Representing Complex Systems within Discrete Event Simulation for Reliability Assessment

Les Warrington • University of Warwick, Coventry
Jeffery A. Jones • University of Warwick, Coventry

Key Words: Simulation, Reliability Modeling, System Reliability

SUMMARY & CONCLUSION

During design of a high-fidelity discrete event simulation (DES) of aircraft reliability and maintenance under realistic operational scenario, modelling of complex system functionality and control of the associated state-space explosion, was a major concern. Hierarchical system decomposition limited this explosion but correct and efficient analysis of the aircraft systems remained a concern.

This paper reviews Markov, Petri net, Fault Tree, Event Tree and RBD, illustrating their respective computer implementations. Computing requirements of each are compared.

Path- and cut-sets, derived from Fault Tree, Event Tree or RBD, were identified to be most efficient computer representations but with the disadvantage that only 'static' systems could be modelled. A static system is one in which order of component failure or other conditional system property, does not determine functionality. The alternate is a 'dynamic' system. However, it is also noted that 'dynamic' system functionality is always a conditional sub-set of an equivalent 'static' path-set. The paper proposes a method that integrates DES with path-sets to allow 'dynamic' system modelling.

Computing storage requirements for the different system analysis methods discussed are:

- Markov -- k^m x k^m sparse binary matrix + k^m binary vector per discrete event
- White Markov – k(m-1) x k^m integer matrix
- Petri net -- k^m x k^m sparse binary matrix + k^m binary vector per transition
- Path-set -- Qty k^m of k(m-1)-binary vectors
- Enhanced Path-set – Qty k^m of k(m-1)-binary vectors + integer vector for each dynamic element (e.g. switch)

Computer processing requirements are:

- Markov -- Vector multiplication + indexing
- White Markov -- Indexing
- Petri net -- Matrix-vector multiplication
- Path-set -- Search
- Modified Path-set – Search

For large systems, path-set representation rapidly becomes more efficient in storage requirements, even when sparse matrix techniques are used with the alternatives.

1. INTRODUCTION

System reliability is achieved by a combination of component reliability, system architecture and maintenance. DES is an important tool to assess the combined influence of these factors. However, despite continued improvement in computing power, efficient modelling of complex systems remains a challenge. Simulation of major systems, including their full operational and maintenance environment, involves significant state-space. Without careful consideration of objectives, performance and memory restrictions, such simulation may not be feasible.

This paper examines the memory efficiency of a number of alternate state-space representations and illustrates their respective advantages and disadvantages. Modularization is highlighted as being an important method to gain maximum efficiency. However, this paper also introduces a novel adaptation of path-set state-space representation that allows increased memory efficiency.

1.1 Acronyms

URAM: Ultra-Reliable Aircraft Model, a discrete event simulation
DES: Discrete event simulation
RBD: Reliability Block Diagram

2. MODELLING OF SAFETY AND RELIABILITY

There are several methods for mathematical analysis of reliability and maintenance. However, such analytical methods do not allow the same richness as discrete event simulation, for capturing the complexity of real world scenario. Indeed, it is where the approximations and simplifications required by analytical methods become too great that discrete event simulation (DES) is particularly used. However, DES may integrate analytical methods.

DES of safety and reliability does not need to model component reliability, system architecture and maintenance using a single methodology. Rather, DES may adopt the most appropriate methodology for each. Therefore, this paper

0-7803-7717-6/03/$17.00 © 2003 IEEE

assumes that component status and maintenance are separately modelled and are inputs to a system state-space model.

3. SYSTEM ANALYSIS

Many systems continue to function despite internal degradation and failure. Such functionality itself may have several conditions. Collectively, all possible combination of internal condition and external function is a state-space.

There are several analytical methods to derive state-space. This paper will consider Fault Trees, Event Trees, Markov diagrams, and Petri-nets. However, DES cannot directly use diagrams such as Figure 1. Rather, they must be converted to a form that promotes computer performance and memory efficiency while still retaining fidelity.

This paper will discuss matrix representations compatible with Markov and Petri analysis, and pathsets compatible with Fault and Event Trees. Each analysis method and each representation have advantages and disadvantages. However, a novel adaptation of the pathset representation will be explained that provides a useful, additional usage for pathsets.

4. PETRI NETS

Petri nets have states (often called places) and transitions, and model the dynamic system state by addition and deletion of 'tokens'. An example Petri net is given in Figure 1, representing the RBD of Figure 2. Petri nets provide opportunity to model every potential system state, and therefore 'success', 'partial success', 'failed' and indeed any other definable system condition of interest can be specifically identified. Therefore, it is a rich analysis methodology.

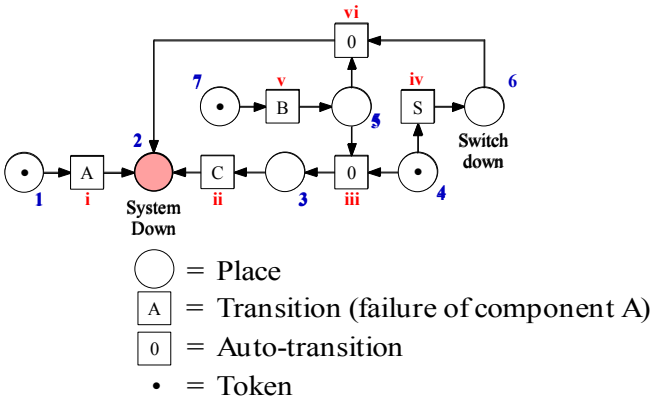

Figure 1 Petri Net (enumerated to link with Figure 3)

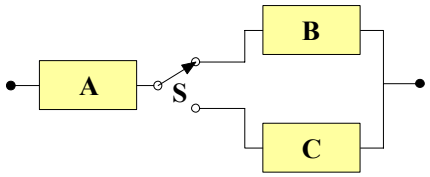

Figure 2 Reliability Block Diagram equivalent of Figure 1

A Petri net is usually represented by a 'transition matrix' and by a 'token vector' prior to further computer-aided analysis, including for DES. Figure 3 illustrates the binary transition matrix and token vector for Figure 1. The matrix describes the addition and deletion of 'tokens' when triggered by certain input conditions, such as component failure. The total presence of 'tokens', as denoted by the 'token vector', denotes the system state. A system may remain in a given state for a period of time (say, using a simple stochastic component failure model) before addition/deletion of tokens, or may wait for external input (discrete event) to trigger such change. The matrix cell values for stochastic models represent the probability of transfer but when a Petri net is linked to external trigger, the cells will be binary. For DES purposes, the most appropriate method is external input and, therefore, the simpler binary transfer matrix is more appropriate.

-1	1	0	0	0	0	0
0	1	-1	0	0	0	0
0	0	1	-1	-1	0	0
0	0	0	-1	0	1	0
0	0	0	0	1	0	-1
0	1	0	0	-1	-1	0

Transition Matrix

1	0	0	1	0	0	1

Token Vector (initial state)

Figure 3 Petri Net Transition Matrix and Token Vector using columns & rows enumerated in Figure 1

When an external event triggers the Petri 'token' vector, a change is calculated by repeated additive matrix/vector multiplication until a steady state is obtained in the 'token' vector. Figure 4 illustrates these calculations following failure of component 'B' of Figure 1. As stated earlier, Petri analysis is a rich analysis method and therefore this new state may prompt subsequent simulation reaction with full fidelity.

Unfortunately, a Petri transition matrix suffers from 'state explosion'. Potentially, a system with 'm' components, each with 'k' possible states (working plus all fault modes), may have k^m valid states. This would create a k^m x k^m binary transition matrix. This transition matrix is generally sparsely populated and, therefore, computing memory requirements may be reduced.

In summary, Petri Net computing requirements are:

- k^m x k^m sparse matrix for transition matrix
- k^m-vector for each transition
- multiple matrix multiplication to calculate each new token vector

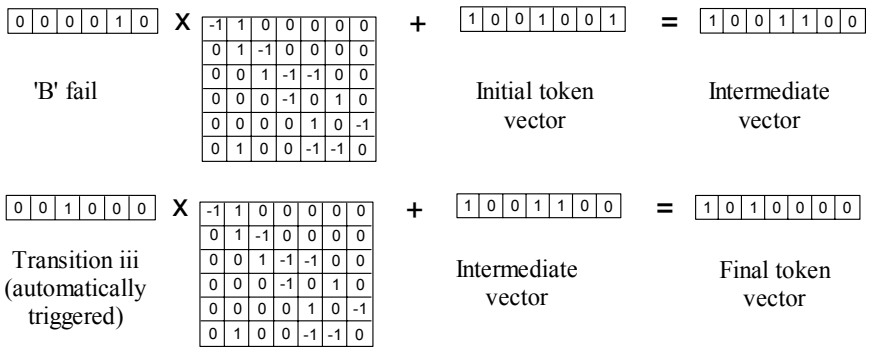

Figure 4 Transition following failure of Component B

5. MARKOV ANALYSIS

Markov analysis is an equally rich analysis method that also identifies all system states individually as shown in Figure 5 (again representing the RBD of Figure 2). As with Petri nets, the system states are usually represented by a matrix. A standard Markov transition matrix is also the form k^m x k^m, where m is the number of components and k the number of possible component states (Figure 6). Each component would have an associated transition vector of length k^m. Therefore, computer representation and calculation of system state derived from Markov analysis would suffer from the same 'state explosion' as from Petri Nets. Sparse matrices could again be used, leading to computer memory footprints similar to the Petri Net representation. However, the matrix and vector size may be minimised if all system failed states are combined (Figure 6), as with Petri Nets.

Computing requirements are:

- k^m x k^m sparse binary matrix
- k^m – binary vector per discrete event (component failure)
- vector multiplication to identify new matrix index

However, White developed an alternate transition matrix representation. (Ref. 1) Instead of using a binary vector (matrix row) to identify the state transfer, he enumerates the states and forms a matrix of 'component fault mode' and 'system state', thus immediately reducing the state matrix to k(m-1) x k^m. Thus, the 'White' matrix for the example 20 element system would be 20 x 10^6. Bitwise coding could not be used, but using 32-bit integers for the enumerated states, the memory usage would again be approximately 100Mb. Sparse matrix techniques may be appropriate for the full matrix but again system failed states may be combined as in Figure 7.

White's method has the advantage that no matrix multiplication is required in order to identify the new state. State change is identified through indexing, which is time efficient.

Computing requirements for White's method are:

- k(m-1) x k^m integer matrix
- Direct indexing for new state

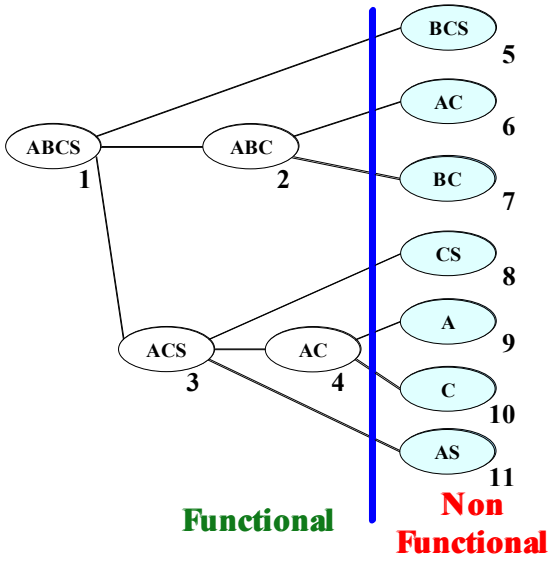

Figure 5 Markov State Diagram (enumerated to link with Figure 6)

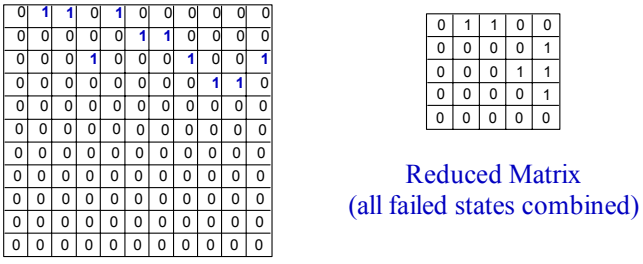

Figure 6 Markov Transition Matrix (using columns/rows enumerated in Figure 5)

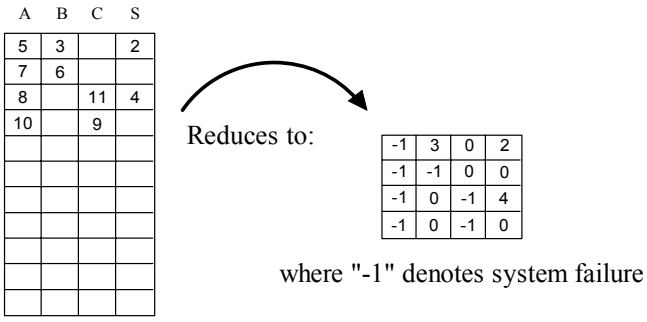

Figure 7 Modified Markov Transition Matrix (Ref. 1)

6. FAULT TREE, EVENT TREE AND RBD

Fault Trees, Event trees and RBD may visually represent dynamic conditions, such as sequence and functional dependency, but such conditions cannot be analysed without first being converted to, say, a Markov representation. Therefore, use of Fault and Event Tree or RBD is often restricted to static conditions.

Results from Fault Tree, Event Tree or RBD are commonly represented as Cut- or Path-sets. These are binary vectors of (for paths) component states that, collectively, indicate system function. A Fault Tree captures system configurations that would lead to a specific 'top' event, usually system failure. An Event-Tree, because it potentially identifies each and every system outcome, will contain path-sets or cut-sets for each outcome category.

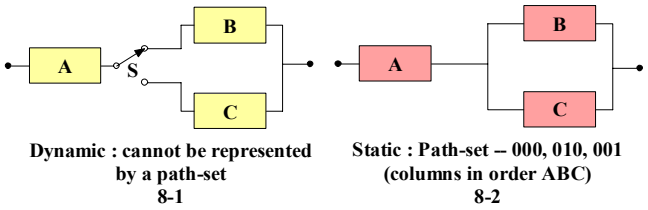

Figure 8 Dynamic and Static RBD

For a 20 element example system, the combined number of system states (path-set and cut-sets) would again be 10^6, with each path or cut being 20 bits long (2 ½ bytes). Thus the combined memory footprint would be approximately 2.5 Mb. This is a considerable reduction over other methods. The disadvantage, of course, is that it is restricted to static conditions.

When applied to DES, system state upon failure or repair of a component is simply an issue of updating the binary state vector, and identifying which category the new vector resides, cut- or path-set. This is a 'find' function, usually of moderate computing efficiency O(ln(n)).

Computing requirements for cut- or path-set fault representation are:

- Qty k^m of k(m-1)-binary vectors
- Search (O(ln(n)) efficiency)

7. DISCRETE EVENT SIMULATION AND PATH-SET INTEGRATION

During development of URAM, a DES of aircraft operating in a complex operational and maintenance scenario (Ref. 3), a need to model dynamic elements was considered likely. Nevertheless, the memory efficiency of Path- and Cut-sets remained highly desirable. Therefore, options to preserve the memory efficiency of Path- and Cut-sets were sought that concurrently allowed dynamic modelling.

7.1 Methodology

7.1.1 Modularity

Dynamic states arise due to differences in results from alternate sequence of events. However, those alternate sequences are not dependent on all input conditions but rather usually only on a sub-set. All other inputs are static. Indeed, modular approaches to Fault Tree Analysis make this very point. (Ref. 2) In which case, modularity would be possible between a Path- and Cut-set representation and, say, a Markov or Petri matrix.

7.1.2 Hierarchical decomposition

Sub-systems may often be modelled hierarchically, with the functional output of sub-systems being used as inputs to a higher-level system structure. However, where such separation is not possible, a combined representation that retains maximal memory and performance efficiency is required.

7.1.3 Extended Path- and Cut-Sets

Path- and Cut-sets collectively describe the full combination of possible component states. Therefore, dynamic systems do not increase the overall number of possibilities but, rather, conditionally change a path into a cut. If a system state is statically failed, it is not reasonable for it to be dynamically functional. Therefore, dynamic systems would not conditionally change a cut into a path.

Thus, it is implicitly possible to model dynamic systems while still using path- and cut-sets, provided an additional data-structure is feasible to capture and identify conditions where a statically functional system is dynamically failed.

7.2 Implementation

Static system behaviour within URAM is derived from an integrated RBD editor, using structure functions to derive path-sets. The memory efficiency of these path-sets is also increased by utilising commonality, whereby only one instance of a particular system path-set is required, no matter how many aircraft are simulated.

Dynamic system behaviour is modelled using additional vector data structures, also derived from the integrated RBD, as described below.

URAM uses bit-wise coding for path vectors and for recording current states. All component failure modes are linked to specific system (path-set and current state) vector columns. Any change to a component state changes the current state, which in turn is compared against available paths.

7.2.1 Stand-by Redundancy

Stand-by redundancy involves a sequential transfer of function to previously non-operational elements (not necessarily non-functional). Sequence is easily modelled as a vector, representing the order in which redundant elements would be brought into use. URAM derives this vector directly from its integrated RBD editor.

The functional properties of the standby switch are linked to an integer vector identifying the switched elements and their order of operation.

Fig. 8-1 would be represented by:

- Static path-set (using Fig. 8-2) : 000, 010, 001
- Switch vector [2,3,0], describing standby redundant elements B and C. (The front element of this integer vector identifies the current active element which, on failure, would be pushed to the rear (cycled) only if the switch is currently functional.)
- Current (starting) system state vector : 000

Upon any change to the switch vector, the current system state vector would always identify if the new front element element is functional at the time and, if not, the switch would be caused to cycle again until either the end of the integer vector is reached (zero entry) or a functional redundant element found.

Consider current system state to be 000, followed by failure of component B:

- New current system state : 010
- Switch is functional, permitting cycling of 'switch' vector [2,3,0] → [3,0,2]. New leading element (component C) is functional.
- Compare new current system state with path-set—system is functional.

However, if the switch has failed (either previously or fail-on-demand), then cycling of the 'switch' vector is denied and the current system state is over-ridden by elements controlled by the integer 'switch' vector being set to 'not functional' within the current state vector.

Consider the current system state to be 000. There are 2 possibilities upon switch failure:

- *Switch fails to change*: component identified as column-3 (component C) would be artifically set to 'non-functional', leading to new current system state 001. Compare new current system state with path-set—system is functional.
- *Switch fails open-circuit*: both columns 2 and 3 (components B and C) would be artifically set to 'non-functional', leading to new current system state 011. Compare new current system state with path-set —system non-functional.

In the current version of URAM, a simplifying restriction is placed on stand-by redundancy, in that only components linking to single path-set columns (fault modes) may be defined as forming stand-by redundancy. Therefore, any sub-systems that contribute to redundancy through multiple path-set columns must, firstly, be modelled separately and introduced hierarchically.

Shared (common) 'cold spares' could be modelled in the same way, with the additional restriction that no spare may be active in more than one 'vector' simultaneously.

General computing requirements for this approach are:

- Qty k^m of k(m-1)-vectors (binary coding)
- k-vector for each switch element
- Search (O(log(m) efficiency)

7.2.2 Functional Dependency

Functional dependence, whereby outputs from otherwise functional elements selectively become denied following failure of a 'control' (trigger) element under particular conditions, is another common dynamic condition.

Functional Dependency may be implemented using path-sets I a similar fashion to stand-by redundancy by incorporating additional data structures..

URAM allows a component to contribute to multiple systems (path-sets). Each component holds a map of those contributions. True failure of a component results in all system states being modified. However, functional dependency may restrict that modification.

Therefore, it is necessary only to link a vector selecting the restricted list of systems from the full set already held by the component. Upon triggering of the control element, that restricted set of systems would be modified, without changing the true state of the component.

7.2.3 Sequence Dependency

Sequence dependency limits the change of system state, notwithstanding component state. Only if component failure occurs in a particular sequence, is system state allowed to change.

This could be implemented by a vector describing the required order, say [1,2,3]. System state change would be prevented

unless and until sequential failure of all elements. Then, the system state would be changed directly to the true state. The static path-set would determine system functionality at all times.

8. DISCUSSION

Therefore, it has been concluded that it is a simple matter to model the behaviour of cold-standby, sequence and functional dependency systems using path-sets and small additional data structures.

There will be instances where system architecture and behaviour is strongly dynamic and therefore a rich analysis method will be required. In such cases, careful sparse-matrix representation of petri nets or markov analysis will be appropriate. However, the demonstrable memory efficiency of pathsets will be appropriate for systems with mostly static properties. Systems may also be modularised, allowing restricted use of Petri or Markov transition matricies and use of pathsets in other static areas.

Allowing additional state description vectors to isolate dynamic properties, particularly standby redundancy, sequence dependency and functional dependency, offers an opportunity to extend the efficiency of path-sets. This methodology has been successfully incorporated into URAM.

During validation of the model, URAM has simulated 50 Tornado aircraft, each with several hundred inter-linked systems, in a matter of seconds on a laptop PC using Gulf War (Desert Storm) data and scenario.

REFERENCES

1. P. White, “Factors for ensuring the integrity of safety analysis of complex systems”, Foresight and Precaution, Vol 2, (Proceedings of ESREL2000), pp1281-1291

2. R. Gulati, J.B. Dugan, “A Modular Approach for Analyzing Static and Dynamic Fault Trees”, Proceedings Annual Reliability and Maintainability Symposium, 1997, pp57 – 63

3. J.A. Jones,, L. Warrington, N. Davis, “The Use of a Discrete Event Simulation to Model the Achievement of Maintenance Free Operating Time for Aerospace Systems”, Proceedings Reliability and Maintainability Symposium, 2001, pp170 – 175

BIOGRAPHIES

Mr Les Warrington, Senior Fellow
Warwick Manufacturing Group,
International Manufacturing Centre,
Department of Engineering,
University of Warwick,
Coventry, CV4 7AL, United Kingdom

E-mail: L.Warrington@Warwick.ac.uk

Les Warrington holds degrees in modern history and aeronautical engineering. He was an engineering officer in the Royal Air Force before joining the University of Warwick in 1992 to develop reliability and maintenance teaching and research. He jointly founded the Warwick Quality and Reliability MSc programme and is course leader of reliability modules in this and other Warwick MSc programmes delivered in UK and overseas. His research interests include the development of improved reliability processes, particularly those that fulfil commercial imperatives and enhance customer benefit. He is project leader of the University of Warwick contribution to the Society of British Aerospace Companies (SBAC) Ultra Reliable Aircraft (URA) research programme. He is a Chartered Engineer and member of the Royal Aeronautical Society.

Jeff Jones, Senior Research Fellow
Warwick Manufacturing Group,
International Manufacturing Centre,
Department of Engineering,
University of Warwick,
Coventry, CV4 7AL, United Kingdom

E-mail: J.A.Jones@Warwick.ac.uk

Jeff Jones is a senior research fellow with Warwick Manufacturing Group, University of Warwick where he is involved in research into the improvement of aircraft reliability. He has a first degree in electronic engineering and physics and has a masters degree by research (MPhil) entitled "The Implementation of a Field Reliability Database". He is an expert and project leader on IEC/TC56. He is also an active member within the UK dependability standards community. He is a Chartered Physicist, a Chartered Engineer and holds memberships of the IEEE, the Institute of Physics and Society of Aerospace Engineers