A Graphical Model-Based Reliability Estimation Tool and Failure Mode & Effects Simulator

David M. Nicol ● College of William and Mary ● Williamsburg
Daniel L. Palumbo ● NASA Langley Research Center ● Hampton
Michael L. Ulrey ● Boeing Defense & Space Group ● Seattle

Key Words: Reliability Estimation, Graphic User Interface, Failure Mode & Effects Simulator

SUMMARY & CONCLUSIONS

A new graphical reliability estimation tool, Reliability Performance Module (RPM), is described. RPM combines the features of a powerful reliability tool, Reliability Estimation System Testbed (REST), developed at NASA Langley, with the convenient graphical modelling and simulation capabilities of an off-the-shelf commercial software package, Block Oriented Network Simulator (BONeS), from the Alta Group of Cadence Design Systems. In order to estimate the reliability of a system, the built-in BONeS graphics capabilities are used to describe the system, and the embedded REST execution engine produces a reliability analysis automatically. An additional benefit of this approach is that a detailed failure modes and effects analysis can be derived by using the simulation capabilities of the tool. The usage of and output from RPM is demonstrated with an example system. As compared to our current design process, RPM promises to reduce overall modelling and analysis time, provide better documentation, make trade studies easier, create reusable modelling components and subsystems, and provide the integration of reliability and timing analysis necessary to guarantee the safety of critical real-time systems. Future work will concentrate on producing a more seamless integration of the reliability and timing analyses. Additional planned enhancements include a distributed (parallel) processing mode, and availability and phased-mission analysis capabilities.

1. INTRODUCTION

In this paper, we describe a software tool, called Reliability Performance Module (RPM). RPM represents the latest phase in an evolutionary process. It borrows heavily, both in philosophy and in actual code, from the Reliability Estimation System Testbed (REST) tool, developed at NASA Langley, and described in Ref. 1. REST, in turn, evolved from the ASSIST/SURE tool set (Ref. 2, 3, 4, and 5) also developed at NASA Langley, and from the Reliability Model Generator (RMG) program, developed at Boeing (Ref. 6). In Ref. 1, the authors show how their unique simultaneous state-space-generation/state-space-analysis technique together with parallel processing combine to reduce dramatically the execution time of a model with almost 500 components. Also explained is the idea of Failure Mode Effects Simulation (FMES), wherein the "local" behavior of system components (entered by the modeller) is expanded by the program into "global" behavior (computed by the tool), which reveals system-wide effects such as system failure. The modelling philosophy of REST is based on a modular, hierarchical approach that is easily adaptable to graphical input methods. The next logical step was to provide a graphical input method, and the result is described herein.

As explained below, there is more to the story than simply replacing text with pictures. The unique feature of RPM is that it is a reliability tool (REST) embedded in the Block Oriented Network Simulator (BONeS) environment. BONeS is a graphical modelling and simulation tool, originally intended to model and analyze complex data and communication networks. The modelling capabilities of BONeS are more flexible and intuitive than a modelling language alone when it comes to describing a complex system.

2. OBJECTIVES of RPM DEVELOPMENT

The primary goal in the development of RPM was to produce a tool to perform the integrated reliability and timing analysis of critical real-time systems. These systems are typically designed with a high degree of fault tolerance. The reliability and timing analyses of such systems are often done separately. Ref. 8 contains an excellent discussion of the dangers of this practice.

This consideration was a strong motivator in the decision to combine REST and BONeS. The BONeS tool is specifically designed for evaluating system performance in terms of throughput and delay measures. One could use one of these measures, say, end-to-end signal delay, to evaluate whether or not the system is still alive, i.e., still performing all safety critical functions. Also, one could make use of the the statistical measures in BONeS, for example, mean response time,

0149-144X/95/$4.00 ©1995 IEEE

to trigger transitions, or again, assess system failure.

In its current state of development, the RPM tool is not capable of combining the reliability and timing models so neatly. What one can do at this stage is create a BONeS model with pure reliability attributes and a separate model with pure performance (timing and throughput) attributes to do reliability and performance analyses. However, one of the main objectives in future work is to achieve a more seamless union of these models so we can claim that the two processes are truly "integrated."

Therefore, we shall restrict the discussion in this paper to the reliability analysis capabilities of the tool, and postpone the discussion of integrated reliability and performance capabilities to a future paper.

As for goals having to do solely with reliability analysis, our experience with the use of reliability packages has produced the following list of things any good tool should do:

(1) Document the configuration and modelling assumptions for each candidate system architecture.

(2) Identify dominant failure modes.

(3) Reveal subtle failure sequences.

(4) Produce understandable results.

In addition, usage of the tool should:

(5) Allow the engineer to use simple language, graphical icons, etc., to define how the system works, the redundancy management operation, and the operational mode fault degradation strategy, i.e., the plan for continuing operation in various degraded states.

(6) Prompt the engineer for missing definitions and simplifying assumptions, and document these assumptions as part of the analysis output.

(7) Output failure sequences and their probabilities to a specified level of detail so the engineer can assess system architecture strengths and weaknesses, thus making it easier to do trade studies.

Most of these objectives have been met through the development of RPM. This will be explained in more detail below.

3. EXHAUSTIVE ANALYSIS & THE MONTE CARLO METHOD

The analysis approaches used in RPM are derived from the REST program. The exhaustive analysis method is a variation on the method used in the SURE program. In both programs, the overall system failure probability is estimated by computing upper and lower bounds on every failure sequence (subject to pruning) and accumulating these to produce upper and lower bounds on the overall system failure probability. See Ref. 4, 5 or 7 for details.

The difference between SURE and REST lies in the way the tasks of state-space generation and state-space analysis are performed. The SURE program requires the entire list of state-space transitions be generated in advance. The REST approach is to perform a depth-first search of all possible "paths" (sequences of state-space transitions), computing upper and lower bound estimates on these paths as they are generated, and discarding a path when its analysis is complete. This technique also makes "trimming (or pruning) on the fly" possible. Just as with SURE, the overall system failure probability is estimated by accumulating the upper and lower bounds on all possible paths, subject to pruning.

The advantage of the REST approach is that the memory requirements are much less severe than for SURE. Also, because of the empirical fact that, for most systems, after a small number of component failures (3-5), the system is either dead or the path probability is insignificant compared to the overall system failure probability, the price paid for this memory gain in terms of execution time is small. See Ref. 7 for details.

An alternative to the exhaustive analysis approach is to use Monte Carlo simulation with importance sampling. The basic idea is to choose transition paths at random, using the relationship between fast and slow transition rates out of a given state to bias the choice towards rare events. The skew in the proability distribution thus introduced is accounted for later using standard importance sampling techniques. The use of this approach within REST is described in Ref. 7. This capability is now also part of RPM.

4. PROGRAM OVERVIEW

The fundamental design concept of RPM has two basic elements: the *executive* and the *behavioral model*.

These elements can be thought of as being part of an experiment, where the executive plays the role of an experimenter attempting to deduce the properties of the behavioral model through stimulation and observed reaction.

From the *user's* (i.e., *design engineer's)* point of view, the executive is of absolutely no concern, while the behavioral model is entirely the engineer's responsibility to construct.

Again from the user's point of view, there are two stages to the use of RPM. In the first stage, the behavioral model is constructed using the BONeS graphical modelling facilities. In the second stage, the user creates a BONeS simulation from the model, sets model parameters, e.g. component failure rates, and starts the simulation. At this point, the executive takes over, stimulates and analyzes the model automatically, eventually producing output in the form of the overall system failure probability, lists of dominant failure sequences and their probabilities, and the probabilities of user-

specified system failure events.

5. THE EXECUTIVE

After the behavioral model has been constructed and the simulation has begun, the operation of RPM at the top level appears to be a rapid-fire "conversation" between the executive and the behavioral model. The primary messages from the executive to the behavioral model are

1. Load a given state.

2. Is the current state a death state?

3. Fire all possible transitions and report the resulting new states.

The behavioral model responds to (1) by configuring itself in the specified state. This activity may involve side effects which cascade through the system. For example, the failure of a particular component may result in messages being sent which inform other components of this situation. Also as a result of this failure, for example, a configuration management module may enter a state indicating that a recovery process has begun. In any case, when the "dust has settled," the behavioral model informs the executive that it is now in the specified state.

In response to (2), the behavioral model checks each of the system failure criteria, which were specified by the engineer as part of the behavioral model, to see if its current state meets any of them. The behavioral model returns a yes or no answer to the executive.

If the answer to (2) is yes, the current state is a death state, the executive performs probability calculations which update the cumulative upper and lower bounds on the system failure probability. This state is then discarded. If there are no states left, the program ends and the final results are output. If the executive has any states left to load, it gives the next one to the behavioral model and the whole process begins anew.

If the answer to (2) is no, the current state is not a death state, the executive asks the behavioral model to fire all possible transitions, which were specified by the engineer as part of the behavioral model. In response to (3), the behavioral model fires transitions one by one. Each time a transition fires, it may happen that a series of messages are sent or other side effects occur. In any case, when this activity ceases, the behavioral model reports the resulting state to the executive, resets itself to the original state specified by the executive, and fires the next transition. This continues until there are no more transitions. The executive then presents a new state to be loaded, if any are left, and the whole process begins anew.

The executive conducts its search of the transition sequence space in one of two modes, *exhaustive analysis mode* or *Monte Carlo mode*. In the exhaustive analysis mode, all possible transition sequences are generated, subject to user-definable pruning criteria. In the Monte Carlo mode, the transition paths are chosen stochastically so as to improve the chance of occurrence of rare events of interest. The response of the behavioral model to a given sequence of transitions is the same in either case.

6. THE BEHAVIORAL MODEL

As mentioned previously, the behavioral model is a description of the relevant characteristics of the system to be modelled and analyzed. As discussed in the objectives section, we will focus on those aspects pertaining to the reliability analysis only. To illustrate the important features of the behavioral model, the example shown in Figures 1 and 2 will be used. (Figures 1-4 are actual BONeS diagrams). The block diagram in Fig. 2 is a template for blocks labelled *String 1, ..., String 4* in Fig. 1.

There are 32 failable components, divided into 4 *strings* of 8 components apiece. Each string consists of one pitch sensor, one a-to-d converter, one i/o card, two processors and three power supplies. The components which are not power supplies in each string are connected in series. One of the power supplies feeds the i/o card, and the other two feed both of the processors. The a-to-d converters and the i/o cards have three modes: GOOD, BAD (active failure) and OFF (passive failure). All other components and power supplies are either GOOD or FAILED.

All failures are viewed as being permanent, and failure detection is assumed to be perfect.

The output of each string is either GOOD, BAD or OFF. For convenience's sake we will say that the string is GOOD, BAD or OFF, respectively, in these cases. It turns OFF (passive failure) if either one of the processors FAILs, both processor power supplies FAIL, the a-to-d converter turns OFF, the i/o card turns OFF, or the i/o power supply FAILs. A string's output is GOOD if there is power to all components which are modelled to have power, and all (non-power supply) components are GOOD. All other conditions cause the string's output to be BAD.

A string attempts to reconfigure to OFF whenever the pitch sensor FAILs, the a-to-d converter goes BAD, or the i/o card goes BAD. Once OFF, a string stays OFF.

There are three system failure conditions. A majority vote failure occurs when the number of BAD strings equals or exceeds the number of GOOD strings. Nearly coincident fault failure occurs whenever two or more strings are simultaneously BAD, i.e., string recovery was unsuccessful. String exhaustion occurs whenever there are no remaining GOOD

strings.

The block labelled *String Configuration Management Module* (SCMM) in Fig. 2 handles the recovery process for the string. The SCMM is omniscient, i.e., it is informed immediately of any change in the state of a component in its particular string. This module, then, is an abstraction made possible by the assumption of perfect fault detection.

The behavior of each of the components in the string is described using a finite state machine (FSM) paradigm. For example, a BONeS representation of the FSM of the A-to-D and I/O modules is shown in Fig. 3. Note that there are three states, GOOD, FAILED and OFF.

The FSM transitions from the GOOD state to either the FAILED or OFF state via *spontaneous transitions*, denoted by the blocks with lightning bolts. These transitions correspond to random failures. They are triggered in response to a command from the executive to fire transitions.

Note also that appropriate messages are output whenever the FSM enters the FAILED or OFF states. These messages are sent to the SCMM.

The BONeS representation of the SCMM is shown in Fig. 4. It has one spontaneous transition, from FAILED to OFF. This models the recovery mechansism for the string. In addition, this FSM can also transition from the GOOD state to the FAILED or OFF state by a *stimulated transition*, denoted by the blocks which look like light switches. That is, the SCMM transitions in response to a message from any of the non-power supply components of the string. This transition is a direct result of a deterministic stimulus, in this case a message, even though the sequence of events leading to this message may be initiated by a random event.

Referring back to Fig. 1 again, the purpose of the blocks labelled *GOOD Signal Generator* and *System Condition Update* is as follows. After a transition fires and the affected string has settled into a new state, the GOOD signal generator sends out a pulse (labelled GOOD) to all four strings. This signal then passes through each of the modules of each string, including the SCMM.

When a signal enters a given module, it comes out GOOD, BAD or OFF based on the input value and the state of the module. When the signal finally reaches the SCMM in a string it is processed as follows. The output signal is OFF if the input signal is OFF, regardless of the SCMM state. The output signal is also OFF if the SCMM state is OFF, regardless of the input signal. The output signal is GOOD if the input signal and the state of the SCMM are both GOOD. Otherwise, the output of the SCMM is BAD.

The outputs from the four strings are then sent to the System Condition Update module, which counts the number of GOOD and BAD strings. This information can then be used by each of the system failure conditions (Majority Vote Failure, Coincident Fault Failure, and String Exhaustion) to decide whether or not the system is still alive. These system failure conditions are represented in BONeS by a collection of arithmetical and logical blocks which computes some function of the system state and then decides whether or not a certain test is passed. The system failure conditions are described inside the blocks of Fig. 1 labelled *System Requirements* (details not shown).

7. RESULTS

Table 1 shows the model parameters, including component failure rates, string recovery rate, and the mission time. The overall system failure probability bounds (upper and lower) are shown in Table 2. Note that the exhaustive analysis bounding interval (4.567e-12, 4.919e-12) is contained in the corresponding Monte Carlo bounding interval (4.57e-12 - 1.68e-12, 4.58 + 1.69e-12) when the (95%) confidence interval limits are taken into account. These results have been repeated using other reliability tools.

8. FAILURE MODE & EFFECTS SIMULATION

As mentioned previously, one of the key concepts of the RPM design philosophy is that the system to be modelled and analyzed is viewed as a behavioral model which can be stimulated and observed. This means that it is possible for the user to take over the role of the executive block and fail selected components in order to observe the effect on the system.

Although this use of the tool helps answer "what if" questions, it is of limited value, since a human cannot compete with the computer when it comes to performing thousands of these operations in a reasonable amount of time.

Therefore, one of the outputs of RPM is a list of the most important failure sequences, ordered from highest probability to lowest. Such a list, corresponding to the example, is shown in Table 3. Because of limited space, only the first five event sequences are shown. Upper and lower bounds on the probabilities of each event sequence are displayed, as well as upper and lower bounds on the cumulative probability of the event sequences up to that point. Also shown is the *effect* of each failure sequence, which in this case means the particular system failure conditions (Majority Vote Failure, Nearly Coincident Fault Failure, or String Exhaustion) which were satisfied by the the occurrence of the given component failure sequence. (The first five failure sequences all resulted in Majority Vote Failure.)

Finally, Table 4 shows the probabilities associated with each of these system failure conditions. (The event labelled *String Exhaustion* has probability zero since the analysis was

arbitrarily terminated after 3 component failures.)

Notice that the first five failure sequences are essentially of the same *type*, that is, two (single) power supplies fail, then a pitch sensor fails. It would be informative to group such similar sequences into a single event, then make a list of these events in order of decreasing probability, similar to the ordering of the raw event sequences in Table 3. In fact this is done in RPM, but limited space prohibits display of these results.

Since this is not a traditional FMEA, but has a similar flavor, we call it Failure Modes and Effects Simulation (FMES). A traditional FMEA would precede the model definition. Obviously, a model has already been defined by the time this FMES is performed.

However, it provides much more insight into the strengths and weaknesses of the system than a single probability-of-failure number can. The FMES feature can be used to succesively redesign the system by focussing attention on the elements of the design which contribute to the most prominent failure sequences. Also, one may be alerted to high probability failure sequences which were not considered important prior to the analysis.

REFERENCES

1. D.M. Nicol, D.L. Palumbo and A. Rifkin, "REST: A Parallelized System for Reliability Estimation", *Annual Reliability and Maintainability Symposium Proceedings*, 1993.
2. R.W. Butler, "An Abstract Language for Specifying Markov Reliability Models," , IEEE Trans. Rel., Vol. R-35, No. 5, pp. 595-601, Dec. 1986.
3. S.C. Johnson, "ASSIST User's Manual,", NASA Tech. Memo 87735, 1986.
4. A.L. White, "Upper and Lower Bounds for Semi-Markov Reliability Models of Reconfigurable Systems," *NASA Contractor Report No. 172340*, April, 1984.
5. R.W. Butler, A. L. White, "SURE Reliability Analysis," *NASA Technical Paper 2764*, NASA Langley Research Center, March 1989.
6. G.C. Cohen, "Reliability Model Generator", *NASA Contractor Report 182005*, 1990.
7. D.M. Nicol, D. L. Palumbo, "Reliability Analysis of Complex Models Using SURE Bounds," *NASA Contractor report 191445* or *ICASE Report No. 93-14*, March, 1993. To appear in *IEEE Transactions on Reliability*.
8. K.G. Shin, P. Ramanathan, "Real-Time Computing: A New Discipline of Computer Science and Engineering," *Proceeedings of the IEEE*, January, 1994.

BIOGRAPHIES

David M. Nicol
Department of Computer Science
College of William and Mary
P.O. Box 8795
Williamsburg, VA 23187-8795 USA
email:nicol@cs.wm.edu
Company Name: College of William and Mary

David M. Nicol received the B.A. degree in mathematics from Carleton College (1979), and the Ph.D. in computer science from the University of Virginia (1985). He is presently an associate professor of computer science at the College of William and Mary. He is associate editor for the ACM Transactions on Modeling and Computer Simulation, and for the ORSA Journal on Computing. He is on the steering committee for the Workshop on Parallel and Distributed Simulation, and has served as program chair, and also general chair for that Workshop's annual conference. He is widely published in the areas of performance modeling, parallel processing, and simulation.

Daniel L. Palumbo
MS 130
NASA Langley Research Center
Hampton, VA 23665-5225 USA
email:dlp@air12.larc.nasa.gov
Company Name: NASA Langley Research Center

Dan Palumbo received his BSEE and MSEE degrees from Rensselaer Polytechnic Institute in 1972 and 1973, respectively. He is a Senior Computer Engineer with NASA Langley Research Center where his current interests are related to the deployment of distributed and integrated flight control systems.

Michael L. Ulrey
Boeing Defense & Space Group, M/S 9E-05
700 S.W. 41st St. (7-81-6 bldg.)
Renton, WA 98055 USA
Phone: (206)-657-5640
FAX: (206)-657-5736
email: ulrey@kenyon.ds.boeing.com
Company Name: Boeing Defense & Space Group

Mike Ulrey is a Principal Engineer in the Advanced Vehicle Management Systems organization of the Boeing Defense & Space Group. He is currently responsible for the development of the RPM tool, to be included in an integrated design process for rapid prototyping of vehicle management systems. He has been involved in large applications software development projects for NCR, TRW, Raytheon and Boeing. He was an assistant professor of mathematics at Wichita State University. He received a Ph.D. degree in mathematics from Ohio State University in 1973. He is a member of SIAM and IEEE.

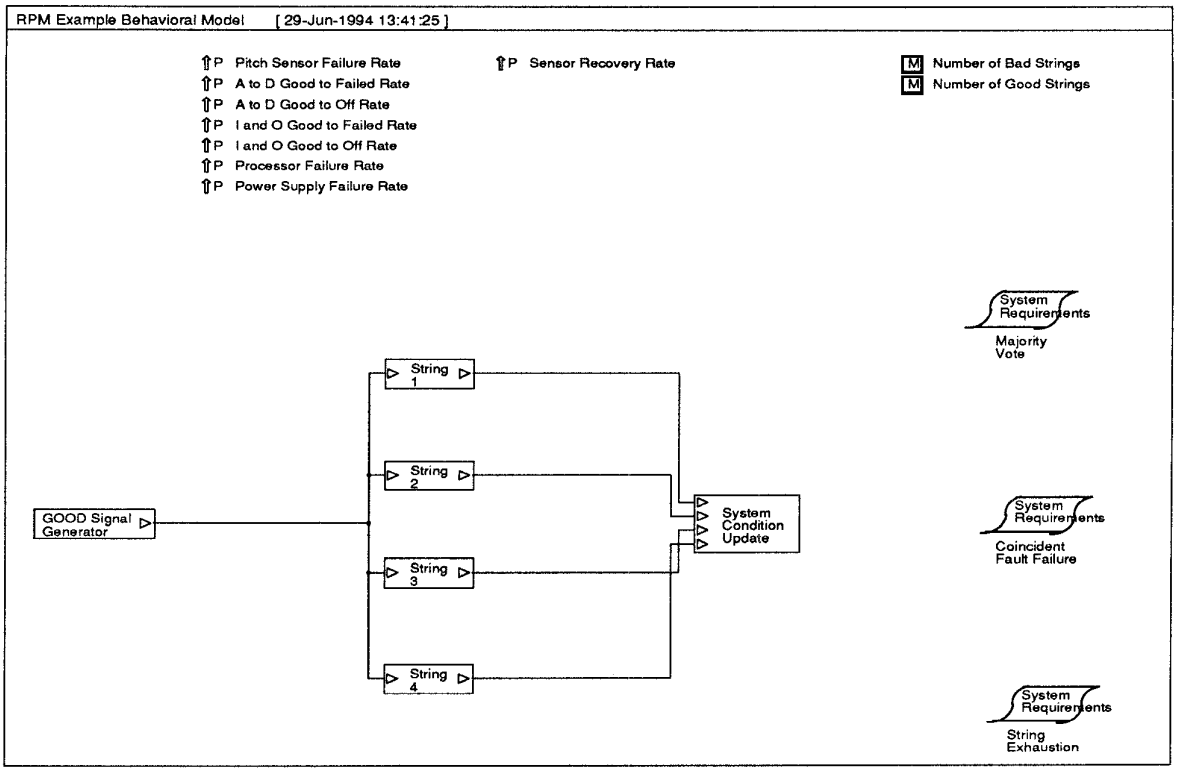

Figure 1. Behavioral Model

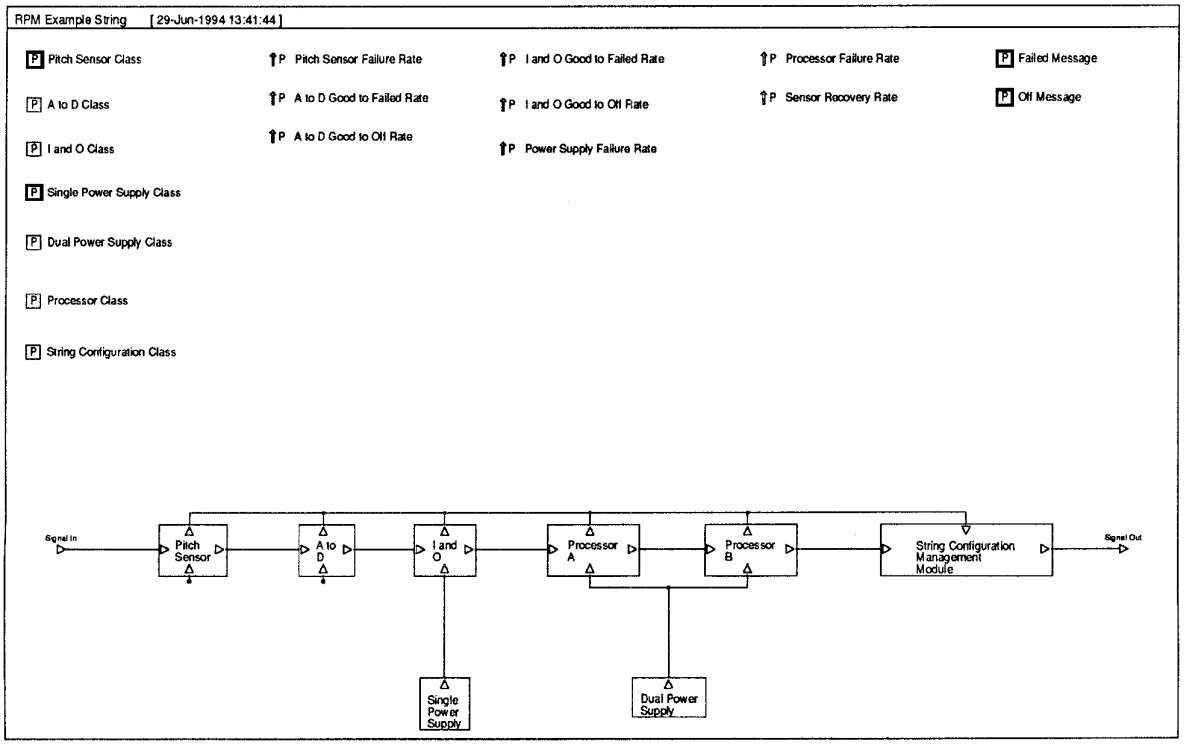

Figure 2. Template for "String"

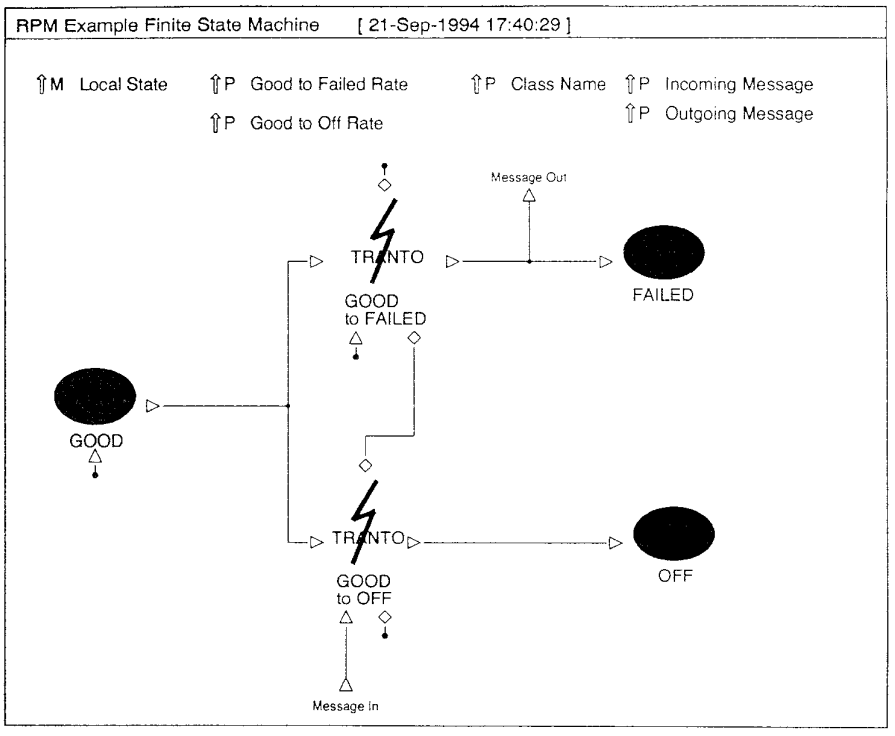

Figure 3. Finite State Machine Template for I/O and A/D Modules

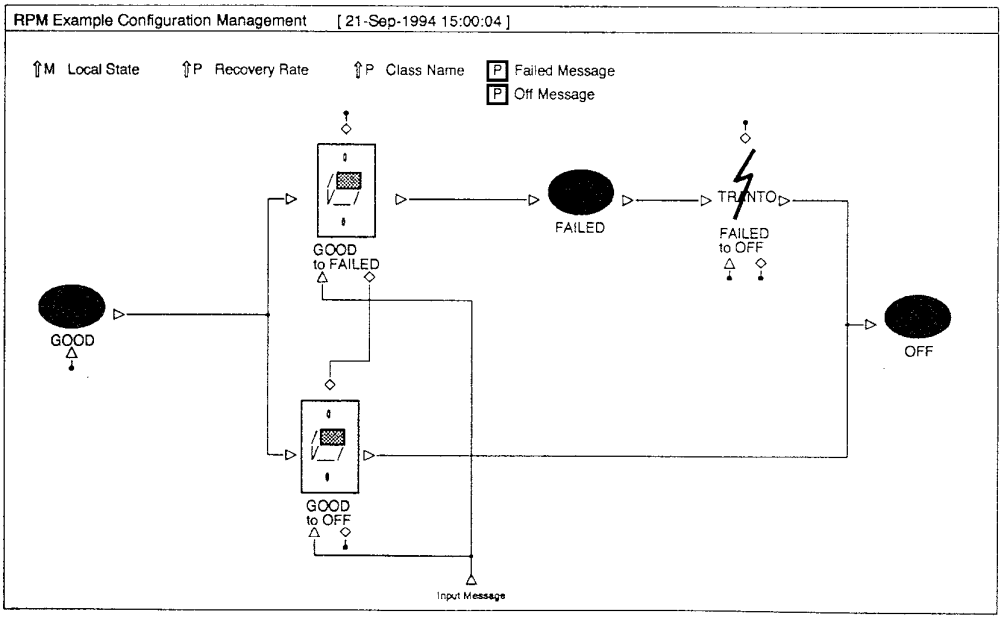

Figure 4. Finite State Machine Template for Configuration Management Module

Pitch Sensor: 2e-6 A/D Passive: 4.14e-6 A/D Active: 4.6e-7 I/O Passive: 6.75e-6 I/O Active: 7.5e-7
Power Supply: 3.71e-5 Processor: 3.12e-5 String Recovery: 3600.0 Mission Time = 3.0 hrs.

TABLE 1. Component Failure Rates (failures/hour) and String Recovery Rate (recoveries/hour)

Exhaustive Analysis:	Death State Probability: 4.567e-12 to 4.578e-12	(at 3 component failure level)
	Prune State Probability: (0.0) to 3.410e-13	
	Total: 4.567e-12 to 4.919e-12	
Monte Carlo:	4.57e-12+/-1.68e-12 to 4.58+/-1.69e-12	(95% confidence level; 5000 replications)

TABLE 2. System Failure Probability Bounds as Computed by the Two Modes

1. Transitions: (String1)(SinglePowerSupply)(GOODtoFAILED)
(String2)(SinglePowerSupply)(GOODtoFAILED)
(String3)(PitchSensor)(GOODtoFAILED)
Effects: (MajorityVote)
Probability: (1.23721e-14, 1.23877e-14) Cumulative: (1.23721e-14, 1.23877e-14)

2. Transitions: (String1)(SinglePowerSupply)(GOODtoFAILED)
(String2)(SinglePowerSupply)(GOODtoFAILED)
(String4)(PitchSensor)(GOODtoFAILED)
Effects: (MajorityVote)
Probability: (1.23721e-14, 1.23877e-14) Cumulative: (2.47442e-14, 2.47754e-14)

3. Transitions: (String1)(SinglePowerSupply)(GOODtoFAILED)
(String3)(SinglePowerSupply)(GOODtoFAILED)
(String2)(PitchSensor)(GOODtoFAILED)
Effects: (MajorityVote)
Probability: (1.23721e-14, 1.23877e-14) Cumulative: (3.71163e-14, 3.71631e-14)

4. Transitions: (String1)(SinglePowerSupply)(GOODtoFAILED)
(String3)(SinglePowerSupply)(GOODtoFAILED)
(String4)(PitchSensor)(GOODtoFAILED)
Effects: (MajorityVote)
Probability: (1.23721e-14, 1.23877e-14) Cumulative: (4.94884e-14, 4.95508e-14)

5. Transitions: (String1)(SinglePowerSupply)(GOODtoFAILED)
(String4)(SinglePowerSupply)(GOODtoFAILED)
(String2)(PitchSensor)(GOODtoFAILED)
Effects: (MajorityVote)
Probability: (1.23721e-14, 1.23877e-14) Cumulative: (6.18605e-14, 6.19385e-14)

TABLE 3. Raw Failure Sequences and Their Effects (Five most probable sequences only are shown)

1. Effect: (MajorityVote)	Probability: (4.56664e-12, 4.57816e-12)
2. Effect: (CoincidentFaultFailure)	Probability: (1.00222e-13, 1.03098e-13)
3. Effect: (StringExhaustion)	Probability: (0.00000e+00, 0.00000e+00)

TABLE 4. Effects and Their Probabilities (Based on three-component failure limit)