

Bridging the gap between Fault Tree Analysis Modeling Tools and the Systems being Modeled

Ragavan Manian • University of Virginia • Charlottesville
David W. Coppit • University of Virginia • Charlottesville
Kevin J. Sullivan • University of Virginia • Charlottesville
Joanne Betcha Dugan • University of Virginia • Charlottesville

Key Words: Dynamic Fault Tree, Gate, Basic Event, Markov Model, Ordinary Differential Equations

SUMMARY AND CONCLUSIONS

Fault tolerant systems comprise of subsystems that interact with each other in complex ways [Joh89]. As a result, modeling the reliability of these systems calls for sophisticated analytical techniques. A powerful technique to address this issue is dynamic fault tree analysis [Dug92]. But because the semantics on which Dynamic Fault Trees are based are themselves complex, there was a question of whether these dynamic fault tree models were indeed faithful representations of the systems, and whether their analytical models are correct. The existing definitions of the modeling constructs employed by Dynamic Fault Trees were not precise or consistent enough, leading to ambiguities in the interpretation of the dynamic gates.

In this paper, we present our efforts at making precise the dynamic fault tree modeling and evaluation process. Our aim was to improve our confidence in the equivalence between system failure behavior and the corresponding fault tree model. By rigorously specifying fault trees and their constituent gates and basic events, we were able to reason about the correctness of the fault tree model, the analytical (Markov) model, and the numerical solution to the analytical model.

1. INTRODUCTION

Fault Trees have been widely used for reliability analysis for over forty years. They were first developed in the 1960s to facilitate analysis of the Minuteman missile system [Wat61] and have been supported by a rich body of research since their inception. In the present context, fault trees can be classified as either static or dynamic, depending on the class of systems that can be modeled using them. Static trees use purely combinatorial logic gates such as AND and OR. Static fault trees are translated into Binary Decision Diagrams (BDDs) and solved [Doy95]. Although static fault trees are

more commonly used, they are limited in their modeling capacity to systems that have no sequential relationships among their component failures. Component failure sequences are best captured using dynamic fault trees. Dynamic fault trees use Markov models as analytical representations. These Markov models are converted into differential equations and solved numerically for the given time period using an Ordinary Differential Equation (ODE) solver [Tri82].

Dynamic fault trees use four gates as the primary modeling elements- the Sequence Enforcing gate (SEQ), the Functional Dependency gate (FDEP), the Priority And gate (PAND) and the Cold, Hot and Warm Spare gates (CSP, HSP and WSP). SEQ gates only allow component failures in the order specified by its inputs; FDEP gates fail their dependent inputs based on whether their trigger input is failed or not; PAND gates output a failure if their inputs fail in order, and the CSP, HSP and WSP gates model primary-spare relationships. These gates are used in conjunction with the static gates viz. AND, OR and k-of-m gates. Using these, one can model a wider range of systems than can be modeled using just static gates.

However, in order to precisely model systems using fault trees, the analyst should have a clear understanding of dynamic gate behavior. This may be achieved by studying the specifications for these gates, their interactions, and the models generated from them. Also in order to reason about the correctness of the analysis, the problem must be decomposed into elementary blocks and the software implementation should closely follow this system decomposition. We developed rigorous specifications for the fault tree constituents (gates, basic events, and complete fault trees). These specifications were both formal [Cop98] and informal, and covered many issues concerning dynamic fault tree analysis and

evaluation, some of which are discussed in this paper.

2. MOTIVATION: RELIABILITY ANALYSIS OF REAL SYSTEMS

The motivation for our work came from collaborating with reliability engineers involved in designing and assessing real systems. Studying these systems made it apparent to us that we needed to define the modeling constructs, resolve ambiguities in specifying these constructs, and consolidate these specifications. The modeling constructs were the gates and basic events, the ambiguities were caused due to the differences in interpreting these constructs by the analyst and by the software developer, and consolidation involved incorporating new features and correcting errors found in previous implementations. Issues raised by these studies included simultaneous failures of basic events or sub-trees, shared spares, dormancy, coverage [Doy95a], time-to-failure distributions and basic event replication.

Word descriptions for four example system configurations are presented below, followed by a discussion of some of the ambiguities they uncovered. These examples were abstracted from real systems modeled by practicing reliability analysts.

Example 1: *A subsystem consists of 8 active units and 2 spares. The spares can fail before being switched into active use, but do so at a lower rate than the active units. Further, the spares are shared between the active units. The first two active units to fail claim the spares. The subsystem fails when fewer than 3 active units remain.*

Example 2: *A particular card has a bus interface unit on it. When the bus interface fails, it's failure causes the system bus (and hence the system) to fail. However if any other component on the card fails before the bus interface unit, then the card is taken off line, and a later bus interface failure can never fail the system bus.*

Example 3: *The system consists of a proprietary component A, a controller and a power converter. When A fails, it takes down the controller and power converter*

UNLESS the controller had already failed. If the controller fails before A then the power converter is isolated from the effects of A's failure.

Example 4: *Data obtained from studying a certain component's times to failure (TTF) showed that the failure rate (number of component failures per unit time) was not constant as in the case of an exponential distribution of TTF. Instead, the failure (or hazard) rate monotonically decreased with time. On further numerical analysis, it was observed that the TTF distribution satisfied a Weibull relation [Wei51]. It is required to use this component in a dynamic fault tree configuration.*

The four cases described above presented very different failure configurations. In the first two cases, it was fairly straightforward for experts to develop dynamic fault tree models. However, the experienced reliability engineer who was a novice dynamic fault tree user was unsure as to the precise operation and interaction of the gates and was unconvinced when presented with the solution. This interaction pointed out the need for user-friendly precise specifications of the dynamic gates, to aid the reliability engineers to confidently develop correct models. The third example exposed an ambiguity in the specification and thus our implementation of the PAND gate: Would (or should) a PAND gate fire if the input events occur simultaneously?. Example 4 involves modeling a nonhomogenous Markov model, as the transition rates are required to be time-dependent. At the time of the request, nonhomogeneous Markov models were not supported.

As a result of our interactions with users of the dynamic fault tree model, we embarked upon a task of rigidly specifying the behavior of the dynamic gates both formally (to aid implementation) and informally (to aid the user). We defined a conceptual break-up of the problem into abstract modules, and developed the software to closely follow the abstractions. The result was an enhanced understanding of how to model and solve systems using dynamic fault trees as reflected in the software.

2. CONCEPTUALIZATION OF PROGRAM FLOW

marked by hazard rates. Thus a "1" entry in a state indicates that the corresponding basic event has one operational replicate. Replicates are identical instances of the same component, and replication is used to reduce the size of the Markov model by lumping together basic events.

5.1. PAND absorbing vs failed states

This case serves to illustrate the fact that absorbing states and failed states do not necessarily coincide. The PAND gate is specified as producing a "Failed" output if its inputs fail in order from left to right. On the other hand, if the inputs do not fail in order, then the gate does not output "Failed". This behavior is represented using a Markov diagram in Figure 4. The issue here is to distinguish between Failed and Absorbing states. The states marked "Oper" and "Fail" are both absorbing states, whereas only one of them corresponds to the PAND gate outputting "Failed".

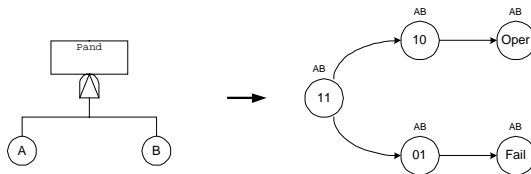


Figure 4: PAND to Markov Chain Translation

5.2. Dormancy factor for warm spare gates

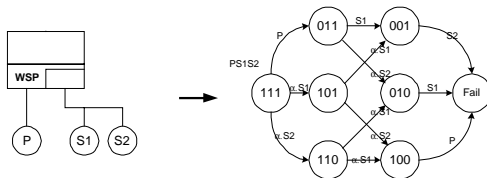


Figure 5: Accounting for Dormancy Factor in Warm Spares

Warm spares fail at a reduced rate compared to hot spares. This reduction is given by a dormancy factor corresponding to the basic event. In Figure 5 above, the dormancy factors for the two spare basic events S1 and S2 are marked by α . The point to note in modeling dormancy with Markov chains is that one should differentiate between in-use and dormant spares,

adjust their failure rates and construct the transitions accordingly.

5.3. Simultaneous failures due to FDEPs

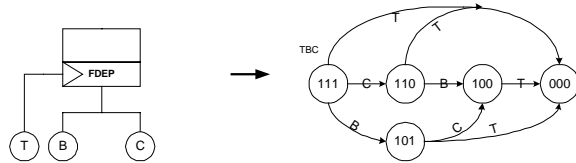


Figure 6: Simultaneous failures due to FDEP trigger failing

Functional Dependency (FDEP) gates can cause simultaneous failures. This is illustrated in Figure 6 above by transitions from states with more than one functional component to the state marked "000". Simultaneous failures of basic events can lead to difficulties in distinguishing between strictly sequential and coincidental failures, and can lead to potential conflicts in switching between spares in a shared spare pool.

5.4. Shared spare pools and state "splitting"

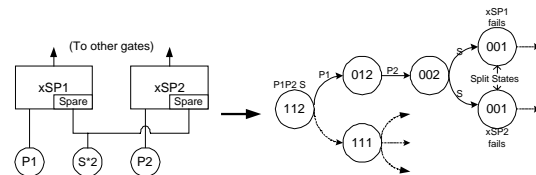


Figure 7: Shared Spare leading to State Splitting

In figure 7 above, the two spare gates xSP1 and xSP2 are shown to be sharing a common pool of spares (two replicates of S, denoted by S*2). A fragment of the corresponding Markov chain is also shown. Of particular interest is the failure path given by (the failures of) P1 followed by P2 followed by one of the two S's. In the state marked "002", both xSP1 and xSP2 are assumed to have switched to the spares. But because of the identical nature of the shared replicated spares, it is not known which replicate is used by which spare gate. Thus when one of the spares fail, it may cause the failure of either of the two spare gates, leading to the notion of "split states".

5.5. Cascading FDEPs

A cascading Functional Dependency (FDEP) effect is obtained in configurations such as the

one shown in Figure 8. The failure of the trigger to the first FDEP, “T” will lead to its dependent

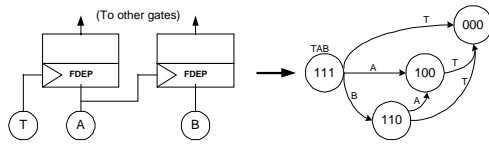


Figure 8: Multiple Cascading FDEPs

“A” failing, and the failure of A will cascade to the next FDEP, where A is the trigger and B is the dependent. The possibility of this domino effect requires the implementation to iteratively propagate failures through all the FDEP gates with interconnected dependencies, until a stable state is reached.

5.6. Replication (aggregation) of basic events

In most cases, the current replication count of a basic event can be multiplied with its hazard rate to obtain the Markov transition rate. However in the case of spares, adjustments must be made for dormancy factors, and the number of in-use replicates. The status (operational or failed) of a replicated basic event can only be defined in the context of an evaluating gate. For instance, an AND gate will evaluate a replicated basic event to be “failed” if and only if all the replications of the basic event have failed whereas a k-of-m gate will evaluate the replicated basic event to be “failed” if k out of the m replicates have failed.

6. ISSUES INVOLVED IN MODELING TIME-VARYING HAZARD RATES

The exponential distribution is the only TTF distribution with constant rate of failure. It is also the easiest to in terms of model generation and evaluation. However, the failure behavior of many components are better described by means of time-varying hazard rates. Examples of such distributions are the Weibull and the log normal distributions. In order to model and evaluate systems with time-varying transitions the relevant hazard rate parameters should be stored in the Markov arcs and evaluated at the required instances of time.

7. THE TRANSLATION ALGORITHMS

The translators implemented algorithms for going from one abstract model to another. These algorithms were of varying levels of

complexities depending on the models involved. Thus the translator from fault tree model to Markov model was the most complex, followed by the translator from Markov model to the differential equations. The ordinary differential equation solver [CVODE] was used as a black box module in solving the differential equations. A special structure called “SolvedNode” stored the fault tree unreliability in the form of a node. This reduced fault tree is useful in case the fault tree was a sub-tree of a larger fault tree, wherein independent sub-trees are identified by a process called modularization and solved separately and the results are integrated to give the complete fault tree unreliability [Gul97].

8. FAULT TREE TRANSLATION ALGORITHM

We designed a recursive algorithm to translate the given dynamic fault tree into the corresponding Markov chain. The new algorithm was rigorous yet easy to understand, and had similarities to the well-known pipe-and-filter software architecture [Shaw96]. There are four procedural stages in this algorithm- the FDEP stage, the Spare Gates stage, the PAND stage, and the SEQ stage. Each stage emulates the behavior of the corresponding set of dynamic gates in terms of state generation, modification, and evaluation. For example, the FDEP stage considers all the FDEP gates in the fault tree and evaluates each one of them in accordance with the FDEP evaluation algorithm.

A Markov state contains information relating to operational basic events, spare gate allocation and PAND gate input failure order. States are evaluated to be either operational or failed based on the information contained within them. Operational states can generate further states while failure states are absorbing states. The algorithm considers all the working components of the state, fails them one by one to produce tentative new states, passes these tentative new states into the procedural stages mentioned earlier, and recursively repeats this process with every newly created operational state. The new states created thus are evaluated to determine whether they are operational or failed, and inserted into the Markov chain. Appropriate transitions are constructed between these states and the parent state. The overall algorithm (called MakeNewStates) is shown in Figure 10.

8.1. Procedural Stages

The main procedural stages of the state generation algorithm correspond to the four types of dynamic gates, marked HandleFDEPs, HandleSpares, HandlePANDs, and HandleSEQ in figure 11. We obtained the specifications for the handler functions by studying the behavioral requirements of the gates, deriving informal specifications from the existing implementation, and accounting for subtleties such as the ones described in Section 5.

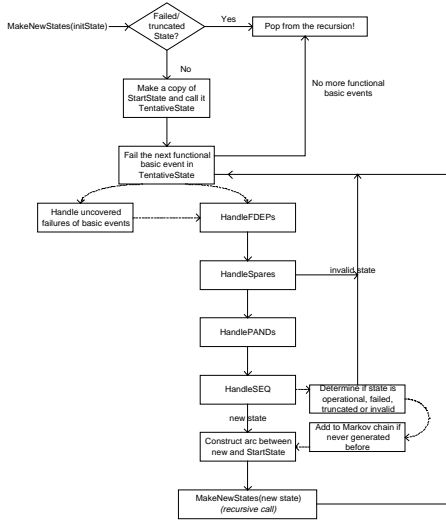
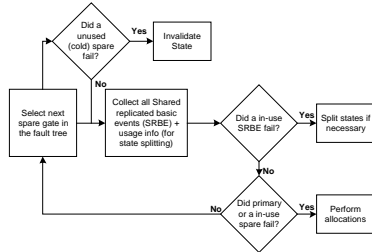


Figure 9: Markov Model Generation



Algorithm

Figure 10: Spare Gates Handler Routine

8.2. Example of a Procedural Stage: The HandleSpares Routine

The chief function of a spare gate is to allocate an available spare component in case a functioning component fails. If the spare gate is a cold spare gate, a state in which a cold spare fails before the primary fails is invalidated. A spare gate is evaluated as *failed* if its primary component has already failed, **and** there are no more available spares. The spare gate handling routine is complicated because of the possibility of pooled spares. We devised an allocation

scheme to dynamically distribute the shared spares between the gates that share them. Replicated basic events can model a pool of shared spare components. Since replicated basic events have no distinct identity, sharing them across spare gates give rise to the problem of how to interpret the failure of any such shared component. As a consequence, multiple (split) states may be generated, with each state corresponding to the failure of one spare gate that shares a replication of the failed component. Section 5 discusses this issue in greater detail under “Shared spare pools and state splitting”.

9. SOFTWARE IMPLEMENTATION

The implementation of the software closely followed the problem conceptualization. This statement is illustrated in figure 9. We first of all identified all the abstractions (models) involved in the fault tree modeling process, starting with the fault tree model followed by the Markov model, the family of (numerical) differential equations, and finally, the reliability estimate. Next, we designed translator classes that convert one abstract model to another. These translator classes are particular cases of a general class of software structures called mediators [Sull94]. By externally defining relationships between the models, these mediators help maintain intellectual control of the software system, enforce modularity, and keep modifications to within module boundaries. Another benefit of following this approach was the ability to potentially replace or repair software modules in an independent fashion, without making any changes to the rest of the system. The conceptual stages are shown along with the corresponding software modules. The software modules were implemented as C++ classes. The model abstractions are shown in gray, while the translator stages are not shaded.

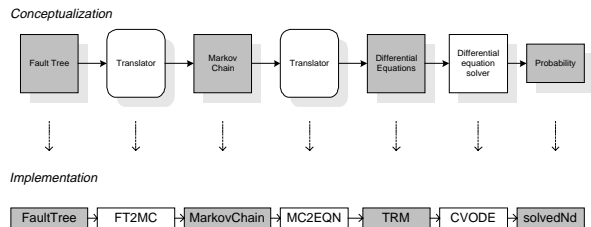


Figure 11: Software implementation

REFERENCES

Bell Telephone Laboratories, Murray Hill, NJ
USA.

- [Cop98] Coppit, D., and Sullivan, K.J., "*Formal specification in collaborative design of critical software tools*", Submitted to High Assurance System Engineering (HASE) Conference- 1999.
- [Doy95] Doyle, S. A., Dugan, J.B., and Boyd, M., "Dependability assessment using Binary Decision Diagrams (BDDs)", *Proceedings of the International Symposium on Fault Tolerant Computing*, page: 249-258, June 1995.
- [Doy95a] Doyle, S.A., Dugan, J.B., and Patterson-Hine, A., "A combinatorial approach to modeling imperfect coverage", *IEEE Transactions on Reliability*, pages 87-94, March 1995.
- [Dug92] Joanne Bechta Dugan, Salvatore Bavuso, Mark Boyd. Dynamic fault tree models for fault tolerant computer systems. *IEEE Transactions on Reliability*, September 1992.
- [Joh89] Johnson, Barry W., "Design and analysis of fault-tolerant digital systems", Addison-Wesley Publishing Company, 1989.
- [Pott96] Potter, B., Sinclair, J., and Till, D., "An introduction to Formal Specification and Z", 2d Ed., Prentice Hill, 1996.
- [Shaw96] Shaw, M., and Garlan, D., "Software architecture", Prentice-Hall, 1996.
- [Sull94] Sullivan, K.J., "Mediators: Easing the Design and Evolution of Integrated Systems", Ph.D. Dissertation, University of Washington, Seattle, Washington, 1994.
- [Tri82] Kishor S. Trivedi. "Probability and statistics with reliability, queing, and computer science applications." Prentice-Hall, 1982.
- [Wei51] Weibull, W., "A statistical distribution function of wide applicability", *Journal of Applied Mechanics*, Vol 18, page 293-297, 1951.
- [Wat61] H.A. Watson and Bell Telephone Laboratories, "Launch Control Safety Study",