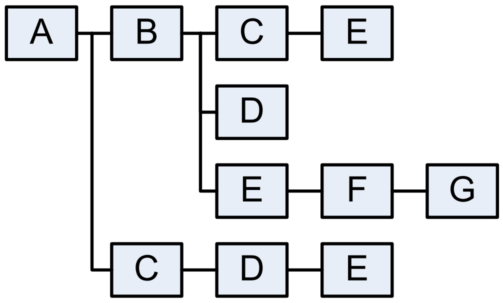


Figure 31: An event sequence diagram generated by the GESD algorithm

5.4.6 *Plan updating*

Updating starts when a predefined number of simulation-runs are completed. The details of the scenarios generated by the simulator are used for two purposes: scenario updating and level-control updating.

For the planner, the updating engine checks every instance of a state change in the detailed scenarios for each element of the simulation. For components, if there is an event related to that component that is called between the changes of state, that event will be considered as the cause of the state transition for that component. If there is no event between state changes, then the previous event will be considered as the source of change in the state of the component although that event is not recalled again in the simulation. Since components are the lowest level elements in the structure tree, there is no other element that can affect or describe their behavior. For subsystems, the last state of each of their subsystems or components is considered as the cause of the change in their states. Figure 32 shows an example line of the simulation log as the input to the updater in the left column and the updating algorithm's output in the right column.

Simulation Log		**Updater Output**
#System:s1 > #Subsystem1:ss1 > #Comp1:c11 > #Comp2:c21 > !Comp1:event1 > !Comp2:event2 > #Comp2:c22 > #Comp1:c12 > #Comp1:c13 >#Subsystem1:ss2 > #System:s2	$\longrightarrow$	1: #Comp1:c11--> !Comp1:event1--> #Comp1:c12 2: #Comp1:c12--> !Comp1:event1--> #Comp1:c13 3: #Comp2:c21--> !Comp2:event2--> #Comp2:c22 4: #Subsystem1:ss1--> @Comp1:c13 AND @Comp2:c22 --> #Subsystem1:ss2 5: #System:s1--> @Subsystem1:ss2 --> #System:s2

Figure 32: An example of the updating algorithm output

In this example, the simulator logs the changes of states of the system, subsystems, and components by putting a '#' sign on the left. It also defines the generation of the events by a '!' sign. Events in the simulation log happen chronologically from left to right. The simulator also records the initial states of the simulation elements at the beginning of each realization.

The logic behind each line of the updating algorithm is easy to follow based upon the rules defined above. Lines 1 through 3 of the updating algorithm's output on the right side are generated as the updater goes searching through the simulation log for changes at the component level. Component 1 is observed to have changed from c11 to c12. When searching for the reason for this change, the updater observes the occurrence of event 1. Therefore, the updater output generates the transition `#Comp1:c11--> !Comp1:event1--> #Comp1:c12` which translates to: component one at state c11 experiences event 1 which moves component one to state c12. The next change identified by the updater in the simulation log is that component one goes from state c12 to state c13. The updater searches for an event that may have occurred between these two different states. When an event is not found, the updater assumes that the state change must be related to the previous event. Line 2, therefore, indicates that the change in component one from state c12 to state c13 is due to event 1. Line 3 of the updater output, like Line 1, reflects the simulation log's record of a change in component 2 from state c21 to state c22 as a result of experiencing event 2. Having identified all state changes at the component level, the updater turns to any state transitions at the subsystem and system level as shown in Lines 4 and 5 of the updater output. Instead of events, however, the states of components and/or subsystems are assumed to be responsible for observed transitions at these levels. Figure 33 shows the algorithm for the updating process and Figure 34 shows the potential benefits of the algorithm for the planner.

```
Read the simulation log file
For each line of the log file {
  For each structure element {
    If element is a component {
        Find the first two instances of state change of the element
        Find the only corresponding event that comes between the states
        Add the transition to the potential transitions list
        While there are more of the element's state changes in the log line {
          If there is an event in between
            Add the transition to the potential transitions list
          Else
            Add previous event and state changes to the potential transitions list
        }
    }
    Else 'if element is a subsystem {
        Find the sub-elements from the structure tree
        While there is a state change for the subsystem in the log line{
          Find the instances of state change of the element
          Find the state of sub-elements at the end of the change
          Add the transition to the potential transitions list
        }
    }
  }
}
Compare the potential transitions list with the ones in the plan
If a transition is missing list it as the output
```

Figure 33: Updating algorithm

Transitions generated by the updating algorithms are compared with the state transitions of the system elements in the system model of the planner, and missing transitions will be suggested for addition while transitions not present in the updating data will be suggested for elimination. There are obviously other constraints on the transitions that the updating algorithm is not capable of determining.

The second purpose of updating is to adjust the level-control information in the plan. The post-simulation analysis uses a level control measure, which is a combination of Shannon's entropy measure (Lindley, 1956; Shannon, 1948; Shannon and Weaver, 1949) and a risk measure.

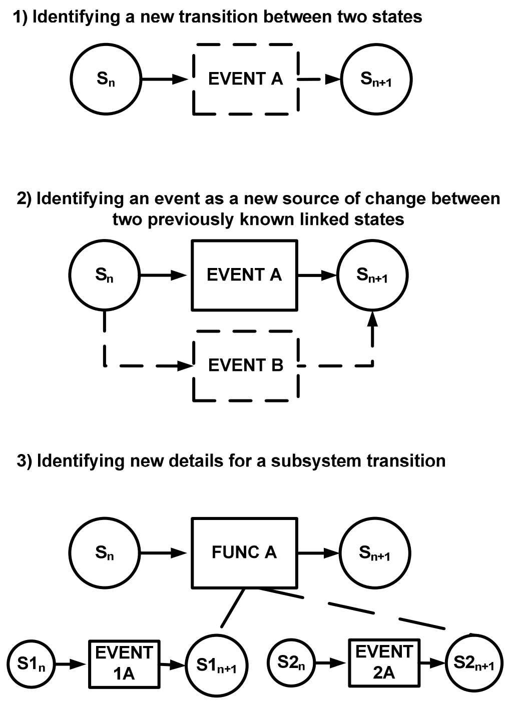

1) Identifying a new transition between two states
S_n
EVENT A
S_{n+1}
2) Identifying an event as a new source of change between two previously known linked states
S_n
EVENT A
S_{n+1}
EVENT B
3) Identifying new details for a subsystem transition
S_n
FUNC A
S_{n+1}
S1_n
EVENT 1A
S1_{n+1}
S2_n
EVENT 2A
S2_{n+1}

Figure 34: The potential benefits of the updating algorithm for the planner

The Shannon entropy value is calculated for each multi-level object in the plan. The risk measure can be either:

1. The conditional probability for the end-state of interest after the multi-level object node. This measure focuses on the consequence of the system when the multi-level objects fail. The conditional probability for the end-state of interest is a value ranging from 0 to 1. A large value indicates that the failure of this multi-level object has a large probability of leading the simulation to the end-state of interest.

2. The importance measure of the multi-level objects in the system. This measure focuses on the importance of the multi-level objects to the end-state of interest.

Birnbaum introduced the concept of importance measure (Birnbaum, 1969). The measure of component *i* is described as:

$$I_i^B = \frac{\partial R_s[r_i(t)]}{\partial r_i(t)}$$

Where

I_i^B = Birnbaum importance of component *i*

$R_s[r_i(t)]$ = Reliability of the system as a function of the reliability of the individual component

$r_i(t)$ = Reliability of component *i*

In the case of simulation, $R_s[r_i(t)]$ is replaced by the probability of the end-state of interest $P_e[r_i(t)]$. $r_i(t)$ represents the probability of failure of the multi-level component *i*.

In the case that the multi-level object only has two different states: fail or success, the Birnbaum importance measure can be calculated as:

$$I_i^B = P_e[r_i(t)=1] - P_e[r_i(t)=0]$$

A large value of the importance measure indicates that the status of this component dramatically influences the end-state of interest. Thus the multi-level objects should be further decomposed to get more accurate results. A small value of the importance measure indicates that the status of this component does not influence the end-state of interest. Thus the multi-level object should be simulated at a relatively high level.

The threshold for the level control measure is defined by the analyst before the simulation begins. It can be further updated after each round of simulation. After each round of simulation, the level control measure for each multi-level object is calculated and compared with the level information in the previous LCN. If the value of the level control measure indicates that the simulation level should be adjusted, the scheduler submits a request to the multi-level objects to check if any updates are available. If no further updates are available for the multi-level objects, the plan is treated as the best plan available. The simulation results are calculated based on this plan.

5.4.7 Good practices

During the process of modeling several systems for plan generation (Hu et al., 2006; Mosleh et al., 2004; Zhu et al., 2005), the following lessons have been learned that make the modeling process easier and smoother:

- It is better to keep details in the component level and make the subsystem level elements as light as possible. This is a very important concept since it can make the modifications to the model much easier and make the planner more expandable.
- If there seems to be a component that belongs to several subsystems, it is likely that the component should be considered as an independent subsystem. An example of this is the power system which usually provides service to most of the subsystems. There is a tendency to consider it as an

element of all subsystems, but in fact, it is better to be considered as an independent subsystem.

- It is better to keep the model as similar to the real system as possible instead of making assumptions to reduce the size of the model. For example, if the real system has 4 layers of subsystems to get to the components, it is better to keep the model the same instead of merging some of the subsystems into each other.

6 Evaluation and comparison with other techniques

In this chapter, the SimPRA planner is evaluated at three different levels.

1- *Within the context of the SimPRA approach as a package:* The PSAM8 benchmark problem has been proposed by NASA and worked on by a number of research teams using different risk assessment approaches and techniques. The capabilities of SimPRA are highlighted through a detailed comparison of its solution to this problem with seven alternate approaches.

2- *As an element within the SimPRA approach:* A holdup tank problem is used to evaluate the importance of the planner in guiding the simulation and generating risk scenarios.

3- *As a stand-alone tool:* The planner is a risk scenario generation tool that captures different types of engineering knowledge with which it automatically generates risk scenarios in the format of Generalized Event Sequence Diagrams (ESDs). This capability is evaluated in LRO satellite risk assessment study. The study uses the SimPRA planner to generate an event sequence diagram of satellite activities during its mission and compares the results with a more traditional FT/ET approach.

6.1 PSAM8 benchmark problem

The proposed benchmark problem consists of an ion propulsion system used in a science mission to the outer solar system. During different mission phases, the propulsion system is turned on and thrust is continually provided until the specified operating time expires. The propulsion system has 5 thruster assemblies and a single propellant tank. In the first phase of the mission only 2 assemblies are needed while in the rest of the phases, 3 assemblies are required for thrust generation. Each assembly has 1 propulsion power unit (PPU) and 2 ion engines. Each engine has a valve that opens and closes when the engine is turned on and off respectively. Standby assemblies and ion engines remain in standby until they are needed to replace a failed unit. In this example, there is also the assumption of Common Cause Failure (CCF) among assemblies and elements that are left to the designer's discretion. The goal of the study was to quantify the time-dependent reliability of the propulsion system over the planned mission considering the mission and design descriptions. For more information about this problem please refer to Appendix B: PSAM8 benchmark problem.

6.1.1 Comparison

Eight groups of researchers worked on the proposed problem and reported the results of their activities in the form of research papers to the PSAM8 conference in 2006. Table 4 lists the papers, their authors and the method used for solving the problem. For more information about the approaches and results please refer to the PSAM8 conference proceedings. It should be noted that some of the approaches, specifically the simulation ones, do not necessarily represent a tool. This makes the capabilities of these approaches limited to what has been done for the benchmark problem and doesn't necessarily show anything in particular about the approach itself.

To compare the suggested approaches to solving the benchmark problem in a more objective way, the following criteria are selected:

1. Exact/ Approximate solution: Is the solution to the problem found by an exact approach or an approximate approach? (Not considering the other approximations made)

Table 4: Papers submitted to PSAM8 conference on the benchmark problem and their authors

Paper #	Title	Authors	Name
67	Direct Monte Carlo Simulation for the Reliability Assessment of a Space Propulsion System Phased Mission	(Zio and Librizzi, 2006)	Direct Monte Carlo Simulation (MC)
111	Advanced PRA Tool Benchmark for Space System Risk Using the Dynamic Flowgraph Methodology	(Olivia and Yau, 2006)	Dynamic Flowgraph Methodology (DFM)
188	A Dynamic Fault Tree of a Propulsion System	(Xu et al., 2006)	Dynamic Fault Tree (DFT)
202	An Intelligent Agent-Oriented Approach to Risk Analysis of Complex Dynamic Systems with Applications in Planetary Missions	(Azarkhail and Modarres, 2006)	Agent-Oriented Monte Carlo Simulation (AO-MC)
252	Space Propulsion System Phased-Mission Probability Analysis Using Conventional PRA Methods	(Knudsen and Smith, 2006)	SAPHIRE
318	An Event Tree/ Fault Tree/ Embedded Markov Model Approach for the PSAM-8 Benchmark Problem Concerning a Phased Mission Space Propulsion System	(Mandelli et al., 2006)	FT/ET/Markov
345	Solution of Phased-Mission Benchmark Problem Using the SimPRA DPRA Methodology	(Hu et al., 2006)	SimPRA
425	Mission Reliability Evaluation for a Space Propulsion System Phased-Mission Benchmark Problem	(Clark et al., 2006)	Discrete Event Simulation (DES TIGER)

2. Binary/multi state: Is the end-state binary or is the consequence multi-stated?
3. Expandable: Is it easy to expand the model or hard?
4. Low Probability High Consequence Scenarios: Does the model consider the low probability high consequence scenarios?
5. Applicable to large systems: Is it feasible to apply the model to large and complex systems or is it only appropriate for small and simple ones?
6. Common Cause: Is it easy to consider common cause in the model or not?
7. Demand/life: Are failures only demand based, time based or both?

8. Model Complexity: Are there any assumptions in the modeling that will affect the outcome of the risk assessment?
9. Problem solved: Was the benchmark problem solved with this approach?

6.1.1.1 *Exact /approximate solution*

Simulation approaches are nondeterministic and therefore provide approximate solutions whose accuracy and precision increase with increased numbers of simulation runs. Truly exact solutions can usually only be found for simple problems. The complicated math associated with complex problems generally takes too much time or is simply impossible to address without resorting to the use of simulation methods at some level of the solution.

As a simulation-based approach, SimPRA provided an approximate solution to the PSAM8 benchmark problem. Based upon this criterion, SimPRA ranks similarly to other simulation-based approaches.

6.1.1.2 *Binary /multi-state outcome*

The analysis provided by some risk assessment tools are limited in that they only consider two possible end-states—success and failure—depending upon whether the nominal path is taken. In contrast, SimPRA and other multi-state solutions consider a variety of outcomes, different levels of degradation, and different levels of severity. Clearly, multi-state solutions are preferable in that they are more realistic and allow for the consideration of a range of outcomes.

6.1.1.3 *Expandability*

Expandability refers to the ease with which additional information can be incorporated into the model of the system. Horizontal expandability allows for the addition of elements to the system. Vertical expandability allows for the addition of information about those elements. This allows the model to very easily become richer in detail and fidelity to the real world as information becomes available and/or the design process progresses.

One of the greatest advantages of the SimPRA approach stems from the hierarchical nature of its structure which allows for expandability in both directions in a manner that is easy and intuitive for the user due to its similarity to common engineering approaches to design and modeling.

6.1.1.4 Low probability/high consequence scenarios

The risk associated with an event is a combination of its probability of occurrence and the consequences stemming from its occurrence. One very important but often overlooked class of scenarios is those that have a very low likelihood of occurrence but extremely serious consequences. Many risk assessment approaches basically ignore this class of scenarios because they generate and screen scenarios based only on their probability of occurrence.

SimPRA's planner provides guidance on the evolution of scenarios in a way that brings attention to the existence of a high-consequence event on a low-probability branch of events. As such, SimPRA will continue generating scenarios along that branch to identify and explicitly consider the high-risk scenarios it contains.

6.1.1.5 Complex systems

Two different factors contribute to the complexity of a system. The first is the size of the system or the number of different components within it. The second is the dynamics of the system meaning the extent to which behavior changes over time. SimPRA is designed to handle both aspects of complexity. Its hierarchical structure allows it to easily organize and incorporate a large number of components into the model. The use of rules on transitions, timing conditions, and conditionality on the states of other systems and components allows it to capture the dynamics of a system with high fidelity to the real world.

6.1.1.6 Common cause failures

Common cause failures are those simultaneous failures of a number of different components due to a single cause. Modeling common cause failures can be extremely complicated because of the difficulty of tracking scenarios and identifying their consequences when several events are occurring at the same time. SimPRA simplifies this process by providing predefined block sets that allow users to easily model common cause failures. Please refer to Hu 2005 for additional information on how SimPRA can be used to model common cause failures.

6.1.1.7 Demand/life

In risk assessment studies, failure events that occur when a component's operation is demanded are inherently different from those that occur during the time that the component is in operation. Indeed, demand failures are usually due to entirely different physical processes and mechanisms than life failures. Incorporating and differentiating between these two different types of events is a challenge to risk assessment approaches insofar as they can not be addressed with the same mathematical model. In order to be able to handle both types of events more sophisticated approaches utilize two mathematical models. SimPRA and other approaches that can handle both types of events are preferable in that they generate higher fidelity scenarios.

6.1.1.8 Model complexity

The assumptions driving some risk assessment approaches limit the ability of the user to model certain relationships between risk elements such as dependent events, latent failures, redundancies, etc. As such, in these approaches, scenario generation is based on models that are overly simplified. This is another area in which SimPRA stands out as there are almost no limitations on modeling the relationships between risk elements.

6.1.1.9 Problem solved

Of the eight risk assessment approaches applied to the PSAM8 problem, six arrived at a solution. Of these six, four, including SimPRA, arrived at answers that were within the order of E-1, strongly implying that the correct solution should be around this magnitude. It is interesting to note that the two approaches that could not provide any solution at all were those that attempted to provide an exact solution.

6.1.2 Summary of comparison

This comparison shows that, in general, only the simulation approaches were able to solve the benchmark problem within the provided time frame and with the full range of assumptions in the problem definition. Among the simulation approaches, SimPRA stands out for its expandability (both horizontally and vertically) and its ability to consider the low probability-high consequence scenarios. Table 5 summarizes the results of this comparison.

Table 5: Comparison between different approaches to the benchmark problem

Name	Exact/ Approx. solution	Binary/ Multi state	Expandable	Low Prob. High Cons. Scenarios	Complex Systems	Common Cause	Demand-Based/ Time-Based	Model Complexity	Problem Solved
MC	Approximate	Binary but multi-state is also possible	Not known	No	Yes	Yes	Both	High	Yes [E-1]
DFM	Analytical	Binary	Yes	Yes	No	Not shown	Both	Can't get too complex	No
DFT	Exact	Binary	Yes	Yes	No	Not shown	Time based only	Not easy to develop	Yes but way too far from other solutions (E-13)
AO-MC	Approximate	Multi state	Yes but only horizontally	No	Yes	Yes	Both	Complex	Yes[E-1]
SAPHIRE	Exact	Multi state	In some cases in a static form	Yes	No	Yes	Both	Very hard to model	No
FT/ET/Markov	Analytical approach, approximate solutions	Multi state	No	Yes	No	Yes	Both	Not easy to develop	Yes but out of range of other solutions [E-3]
SimPRA	Approximate	Multi state	Yes, both horizontally and vertically (Hierarchical)	Yes	Yes	Yes	Both	Complex	Yes [E-1]
DES (TIGER)	Approximate	Multi state	Not known	Not known	Yes	Yes with difficulty	Time based. Demand based with difficulty	Not too complex	Yes [E-1]

6.2 Holdup tank control system example

Variations of the holdup tank example are commonly used in the PRA literature to demonstrate and compare the capabilities of different methods suggested for the field (Figure 35). Aldemir 1987, Deoss and Siu 1989, Siu 1994, Cojazzi 1996, and Hu 2005 are examples of studies that use a holdup tank as the case for the study.

There are different variations of the problem but in general the holdup tank control system consists of two pumps, a valve and a number of sensors and actuators that keep the level of liquid chemical of the tank within a predefined range. The first pump has the same capacity for pumping liquid that the valve has for dispensing and the second pump has half the capacity of the first pump. At time = t_0, the level of liquid in the tank is somewhere between the High and Low levels (acceptable range). If for any reason (such as a hole in the tank or a change in the balance of in-flow and out-flow of the tank) the level of liquid in the tank goes out of the acceptable range, the control system will intervene. If the tank level (L) rises much higher than the high level of acceptable range and reaches the overflow level, or falls much lower than the low level of acceptable range to dry-out level, a system failure has occurred. The goal of the studies cited above was to identify the time-dependent probability of system failure and present the type of scenarios considered.

Hu, 2005, used the holdup tank example to show how SimPRA's risk assessment results compare with other approaches mentioned in the literature. He found that the SimPRA approach generates the same numerical results when there are only a few dynamic elements in the model. In more complicated cases, however, SimPRA was able to generate scenarios that were not observed by other methods. Hu also determined that SimPRA's use of a combination of discrete and continuous time models make it more suitable for studying systems that are usually fairly stable before a deviation from the normal path by, for example, a component failure, at which point the system evolution becomes very fast.

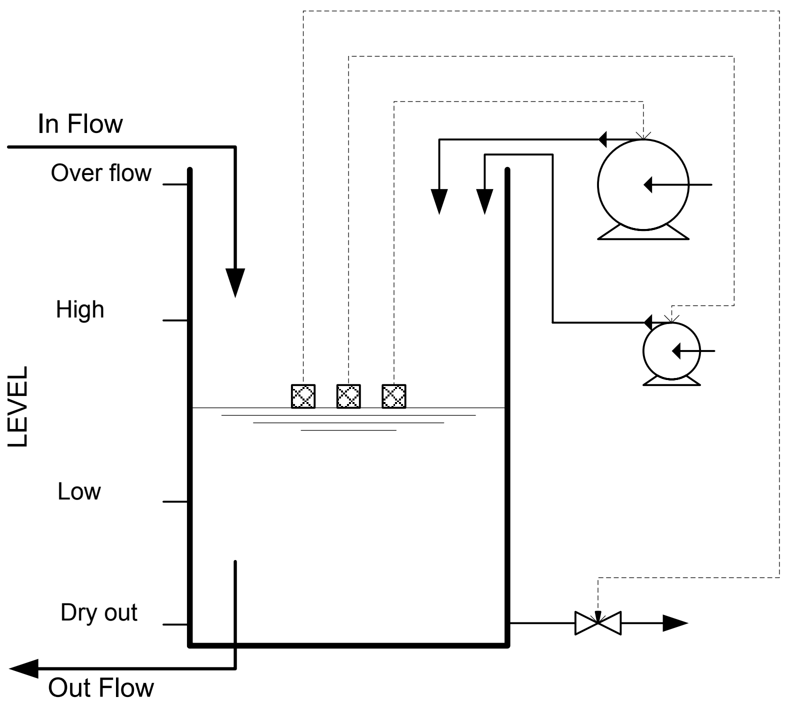


Figure 35: Schematic diagram of the holdup tank example

More important to the purposes of this section, Hu found that the planner had a critical role to play in ensuring the exploration of high-consequence / low-probability events. More specifically, when the planner gave a scenario with a fairly low probability, the scheduler was able to generate detailed scenarios based on those cases. Some of the low probability scenarios were cases in which a component failure had to happen after a number of successful uses of that component and a simple increase of the failure probability could not generate those scenarios frequently enough.

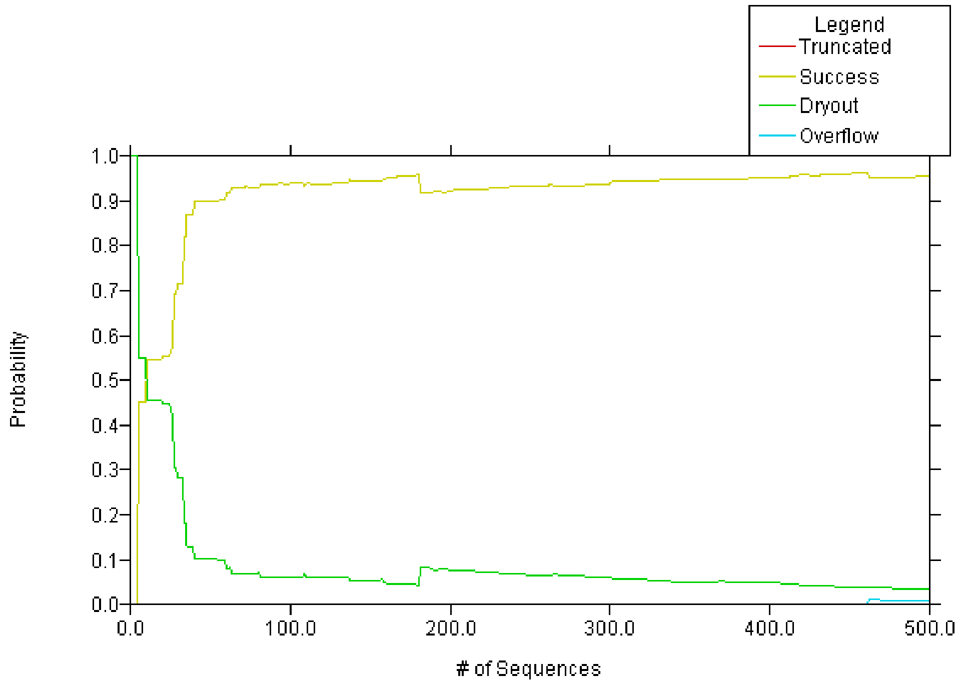


Figure 36: An example of the change of the end-state probabilities by increase in the number of realizations

As another test of the critical role of the planner in guiding the simulation, SimPRA was run three times with and without the planner component for 500 realizations (Figure 36 shows an example). The results of these simulations are given in Table 6. As can be seen, those simulations that included the planner component resulted in higher instances of scenarios with lower probabilities. This suggests that the presence of the plan helped the SimPRA scheduler better guide the simulation to generate low-probability scenarios. Another important conclusion can be made by comparing the standard deviations of the probabilities of the end-states in each group. The smaller standard deviations (in the order of magnitudes) for those simulations that included the plan suggest that the planner caused the scheduler to get on a faster track of convergence.

Table 6: Comparison between the final probability and standard deviation of 3 different simulation runs with and without having a plan

		Probability			SD	# Sequences		
		case 1	case 2	case 3		case 1	case 2	case 3
With Plan	Success	9.90E-01	9.88E-01	9.92E-01	2.00E-03	103	97	73
	Dry-out	9.73E-03	1.18E-02	7.88E-03	1.96E-03	350	353	371
	Overflow	9.02E-05	1.64E-04	1.39E-04	3.75E-05	47	50	56
Without Plan	Success	9.44E-01	9.53E-01	8.89E-01	3.46E-02	448	441	453
	Dry-out	5.57E-02	3.63E-02	9.87E-02	3.19E-02	27	35	26
	Overflow	4.88E-06	1.05E-02	1.25E-02	6.71E-03	25	24	21

6.3 LRO satellite example

In a recent study by Haghani (Haghani 2007), the risk scenario generation capability of the SimPRA planner was evaluated. NASA's Lunar Reconnaissance Orbiter (LRO) satellite was used as an example of a complex system. The SimPRA planning approach was applied to the LRO and the risk scenarios generated were compared with a focus on the Command and Data Handling Subsystem (C&DH). Given the relevance and significance of its findings, a fairly extensive summary of the Haghani study follows.

6.3.1 Mission overview

The Lunar Reconnaissance Orbiter is a very high-profile mission within the Robotic Lunar Exploration Program (RLEP). LRO is the first in a series of missions for robotic lunar exploration which plans to develop new approaches and technologies to support human life on Mars and other destinations. Proof of water will establish NASA's primary goal to first determine whether life can be sustained on the moon for long periods of time. On the moon, man can "practice living, working and doing science" in order to prepare for longer and more dangerous missions (Opperhauser 2006). LRO supports a number of instruments that will allow it to achieve its primary mission objectives including: characterizing the moon's radiation environment to evaluate biological impacts and their possible mitigation; determining the moon's topography through a high resolution 3-D geodetic grid in order to identify landing sites and determine the availability of resources (such as water); mapping temperature, lighting and surface imaging, etc.

The LRO and subsequent missions are scheduled to launch on a yearly basis starting no later than 2008. With a mission life of 14 months, the LRO and its seven instruments will transmit data to Earth that will allow scientists to determine the best approach to supporting life on the moon. An extended human life venture to the moon is scheduled as early as 2015 (Opperhauser, 2006).

6.3.2 LRO subsystems

The LRO has several subsystems and components that must be integrated together for a successful mission. Figure 37 is a block diagram view of the entire spacecraft.

For the purposes of evaluating SimPRA, some of these subsystems are treated as a high-level structure while other subsystems are explored more deeply (i.e. C&DH). A brief description of each subsystem is provided below.

- Attitude Control Subsystem (ACS) – Responsible for controlling the movement and location of the spacecraft in orbit for all three axes.

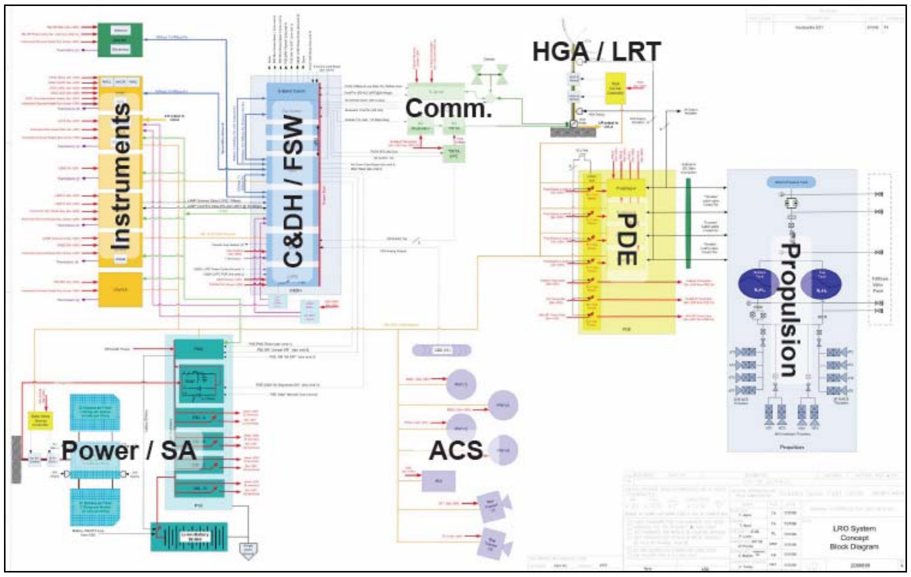


Figure 37: LRO system overview (Houghton, 2006)

- Power Supply Electronics (PSE) – Responsible for converting power from the solar arrays and supplying the primary bus voltage to the other subsystems and their hardware.
- Radio Frequency (RF) Comm – Responsible for the hardware that provides communication link between the ground and the spacecraft
- Propulsion – Contains the tanks of fuel, oxygen, and hydrogen used in spacecraft maneuvering
- Propulsion Deployment Electronics (PDE) – Responsible for controlling and monitoring the Propulsion subsystem
- Command and Data Handling (C&DH) – Sole communication between the satellite and the ground station on Earth (see below for additional details)

6.3.3 C&DH subsystem

The Command and Data Handling (C&DH) subsystem plays a pivotal role in any spacecraft developments. The C&DH is the sole communication between the satellite and the ground station on Earth. The C&DH box is responsible for down-linking science and housekeeping data as well as receiving commands (uplink) from the ground and transmitting them to other subsystems on the spacecraft. The C&DH is comprised of a number of electrical circuit boards that communicate with one other and interface with key elements outside of the subsystem. At a bare minimum, this subsystem contains some type of a processor, and source of power for the box, as well as a communication interface for the radio frequency (RF) antenna. More sophisticated versions may contain a form of memory for data storage and/or an interface for specific instruments.

Because of the significance of the C&DH box to the spacecraft, engineers must carefully consider the overall system architecture of this subsystem. A redundant box is often built and placed on the spacecraft in standby mode to increase the reliability of the overall mission. Due to the limited resources as well as the tight schedule of LRO, however, the decision was made to make the C&DH box in this spacecraft single string. In other words, the loss of any functionality in the C&DH box will lead to mission degradation and quite possibly mission failure.

6.3.4 SimPRA application to LRO

As discussed above, SimPRA's planner component develops high-level scenarios which are later used by the scheduler for simulation. To generate scenarios, the SimPRA planner requires that several stages of modeling be completed. These include developing the Functionality Tree, Structure Tree, Functionality-Structure Map, State Transition Diagrams, and Functionality Details List. Upon the completion of these phases, scenarios can be generated in the Plan Generation phase.

In order to generate risk scenarios for the LRO project, the hierarchical structure of the system was designed with additional detail given to specific lower level subsystems. Functionalities were identified for all components defined and mapped to these structures. State diagrams were generated for each subsystem and component in the design. Finally, the higher level subsystems' transitions were linked to their lower level component states. With this complete, the end-state scenarios for the top level LRO system were generated. Those scenarios that result in the 'Failed' or 'Degraded' state of each subsystem were then available for use in developing a thorough risk assessment of the system.

6.3.5 LRO PRA approach

NASA Headquarters Office of Safety and Mission Assurance (OSMA) defines risk as a concept which includes consequences that are undesirable (i.e. harmful) in addition to the probability of the occurrence of such consequences. Risks are determined by first defining the initial event that could upset the system. Then the consequences of such an event are investigated along with the frequency of occurrence. Once all the risks are determined, they are integrated into the system risk profile.

As applied to the LRO example, NASA's current practice of PRA would employ a combination of classical reliability techniques such as Reliability Block Diagrams (RBDs), Fault Trees (FTs), and Failure Modes and Effects Analysis (FMEA). In the case of the C&DH subsystem, a Fault Tree Analysis was employed as shown in Figure 38. Individual card-level fault trees were *not*, however, developed because reliability engineers were not necessarily familiar with the details of each particular design (Haghani, 2007). To develop a fault tree at the card level would, therefore, have required increases in financial and manpower resources that were not available.

6.3.6 Comparison

The SimPRA results compares favorably with current PRA practices on at least three counts (Haghani, 2007):

First, FT/ET is unable to explore scenarios that unfold at a lower, component level of the system. Card-level fault trees are, however, very significant in determining the critical causes of failure per board and thus the subsystem. The fault tree approach is unable to produce fault trees at this level without prohibitively expensive inputs of money and manpower. In contrast, the SimPRA planner generates scenarios that unfold at all levels of the C&DH subsystem (Figure 39). As such, even if the quantification of probabilities still require data that is not available, the scenarios generated by the SimPRA planner will ensure that the risk analyst at least qualitatively considers more possible (and unanticipated) risks to the subsystem.

Second, SimPRA automatically generates a considerable (779 in this example) number of risk scenarios most of which are not even identified by the FT/ET approach. As shown in Figure 38, by restricting the analysis to the subsystem level, LRO PRA considered only seven risk scenarios.

Third, the SimPRA planner provides very dynamic scenarios. A comparison of Figure 38 and Figure 39 reveals this difference. Every one of the seven subcomponent failure scenarios considered by FT/ET approach and presented in Figure 38 is comprised of the failure of one and only one component. By way of contrast, the SimPRA planner can generate rich and dynamic scenarios like the one in Figure 39 that presents the evolution of component state changes based on component-level events and the subsequent consequences for subsystem failure. The advantage is that the SimPRA planner can identify scenarios under which the failure of a component leads to the degradation but not necessarily failure of the subsystem. FT/ET is limited and pessimistic in its simplified assumption—due to a lack of information—that all component failures automatically result in subsystem failure.

Given the sensitivity of LRO's mission, one might expect risk analysis to be conducted at a more detailed level that explicitly considers a greater number of risk scenarios. The problem, however, is that classical PRA methods are not well-suited for capturing the dynamic elements of complex systems like LRO. Even attempting to model the failure scenarios that the SimPRA planner generates automatically proves to be a tedious and grueling process for FT/ET approaches because they require the analyst to consider events occurring at all levels of the system simultaneously. In contrast, SimPRA allows the analyst to model the behavior of sub-systems in isolation and then link the changes in behavior to the states of its sub-elements. The planning algorithm then uses these inputs to automatically generate all of the scenarios.

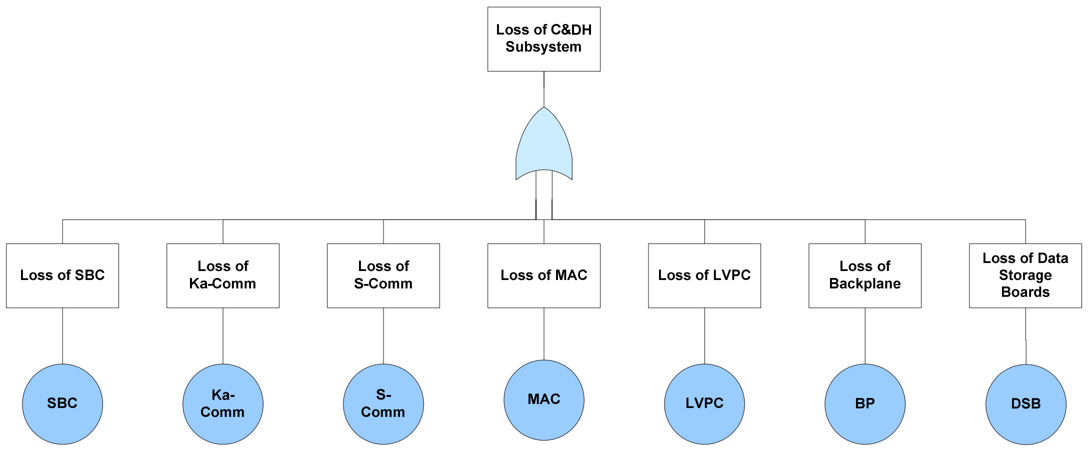


Figure 38: C&DH top-level fault tree

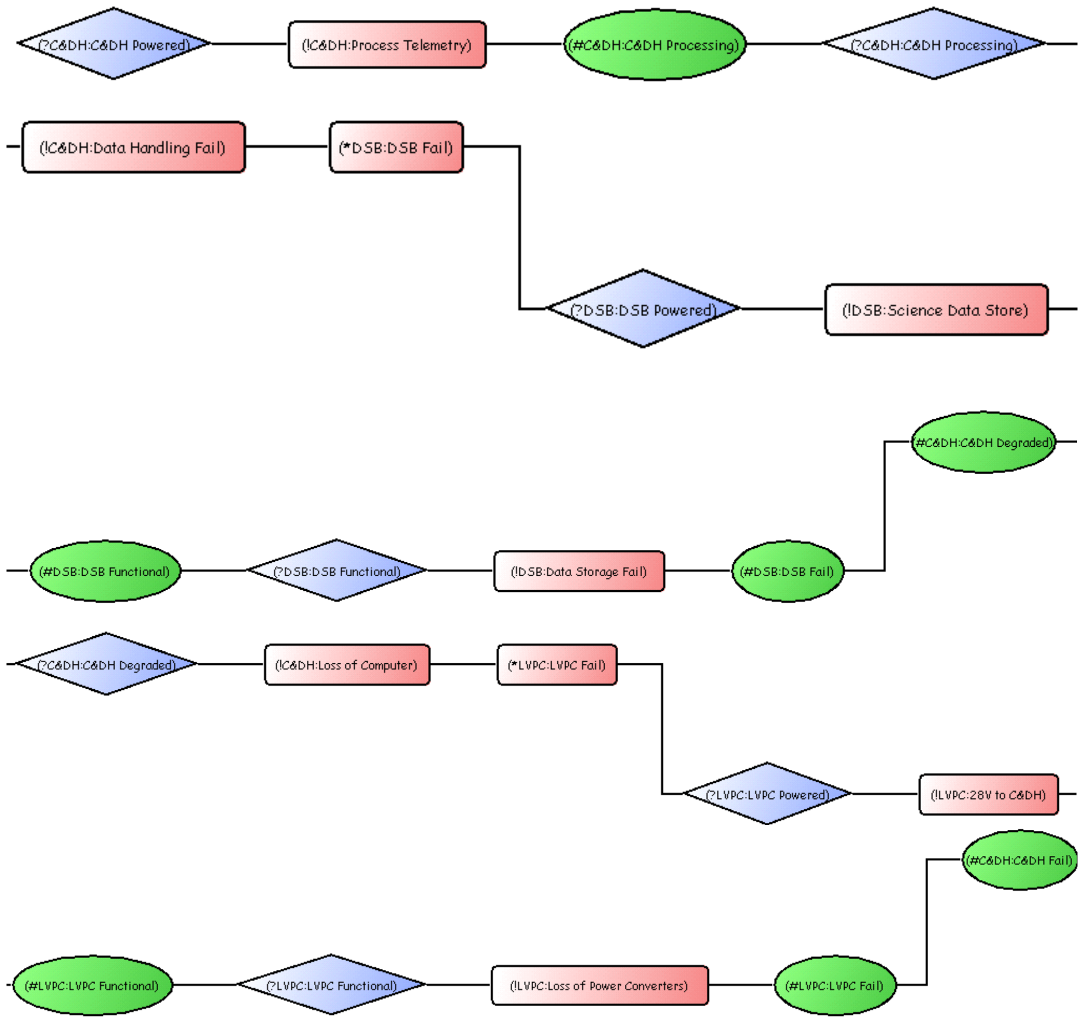


Figure 39: A C&DH failure scenario

7 Application: Risk based design

In this chapter, the application of SimPRA and the planner in Risk-Based design is explained. Within the SimPRA framework, this method is particularly useful for addressing concerns that arise during the design of highly sensitive and complex systems whose failure may lead to serious consequences that put financial resources, the environment, and human lives at risk. The remainder of this chapter highlights the utility of SimPRA to the field of risk based design.

7.1 Background

The classical approach to design is based on a deterministic perspective where the assumption is that the system and its environment are fully predictable and their behavior is completely known to the designer. Input variables are assumed to be fixed and safety factors are applied to ensure the reliability and robustness of the design (Mahadevan and Smith, 2003). Although this approach may work fairly well for regular design problems—given, of course, the possible extra costs of over-designing for the sake of safety and reliability—it is not satisfactory for the design of highly sensitive and complex systems where significant resources and even lives are at risk.

Defining 'Design' as "the interplay between what we want to achieve and how we want to achieve it" (Suh, 2001) and 'Risk' as the probability and consequences of something happening that will have an adverse impact upon objectives, 'Risk-Based Design' can be defined as an interactive process by which the repeated assessment of risk within a system is used to inform modifications that ultimately lead to a more mature and refined system design. It has also been defined as a process whereby the analysis of uncertainties within the system and its environment as well as their impact upon the system's performance is incorporated directly into the design process (Crowe, 2001; Priest, 1988; and Cruse and Mahadevan, 1997).

7.2 Review of SimPRA

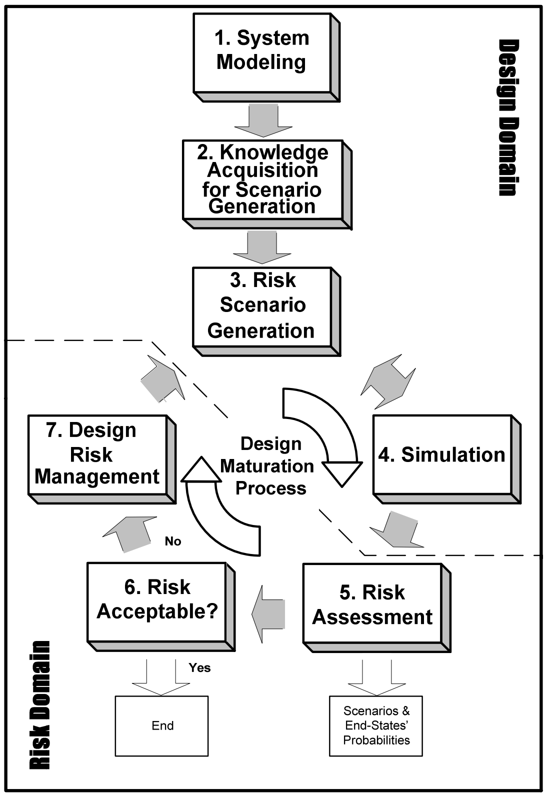
In practice, risk assessment is performed by first identifying how a system might deviate from its intended performance, second deciding how probable these deviations are and third determining what the consequences of these deviations might be (Kaplan, Haimes and Garrick, 2001). SimPRA allows the designer to use the knowledge that can be expected to exist at the design stage—which is based on the mapping of functional requirements in the functional domain to the design parameters in the physical domain (Suh, 2001)—to: a) generate high level risk scenarios, that is, to identify how deviations can occur; and then b) apply these high level scenarios to a rich simulation model of the system to generate detailed scenarios and identify the probability and consequences of these scenarios.

As discussed in chapter 2, Dynamic Probabilistic Risk Assessment (DPRA) is usually interpreted as an exploration of the space of possible event sequences to gain risk information. A number of simulation methods have been used in DPRA to help the analyst understand the behavior of the system under a variety of conditions (Mahadevan and Raghothamachar, 2000; Marsequerra et. al., 2000) especially those leading to risky outcomes. However, the SimPRA approach is much more efficient in covering the large space of possible scenarios as compared with, for example, biased Monte Carlo simulations because of the planner element which uses engineering knowledge to guide the simulation process. This allows SimPRA to avoid the slow convergence of most biased Monte Carlo methods which aim at finding an optimal sampling function while disregarding the structure of the system under investigation. The planner uses engineering knowledge to generate high level scenarios.

The SimPRA framework may be used for the risk assessment of a design to identify the worst case scenarios and the probability of their occurrence as well as the total risk of the system's behavior under uncertain conditions. The value-added of this approach is that it enables the designer to observe system behavior under many different conditions. The designer can also modify the design to compare the results of the risk assessment under different design specifications. This process will lead to a risk-informed design in which the risk of negative consequences are either eliminated entirely or reduced to an acceptable range.

The overall structure of the SimPRA approach to Risk-Based Design is illustrated in Figure 40. The first step—System Modeling—is to create a virtual model of the system with the appropriate level of detail. The second step—Knowledge Acquisition for Scenario Generation—is to acquire additional information about the system and its environment that will be used to automatically generate risk scenarios in the third step—Risk Scenario Generation. The fourth step—Simulation—consists of the actual running of the simulation, the results of which are reported as the probabilities of the end-states as well as the worst case scenarios in the fifth step—Risk Assessment. In the sixth step—Risk Acceptable—the designer or risk analyst makes a determination as to whether or not the risks associated with the scenarios and end-states are acceptable. If the risks are not acceptable and must be reduced through changes in the design of the system, those changes are applied to the simulation model (and to the planner model if necessary) in the seventh step—Design

Risk Management. The risk assessment process is repeated as many times as necessary and ends only when the designer is satisfied with the level of risk in the system or has found an acceptable range of behavior for each component of the system that does not jeopardize the robustness of the overall system's behavior.



Design Domain
1. System Modeling
2. Knowledge Acquisition for Scenario Generation
3. Risk Scenario Generation
4. Simulation
5. Risk Assessment
6. Risk Acceptable?
7. Design Risk Management
Design Maturation Process
No
Yes
End
Scenarios & End-States' Probabilities
Risk Domain

Figure 40: Overall structure of SimPRA approach to risk-based design

7.3 Risk acceptance and risk management

The level of acceptable risk is usually defined by the requirements. Any deviation from the expected functionality and behavior of a system can lead to an unwanted consequence and needs to be considered in the risk assessment process. Safety, robustness, dependability and reliability are all some of the characteristics that can be affected by changes in the behavior of a system and can be evaluated through the simulation process.

When studying the life-cycle of a component in a system that has triggered an accident, it is usually possible to distinguish four time intervals:

1. Assuming that the component was not defective when it was initially put into the system, there is a time frame in which the component does its job properly and within the range of expectations. This time frame can be named the "*productive time*".
2. The second time frame is that in which the component has not yet failed to perform its function but is pushing toward the edges of acceptable performance range. This is usually accompanied by some sorts of changes in the system behavior that can signal an observer that something has changed in the system. This time frame is called the "*degradation time*"
3. The third time frame is when the component finally fails to perform its job and the system starts moving towards a failure. This is called the "*failure time*".
4. Finally, most failures can become triggering events for additional series of failures if their damage is not properly controlled. The time it takes to control the effects of a failure on its environment is called the "*failure recovery and management time*".

A familiar example of the failure life-cycle can be seen in the performance of automobile brakes. When a car's brake pads are new, they usually work very well for a long time. When they begin to wear out, they make a squeaking noise when the driver tries to stop the car. This is the degradation time during which the brakes are still working but are pushing toward the edges of acceptable performance range. If not soon replaced, the brakes will soon stop working altogether and may cause a serious accident. In the event that a car accident does occur, the recovery and management time may be devoted to ensuring the safety of other cars in the area.

Different risk management tools are designed to work within different time frames of the failure life cycle and help to prevent or at least minimize the consequences of the failure or the recovery effort that follows. These risk management tools can be divided into 7 categories:

1. Life design: This is the most common approach to risk management. If the "productive time" of a component is short, the component can be replaced with a more reliable one. Alternatively, the conditions under which the component is performing may be improved so that it experiences less stress while performing its job.
2. Maintenance: Maintenance can significantly increase the "productive time" of a system's lifecycle by replacing components or increasing components' lives by eliminating/decreasing the environmental stressors that affect it. One of the potential issues with this approach to risk management, however, is that it usually inserts a human into the loop, thereby making the system more susceptible to negligence or other human errors in the maintenance process.
3. Alarms: Alarms act during the "degradation time" of a system's lifecycle and are useful when the system can tolerate a little bit of change while an intervention is made to impede progress toward an actual failure. Like

maintenance, this approach to risk management is also susceptible to human mistakes.

4. Mitigations: These processes lengthen the “degradation time” without having an effect on the consequences of a failure.
5. Controls: Controls act like a safety net. They do not affect the life of a component or the duration of a failure, but minimize the consequences of failures.
6. Containments: Containments do not prevent a failure from occurring but can minimize or eliminate a chain reaction in which one failure causes additional accidents. An example is the containment structure on top of nuclear power plants
7. Design Change: Change is another tool in risk management by which the designer completely changes his or her approach to a problem. Of course, such a step will change the failure scenario and may require a whole new set of risk modeling.

7.4 Satellite example

For illustrative purposes, we return to the satellite example first introduced in chapter 5. The goal now is to consider the early stages of the satellite’s design as an example of how the proposed methodology improves the design by making it risk-informed. This example highlights the flexibility of the risk scenario planning process in adapting itself to the new system settings after applying design risk management tools.

Let us assume that the risk assessment of the satellite design shows that the probability of the system going into the degraded state is higher than the risk acceptability level. At this point the designer studies the scenarios leading to this state and finds out that the actual reason behind this problem is the interaction between software and hardware in the downlink process. The designer determines that one way to solve this problem is to detect the problem early in the downlink process and send the satellite into safe mode for a short period of time to restart the process. Two alternatives for the early detection process are: 1) to use an alarm mechanism on the ground, or 2) to use a control mechanism on the satellite. The design of the system can then be altered to accommodate these changes:

Option I: An alarm mechanism is designed to check the quality of the data automatically and warn users of possible problems with the data before it is too late to ask the satellite to resend it. The alarm system that is modeled here has three states, DETECT, ACTIVATED and NOT_ACTIVATED. There is also a human decision model added with three states, OBSERVE, SEND_TO_SM and NO_SM. Figure 41 illustrates the changes made in the model to accommodate for applying the alarm system.

Since Alarm and Human are not on the control variable list, their effect is only modeled by detecting their presence. This means that the planner will only generate these scenarios if it detects the presence of these signals in the simulation.

Option II: The same problem is addressed with a control rather than with an alarm. A quality control unit independently checks the downlink data on the satellite. If it detects a problem with the quality of data, it will presume that the Txmitter is degraded so that the system will go to safe mode for repair. Figure 42 illustrates this process.

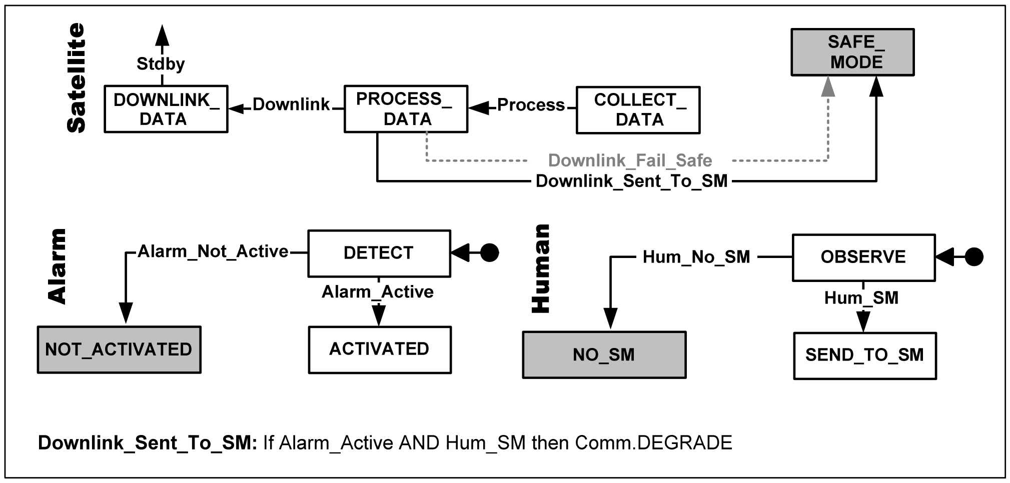


Figure 41: The alarm system added to the satellite system

New risk scenarios are generated based on these new designs. After running the risk simulation for each alternative, it is observed that although both options can reduce the probability of the degrade state, the use of an alarm on the ground is preferable because of the higher accuracy achieved by having a human checking the quality of transferred data as well.

As the above examples indicate, the risk scenario planning model can accommodate these design risk management changes with a few relatively minor changes. However, these two scenarios have very different effects on the system. One will send the system to Safe Mode with some delay and only after alarming a human, while the other one automatically sends the system to Safe Mode without involving the ground station. One of the major factors for deciding between these risk management tools is how they affect the accuracy of the system as reflected in the results of the simulation itself. The analyst will apply these changes one by one and compare the results of the risk analysis for each of these designs and choose the one that best suits the needs of the system.

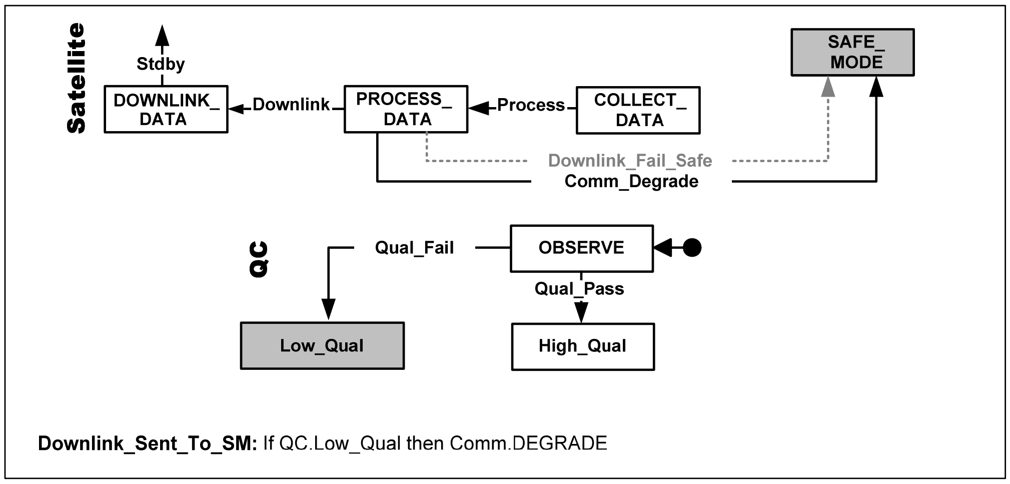


Figure 42: A quality control unit added to the satellite system

7.5 SimPRA application in RBD conclusion

Guided simulation is a very effective tool for risk-assessment in the design of large and complex systems. This chapter detailed how SimPRA's planner component can automatically generate a large number of scenarios that guide the simulation toward different states of the system and automatically explore unplanned scenarios to find new vulnerabilities in the design of the system. Risk assessment of the system generates probabilities of the end-states and scenarios that lead to those end-states. The system designer will then decide if the risks are acceptable or not. Because of the hierarchical structure of the planner in the case that risks are still not acceptable, the designer can easily change the planner and simulation model to manage the risk and enhance the design.

8 Conclusion

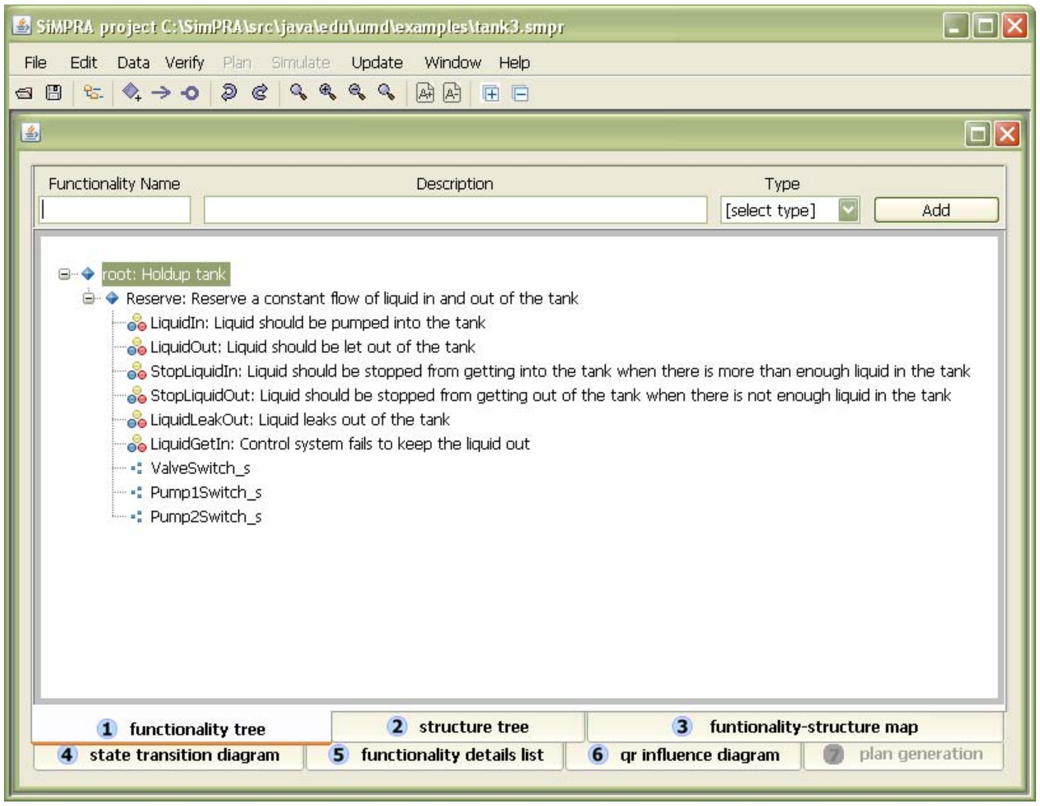
The primary contribution of this dissertation is in offering a new method for capturing different types of engineering knowledge which are used to automatically generate dynamic risk scenarios that are presented as generalized event sequence diagrams. As an integral element within the SimPRA framework, the planner has been shown to be an important innovation in the field of dynamic probabilistic risk assessment methods. Comparisons at three different levels showed that: the SimPRA approach is superior to other DPRA methods of solving a propulsion system benchmark problem; the planner element significantly improves convergence and coverage of risk scenarios; and the planner itself compares favorably with other PRA methods in generating risk scenarios. A software code designed to implement the planner has been developed and refined based upon the results of its application to several problems.

The utility of the SimPRA framework was also highlighted in its application to the field of risk-based design. The planner's automatic generation of a large number of scenarios that guide the simulation toward different states of the system and automatically explore unplanned scenarios was found to be particularly useful in identifying new vulnerabilities in the design of the system. The hierarchical structure of the planner was also shown to be useful in allowing for the accommodation of design risk management changes with relatively few minor changes.

The next step in continuing this research can be the development of a more active planner and scheduler that can guide the simulation in real time and use the captured engineering knowledge to schedule the timing of the events in a more effective way. The ability of the planner to summarize simulation results can also be explored for applications in risk management. Although the contribution of this dissertation is in generating risk scenarios for dynamic systems and guiding the simulation by providing it with the knowledge of what to bias, not how to bias, a comparison between the SimPRA framework as a DPRA method and other biased Monte Carlo approaches in terms of their ability to generate low probability high consequence scenarios would be of interest.

Appendix A: Software implementation of the SimPRA multi-state planner

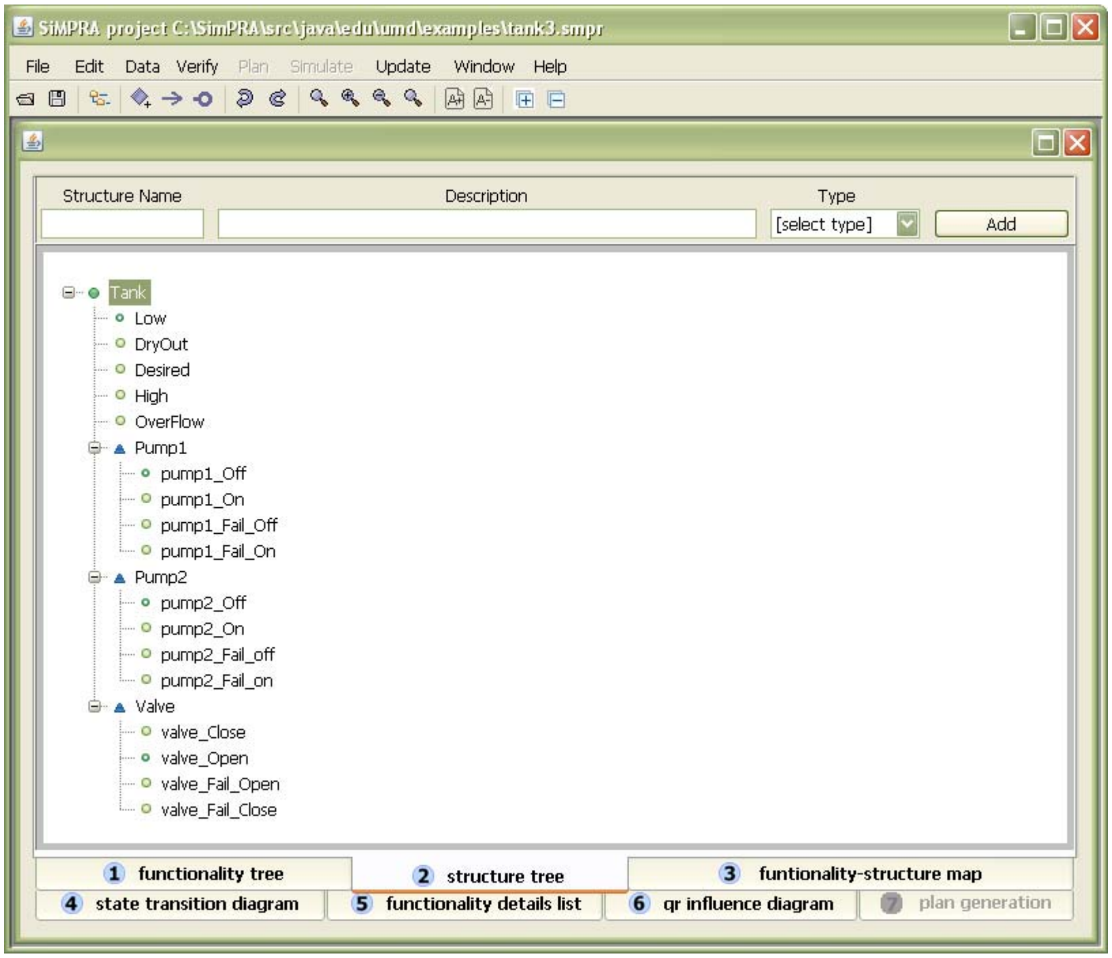
The primary contribution of this thesis is in offering a new method (called the planner) for capturing different types of engineering knowledge and using them to automatically generate risk scenarios for dynamic systems that are presented as generalized event sequence diagrams. A software code designed to implement the planner has been developed in JAVA programming language and used for several projects including the Lunar Reconnaissance Orbiter (LRO) project discussed in section 6.3 and the hold-up tank example discussed in section 6.2. (The code for the scheduler was also developed in JAVA while the simulation model was developed in the Simulink® environment which is fully compatible with JAVA).


SiMPRA project C:\SimPRA\src\java\edu\umd\examples\tank3.smpr
File Edit Data Verify Plan Simulate Update Window Help
Functionality Name
Description
Type
[select type]
Add
1 functionality tree
2 structure tree
3 funtionality-structure map
4 state transition diagram
5 functionality details list
6 qr influence diagram
7 plan generation

root: Holdup tank
Reserve: Reserve a constant flow of liquid in and out of the tank
LiquidIn: Liquid should be pumped into the tank
LiquidOut: Liquid should be let out of the tank
StopLiquidIn: Liquid should be stopped from getting into the tank when there is more than enough liquid in the tank
StopLiquidOut: Liquid should be stopped from getting out of the tank when there is not enough liquid in the tank
LiquidLeakOut: Liquid leaks out of the tank
LiquidGetIn: Control system fails to keep the liquid out
ValveSwitch_s
Pump1Switch_s
Pump2Switch_s

Figure 43: Functionality tree

In this section, screenshots of the software's user interface at different stages of soliciting engineering knowledge for the hold up tank project are shown and discussed. The purpose is to give the reader a practical and step-by-step sense of how the method outlined in this thesis has been operationalized.

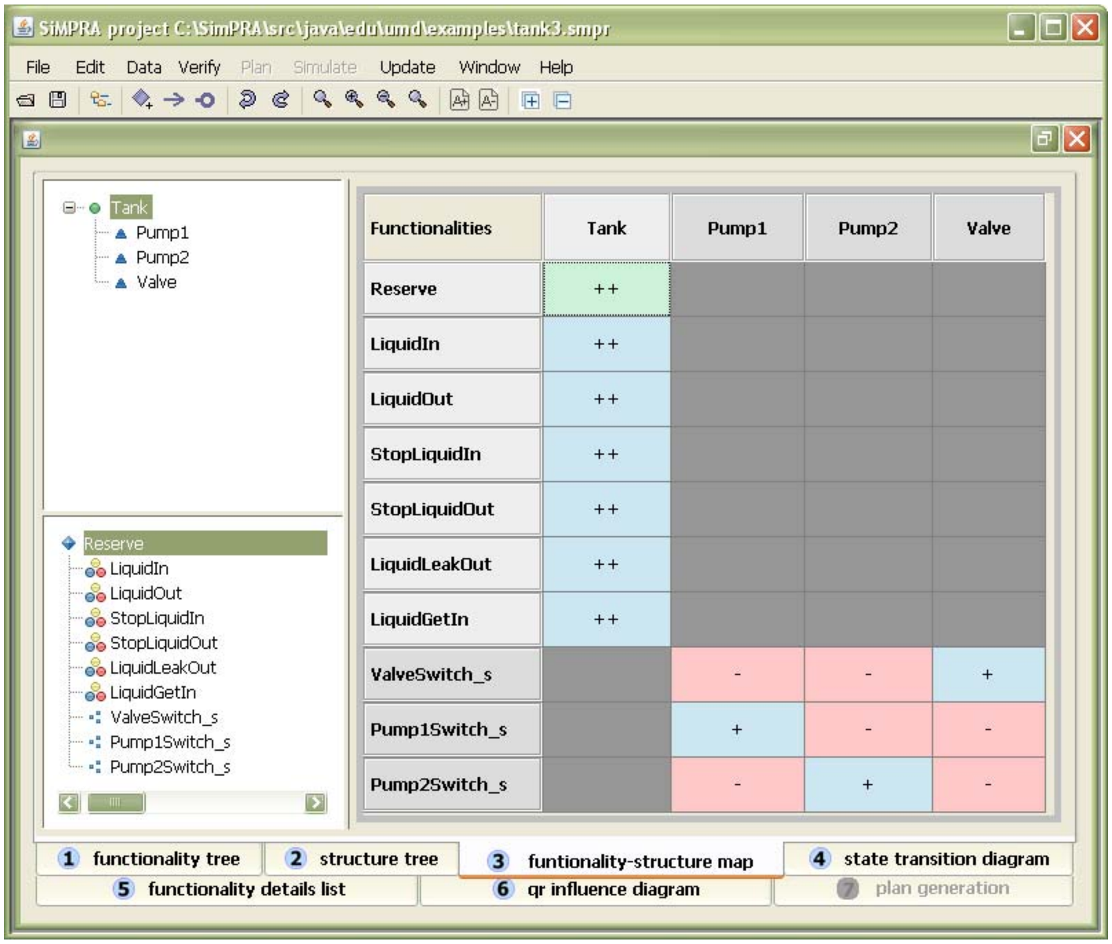

SiMPRA project C:\SimPRA\src\java\edu\umd\examples\tank3.smpr
File Edit Data Verify Plan Simulate Update Window Help
Structure Name
Description
Type
[select type]
Add
1 functionality tree
2 structure tree
3 funtionality-structure map
4 state transition diagram
5 functionality details list
6 qr influence diagram
7 plan generation

- Tank
  - Low
  - DryOut
  - Desired
  - High
  - OverFlow
  - Pump1
    - pump1_Off
    - pump1_On
    - pump1_Fail_Off
    - pump1_Fail_On
  - Pump2
    - pump2_Off
    - pump2_On
    - pump2_Fail_off
    - pump2_Fail_on
  - Valve
    - valve_Close
    - valve_Open
    - valve_Fail_Open
    - valve_Fail_Close

Figure 44: Structure tree

Engineering knowledge is solicited from the operator in six steps, each of which is handled by a separate tab in the user interface. As shown at the bottom of Figure 43, these six steps/tabs are for capturing the: 1) Functionality tree; 2) Structure tree; 3) Functionality-structure map; 4) State transition diagrams; 5) Functionality details list; and 6) QR influence diagram.

Figure 43 displays a screenshot of Tab 1. In this step, the program solicits information from the user about the hierarchy of the system's functionalities and incorporates this information into a functionality tree. The user can add, delete, and modify functionalities (and sub-functionalities), actions and/or events as the requirements of the system are defined.



Functionalities	Tank	Pump1	Pump2	Valve
Reserve	++			
LiquidIn	++			
LiquidOut	++			
StopLiquidIn	++			
StopLiquidOut	++			
LiquidLeakOut	++			
LiquidGetIn	++			
ValveSwitch_s		-	-	+
Pump1Switch_s		+	-	-
Pump2Switch_s		-	+	-

Figure 45: Functionality-structure map

A screenshot of the Structure Tree tab is presented in Figure 44. In this step, the user defines the hierarchical structure of the system along with the states of each element. This hierarchy can have as many levels as necessary to adequately reflect the complexity of the system.

Figure 45 displays the functionality-structure map that is used to solicit information from the user about the functionalities that are provided by each structure element. A '+' sign in the map indicates that the functionality described in the left-most column is provided by the structure described in the top row. As mentioned earlier, events only map to components while actions and functionalities map to subsystems. A gray cell in the cross section means that because of the difference between the levels of component in its hierarchy with the level of functionality in its own hierarchy, there is no chance of a mapping between them.

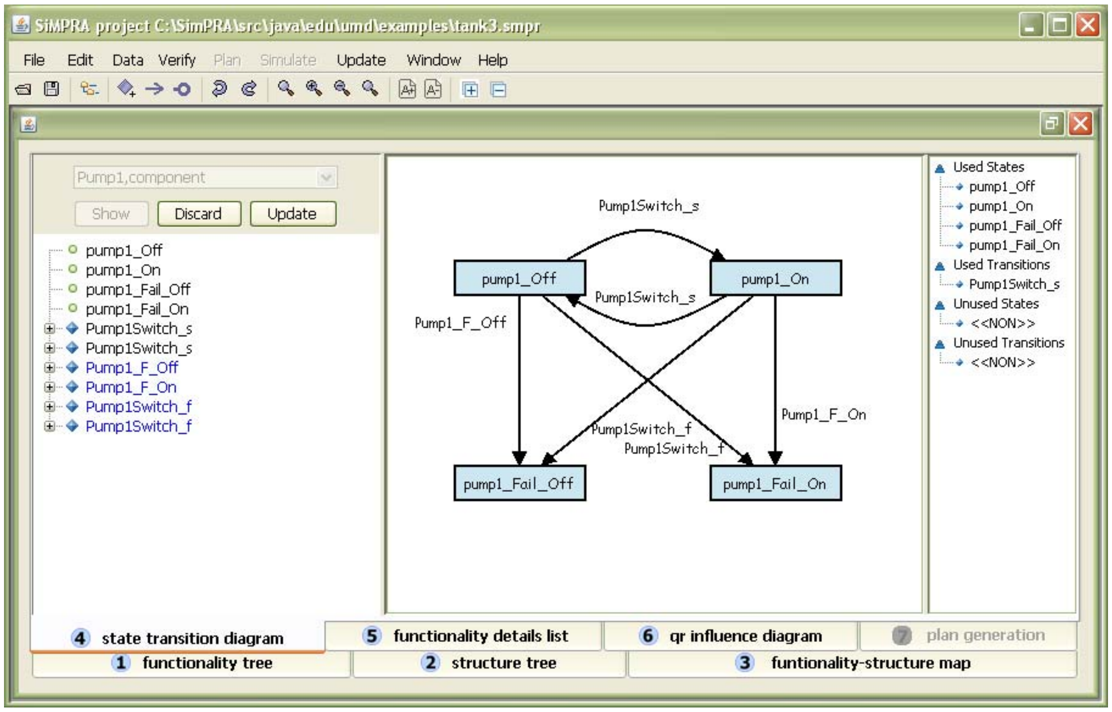


Figure 46: State transition diagrams

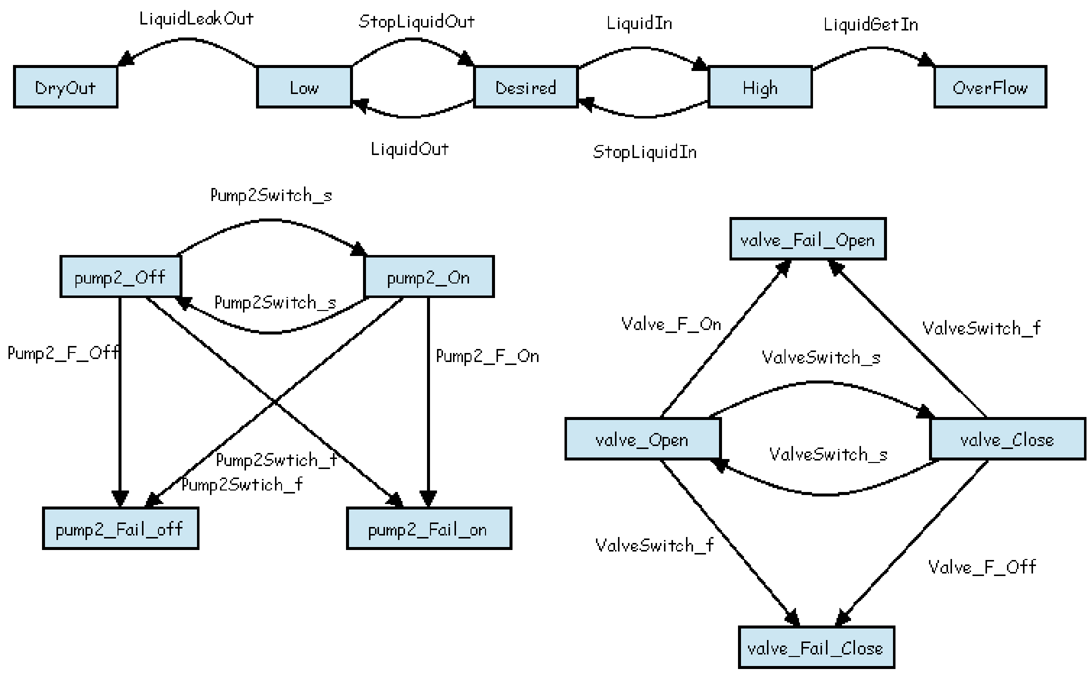
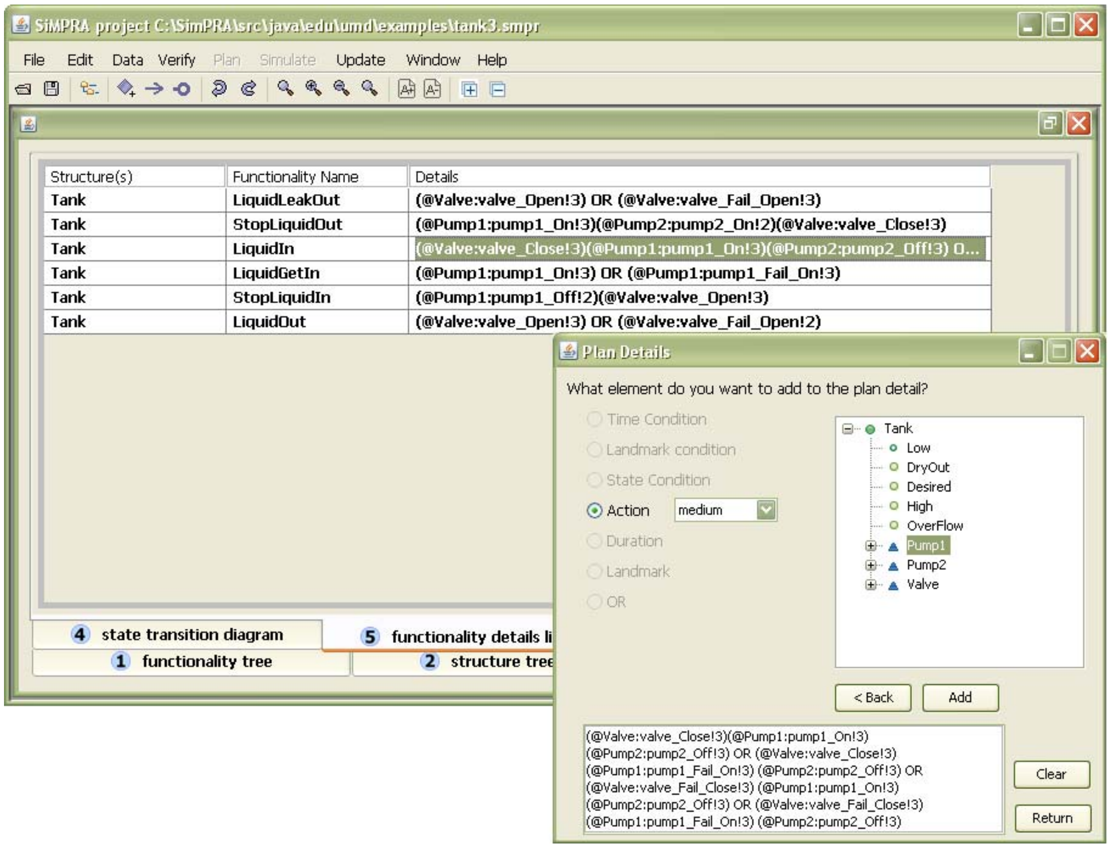


Figure 47: State transition diagrams for other elements of the holdup tank example

Tab 4, the state transition diagrams presented in Figure 46 and Figure 47, is where the relationships between different states of the elements and the functionalities provided by them are defined. System elements change the functionalities that they provide in response to changes in the states of the system. This knowledge provides insight into the behavior of the components of the system elements regardless of their interactions with the rest of the system.

Figure 48 presents the step in which the details of the functionalities including time, landmark, and state conditions, duration, and sub-goals (actions) are defined. These details link the elements' functionalities together and provide the main thrust of the planning process.


SiMPRA project C:\SimPRA\src\java\edu\umd\examples\tank3.smpr
File Edit Data Verify Plan Simulate Update Window Help
4 state transition diagram
5 functionality details li
1 functionality tree
2 structure tree
Plan Details
< Back
Add
Clear
Return

Structure(s)	Functionality Name	Details
Tank	LiquidLeakOut	(@Valve:valve_Open!3) OR (@Valve:valve_Fail_Open!3)
Tank	StopLiquidOut	(@Pump1:pump1_On!3)(@Pump2:pump2_On!2)(@Valve:valve_Close!3)
Tank	LiquidIn	(@Valve:valve_Close!3)(@Pump1:pump1_On!3)(@Pump2:pump2_Off!3) O...
Tank	LiquidGetIn	(@Pump1:pump1_On!3) OR (@Pump1:pump1_Fail_On!3)
Tank	StopLiquidIn	(@Pump1:pump1_Off!2)(@Valve:valve_Open!3)
Tank	LiquidOut	(@Valve:valve_Open!3) OR (@Valve:valve_Fail_Open!2)

What element do you want to add to the plan detail?

- Time Condition
- Landmark condition
- State Condition
- [✓ Action] medium
- Duration
- Landmark
- OR

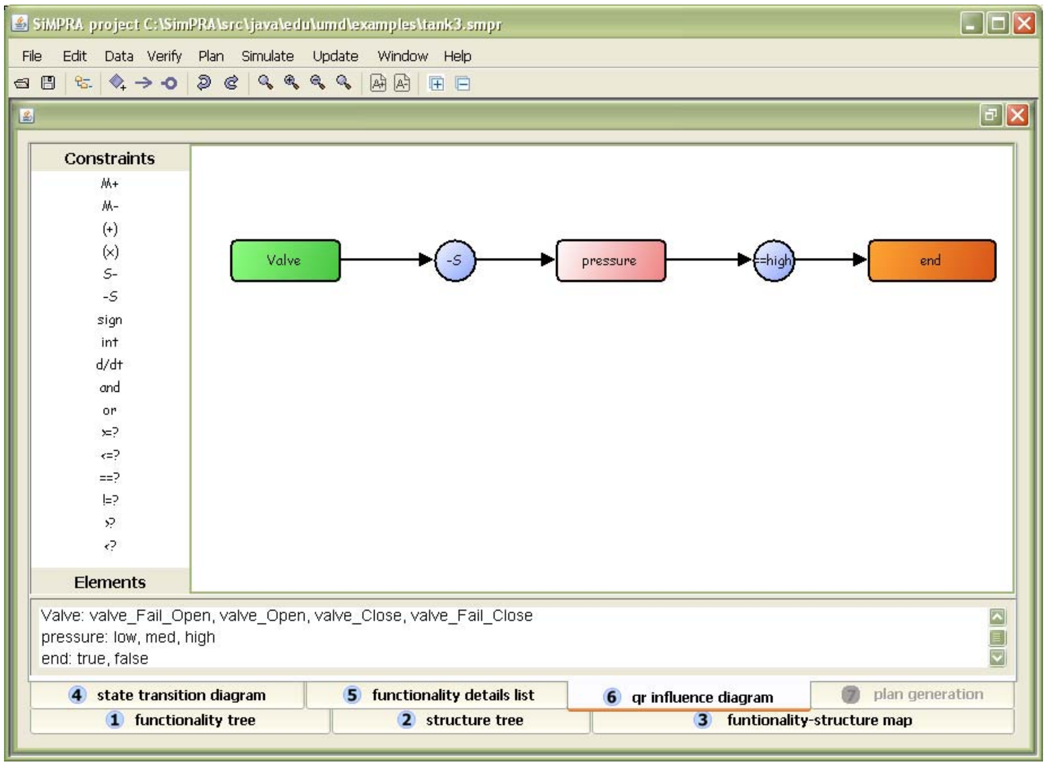
- Tank
  - Low
  - DryOut
  - Desired
  - High
  - OverFlow
  - Pump1
  - Pump2
  - Valve

```
(@Valve:valve_Close!3)(@Pump1:pump1_On!3)
(@Pump2:pump2_Off!3) OR (@Valve:valve_Close!3)
(@Pump1:pump1_Fail_On!3) (@Pump2:pump2_Off!3) OR
(@Valve:valve_Fail_Close!3) (@Pump1:pump1_On!3)
(@Pump2:pump2_Off!3) OR (@Valve:valve_Fail_Close!3)
(@Pump1:pump1_Fail_On!3) (@Pump2:pump2_Off!3)
```

Figure 48: Functionality details list

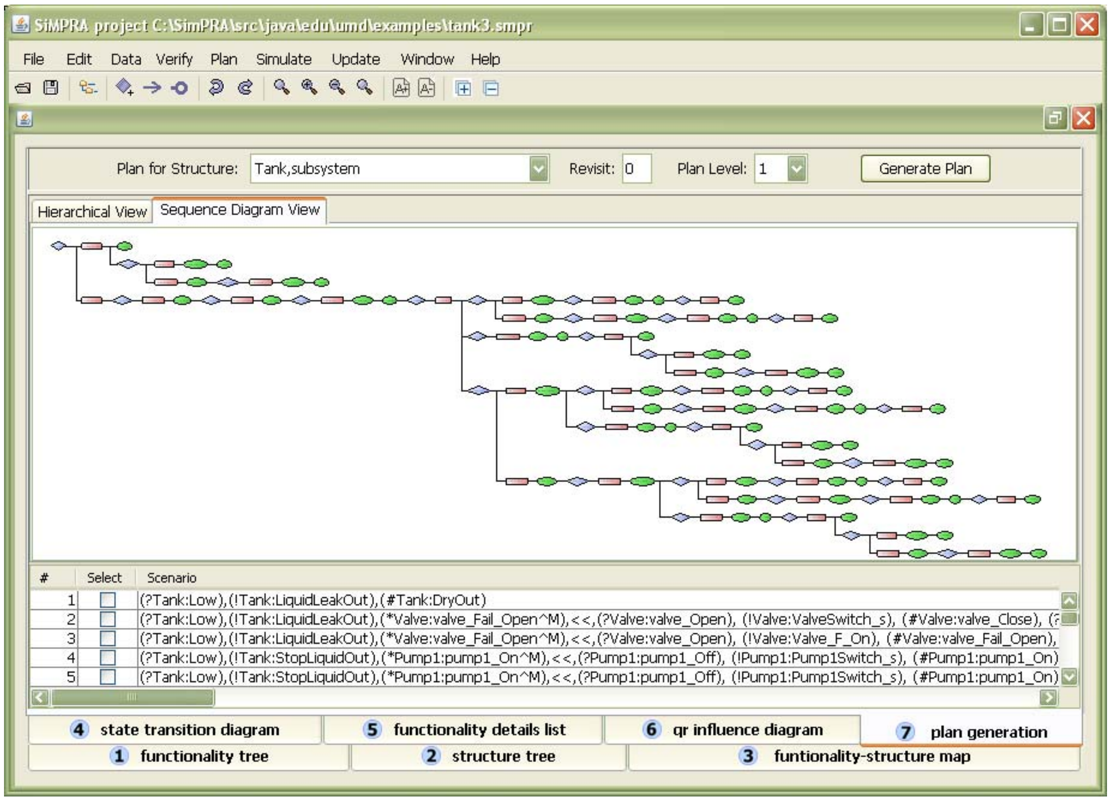
Figure 49 shows a screenshot of a qualitative reasoning influence diagram. The risk knowledge captured in this diagram is used to identify the risky or important scenarios from the field of all possible system behaviors. Please refer to section 5.4.3 for more detail.

Based upon the engineering knowledge solicited in these six steps (tabs) the planner automatically generates risk scenarios. As shown in Figure 50, these risk scenarios are output as a list of high-level scenarios which are also grouped together as a Generalized Event Sequence Diagram.


SiMPRA project C:\SimPRA\src\java\edu\umd\examples\tank3.smpr
File Edit Data Verify Plan Simulate Update Window Help
Constraints
M+
M-
(+)
(x)
S-
-S
sign
int
d/dt
and
or
>=?
<=?
==?
!=?
>?
<?
Elements
Valve
-S
pressure
==high
end
4 state transition diagram
5 functionality details list
6 qr influence diagram
7 plan generation
1 functionality tree
2 structure tree
3 funtionality-structure map

Valve: valve_Fail_Open, valve_Open, valve_Close, valve_Fail_Close
pressure: low, med, high
end: true, false

Figure 49: QR influence diagram for the tank example

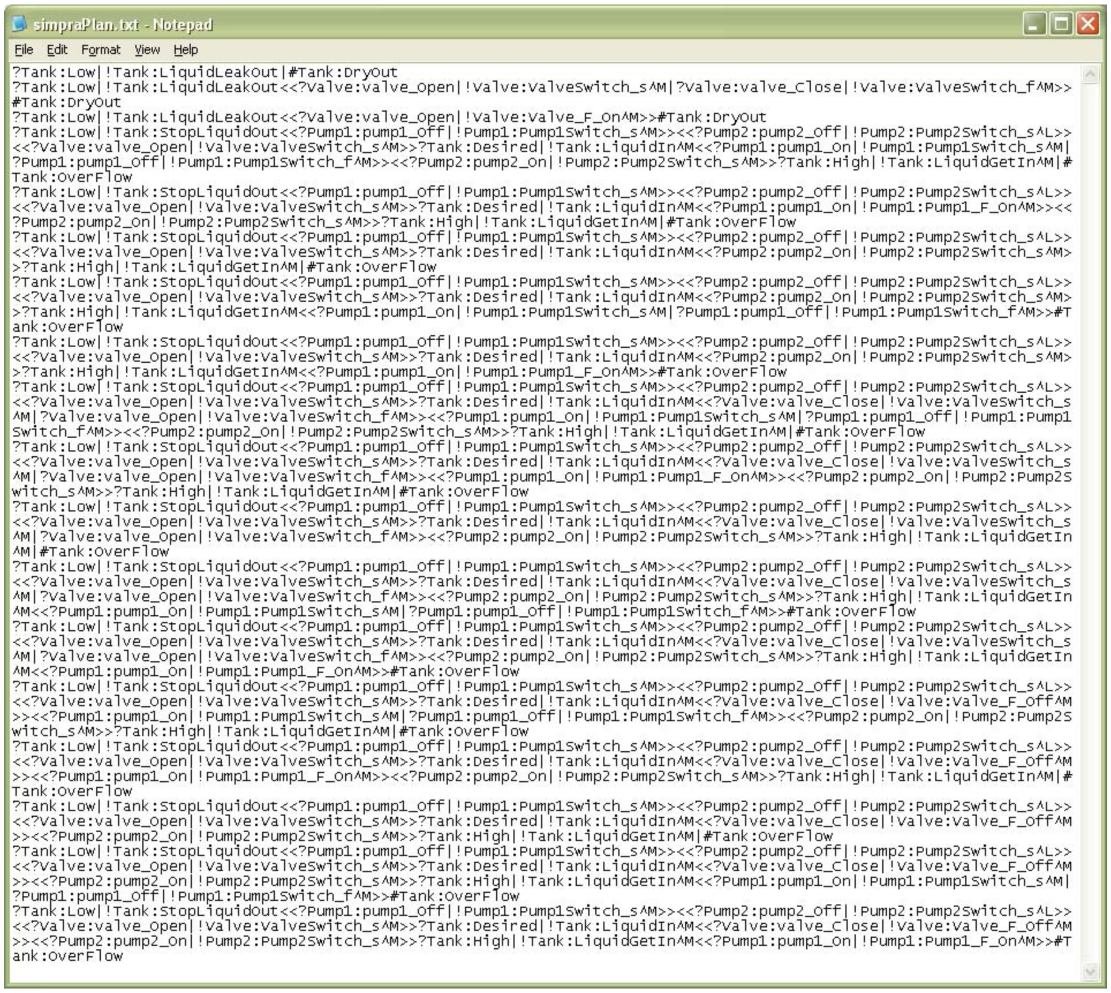

SiMPRA project C:\SimPRA\src\java\edu\umd\examples\tank3.smpr
File Edit Data Verify Plan Simulate Update Window Help
Generate Plan
Hierarchical View
Sequence Diagram View
4 state transition diagram
5 functionality details list
6 qr influence diagram
7 plan generation
1 functionality tree
2 structure tree
3 funtionality-structure map

Plan for Structure: Tank,subsystem Revisit: 0 Plan Level: 1

#	Select	Scenario
1	☐	(?Tank:Low),(!Tank:LiquidLeakOut),(#Tank:DryOut)
2	☐	(?Tank:Low),(!Tank:LiquidLeakOut),(*Valve:valve_Fail_Open^M),<<,(?Valve:valve_Open), (!Valve:ValveSwitch_s), (#Valve:valve_Close), (?
3	☐	(?Tank:Low),(!Tank:LiquidLeakOut),(*Valve:valve_Fail_Open^M),<<,(?Valve:valve_Open), (!Valve:Valve_F_On), (#Valve:valve_Fail_Open),
4	☐	(?Tank:Low),(!Tank:StopLiquidOut),(*Pump1:pump1_On^M),<<,(?Pump1:pump1_Off), (!Pump1:Pump1Switch_s), (#Pump1:pump1_On)
5	☐	(?Tank:Low),(!Tank:StopLiquidOut),(*Pump1:pump1_On^M),<<,(?Pump1:pump1_Off), (!Pump1:Pump1Switch_s), (#Pump1:pump1_On)

Figure 50: Planner output in the format of GESD and list of scenarios

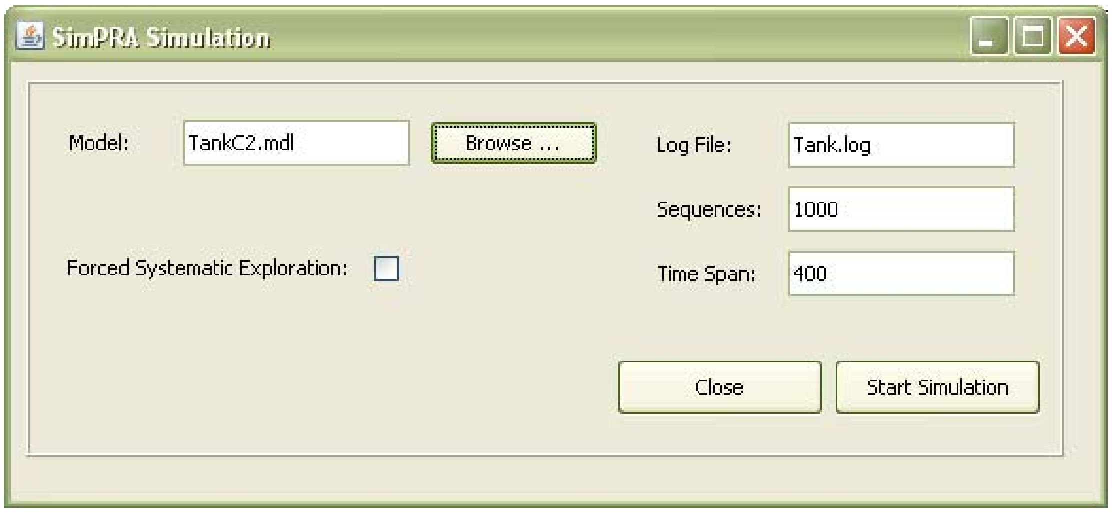
After generating the risk scenarios, the plan file (an example is shown in Figure 51) is passed to the scheduler for running the simulation.



```
?Tank:Low|!Tank:LiquidLeakOut|#Tank:DryOut
?Tank:Low|!Tank:LiquidLeakOut<<?Valve:valve_Open|!Valve:ValveSwitch_s^M|?Valve:valve_Close|!Valve:ValveSwitch_f^M>>
#Tank:DryOut
?Tank:Low|!Tank:LiquidLeakOut<<?Valve:valve_Open|!Valve:Valve_F_On^M>>#Tank:DryOut
?Tank:Low|!Tank:StopLiquidOut<<?Pump1:pump1_Off|!Pump1:Pump1Switch_s^M>><<?Pump2:pump2_Off|!Pump2:Pump2Switch_s^L>>
<<?Valve:valve_Open|!Valve:ValveSwitch_s^M>>?Tank:Desired|!Tank:LiquidIn^M<<?Pump1:pump1_On|!Pump1:Pump1Switch_s^M|
?Pump1:pump1_Off|!Pump1:Pump1Switch_f^M>><<?Pump2:pump2_On|!Pump2:Pump2Switch_s^M>>?Tank:High|!Tank:LiquidGetIn^M|#
Tank:OverFlow
?Tank:Low|!Tank:StopLiquidOut<<?Pump1:pump1_Off|!Pump1:Pump1Switch_s^M>><<?Pump2:pump2_Off|!Pump2:Pump2Switch_s^L>>
<<?Valve:valve_Open|!Valve:ValveSwitch_s^M>>?Tank:Desired|!Tank:LiquidIn^M<<?Pump1:pump1_On|!Pump1:Pump1_F_On^M>><<
?Pump2:pump2_On|!Pump2:Pump2Switch_s^M>>?Tank:High|!Tank:LiquidGetIn^M|#Tank:OverFlow
?Tank:Low|!Tank:StopLiquidOut<<?Pump1:pump1_Off|!Pump1:Pump1Switch_s^M>><<?Pump2:pump2_Off|!Pump2:Pump2Switch_s^L>>
<<?Valve:valve_Open|!Valve:ValveSwitch_s^M>>?Tank:Desired|!Tank:LiquidIn^M<<?Pump2:pump2_On|!Pump2:Pump2Switch_s^M>
>?Tank:High|!Tank:LiquidGetIn^M|#Tank:OverFlow
?Tank:Low|!Tank:StopLiquidOut<<?Pump1:pump1_Off|!Pump1:Pump1Switch_s^M>><<?Pump2:pump2_Off|!Pump2:Pump2Switch_s^L>>
<<?Valve:valve_Open|!Valve:ValveSwitch_s^M>>?Tank:Desired|!Tank:LiquidIn^M<<?Pump2:pump2_On|!Pump2:Pump2Switch_s^M>
>?Tank:High|!Tank:LiquidGetIn^M<<?Pump1:pump1_On|!Pump1:Pump1Switch_s^M|?Pump1:pump1_Off|!Pump1:Pump1Switch_f^M>>#T
ank:OverFlow
?Tank:Low|!Tank:StopLiquidOut<<?Pump1:pump1_Off|!Pump1:Pump1Switch_s^M>><<?Pump2:pump2_Off|!Pump2:Pump2Switch_s^L>>
<<?Valve:valve_Open|!Valve:ValveSwitch_s^M>>?Tank:Desired|!Tank:LiquidIn^M<<?Pump2:pump2_On|!Pump2:Pump2Switch_s^M>
>?Tank:High|!Tank:LiquidGetIn^M<<?Pump1:pump1_On|!Pump1:Pump1_F_On^M>>#Tank:OverFlow
?Tank:Low|!Tank:StopLiquidOut<<?Pump1:pump1_Off|!Pump1:Pump1Switch_s^M>><<?Pump2:pump2_Off|!Pump2:Pump2Switch_s^L>>
<<?Valve:valve_Open|!Valve:ValveSwitch_s^M>>?Tank:Desired|!Tank:LiquidIn^M<<?Valve:valve_Close|!Valve:ValveSwitch_s
^M|?Valve:valve_Open|!Valve:ValveSwitch_f^M>><<?Pump1:pump1_On|!Pump1:Pump1Switch_s^M|?Pump1:pump1_Off|!Pump1:Pump1
Switch_f^M>><<?Pump2:pump2_On|!Pump2:Pump2Switch_s^M>>?Tank:High|!Tank:LiquidGetIn^M|#Tank:OverFlow
?Tank:Low|!Tank:StopLiquidOut<<?Pump1:pump1_Off|!Pump1:Pump1Switch_s^M>><<?Pump2:pump2_Off|!Pump2:Pump2Switch_s^L>>
<<?Valve:valve_Open|!Valve:ValveSwitch_s^M>>?Tank:Desired|!Tank:LiquidIn^M<<?Valve:valve_Close|!Valve:ValveSwitch_s
^M|?Valve:valve_Open|!Valve:ValveSwitch_f^M>><<?Pump1:pump1_On|!Pump1:Pump1_F_On^M>><<?Pump2:pump2_On|!Pump2:Pump2S
witch_s^M>>?Tank:High|!Tank:LiquidGetIn^M|#Tank:OverFlow
?Tank:Low|!Tank:StopLiquidOut<<?Pump1:pump1_Off|!Pump1:Pump1Switch_s^M>><<?Pump2:pump2_Off|!Pump2:Pump2Switch_s^L>>
<<?Valve:valve_Open|!Valve:ValveSwitch_s^M>>?Tank:Desired|!Tank:LiquidIn^M<<?Valve:valve_Close|!Valve:ValveSwitch_s
^M|?Valve:valve_Open|!Valve:ValveSwitch_f^M>><<?Pump2:pump2_On|!Pump2:Pump2Switch_s^M>>?Tank:High|!Tank:LiquidGetIn
^M|#Tank:OverFlow
?Tank:Low|!Tank:StopLiquidOut<<?Pump1:pump1_Off|!Pump1:Pump1Switch_s^M>><<?Pump2:pump2_Off|!Pump2:Pump2Switch_s^L>>
<<?Valve:valve_Open|!Valve:ValveSwitch_s^M>>?Tank:Desired|!Tank:LiquidIn^M<<?Valve:valve_Close|!Valve:ValveSwitch_s
^M|?Valve:valve_Open|!Valve:ValveSwitch_f^M>><<?Pump2:pump2_On|!Pump2:Pump2Switch_s^M>>?Tank:High|!Tank:LiquidGetIn
^M<<?Pump1:pump1_On|!Pump1:Pump1Switch_s^M|?Pump1:pump1_Off|!Pump1:Pump1Switch_f^M>>#Tank:OverFlow
?Tank:Low|!Tank:StopLiquidOut<<?Pump1:pump1_Off|!Pump1:Pump1Switch_s^M>><<?Pump2:pump2_Off|!Pump2:Pump2Switch_s^L>>
<<?Valve:valve_Open|!Valve:ValveSwitch_s^M>>?Tank:Desired|!Tank:LiquidIn^M<<?Valve:valve_Close|!Valve:ValveSwitch_s
^M|?Valve:valve_Open|!Valve:ValveSwitch_f^M>><<?Pump2:pump2_On|!Pump2:Pump2Switch_s^M>>?Tank:High|!Tank:LiquidGetIn
^M<<?Pump1:pump1_On|!Pump1:Pump1_F_On^M>>#Tank:OverFlow
?Tank:Low|!Tank:StopLiquidOut<<?Pump1:pump1_Off|!Pump1:Pump1Switch_s^M>><<?Pump2:pump2_Off|!Pump2:Pump2Switch_s^L>>
<<?Valve:valve_Open|!Valve:ValveSwitch_s^M>>?Tank:Desired|!Tank:LiquidIn^M<<?Valve:valve_Close|!Valve:Valve_F_Off^M
>><<?Pump1:pump1_On|!Pump1:Pump1Switch_s^M|?Pump1:pump1_Off|!Pump1:Pump1Switch_f^M>><<?Pump2:pump2_On|!Pump2:Pump2S
witch_s^M>>?Tank:High|!Tank:LiquidGetIn^M|#Tank:OverFlow
?Tank:Low|!Tank:StopLiquidOut<<?Pump1:pump1_Off|!Pump1:Pump1Switch_s^M>><<?Pump2:pump2_Off|!Pump2:Pump2Switch_s^L>>
<<?Valve:valve_Open|!Valve:ValveSwitch_s^M>>?Tank:Desired|!Tank:LiquidIn^M<<?Valve:valve_Close|!Valve:Valve_F_Off^M
>><<?Pump1:pump1_On|!Pump1:Pump1_F_On^M>><<?Pump2:pump2_On|!Pump2:Pump2Switch_s^M>>?Tank:High|!Tank:LiquidGetIn^M|#
Tank:OverFlow
?Tank:Low|!Tank:StopLiquidOut<<?Pump1:pump1_Off|!Pump1:Pump1Switch_s^M>><<?Pump2:pump2_Off|!Pump2:Pump2Switch_s^L>>
<<?Valve:valve_Open|!Valve:ValveSwitch_s^M>>?Tank:Desired|!Tank:LiquidIn^M<<?Valve:valve_Close|!Valve:Valve_F_Off^M
>><<?Pump2:pump2_On|!Pump2:Pump2Switch_s^M>>?Tank:High|!Tank:LiquidGetIn^M|#Tank:OverFlow
?Tank:Low|!Tank:StopLiquidOut<<?Pump1:pump1_Off|!Pump1:Pump1Switch_s^M>><<?Pump2:pump2_Off|!Pump2:Pump2Switch_s^L>>
<<?Valve:valve_Open|!Valve:ValveSwitch_s^M>>?Tank:Desired|!Tank:LiquidIn^M<<?Valve:valve_Close|!Valve:Valve_F_Off^M
>><<?Pump2:pump2_On|!Pump2:Pump2Switch_s^M>>?Tank:High|!Tank:LiquidGetIn^M<<?Pump1:pump1_On|!Pump1:Pump1Switch_s^M|
?Pump1:pump1_Off|!Pump1:Pump1Switch_f^M>>#Tank:OverFlow
?Tank:Low|!Tank:StopLiquidOut<<?Pump1:pump1_Off|!Pump1:Pump1Switch_s^M>><<?Pump2:pump2_Off|!Pump2:Pump2Switch_s^L>>
<<?Valve:valve_Open|!Valve:ValveSwitch_s^M>>?Tank:Desired|!Tank:LiquidIn^M<<?Valve:valve_Close|!Valve:Valve_F_Off^M
>><<?Pump2:pump2_On|!Pump2:Pump2Switch_s^M>>?Tank:High|!Tank:LiquidGetIn^M<<?Pump1:pump1_On|!Pump1:Pump1_F_On^M>>#T
ank:OverFlow
```

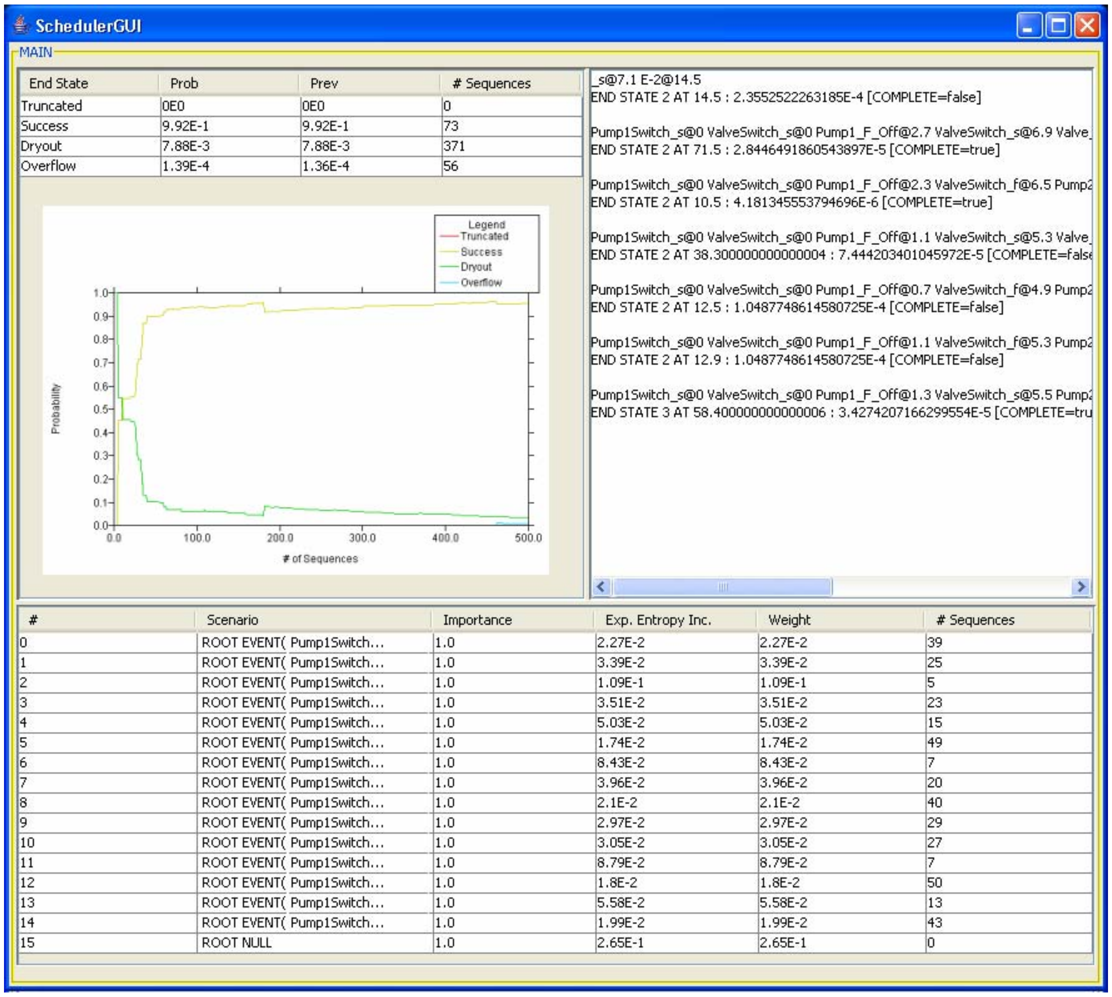
Figure 51: An example of a plan file

Figure 52 shows the screen used to run the simulation. Figure 53 shows the simulation runs and outputs. Updating algorithms are applied on the simulation log to update the plan. Figure 54 shows the potential functionality details suggested by the updating algorithm for updating the plan.



Model: TankC2.mdl
Log File: Tank.log
Sequences: 1000
Forced Systematic Exploration: ☐
Time Span: 400

Figure 52: Simulation run command screen

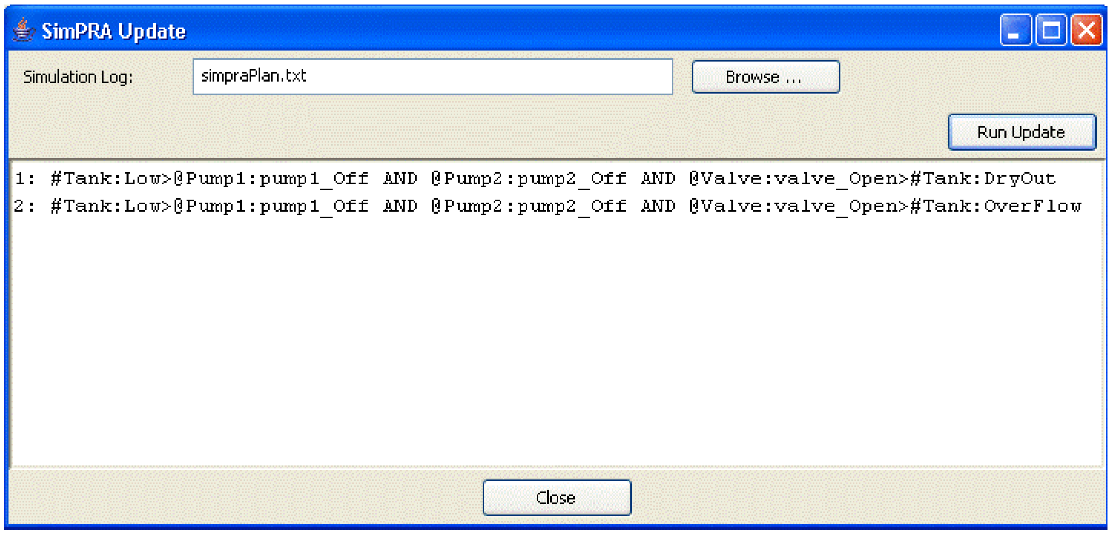


End State	Prob	Prev	# Sequences
Truncated	0E0	0E0	0
Success	9.92E-1	9.92E-1	73
Dryout	7.88E-3	7.88E-3	371
Overflow	1.39E-4	1.36E-4	56

```
_s@7.1 E-2@14.5
END STATE 2 AT 14.5 : 2.3552522263185E-4 [COMPLETE=false]

Pump1Switch_s@0 ValveSwitch_s@0 Pump1_F_Off@2.7 ValveSwitch_s@6.9 Valve_
END STATE 2 AT 71.5 : 2.8446491860543897E-5 [COMPLETE=true]

Pump1Switch_s@0 ValveSwitch_s@0 Pump1_F_Off@2.3 ValveSwitch_f@6.5 Pump2
END STATE 2 AT 10.5 : 4.181345553794696E-6 [COMPLETE=true]

Pump1Switch_s@0 ValveSwitch_s@0 Pump1_F_Off@1.1 ValveSwitch_s@5.3 Valve_
END STATE 2 AT 38.300000000000004 : 7.444203401045972E-5 [COMPLETE=false

Pump1Switch_s@0 ValveSwitch_s@0 Pump1_F_Off@0.7 ValveSwitch_f@4.9 Pump2
END STATE 2 AT 12.5 : 1.0487748614580725E-4 [COMPLETE=false]

Pump1Switch_s@0 ValveSwitch_s@0 Pump1_F_Off@1.1 ValveSwitch_f@5.3 Pump2
END STATE 2 AT 12.9 : 1.0487748614580725E-4 [COMPLETE=false]

Pump1Switch_s@0 ValveSwitch_s@0 Pump1_F_Off@1.3 ValveSwitch_s@5.5 Pump2
END STATE 3 AT 58.400000000000006 : 3.4274207166299554E-5 [COMPLETE=tru
```

#	Scenario	Importance	Exp. Entropy Inc.	Weight	# Sequences
0	ROOT EVENT(Pump1Switch...	1.0	2.27E-2	2.27E-2	39
1	ROOT EVENT(Pump1Switch...	1.0	3.39E-2	3.39E-2	25
2	ROOT EVENT(Pump1Switch...	1.0	1.09E-1	1.09E-1	5
3	ROOT EVENT(Pump1Switch...	1.0	3.51E-2	3.51E-2	23
4	ROOT EVENT(Pump1Switch...	1.0	5.03E-2	5.03E-2	15
5	ROOT EVENT(Pump1Switch...	1.0	1.74E-2	1.74E-2	49
6	ROOT EVENT(Pump1Switch...	1.0	8.43E-2	8.43E-2	7
7	ROOT EVENT(Pump1Switch...	1.0	3.96E-2	3.96E-2	20
8	ROOT EVENT(Pump1Switch...	1.0	2.1E-2	2.1E-2	40
9	ROOT EVENT(Pump1Switch...	1.0	2.97E-2	2.97E-2	29
10	ROOT EVENT(Pump1Switch...	1.0	3.05E-2	3.05E-2	27
11	ROOT EVENT(Pump1Switch...	1.0	8.79E-2	8.79E-2	7
12	ROOT EVENT(Pump1Switch...	1.0	1.8E-2	1.8E-2	50
13	ROOT EVENT(Pump1Switch...	1.0	5.58E-2	5.58E-2	13
14	ROOT EVENT(Pump1Switch...	1.0	1.99E-2	1.99E-2	43
15	ROOT NULL	1.0	2.65E-1	2.65E-1	0

Figure 53: Simulation runs and results



```
1: #Tank:Low>@Pump1:pump1_Off AND @Pump2:pump2_Off AND @Valve:valve_Open>#Tank:DryOut
2: #Tank:Low>@Pump1:pump1_Off AND @Pump2:pump2_Off AND @Valve:valve_Open>#Tank:OverFlow
```

Figure 54: Simulation updating screen

Appendix B: PSAM8 benchmark problem

The PSAM8 benchmark problem (discussed in section 6.1) has been proposed by NASA and worked on by a number of research teams using different risk assessment approaches and techniques. In this appendix, the details of this problem are presented.

Mission profile

An ion propulsion system is needed for a science mission to the outer solar system. Figure 55 depicts the mission phases, along with the propulsion system operating time during each phase in hours of Mission Elapsed Time (MET). Table 7 conveys the same information in tabular form. For those phases where the propulsion system only operates during part of the phase (e.g. Phases 4 & 5), thrust is continually provided from the beginning of the phase until the specified operating time expires.

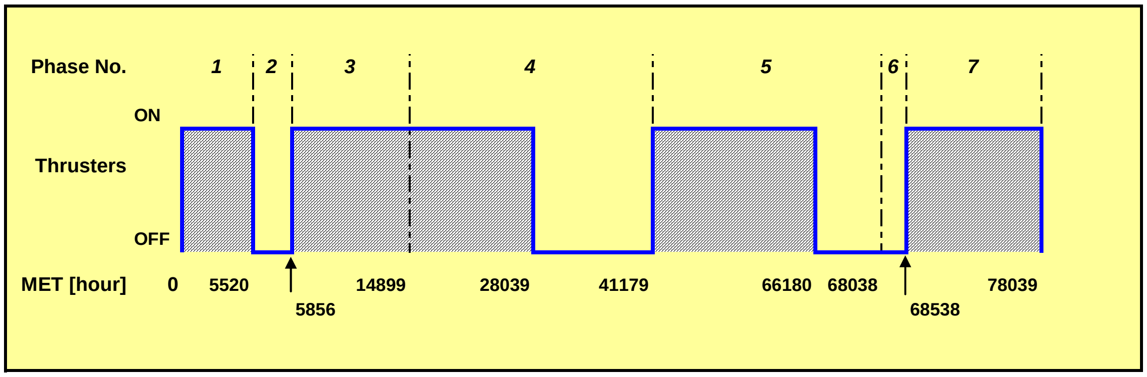


Figure 55: Propulsion system mission profile

Design description

The propulsion system consists of 5 thruster assemblies and a single propellant supply. Each assembly has:

1 propulsion power unit (PPU), and

2 ion engines

When an assembly is operating, the PPU provides power to just one ion engine. The other engine will be in a standby mode, unless failed.

During Phase 1 the success criterion is propulsion from 2 assemblies. In all subsequent phases where the propulsion system is operating, the success criterion is propulsion from 3 assemblies.

Table 7: Mission profile

Mission Phase No.	Duration (Hours)	Propulsion System Operating Time (Hours)
1	5520.0	5520.0
2	336.0	0
3	9043.2	9043.2
4	26280.0	13140.0
5	26858.5	25001.0
6	500.0	0
7	9501.5	9501.5

Relative to the assembly operation, the strategy is to use Assemblies 1 through 2 during the first phase. During subsequent phases, Assemblies 1 through 3 will furnish propulsion, if available.

Failure of an assembly causes it to be replaced by the lowest numbered standby assembly. For example, if assembly 1 fails in Phase 1, the strategy is to actuate Assembly 3. If no further failures occur during Phase 1, assemblies 2, 3 & 4 will furnish propulsion at the beginning of Phase 3.

Basically, standby assemblies remain in standby until they are needed to replace a failed assembly, and they are actuated in series (i.e., the lowest numbered assembly is first selected).

Figure 56 is a schematic of a thruster assembly. In assessing the mission risk input power failures are modeled separately, so the propulsion system model can ignore a loss of power from that support system.

The strategy for thruster assembly operation is to begin with power from the PPU going to Ion Engine A. Ion Engine A will continue to be the operating engine of the assembly until the engine fails. At that time the strategy is to:

Shutdown the PPU;

Switch the PPU to Ion Engine B; then

Reenergize the PPU and operate with Ion Engine B.

There are no intermediate switches between a PPU and the ion engines. All switches are integral to the PPU.

Figure 56 also depicts a propellant supply to each engine. The propellant is a noble gas from a common storage tank. The engine ionizes and accelerates the propellant to produce thrust. Since the propellant supply is part of the propulsion system, it must be included in the system model.

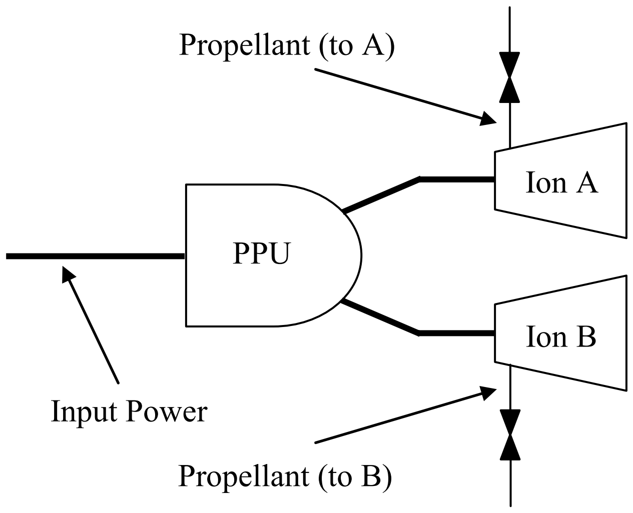


Figure 56: Thruster assembly schematic

Common cause failures (CCF) should be assessed using the conditional probability values from Table 8 by the CCF model of choice. No specific CCF model is endorsed, but any simplification or approximation of CCF probabilities must be based on calculations using the values below.

Table 8: Common cause failure modeling values

Group Size	Group Conditional Failure Probability [%]
2	8.0
3	4.0
4	2.0
5	1.0

Table 9 is a failure mode and effects analysis for the propulsion system. Reliability data are listed in Table 10.

Table 9: Failure mode and effects analysis

Component	Failure Mode	Effect
PPU	Fails to start on demand	Assembly failure
	Failure to operate	
	Failure to shutdown on demand	
Ion Engine A	Fails to start on demand	Loss of redundancy
	Failure to operate	
Ion Engine B	Fails to start on demand	Assembly failure
	Failure to operate	
Propellant Valve A	Failure to open on demand	Loss of Ion Engine A
	Failure to close on demand	System failure
	External leakage	
Propellant Valve B	Failure to open on demand	Loss of Ion Engine B
	Failure to close on demand	System failure
	External leakage	
Propellant tank	External leakage	System failure
Propellant distribution lines	External leakage	System failure

Table 10: Reliability data

Component Type	Failure Mode	Value
PPU	Fails to start on demand	1×10^{-4} (per demand)
	Failure to operate	1×10^{-6} (per hour)
	Failure to shutdown on demand	1×10^{-5} (per demand)
	Fails to switch to Ion Engine B	2×10^{-6} (per demand)
Ion Engine	Fails to start on demand	3×10^{-5} (per demand)
	Failure to operate	2×10^{-5} (per hour)
	Failure to shutdown on demand	3×10^{-6} (per demand)
Propellant Valve	Failure to open on demand	3×10^{-4} (per demand)
	Failure to close on demand	3×10^{-4} (per demand)
	External leakage	5×10^{-5} (per hour)
Propellant tank	External leakage	1×10^{-6} (per hour)
Propellant distribution lines	External leakage	1×10^{-6} (per hour)

Predicated upon the above mission and design descriptions, the time-dependent reliability of the propulsion system over the planned mission should be quantified.

Glossary of terms

****Please note that the following list of terms is ordered such that each new concept relies upon previously defined concepts. It is therefore, highly recommended that the list is read in order.****

- ***System:*** The Encarta dictionary defines a system as a combination of related elements organized into a complex whole. In the engineering field, a system usually means an assembly of mechanical and/or electronic components that function together as a unit. A system can be comprised of *subsystems* and/or *components*. Components are the smallest elements of a system.
- ***Functionality***: Use, action or purpose. Considered with an object, a normative relation of object to its use.
- ***Parameter:*** A quantity that defines a relatively constant characteristic of a system or function.
- ***Model***: An abstraction of a system. Models are used to obtain predictions of the behavior of systems, especially how one or more changes in various aspects of the modeled system would affect the other aspects of the system.
- ***State***: The condition that a system or component is in at a particular time. A system provides or denies its functionalities by remaining in or transitioning from one state to another. The state of a system can be defined by an optimal ensemble of system parameters which characterize it independent of its history and surroundings. When a system is in a state, its history of arriving at that state is irrelevant to the future of its behavior. Systems or components are always in one

and only one state and the number of states of a system is finite. A state can be presented by a set of conditions.

- ***End-state***: A state of the system that can be entered but either cannot be exited or for which the observation of subsequent system behavior holds no interest. A system must have at least one but may have several end-states. It is important to emphasize that in the context of this thesis an end-state is only defined for the system as a whole and not for its elements. Within the context of PRA and DPRA, end-states are normally specified as one of the discrete end-state types, which typically indicate the severity of the condition.
- ***Init-state***: The initial state of a system/subsystem or component. Each system or its element is supposed to be in an init-state at $T = t_0$.
- ***System Configuration***: This is defined here as a combination of the states of a system's components and it is indexed by a positive integer $c \in N$.
- ***System Status***: System status includes continuous process variables, (a real number vector $\overline{X}$) and a discrete system configuration (a positive integer ***c***). It is defined on $S = \Re^n \times N$. The process variables are governed by a set of deterministic equations $\frac{d\bar{x}}{dt} = \bar{f}_i(\bar{x}), \bar{x}(0) = \bar{x}_0, \bar{x} \in \Re^N$. 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****:* This term is used in two different contexts. An event can occur at the *system* level and at the *component* level. At the system level, following the convention of discrete event simulation, an event is defined as an instantaneous

occurrence that changes the system configuration $\delta : c_i \rightarrow c_j$. When used in the context of the component level, an event refers to the simple transition of a component from one state to another. In general, there are two kinds of events that can occur at either the system or the component level:

1. ***Random Events***: 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 a random event is a time-distributed component failure modeled by a Weibull distribution.
2. ***Deterministic Events***: Those events that are induced by the deterministic rules. An example of a deterministic event is that the threshold pressure or temperature is reached.

- ***Action****:* The transition of a system/subsystem from one state to another. Unlike an event at the system level, an action is defined by the transition of a system state instead of the system configuration.
- ***Scenario***: There are high-level scenarios and low-level (also called detailed) scenarios:

1. High-level Scenarios are the sequences of actions and events with the conditions that need to be met—such as reaching a particular state or passing a landmark—in between to take the system and its elements from one state to another. High-level scenarios act as a guide for what needs to be done under what circumstances to take the system from one state (usually the initial-state) to another (usually an end-state).

2. Low-level (also called detailed) scenarios are the list of actions and events with the exact time of their occurrence along with the changes in the system/subsystems/components states and observation of landmarks. Low-level scenarios are the real simulation instances recorded for future observations.

- ***Plan***: A plan is defined in this paper as a collection of high level scenarios.
- ***Planning****:* The process of modeling and automatically generating the plan.
- ***Event Sequence***: A system trajectory generated by the simulation model that consists of sequences of events along with the deterministic behavior of the system and its elements. Every event sequence should be unique. It is an instance of the system status evolution through the time line. $ES = \delta \times T$
- ***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 random drawing of realizations of event sequences from the event sequence space.
- ***Scheduling***: the process of controlling the generation of event sequences. This is accomplished 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 the occurrence of possible events.

Some of these concepts are illustrated in Figure 57 and Figure 58.

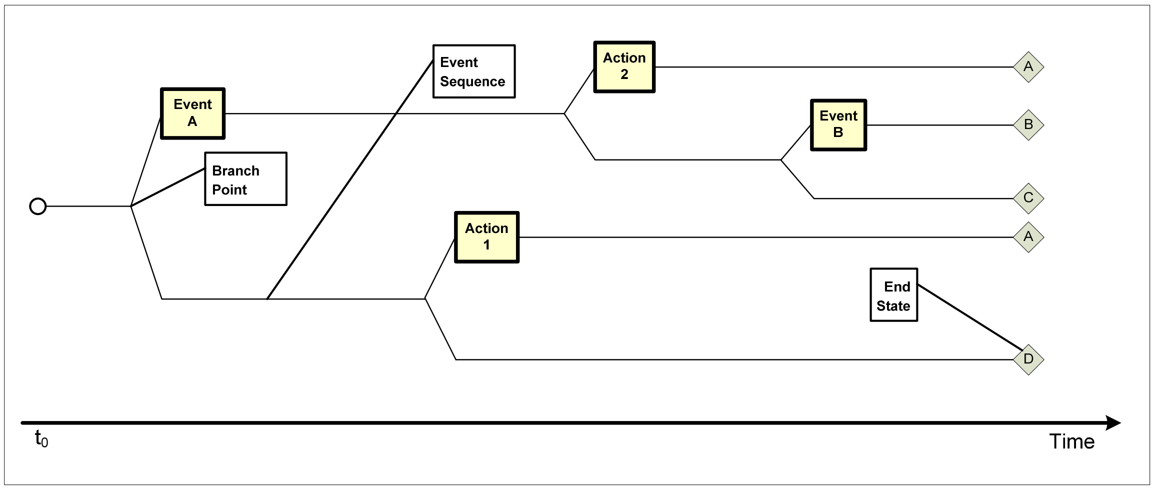


Figure 57: Illustration of DPRA terminology

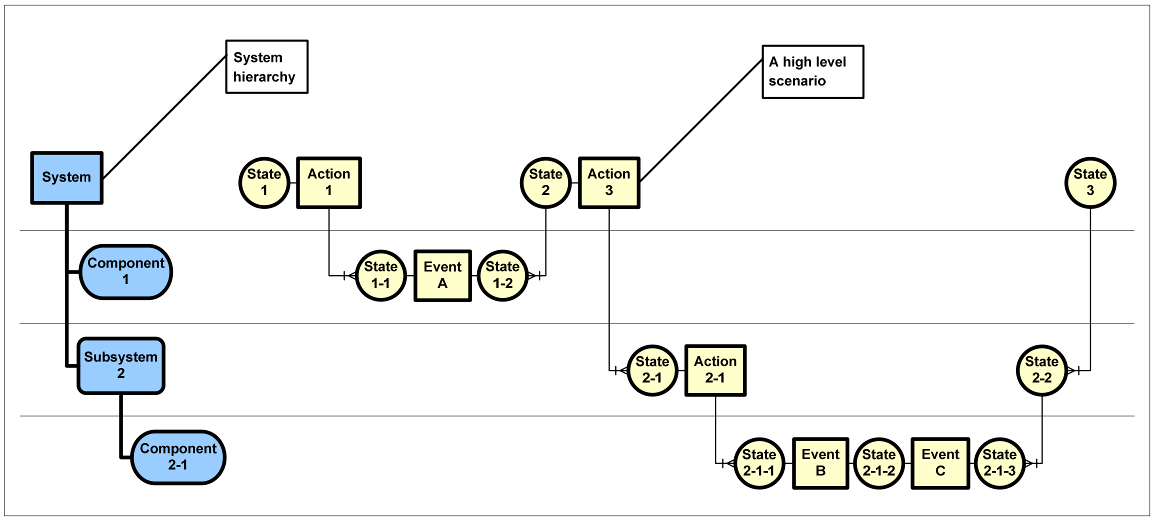


Figure 58: Illustration of planner terminology

References

Albus, J. S., and A. M. Meystel (2001). *Engineering of Mind: an introduction to the science of intelligent systems, Wiley Series on Intelligent Systems.* John Wiley & Sons, New York, NY.

Aldemir, T. (1987). Computer-Assisted Markov Failure Modeling Of Process-Control Systems, *IEEE Transactions On Reliability, 36*(1), pp. 133-149.

Aldemir, T. and E. Zio (1998). New Domain of Application: Discussion Group II, *Fifth International Workshop on Dynamic Reliability: Future Directions*.

Amari, S., G. Dill, et al. (2003). A new approach to solve dynamic fault trees, *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): pp. 3-25.

Amendola, A. and G. Reina (1981). Event Sequences And Consequence Spectrum - A Methodology For Probabilistic Transient Analysis, *Nuclear Science And Engineering 77*(3): pp. 297-315.

Azarkhail, M., and M. Modarres (2006). An Intelligent Agent-Oriented Approach to Risk Analysis of Complex Dynamic Systems with Applications in Planetary Missions, *Eighth International Conference on Probabilistic Safety assessment and Management (PSAM8).* New Orleans, LA.

Birnbaum, Z. W. (1969). On the importance of different components in a multicomponent system, chapter 11 in *Multivariate Analysis* (P.R.Krishnaiah, Ed), Academic Press.

Börger, E., and R. Stärk (2003). *Abstract State Machines: A method for high-level system design and analysis*, Springer.

Bucklew, J. A. (2004). *Introduction to Rare Event Simulation*, Springer.

Cacciabue, P. C., A. Carpignano, et al. (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): pp. 127-136.

Cacciabue, P. C. and G. Cojazzi (1994). A Human-Factors Methodology For Safety Assessment Based On The Dylam Approach, *Reliability Engineering & System Safety, 45*(1-2): pp. 127-138.

Campioni, L. and P. Vestrucci (2004). Monte Carlo importance sampling optimization for system reliability applications, *Annals Of Nuclear Energy, 31*(9): pp. 1005-1025.

Carson, J. S. (2004). Introduction to Modeling and Simulation, *Winter Simulation Conference2004 Proceedings*:9-16. Washington, DC.

Cepin, M. and B. Mavko (2002). A dynamic fault tree, *Reliability Engineering & System Safety, 75*(1): pp. 83-91.

Chang, Y.-H. (1999). *Cognitive Modeling and Dynamic Probabilistic Simulation of Operating Crew Response to Complex System Accident (ADS-IDACrew)*, Ph.D. Dissertation, University Of Maryland, College Park, MD.

Christensen, P. and M. Lind (1996). Functional Modeling for a Pipe Surveying Autonomous Submarine, *Fourth International Workshop on Functional Modeling of Complex Technical Systems*, (Mohammad Modarres, Ed.) Athens, Greece.

Clark, J., K. Fry, and R. Youngblood (2006). Mission Reliability Evaluation for a Space Propulsion System Phased-Mission Benchmark Problem. *Eighth International Conference on Probabilistic Safety assessment and Management (PSAM8),* New Orleans, LA.

Cojazzi, G. (1996). The DYLAM approach for the dynamic reliability analysis of systems *Reliability Engineering & System Safety, 52*(3): pp. 279-296.

Crowe, D., A. Feinberg (2001). *Design for reliability*, CRC Press.

Cruse, T. A. and S. Mahadevan (1997). *Reliability-based mechanical design*, Marcel Dekker.

Dang, V. (1998). Frameworks for Dynamic Risk Assessment and their Implications for Operator Modeling, *Fifth International Workshop on Dynamic Reliability: Future Directions*, Greenbelt, Maryland.

Deoss, D. L. and N. Siu (1989). *A Simulation Model for Dynamic System Availability Analysis*: Massachusetts Institute of Technology.

Devooght, J. (1998). Dynamic Reliability: The Challenges Ahead, *Fifth International Workshop on Dynamic Reliability: Future Directions*, Greenbelt, Maryland.

Devooght, J. and C. Smidts (1992). Probabilistic Reactor Dynamics.1. The Theory Of Continuous Event Trees, *Nuclear Science and Engineering, 111*(3): pp. 229-240.

Devooght, J. and C. Smidts (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): pp. 101-113.

Devooght, J. and C. Smidts (1996). Probabilistic dynamics as a tool for dynamic PSA, *Reliability Engineering & System Safety, 52*(3): pp. 185-196.

Dubi, A. (1998). Analytic approach & Monte Carlo methods for realistic systems analysis, *Mathematics And Computers In Simulation, 47*(2-5): pp. 243-269.

Dubi, A. and S. A. W. Gerstl (1980). Application Of Biasing Techniques To The Contribution Monte-Carlo Method, *Nuclear Science And Engineering, 76*(2): pp. 198-217.

Dugan, J. B. (1991). Automated-Analysis Of Phased-Mission Reliability, *IEEE Transactions On Reliability 40*(1): pp. 45 - 55.

Dugan, J. B. (2000). Galileo: A tool for dynamic fault tree analysis, *Computer Performance Evaluation, Proceedings*, *1786*: pp. 328-331.

Dugan, J. B., and K. J. Sullivan, et al. (2000). Developing a low-cost high-quality software tool for dynamic fault-tree analysis, *IEEE Transactions On Reliability*, *49*(1): pp. 49-59.

Fikes, R. and N. Nilson (1971). Strips: A new approach to the application of theorem proving to problem solving, *Articial Intelligence*, *2*(3-4): pp. 189-208.

Fragola, J. R. (1995). *Probabilistic Risk Assessment of the Space Shuttle*, NASA CR-197808, SAIC, New York.

Ghallab, M., D. Nau, and P. Traverso (2004). *Automated Planning, Theory and Practice*, Elsevier.

Garrett, C. J., S. B. Guarro (1995). The Dynamic Flowgraph Methodology For Assessing The Dependability Of Embedded Software Systems, *IEEE Transactions On Systems Man And Cybernetics 25*(5): pp. 824-840.

Groen, F. J., C. S. Smidts (2002). QRAS - the Quantitative Risk Assessment System, *Reliability and Maintainability Symposium, 2002. Proceedings,* Seattle, WA.

Groen, Frank, Hamed Nejad, Yunwei Hu, Thiago Pirest, Dongfeng Zhu, and Ali Mosleh (2005). Simulation-Based Probabilistic Risk Analysis, *Report of Research Activities,* University of Maryland, College Park.

Haghani, N., (2007). SimPRA Approach to Risk Analysis: An Application, *M.S. research paper,* University of Maryland, College Park.

Horrocks, I. (1999). *Constructing the User Interface with Statecharts*, Addison Wesley.

Houghton, Martin, et al. (2006). *LRO Critical Design Review (CDR): Mission Design and Operations*, <http://lunar.gsfc.nasa.gov/> , p. 13.

Houtermans, M., G. Apostolakis (2002). The dynamic flowgraph methodology as a safety analysis tool: programmable electronic system design and verification, *Safety Science, 40*(9): pp. 813-833.

Hsueh, K. S. and A. Mosleh (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): pp. 297-314.

Hu, Y., F. Groen, and A. Mosleh (2004). An Entropy-Based Exploration Strategy in Dynamic PRA, *International Conference on Probabilistic Safety Assessment and Management (PSAM7),* Berlin, Germany.

Hu, Y. (2005). *A Guided Simulation Methodology for Dynamic Probabilistic Risk Assessment of Complex Systems*, Ph.D Dissertation University of Maryland, College Park, MD.

Hu, Y., H. Nejad, D. Zhu, and A. Mosleh (2006). Solution of the Phased-Mission Benchmark Problem Using the SimPRA Dynamic PRA Approach, *Eighth International Conference on Probabilistic Safety Assessment and Management (PSAM8),* New Orleans, LA.

Kaplan, S. and B. J. Garrick (1981). On the Quantitavive Definition of Risk *Risk Analysis 1*: pp. 11-27.

Kaplan, Stan, Yacov Y. Haimes, and B. John Garrick (2001). Fitting Hierarchical Holographic Modeling into the Theory of Scenario Structuring and a Resulting Refinement to the Qualitative Definition of Risk, *Journal of Risk Analysis, 21*, No.5.

Knudsen, J., and C. Smith (2006). Space Propulsion System Phased-Mission Probability Analysis Using Conventional PRA Methods, *Eighth International Conference on Probabilistic Safety assessment and Management (PSAM8)*, New Orleans, LA.

Labeau, P.-E. and E. Zio (2001). *Biasing schemes in component-based and system-based Monte Carlo algorithms in system engineering*. Esrel, Torino, Italy.

Labeau, P. E., C. Smidts (2000). Dynamic reliability: towards an integrated platform for probabilistic risk assessment, *Reliability Engineering & System Safety, 68*(3): pp. 219-254.

Labeau, P. E. and E. Zio (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): pp. 347-360.

Labeau, P. E. and E. Zio (2002). Procedures of Monte Carlo transport simulation for applications in system engineering, *Reliability Engineering & System Safety 77*(3): pp. 217-228.

Labeau, P. E., Carol Smidts, and S. Swaminathan, Dynamic reliability: towards an integrated platform for probabilistic risk assessment, *Reliability Engineering & System Safety*, *68*(3), pp. 219-254.

Lindley, D. V. (1956). On the Measure of Information Provided by an Experiment, *Annals of Statistics, 27*(4): pp. 986-1005.

Mahadevan, Sankaran, and Natasha L. Smith (2003). System Risk Assessment and Allocation in Conceptual Design *NASA/CR-2003-212162.*

Mahadevan S, and P. Raghothamachar (2000). Adaptive Simulation for System Reliability Analysis of Large Structures, *Computers & Structures*, *77* (6): pp. 725-734.

Mandelli, D., T. Aldemir, and E. Zio (2006). An Event Tree/ Fault Tree/ Embedded Markov Model Approach for the PSAM-8 Benchmark Problem Concerning a Phased Mission Space Propulsion System, *Eighth International Conference on Probabilistic Safety assessment and Management (PSAM8),* New Orleans, LA.

Marseguerra, M. and E. Zio (1993). Nonlinear Monte Carlo Reliability Analysis With Biasing Towards Top Event, *Reliability Engineering & System Safety, 40*(1): pp. 31-42.

Marseguerra, M., E. Zio, et al. (2002). Biased Monte Carlo unavailability analysis for systems with time-dependent failure rates, *Reliability Engineering and System Safety*, *76*(1): pp. 11-17.

Marseguerra, M., E. Zio (1998). A concept paper on dynamic reliability via Monte Carlo simulation, *Mathematics And Computers In Simulation, 47*(2-5): pp. 371-382.

Marseguerra, Marzio, E. Zio, E. Patelli, F. G. Ventura, and G. Mingrone (2000). Monte Carlo simulation of contaminant release from a radioactive waste deposit, *Mathematics and Computers in Simulation, 62(3):* pp. 421 - 430.

Matsuoka, T. (2004). Improvement of the GO-FLOW Methodology, *Proceedings of the PSAM 7 ESREL 2004 conference*, Berlin, Germany.

Matsuoka, T. and M. Kobayashi (1988). GO-FLOW: A New Reliability Analysis Methodology *Nuclear Science and Engineering, 98*: pp. 64-78.

Modarres, M. (1996). Functional Modeling for Integration of Human-Software-Hardware in Complex Physical Systems, *Fourth International Workshop on Functional Modeling of Complex Technical Systems*, Athens, Greece :75-90.

Mosleh, A. and V. Bier (1992). On Decomposition and Aggregation Error in Estimation: Some Basic Principles and Examples, *Risk Analysis, 12*(2): pp. 203-214.

Mosleh, A., F. Groen, Y. Hu, H. Nejad, D. Zhu, and T. Piers (2004). *Simulation-Based Probabilistic Risk Analysis*, Center for Risk and Reliability, University of Maryland, College Park, MD.

Nau, D., T.C. Au, O. Ilghami, U. Kuter, W. Murdock, D. Wu, and F.Yaman (2003). SHOP2: An HTN Planning System; *Journal of Artificial Intelligence Research*, *20*, pp. 379-404.

Nejad, H., A. Mosleh (2005). Automated Risk Scenario Generation Using System Functional and Structural Knowledge, *Proceedings of ASME International Mechanical Engineering Congress and Exposition,* Orlando, FL.

Nivolianitou, Z., A. Amendola (1986). Reliability-Analysis Of Chemical Processes By The Dylam Approach, *Reliability Engineering & System Safety, 14*(3): pp. 163-182.

NRC (1975). *Reactor Safty Study: an Assessment of Accident Risks in US Commercial Nuclear Power Plants*. Nuclear Regulatory Commission.

NRC (1995). Review of NRC's Individual Plant Examination (IPE) Program, *OIG/95A-16*, http://www.nrc.gov/reading-rm/doc-collections/insp-gen/1996/95a-16.html, Nuclear Regulatory Commision.

Olivia S., M. Yau (2006). Advanced PRA Tool Benchmark for Space System Risk Using the Dynamic Flowgraph Methodology, *Eighth International Conference on Probabilistic Safety assessment and Management (PSAM8)*, New Orleans, LA.

Opperhauser, K. (2006). Lunar Reconnaissance Orbiter, *NASA/Goddard Space Flight Center*. Last Accessed: 7 May 2006. Available at: < http://lunar.gsfc.nasa.gov/>.

Pickard (1983). Pickard, Lowe and Garrick, Inc. Seabrook station probabilistic safety assessment (SSPSA), Prepared for Public Service Company of New Hampshire and Yankee Atomic Electric Company (PLG-0300). Newport Beach, CA, Lowe and Garrick Inc.

Priest, John W. (1988). *Engineering design for producibility and reliability*, Marcel Dekker Inc.

Robinson, S. (2004). *Simulation: The Practice of Model Development and Use*, John Wiley & Sons.

Rubinstein, R. Y. and B. Melamed (1998). *Modern Simulation and Modeling*, Wiley Interscience.

Shannon, C. (1948). Mathematical theory of communication, *The Bell Labs Technical Journal, 27*: pp. 379-457.

Shannon, C. E. and W. Weaver (1949). *The mathematical theory of communication*, University of Illinois.

Siu, N. (1994). Risk Assessment For Dynamic-Systems - An Overview, *Reliability Engineering & System Safety, 43*(1): pp. 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): pp. 189-206.

Smidts, C. and J. Devooght (1992). Probabilistic reactor dynamics. II. A Monte Carlo study of a fast reactor transient, *Nuclear Science and Engineering, 111*: pp. 241-256.

Stamatelatos, M., (2000). Probabilistic Risk Assessment: What Is It And Why Is It Worth Performing It, Available at: <http://www.hq.nasa.gov/office/codeq/qnews/pra.pdf>*,* Office of Mission and Safety Assurance, NASA.

Stamatelatos, M., (2004). NASA Perspective on Risk Assessment, *NRC Workshop on Stepping Stones in Space,* NASA.

Stamatelatos, M., et al. (2002). *Probabilistic Risk Assessment Procedures Guide for NASA Managers and Practitioners*, NASA.

Suh, N. P. (2001). *Axiomatic Design: Advances and Applications*, Oxford University Press.

Suh, N.P. (2005). *Complexity: Theory and Applications*, MIT-Pappalardo Series in Mechanical Engineering, Oxford University Press.

Swaminathan, S. and C. Smidts (1999). The event sequence diagram framework for dynamic probabilistic risk assessment *Reliability Engineering & System Safety 63*(1): pp. 73-90.

Swaminathan, S. and C. Smidts (1999). Identification of missing scenarios in ESDs using probabilistic dynamics, *Reliability Engineering & System Safety, 66*(3): pp. 275-279.

Swaminathan, S. and C. Smidts (1999). The mathematical formulation for the event sequence diagram framework, *Reliability Engineering & System Safety, 65*(2): pp. 103-118.

Tombuyses, B., P. R. DeLuca (1998). Backward Monte Carlo for probabilistic dynamics, *Mathematics And Computers In Simulation, 47*(2-5): pp. 493-505.

Wilson, R.A., and Keil, F. (2001). *The MIT Encyclopedia of the Cognitive Sciences (MITECS)*, The MIT Press.

Xu, H., C. Brown, J. Dugan, and L. Meshkat (2006). A Dynamic Fault Tree of a Propulsion System, *Eighth International Conference on Probabilistic Safety Assessment and Management (PSAM8)*, New Orleans, LA.

Zhu, D., A. Mosleh, and C. Smidts (2005). Software Modeling Framework for Dynamic PRA, *European Safety and Reliability Conference*: pp. 2099-2107, Tri-City, Poland.

Zhu, D., A. Mosleh, and C. Smidts (2006). A Framework to Integrate Software Behavior into Dynamic Probabilistic Risk Assessment, *Reliability Engineering and System Safety*.

Zio, E., and M. Librizzi (2006). Direct Monte Carlo Simulation for the Reliability Assessment of a Space Propulsion System Phased Mission, *Eighth International Conference on Probabilistic Safety assessment and Management (PSAM8),* New Orleans, LA.