IN-61
32308
P57

NASA Technical Memorandum 104068

A GENERAL GRAPHICAL USER INTERFACE FOR AUTOMATIC RELIABILITY MODELING

Carlos A. Liceaga and Daniel P. Siewiorek

May 1991

(NASA-TM-104068) A GENERAL GRAPHICAL USER INTERFACE FOR AUTOMATIC RELIABILITY MODELING (NASA) 57 p CSCL 09B
N91-29745
Unclas
G3/61 0032308

NASA
National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia 23665-5225

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer systems are growing in complexity and sophistication as multiprocessors and distributed computers are coming into widespread use to achieve higher performance and reliability. This growth is being assisted by the availability of successively more complex building blocks. This trend has increased the importance of fault tolerance and system reliability as design parameters. Thus the computation of system reliability measures has become one of the system design tasks. Several efforts have been reported in the literature and are in progress to make the task of computing system reliability measures easier and more efficient by providing designers with reliability evaluation tools.

The analysis and evaluation of system reliability for complex computer systems is very tedious and prone to error even for experienced reliability analysts. With the exception of the ADVISER program, discussed in Section 1.2, existing software tools usually assume an understanding of reliability analysis techniques and therefore are more in the nature of computational aids once the preliminary system decomposition and analysis has been manually achieved. Although ADVISER does not make this assumption it uses combinatorial techniques and is therefore limited in the complexity of systems and fault types it can analyze.

More advanced techniques are required to analyze computer architectures that use standby redundancy, can be repaired, and are susceptible to transient or intermittent faults. One possibility is the Markov model, discussed in Section 1.1. The advantages offered by Markov models are that they are in widespread use among reliability analysts and several programs, discussed in Section 1.2, have been developed to solve them. However, Markov models can not be used to analyze nonexponentially distributed concurrent events. For example, a fault that arrives while the system is reconfiguring itself around a previous fault would be represented by a transition to a state where two faults are present. This new state would not take into account the time the system already spent reconfiguring from the first fault.

Another possibility is the extended stochastic Petri net (ESPN) described in [DTGN84]. The advantages offered by the ESPN is that it can analyze concurrent events and model systems at a lower level of detail than Markov models. The ESPN "tokens" can be simultaneously enabled to move concurrently at independent transition times. The low level modeling capability is due to mechanisms such as queues and counters that can simulate the algorithm of the process being modeled. To solve an ESPN analytically or numerically it must be converted to a Markov model. This conversion is not possible if tokens are moving concurrently at independent transition times that are not exponentially distributed, because this makes the process non-Markovian (i.e., the transition probabilities depend on past states). In general an ESPN must be solved by simulation.

Simulations can include any level of detail, and are thus flexible, but many repetitions of the simulation are needed to ensure accuracy. For example, in life critical applications that require a probability of failure of 10^{-9} with a relative error no more than 10% within a confidence interval of 95%, approximately 3.8×10^{11} simulation repetitions are necessary [LS86]. In general those applications require a Markov model because it can be solved analytically or numerically.

The purpose of this paper is to present a general graphical user interface (GUI) for automatic reliability modeling of processor-memory-switch (PMS) structures using a Markov model so that it can be used for life critical applications. This GUI is based on a hierarchy of windows implemented in C using the TAE Plus user interface development tool for building X window-based applications [Szc90]. One window has graphical editing capabilities for specifying the system's communication structure, hierarchy, reconfiguration capabilities, and requirements. This window is implemented using the schematic drawing editor Schem [Vli90]. Other windows have text fields, pop-up menus, and buttons for specifying parameters and selecting actions. The advantages of such an approach are (a) utility to a larger class of users, not necessarily expert in reliability analysis, and (b) a lower probability of human error in the computation.

A brief background on reliability calculation at the PMS level using Markov models is presented in Section 1.1. Previous work in the generation and evaluation of reliability models is surveyed in Section 1.2. The proposed GUI is defined and illustrated in Chapter 2. An example application of the GUI is presented in Chapter 3 using the Fault-Tolerant Multiprocessor (FTMP) described in [LS83]. Conclusions are drawn in Chapter 4.

1.1 Background

Present day computer systems can be viewed at varying levels of detail, and therefore so can the process of designing and analyzing them. Four levels were defined in [SBN82]. These range from the circuit level, through the logic and programming levels, to the PMS level. The PMS level view of digital systems is one where the primitives are processors, memories, switches, transducers, etc. as opposed to the logic level where the primitives may be gates, registers, multiplexers, etc.

Hardware components are susceptible to hard and soft faults as discussed in [SS82]. A fault is an incorrect state of hardware or software resulting from a physical change in the hardware, interference from the environment, or design mistakes [Lap85]. Hard or permanent faults are continuous and stable, and result from an irreversible physical change. Soft faults can be transient or intermittent. Transient faults result from temporary environmental conditions. Intermittent faults are occasionally active due to unstable hardware, or varying hardware or software states (e.g., as a function of load or activity). Depending on whether the intermittent fault is benign or active the output of the component will be correct or not, respectively.

Fault-tolerant computer systems can be affected by a limited set of faults without interruptions in their operation. Some computer systems achieve fault tolerance by using redundant groups of components to perform the same operations. The system must determine which is the correct output using diagnostics or majority voting. The various redundancy techniques are discussed in [SS82], the more relevant ones are defined below.

Static redundancy. Faults are masked through a majority vote involving a fixed group of redundant components. Thus, when the masking redundancy is exhausted by component failures, any further faults will cause errors at the output. Figure 1.1 illustrates a statically redundant processor triad.

Dynamic redundancy. Faults are not masked from causing errors at the output, but the faulty components are detected, isolated, and reconfigured out of the system. The faulty components are replaced when spares are available. Figure 1.2 illustrates a dynamically redundant active processor with n spares.

Hybrid redundancy. Faults are masked through a majority vote involving a group of redundant components that is reconfigured when spares are available. Thus, when the masking redundancy is exhausted by component failures, any further faults that occur before a faulty component is replaced by a spare will cause errors at the output. Figure 1.3 illustrates a hybridly redundant triad of active processors with n spares.

Adaptive voting. Faults are masked through a majority vote involving a variable group of redundant components without spares. Faulty components are reconfigured out of the system by excluding them from the voting process. Thus, when the masking redundancy is exhausted by component failures, any further faults that occur before a faulty component is reconfigured out of the voting process will cause errors at the output. Figure 1.4 illustrates adaptive voting with n processors.

Adaptive hybrid. Faults are masked through a majority vote involving a variable group of redundant components which are replaced when spares are available. If spares are not available, faulty components are reconfigured out of the system by excluding them from the voting process. Thus, when the masking redundancy is exhausted by component failures, any further faults that occur before a faulty component is replaced by a spare or reconfigured out of the voting process will cause errors at the output. Figure 1.5 illustrates an adaptive hybrid n-tuple of active processors with m spares.

For example, if a triad (a group of 3 components) that uses hybrid redundancy "recovers" from a fault by replacing the faulty component with a spare, it can then tolerate a second fault. Recovery is the process of detecting, isolating, and reconfiguring the faulty component out of the system. The fault coverage of a component is the probability that the system can survive a fault in this component and successfully

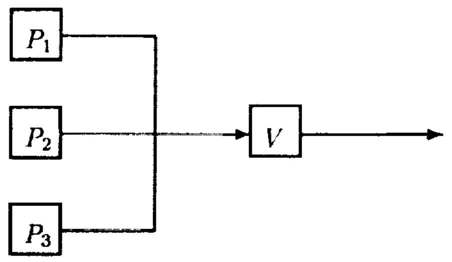


Figure 1.1: A statically redundant processor triad.

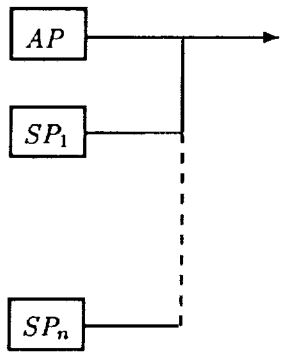


Figure 1.2: A dynamically redundant active processor with n spares.

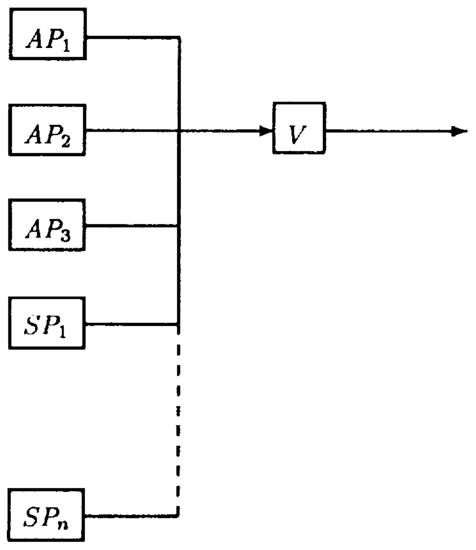


Figure 1.3: A hybridly redundant triad of active processors with n spares.

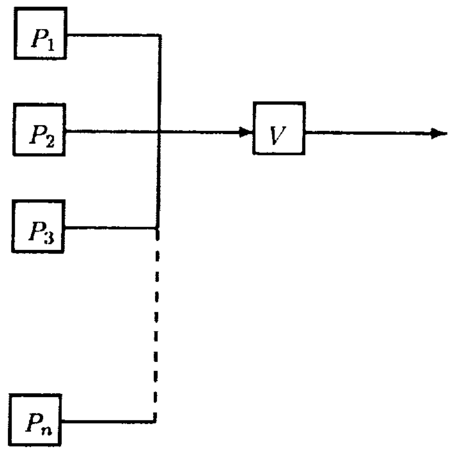


Figure 1.4: Adaptive voting with n processors.

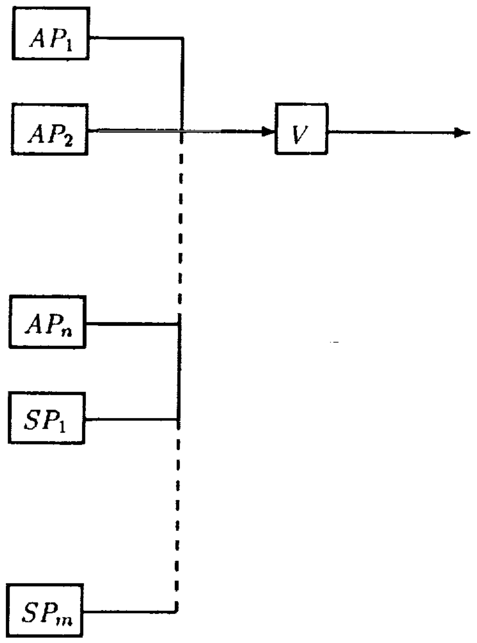


Figure 1.5: An adaptive hybrid n-tuple of active processors with m spares.

recover.[1] If the system can always recover it has a "perfect" coverage of one.

Spares are sometimes left unpowered until they become part of the active configuration to reduce their failure rates [AGM+71]. They are sometimes said to be: cold if their failure rates are assumed to be zero, warm if their failure rates are reduced but not zero, or hot if their failure rates are not reduced [BJ90].

Reliability measures are defined in terms of probabilities because the failure processes in hardware components are nondeterministic. These various measures are discussed in [SS82], the more relevant ones are defined below.

Reliability. The conditional probability, $R(t)$, as a function of time t that the system has survived the interval $[0, t]$ given that it was operational at time zero. It is a nonincreasing function whose initial value is one.

Mean Time To Failure (MTTF). The expected time of the first system failure assuming a new (perfect) system at time zero.

Availability. The probability, $A(t)$, as a function of time t that the system is operational at that instant of time t.

If the limit of $A(t)$ exists as t goes to infinity, it expresses the expected fraction of time that the system is available to perform useful computations. Availability is typically used as a figure of merit in systems in which service can be delayed or denied for short periods to perform preventive maintenance or repair without serious consequences. The availability is important in the computation of system life-cycle costs.

Reliability is used to describe systems in which repair is typically infeasible such as aerospace applications. The MTTF can be derived from $R(t)$ as follows:

$$\text{MTTF} = \int_0^\infty R(t)\,dt$$

The most commonly used reliability function for a single component is based on a Poisson process with

[1] These are the definitions that will be used in this paper, but recovery and coverage do not have universally accepted definitions.

an exponential distribution. This is called the exponential reliability function and has the form:

$$R(t) = e^{-\lambda t}$$

where λ is the hazard or failure rate. The failure rate is a constant which reflects the reliability of the component and for highly reliable components is usually expressed in failures per million hours. The exponential reliability function is used when the failure rate is time-independent, such as when components do not age. It is often observed that, after a burn-in period, permanent faults in electronic components follow a relatively constant failure rate. The MTTF for the exponential reliability function has the form:

$$\text{MTTF} = \frac{1}{\lambda}$$

Many other reliability functions have been formulated. The second most common reliability function is based on the Weibull distribution. This is called the Weibull reliability function and has the form:

$$R(t) = e^{-(\lambda t)^{\alpha}}$$

where λ is the scale parameter and α is the shape parameter (other reparameterized forms are also common). It is equivalent to the exponential function when α is one. The Weibull reliability function is used when the failure rate is time-dependent. Permanent faults for components that age can be described using an increasing failure rate ($\alpha > 1$) and in this case the system is not as good as new when repair takes place. Data presented in [McC81, CMS82] indicates that transient faults follow a decreasing failure rate ($\alpha < 1$).

The failure processes of different components will be assumed to be independent of each other. This assumption is not strictly true, such as when electrical, mechanical, or thermal conditions in one component affect other components in its proximity. However, it is close enough in practice to be used to simplify the analysis.

The state of a system represents all that must be known to describe the system at any instant. As the system changes, such as when components fail or are repaired, so does its state. These changes of state are called state transitions. If all possible states are assumed to be known a discrete-state system model is used; if this assumption is not made a continuous-state system model is used. If the state transition times are assumed to be restricted to some multiple of a given time interval a discrete-time system model is used. If it is assumed that state transitions can occur at any time a continuous-time system model is used. Most systems can be classified according to their state space and time parameter as

a) discrete-state and discrete-time

b) discrete-state and continuous-time

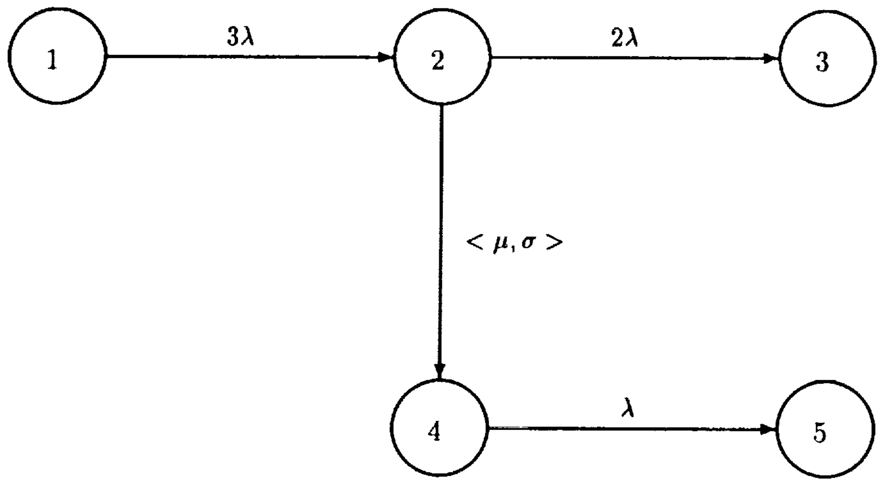
c) continuous-state and discrete-time

d) continuous-state and continuous-time

For a discrete-state system a state transition diagram (STD) may be drawn. The transition diagram is a directed graph. The nodes correspond to system states, and the directed arcs indicate allowable state transitions. Each arc has a label that identifies the distribution of the conditional probability that the system will go from the originating node to the destination node of that directed arc given the previous history of the system and that the system was initially at the originating node. The label used depends on the distribution. For example, the label could be the hazard rate for the exponential distribution, the scale and shape parameters for the Weibull distribution, or the mean and standard deviation for a general distribution.

If transitions are allowed from failed states to operational states, then the STD is an Availability graph and $A(t)$ may be obtained from it. $R(t)$ may be obtained by specifically disallowing failed to working state transitions from the STD, thus making it a Reliability graph.

A Reliability graph of a triad is given in Figure 1.6. In this model it is assumed that the components have a perfect coverage of 1. The horizontal transitions represent fault arrivals. These follow an exponential distribution, and consequently λ represents the constant hazard rate. The coefficients of λ represent the number of working processors that are being actively used in the configuration. The vertical transition represents recovery from a fault. These follow a general distribution, and consequently μ and σ represent its mean and standard deviation. There is a race between the two transitions leaving state 2. If the second



Key: State Description

1	3 working
2	2 working
3	system failed
4	2 working, uses 1
5	system failed

Figure 1.6: Reliability graph of a triad.

fault wins the race, then system failure occurs. If the removal of the first fault wins the race, then the system reconfigures into a simplex (i.e., only uses one of the two working components). Unless otherwise noted in the state descriptions, all working processors are being actively used in the configuration.

The information conveyed by the STD is often summarized in a square matrix called the state transition matrix (STM). The STM element in row i and column j is the label in the arc from state i to state j.

The terminology used in this paper to denote the various types of Markov models, and the assumptions they are based on are defined below. The hierarchy of Markov models is illustrated in Figure 1.7.

Markov model. A stochastic process model whose future state depends only upon the present state, and not upon the history that led to its present state.

Homogeneous Markov model. A Markov model whose state transition probabilities are time-independent. For the continuous-time homogeneous Markov model this implies that the state transition times follow an exponential distribution. This type of model is discussed in [Chu67, Rom70] and applied to computer systems in [MA82].

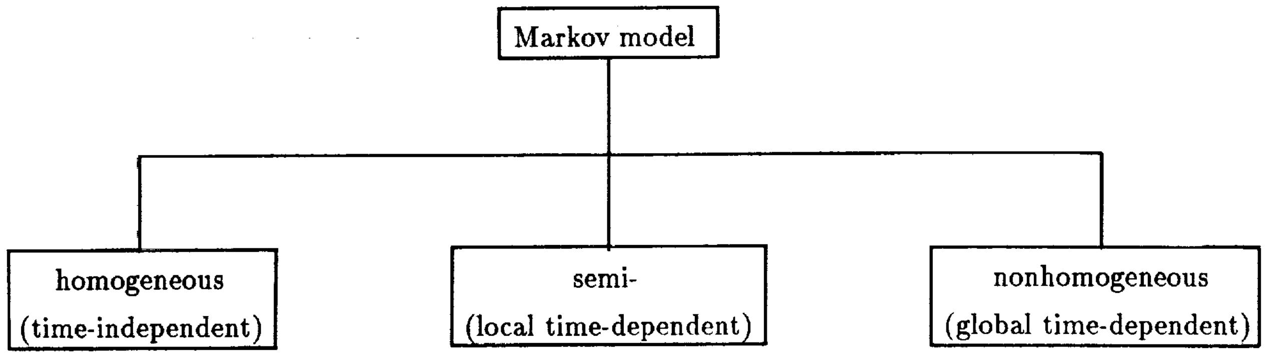

Markov model
homogeneous (time-independent)
semi- (local time-dependent)
nonhomogeneous (global time-dependent)

Figure 1.7: Hierarchy of Markov models.

Semi-Markov model. A Markov model whose state transition probabilities depend upon the time spent in the present state, called the local time. For the continuous-time semi-Markov model this implies that the state transition times do not follow an exponential distribution, they might follow a Weibull distribution or any other distribution. This type of model is discussed and applied to computer systems in [Whi84].

Nonhomogeneous Markov model. A Markov model whose state transition probabilities depend upon the time since the system was first put into operation, called the global time. For the continuous-time nonhomogeneous Markov model this implies that the state transition times do not follow an exponential distribution. Often they are assumed to follow a Weibull distribution, but they can follow any other distribution. This type of model is discussed and applied to computer systems in [TG81].

The probability of being in a particular state for a discrete-state, continuous-time Markov model can be expressed with a differential equation. The set of simultaneous differential equations that describe these models are called the continuous-time Chapman-Kolmogorov equations. For homogeneous Markov models these equations can be solved using matrix or Laplace transformations.

If the state transition probabilities are time-dependent, it may be quite difficult to obtain explicit solutions to the continuous-time Chapman-Kolmogorov equations. To obtain the exact probability of reaching a state through a particular path of transitions requires the solution of a multiple integral, where each integral represents the probability of making one of the transitions in the path. Often the integrals are approximated using numerical integration techniques [SBG79]. An alternative method is to approximate the continuous-time model with discrete-time equivalents [SS82]. The major difficulty with the second method is that many transition rates that are effectively zero in the continuous-time model assume small, but nonzero, probabilities in a discrete-time model.

1.2 Previous Work

There are several programs that use Markov models to evaluate the reliability and/or availability of systems that use standby redundancy or can be repaired, and are susceptible to hard, transient, and intermittent faults, such as CARE III, ARIES, SURE, PAWS, STEM, SURF, and HARP. All of these programs can evaluate reliability. ARIES, SURF, and HARP can also evaluate availability. Except for CARE III, they all have as one of the system specification methods the state transition matrix.

CARE III (Computer-Aided Reliability Estimation), described in [BPR84], can evaluate the reliability of systems that use reconfiguration to tolerate component faults, but do not repair the faulty components. It uses a behavioral decomposition/aggregation solution technique described in [TG81]. This technique assumes that the fault-occurrence behavior is composed of relatively infrequent events while the fault-handling behavior is composed of relatively frequent events. The fault-handling behavior is separately analyzed using a fixed semi-Markov model that can use exponential and uniform distributions. The fault occurrence behavior is analyzed using an aggregate nonhomogeneous Markov model that can use exponential and Weibull distributions. The fault handling behavior is reflected by parameters in the aggregate nonhomogeneous Markov model. Numerical integration techniques are used to solve these Markov models. The fault-occurrence behavior is specified using extended fault trees, which are automatically converted to the nonhomogeneous Markov model. The fault-handling behavior is specified by providing the transition parameters of the fixed semi-Markov model. CARE III was developed at Raytheon under NASA's Langley Research Center sponsorship. It is written in FORTRAN 77 and runs on Cyber and VAX/VMS computers.

ARIES (Automated Reliability Interactive Estimation System), described in [MAG82], is restricted to homogeneous Markov models. The system can be specified using a state transition matrix or as a series of independent subsystems each containing identical modules that are either active or serve as spares. It uses a matrix transformation solution technique that assumes distinct eigenvalues for the state transition matrix.

It was developed at UCLA and runs on a VAX.

SURE (Semi-Markov Unreliability Range Evaluator), described in [BW88], evaluates the unreliability upper and lower bounds of semi-Markov models. It uses new mathematical theorems proven in [Whi84, Lee85]. These theorems provide a means of bounding the probability of traversing a specific path in the model within a specified time. By applying the theorems to every path of the model, the probability of the system reaching any death state can be determined within usually very close bounds. These theorems assume that slow (with respect to the mission time) exponential transitions describe the occurrence of faults and fast transitions, that follow a general distribution specified by its mean and standard deviation, describe the recovery process. It provides the option of pruning the model during its evaluation by conservatively assuming system failure once the probability of reaching a state falls below a specified or automatically selected prune level. Faults can be modeled as permanent, transient, or intermittent. Its only input method is the state transition matrix. SURE was developed at NASA's Langley Research Center. It is written in Pascal and runs on VAX/VMS and SUN computers.

The PAWS (Padé Approximation With Scaling) and STEM (Scaled Taylor Exponential Matrix) programs, described in [BS88], evaluate the unreliability of homogeneous Markov models. The input language for these two programs is essentially the same as for the SURE program. PAWS and STEM were developed at NASA's Langley Research Center. They are written in Pascal and FORTRAN 77 and run on VAX/VMS and SUN computers.

SURF, described in [LL78], can solve semi-Markov models that use exponential distributions or non-exponential distributions that are related to the exponential (e.g., Gamma, Erlang, etc.). The method of stages [CM65] is used to produce a homogeneous Markov model. Matrix transformations are used to obtain time-independent values, such as MTTF and the limiting availability. The Laplace transform is used to obtain time-dependent values, such as availability and reliability. SURF was developed in Toulouse, France. Written in PL/I, it runs on an IBM System/370 at the IBM research facility in Yorktown Heights, New York.

For HARP (Hybrid Automated Reliability Predictor), described in [DTSG86, HBH90], the state transition probabilities can have exponential, uniform, Weibull, or general (a histogram must be provided) distributions. If the state transition matrix is given by the user, HARP can only evaluate the availability of systems with constant repair rates. HARP has several additional methods of specifying the fault-occurrence behavior (e.g., fault trees), all of which are automatically converted to a nonhomogeneous Markov model. The fault-handling behavior can also be specified by providing the transition parameters of one of several models. It uses the same behavioral decomposition/aggregation solution technique as CARE III, but the various models are solved in a hybrid fashion. Markov models are solved using numerical integration techniques, and extended stochastic Petri nets are solved by simulation. HARP was developed at Duke University and Clemson University under NASA's Langley Research Center sponsorship. It is written in FORTRAN 77 and C, and runs on the following computers: VAX/VMS, SUN, and IBM compatible PCs.

An abstract specification language for Markov reliability models was described in [But85]. The language has statements to specify: (a) the state space, by defining the state variables and their range; (b) the start state, by the initial values of the state variables; (c) the death states, by a Boolean expression of the state variables; and (d) the state transitions, by a set of if-then rules that define, in terms of the state variables, the possible transitions, their rates, and their destination states. This language has been implemented in the ASSIST (Abstract Semi-Markov Specification Interface to the SURE Tool) program to generate Markov reliability models in the SURE input language [Joh86]. This implementation provides three optional state space reduction techniques. First is pruning the model during its generation by conservatively assuming system failure once a state satisfies a prune condition specified as a Boolean expression of the state variables [Joh88]. Second is trimming the model by conservatively altering states with outgoing recovery transitions [WP90]. Their outgoing failure transitions that do not go to death states are changed to go to a single trim state. The third technique combines pruning and trimming by changing all states that meet a prune condition to trim states. Each trim state has a single transition to a death state at some trim rate. The trim rate must be the sum of the failure rates of all remaining components. ASSIST was developed at NASA's Langley Research Center. It is written in Pascal and runs on VAX/VMS and SUN computers.

ADVISER (Advanced Interactive Symbolic Evaluator of Reliability), described in [KS82], automatically generates symbolic reliability functions for PMS structures. Its assumptions are: (a) all faults are permanent and stochastically independent, (b) the PMS system has a perfect coverage, and (c) failed components are

Program Name	Primary Inputs	Primary Outputs
CARE III	fault tree and semi-Markov model parameters	reliability estimate
ARIES	homogeneous Markov model	reliability or availability estimate
SURE	semi-Markov model	reliability bounds
PAWS/STEM	homogeneous Markov model	reliability estimate
HARP	fault tree or nonhomogeneous Markov model	reliability or availability estimate
ASSIST	semi-Markov model specification	semi-Markov model
ADVISER	PMS structure	symbolic reliability function

Table 1.1: Summary of previous work.

not repaired and returned to a nonfaulty state. Its primary input is the interconnection graph of the PMS structure. Other program inputs describe the components of the PMS structure by their types, reliability functions, internal port connections, and ability to communicate with components of the same type. The program also takes as input the requirements for the system and its subsystems or clusters, in the form of modified Boolean expressions. ADVISER was developed at CMU. It is written in BLISS and runs on a PDP-10.

A summary of the programs described in this section is presented in Table 1.1 in terms of their primary inputs and outputs. None of these programs is able to generate a Markov model or its specification from the PMS structure.

Chapter 2

Graphical User Interface

This GUI is to become the first of four steps in an automated reliability modeling process using a Markov model. The second step will be automating the specification of the model in the ASSIST language. We are implementing this second step in a program named ARM (Automated Reliability Modeling). The last two steps, automating the generation and evaluation of the model, have already been implemented in the ASSIST and SURE programs.

In order of importance, the major goals of the GUI are defined below.

General. New fault-tolerant techniques and system designs can be accommodated.

Hierarchical. Systems and subsystems can be defined in terms of their subsystems and components, respectively.

Compact. Subsystem classes need only be defined with their component types as formal parameters once.

Subsystems are in the same *class* if they have the same hierarchy and requirements (e.g., triads which require two of their three components). *Subsystems* are of the same *type* if they are in the same class, are composed of the same component types, and have the same recovery parameters, if any (e.g., processor triads). *Components* are of the same *type* if they have the same function and parameters (e.g., processors). These categories of subsystems and components are summarized and ranked relative to each other in Table 2.1. For the sake of generality, the GUI does not predefine any category.

Category	Common Attributes	Rank
subsystem class	hierarchy requirements	{[+++][+++][+++]}
subsystem type	component types recovery parameters	[+++]
component type	function parameters	+

Table 2.1: Categories of subsystems and components.

Major GUI Input Category	Source
requirements	application
architecture	design
parameters	implementation technology

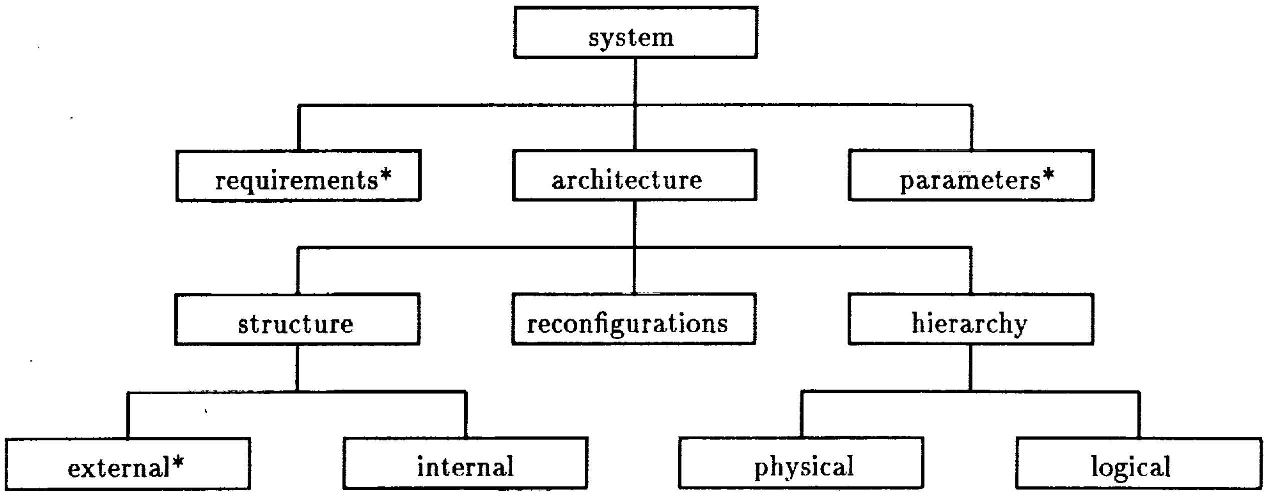
Table 2.2: Sources of major GUI input categories.

Each category is represented by an identifier that starts with a letter and can contain letters, underscores (_), and numbers (e.g., a processor could be represented by p). This identifier can be preceded by an integer greater than one to represent multiple elements of the same category (e.g., two processors could be represented by $2p$). A subsystem identifier can also end with a set of parentheses that enclose a list of parameters separated by commas [e.g., $T(p)$]. Formal parameters, identifiers which are not used to represent anything else, are used in the identifier of a subsystem class. Component types are used instead of the formal parameters in the identifier of a subsystem type.

The system's description is divided into requirements, architecture, and parameters. The requirements depend on the application of the system. How the system was designed determines the architecture. The technology used to implement the system components determines the parameter values (e.g., failure rates). The sources of the major GUI input categories are summarized in Table 2.2. Figure 2.1 shows the hierarchy of the system description.

The GUI starts by displaying the main window shown in Figure 2.2. It contains: text fields for entering the system name and the name of the current selection; the graphs, parameters, and model pull-down menus; and a button to quit the GUI. The current selection is the initial name used by windows that describe a component type, subsystem type, or reconfiguration. It changes automatically to the last name entered in the first text field of any such window, but can also be changed manually.

The graphs menu, shown in Figure 2.3, displays a window for editing the graphs described in Section 2.1.



Key: Parts that are not optional are marked with an asterisk (*).

Figure 2.1: System description hierarchy.

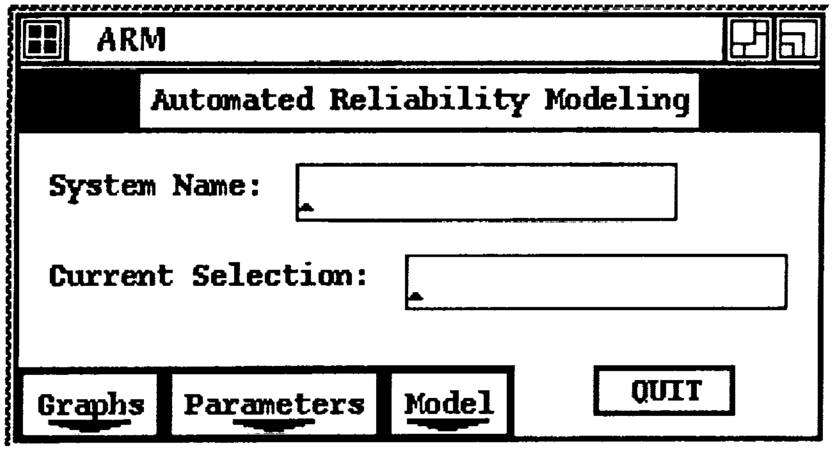


Figure 2.2: Main window.

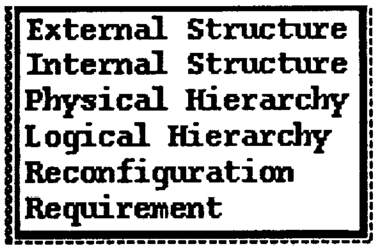


Figure 2.3: Graphs menu.

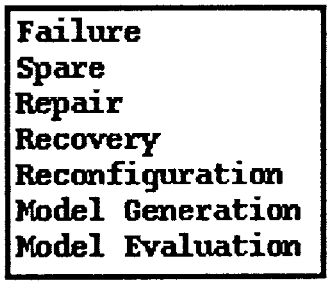


Figure 2.4: Parameters menu.

The parameters menu, shown in Figure 2.4, displays windows, with text fields and buttons for parameter specification, which are described in Section 2.2. The model menu, shown in Figure 2.5, executes the programs that will specify, generate, and evaluate the Markov model, based on the system description given through the GUI. ARM will notify the user if the system description is incomplete (e.g., if some parameters have not been specified).

2.1 Graphs

The following subsections describe the graphs used for specifying the system's communication structure, hierarchy, reconfiguration capabilities, and requirements.

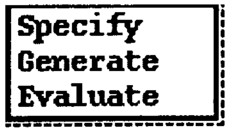


Figure 2.5: Model menu.

2.1.1 Structure

Graphs with unidirectional and bidirectional edges describe the system's communication structure. It is assumed that critical components, those required for the system to be operational, must be able to communicate. The main purpose of the communication structure description is to analyze which component failures will prevent communication between critical components and therefore cause system failure. This structure is divided into external and internal.

2.1.1.1 External

A system's external structure is defined as the communication interconnection of all of its components. In the external structure graph, the nodes represent one or more components of the same type. An unidirectional edge between two nodes indicates that all the components of the source node can communicate with all the components of the target node. A bidirectional edge between two nodes indicates that all the components of one node can communicate with all the components of the other node, and vice versa.

A plus sign (+) at the end of a component type identifier indicates that this is a self-talking component. The majority of components of like type are passive and do not need to communicate. Examples of passive components are memories, buses, and input/output transducers. Self-talking components need to exchange information amongst each other. Examples of self-talking components are processors, direct-memory-access device controllers, and other "smart" controllers. If not specified, the default is for components to be passive and not communicate with their own type. This information is needed to prevent ARM from requiring communication paths between components of the same type that never exchange information. Not taking this behavior into account would lead to a pessimistic evaluation of the system reliability.

An asterisk (*) at the end of a component type identifier indicates that every communication port of this component is internally connected bidirectionally to all other ports of the component. Most buses have this internal structure. If not indicated in this way or as described in Subsubsection 2.1.1.2, the default is for every port of a component to be disconnected from the other ports of the component.

The graph in Figure 2.6 describes the external structure of a multiprocessor composed of six processors (p), six memories (m), six watchdog timers (w), four transmit buses (tb), four receive buses (rb), and four

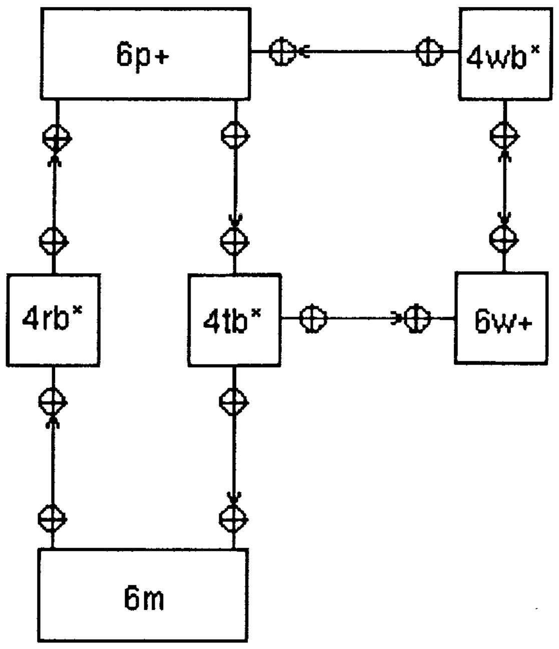


Figure 2.6: External structure of a multiprocessor.

watchdog buses (*wb*). The processors and watchdog timers need to communicate with components of their own type. The processors communicate through the memory as described in Subsubsection 2.1.1.2. The watchdog timers communicate through the watchdog bus. All of the buses have the typical internal structure described above. This multiprocessor will be used as a running example throughout this chapter.

2.1.1.2 Internal

A component's internal structure is defined as the communication interconnection of its ports. This internal structure of one or more components can be described by a graph inside of a component with its external port connections labeled on the outside of the component. The absence of an edge between two ports indicates that they can not communicate through this component.

The internal structure of the six memories is described by the graph in Figure 2.7. It indicates that the processors can communicate through the memory.

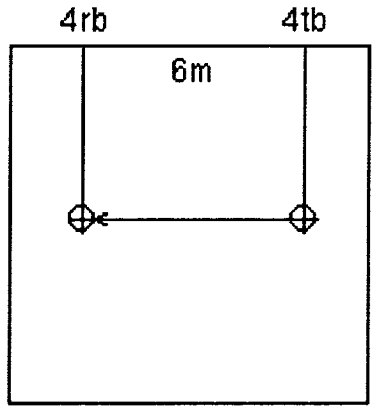


Figure 2.7: Internal structure of the six memory components.

2.1.2 Hierarchy

A system can have physical and/or logical hierarchies which contain physical and logical subsystems, respectively. These hierarchies are different partial views of the same system; therefore, a component of a physical subsystem may also be a component of a logical subsystem. The difference between a physical and a logical subsystem is in their ability to be reconfigured and in how their failure affects the system's operation, as explained in the next two subsubsections. If present, the system hierarchies (a) show what subsystems are in the initial system configuration and (b) define the composition of the subsystems that may be part of those hierarchies.

A group of components with their own set of requirements constitutes a subsystem. A subsystem can also be composed of other subsystems. If a subsystem does not meet its requirements, then none of its components are able to perform their function.

Redundant subsystems are composed of multiple components with the same function to increase their reliability. Some of these redundant subsystems may be part of the initial system configuration, while others serve as alternatives for system reconfiguration (e.g., a quad subsystem that reconfigures into a triad).

A system hierarchy is described by nondirectional tree graphs. Root nodes (identified by a circle) represent the system or one of its subsystems. Other nodes (identified by a rectangle) represent one or more identical subsystems or components.

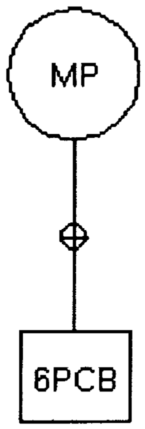


Figure 2.8: Physical hierarchy of the multiprocessor.

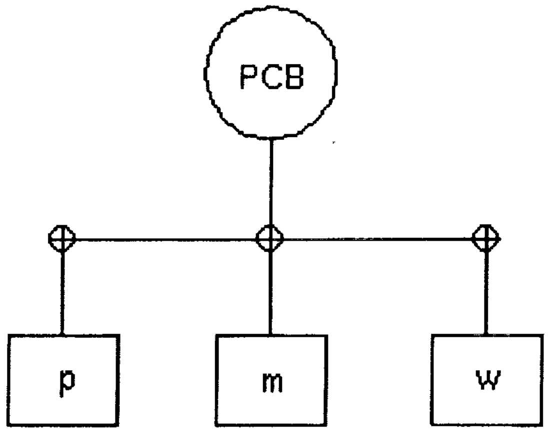


Figure 2.9: Physical hierarchy of the printed circuit board subsystem type.

2.1.2.1 Physical

Physical subsystems can not be reconfigured. However, the failure of a physical subsystem does not preclude the system from operating, as long as the system requirements are met.

Figures 2.8 and 2.9 describe the physical hierarchy of the multiprocessor. Initially, the multiprocessor contains six printed circuit boards, which belong to the same physical subsystem type. Each board contains a processor, memory, and a watchdog timer.

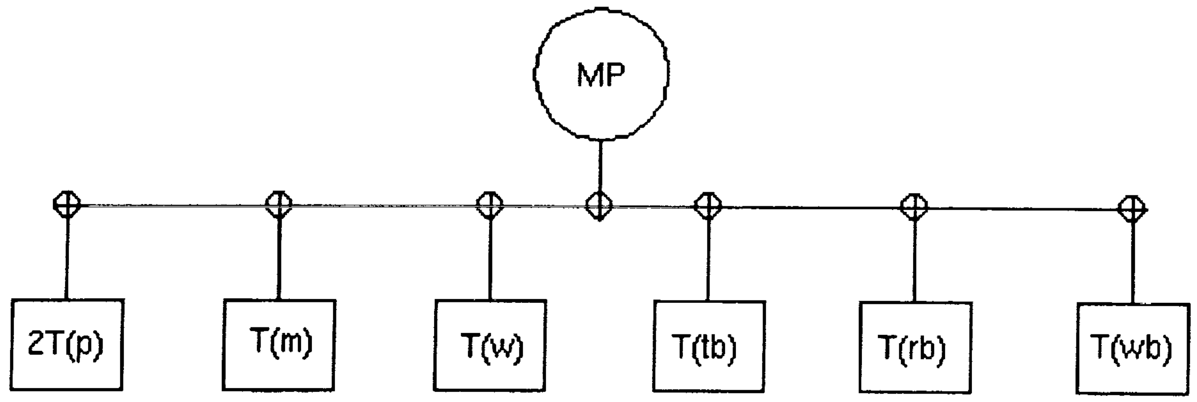


Figure 2.10: Logical hierarchy of the multiprocessor.

2.1.2.2 Logical

Logical subsystems can be reconfigured. Before component failures cause them to fail, they can recover by replacing the failed components with spares. If there are not enough spares available, the system can degrade to a lesser number of subsystems or a less redundant subsystem. When a logical subsystem fails, the system also fails unless it can be reinitialized by a separate subsystem or component.

Figures 2.10 and 2.11 describe the logical hierarchy of the multiprocessor. Initially, the multiprocessor contains two processor triads, one memory triad, one watchdog triad, one transmit bus triad, one receive bus triad, and one watchdog bus triad. These triads are each composed of three components of the same type.

ARM will automatically determine what components are spares by comparing the external structure with the logical hierarchy; any extra instances of components in the external structure, beyond what is included in the logical hierarchy, will be assumed to be spares. Therefore, from Figures 2.6 and 2.10 it will be assumed that the spare components are: three memories, three watchdog timers, one transmit bus, one receive bus, and one watchdog bus.

2.1.3 System Reconfiguration

The future system configurations are described in terms of the reconfigurations allowed. A change in the system's configuration in response to some event is defined as a reconfiguration. A reconfiguration occurs

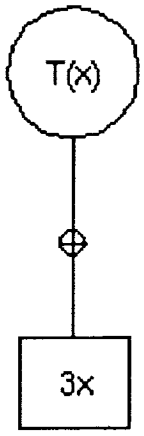


Figure 2.11: Logical hierarchy of the triad subsystem class.

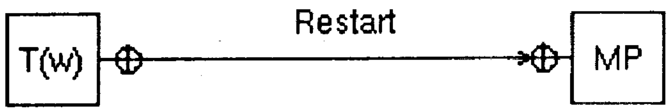


Figure 2.12: Reinitialization of the multiprocessor.

when the system is reinitialized because of a logical subsystem failure, or when the system degrades to a lesser number of subsystems or a less redundant subsystem because there are no spares to replace a failed component. After a certain time period the mission phase may change causing the system to reconfigure.

Reconfigurations are described by unidirectional graphs. The source nodes are the components and subsystems (physical or logical) that must be active before the reconfiguration. The destination nodes are the reinitialized system, or the logical subsystems that will be active after the reconfiguration, in place of the logical subsystems identified by the source nodes. Each edge is labeled with the name of a specification that will provide the reconfiguration parameters.

Figure 2.12 describes the reinitialization of the multiprocessor by the watchdog triad. Figure 2.13 describes the degradation of the multiprocessor from two to one processor triad. The working processors in the deactivated triad are assumed to become spares.

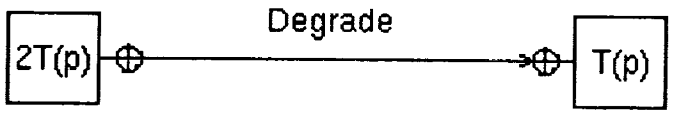


Figure 2.13: Degradation of the multiprocessor.

2.1.4 Requirement

The requirements of a system, subsystem, or performance level are defined as the minimum set of subsystems, components, and performance levels needed. These levels are used to identify the nondegraded mode and the various degraded modes of operation a system might have.

These requirements are described by one or more success trees. Root nodes (identified by a circle) represent the system, one of its subsystems, or a performance level. Other nodes (identified by a rectangle) represent one or more identical subsystems, a performance level, or one or more identical components. Their advantages over fault trees are that: (a) they are more intuitive for a computer engineer concerned with making the system work and not with how it can fail; and (b) a conservative reliability estimate is produced if some modes of operation are left out of the success tree, because system failure is assumed for those modes of operation, whereas an optimistic reliability estimate is produced if a failure mode is left out of a fault tree.

The graphs in Figures 2.14 through 2.16 describe the system and subsystem requirements of the multiprocessor. This multiprocessor can operate at one of two performance levels. To achieve full performance (FP) both processor triads, the watchdog triad, and the memory triad must be operational. The requirements for degraded performance (DP) are the same except that only one processor triad is needed. The components in the printed circuit board can operate as long as the memory on the board has not failed.

2.2 Parameters

The following subsections describe the parameter specification windows. Any time unit may be used for the parameter values as long as it is the same one for all of them. The OK and CANCEL buttons in each window save and discard the parameter changes made, respectively. Both buttons make the window disappear.

Initially, numeric and selection parameters are assigned an appropriate default value. Probabilities

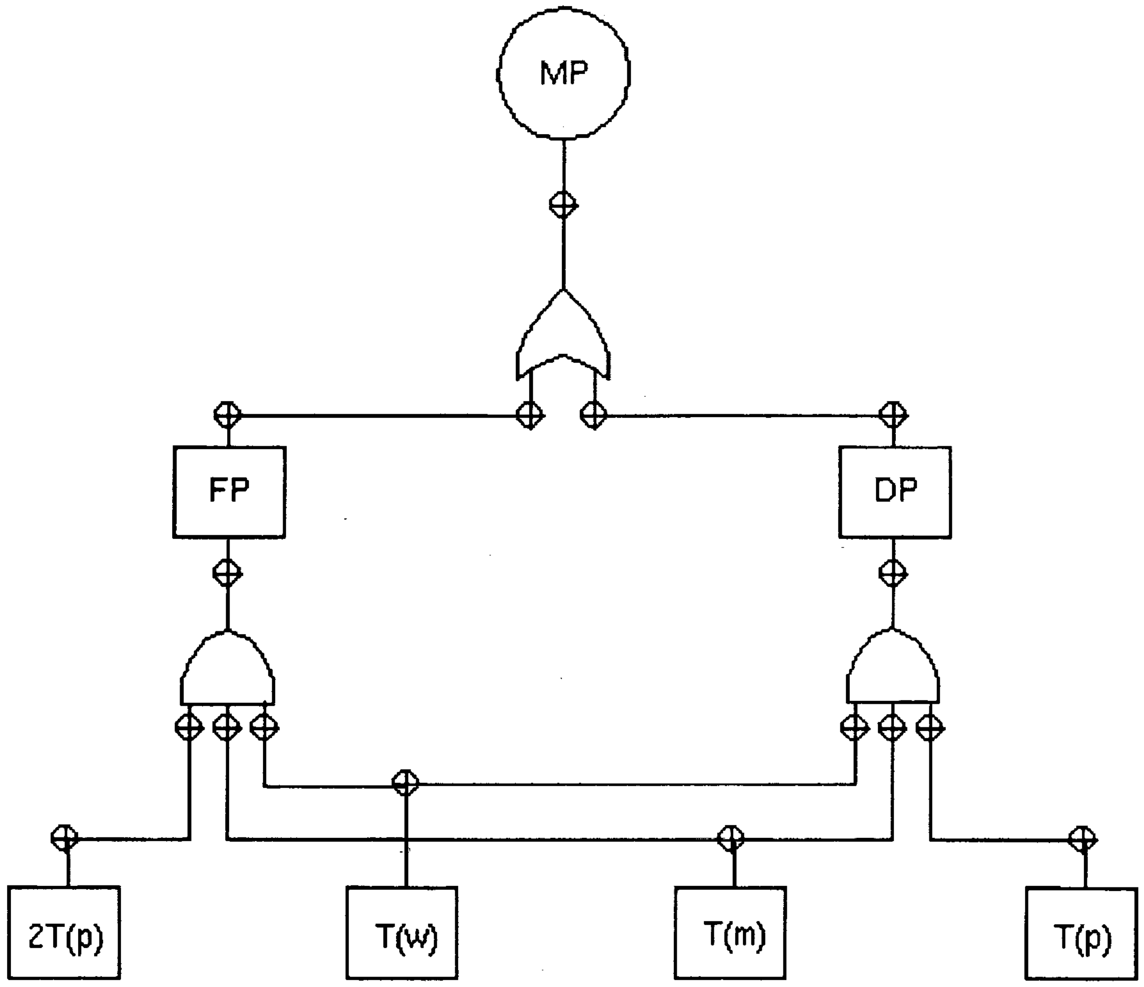


Figure 2.14: Requirements of the multiprocessor.

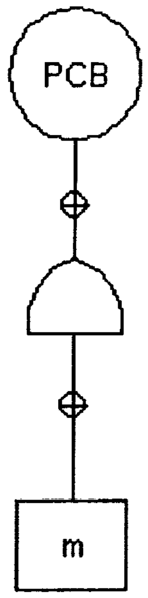


Figure 2.15: Requirement of the printed circuit board subsystem type.

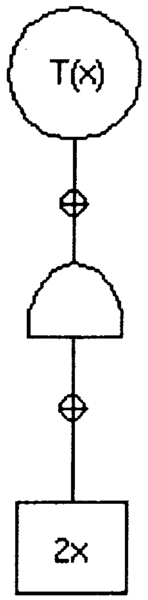


Figure 2.16: Requirements of the triad subsystem class.

default to one or zero depending on what is most commonly used. Rates, means, and standard deviations default to zero. Integer and selection parameters default to their most commonly used values.

Instead of values, numeric parameters can also be given variable identifiers that start with a letter and can contain letters, underscores (_), and numbers. One of these variables can be given a range as described in Subsection 2.2.7. The SURE program will prompt for the value of variables without a range.

Numeric parameters will be assumed to be independent of the system state. This assumption is not strictly true. However, it is often close enough in practice to be used to simplify the analysis.

2.2.1 Active Component Failure

The active component failure parameters are shown in Figure 2.17 with example values. These are the only parameters that are always required for each component type, the other parameters are optional. First is the name of the component type.

Second is the probability (ϵ) a fault in an active component of this type will be a single point of failure for the system, either immediately or during the recovery process. This is equal to one minus the fault coverage of an active component of this type.

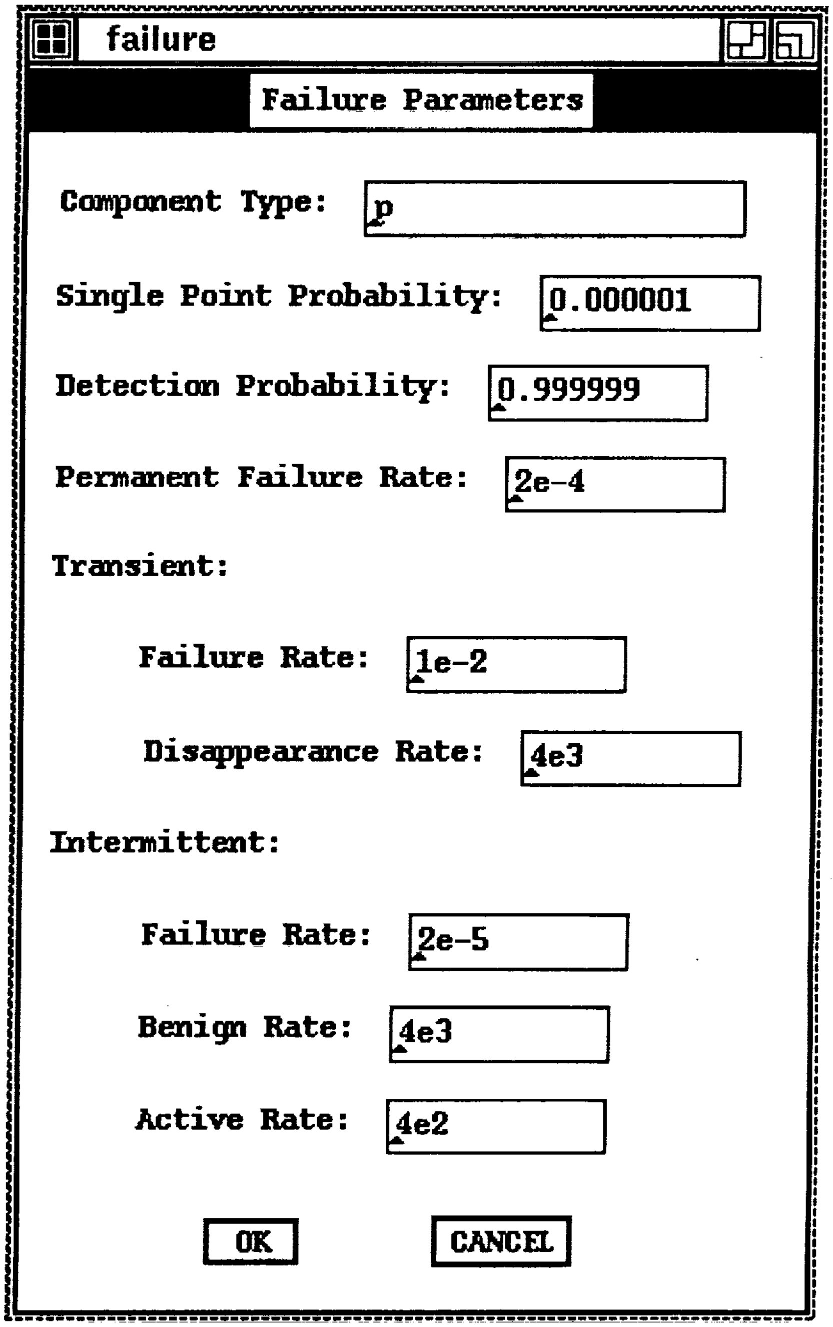
Third is the probability (θ) a fault in an active component of this type will be detected and correctly isolated. Fourth is the exponential arrival rate (λ) of permanent faults. The next two are the exponential arrival and disappearance rates (τ and δ) of transient faults. The last three are the exponential rates (ω, β, and α) at which intermittent faults arrive, become benign, and become active again.

2.2.2 Spare Component Failure

The spare component failure parameters are shown in Figure 2.18 with example values. First is the name of the component type.

Second is the failure rates factor used to indicate which type of spare this is. It is: zero for cold, in the exclusive range of zero to one for warm, and one for hot. This failure rates factor is the spare's fraction of the active component failure rates.

Third is the probability a fault in a spare component of this type will be a single point of failure for the



Failure Parameters

Component Type: p

Single Point Probability: 0.000001

Detection Probability: 0.999999

Permanent Failure Rate: 2e-4

Transient:

Failure Rate: 1e-2

Disappearance Rate: 4e3

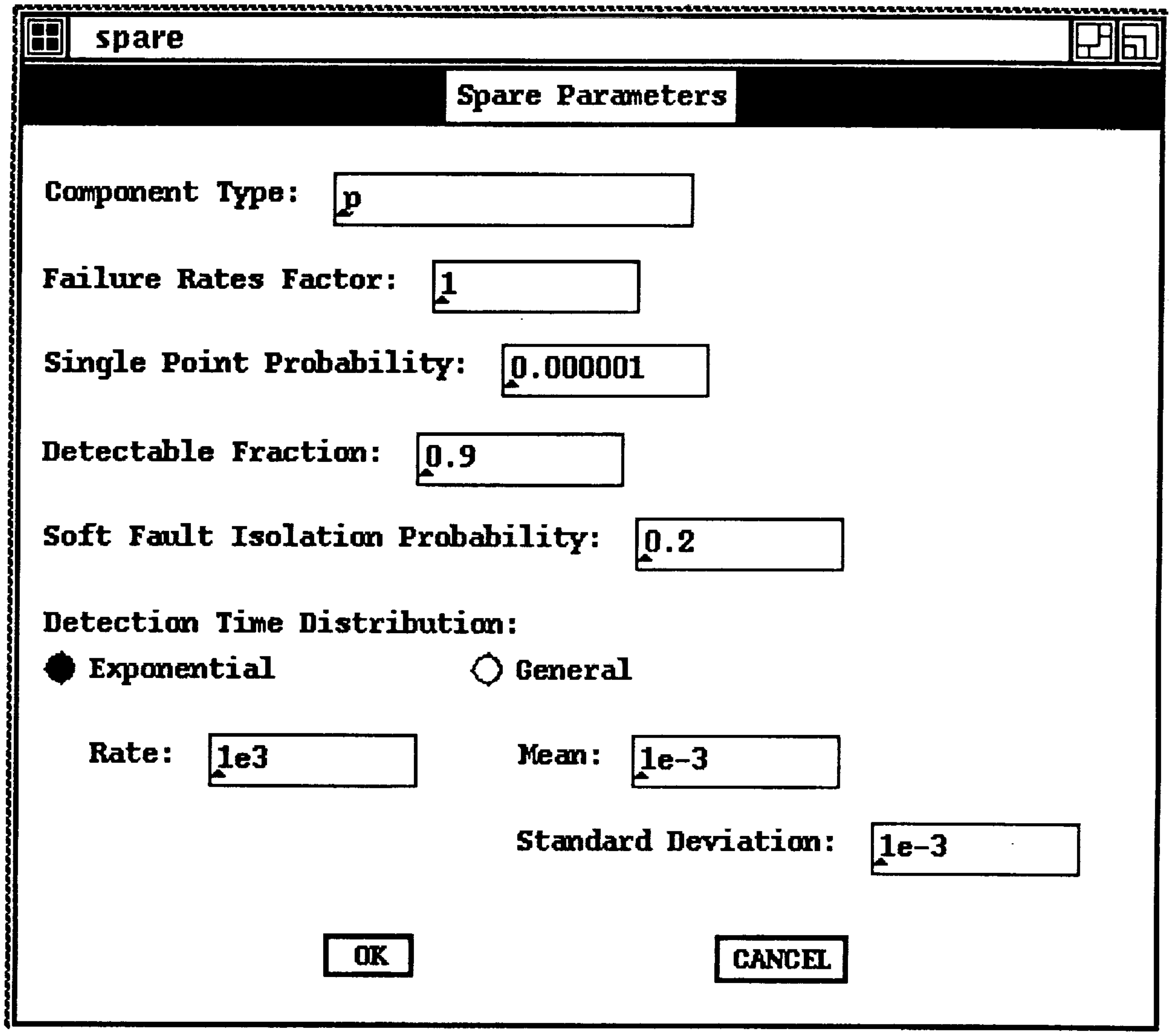
Intermittent:

Failure Rate: 2e-5

Benign Rate: 4e3

Active Rate: 4e2

Figure 2.17: Active component failure parameters.



Spare Parameters

Component Type: p

Failure Rates Factor: 1

Single Point Probability: 0.000001

Detectable Fraction: 0.9

Soft Fault Isolation Probability: 0.2

Detection Time Distribution:

[✓ Exponential] General

Rate: 1e3

Mean: 1e-3

Standard Deviation: 1e-3

Figure 2.18: Spare component failure parameters.

system, either immediately or during the recovery process. This is equal to one minus the fault coverage of a spare component of this type.

Fourth is the fraction of faults that can be detected in a component of this type while it is a spare. Fifth is the probability a soft fault can be isolated, before it disappears if it is transient or becomes benign if it is intermittent, such that the faulty spare component can be reconfigured out of the system.

Soft fault isolation probabilities are assumed to be the same for transients and intermittents. This assumption is not strictly true. However, it is often close enough in practice to be used to simplify the analysis.

The sixth parameter indicates whether the detection time, of those faults that can be detected, follows an exponential or general distribution. The next three parameters are the detection time: rate for an exponential distribution, and mean and standard deviation for a general distribution.

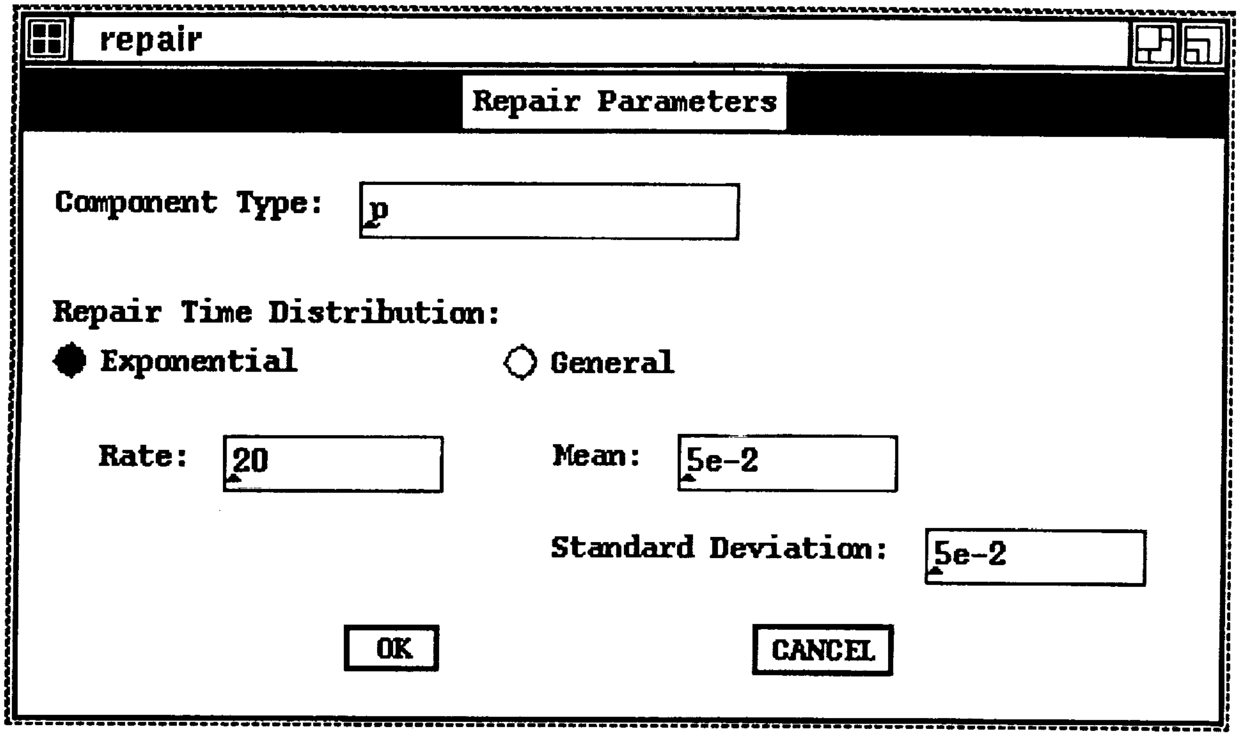
2.2.3 Component Repair

The component repair parameters are shown in Figure 2.19 with example values. First is the name of the component type. The second parameter indicates whether the repair time follows an exponential or general distribution. The next three parameters are the repair time: rate for an exponential distribution, and mean and standard deviation for a general distribution.

2.2.4 Subsystem Recovery

The recovery parameters are shown in Figure 2.20 with example values. First is the name of the subsystem type. The second parameter is the probability (ρ) a soft fault can be isolated such that the subsystem can recover. The third parameter indicates whether the recovery time follows an exponential or general distribution. The next three parameters are the recovery time: rate for an exponential distribution, and mean (μ) and standard deviation (σ) for a general distribution.

Figure 2.21 illustrates the meaning of the failure and recovery parameters using a partial Markov model. Except for states 0 and 1, they all have additional transitions to additional states, neither of which are shown. A transition that follows an exponential time distribution is labeled with a single expression that


repair

Repair Parameters

Component Type: p

Repair Time Distribution:

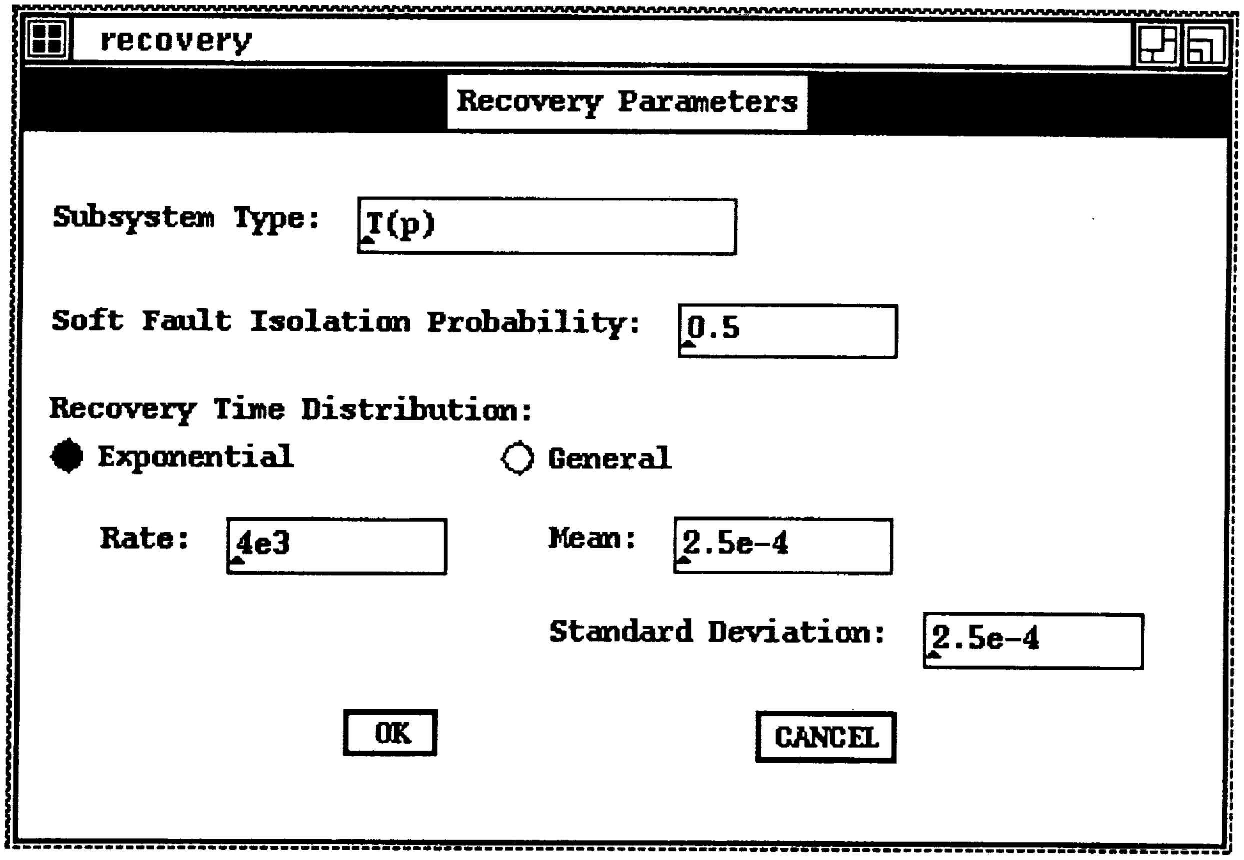
● Exponential ○ General

Rate: 20

Mean: 5e-2

Standard Deviation: 5e-2

Figure 2.19: Component repair parameters.


recovery

Recovery Parameters

Subsystem Type: T(p)

Soft Fault Isolation Probability: 0.5

Recovery Time Distribution:

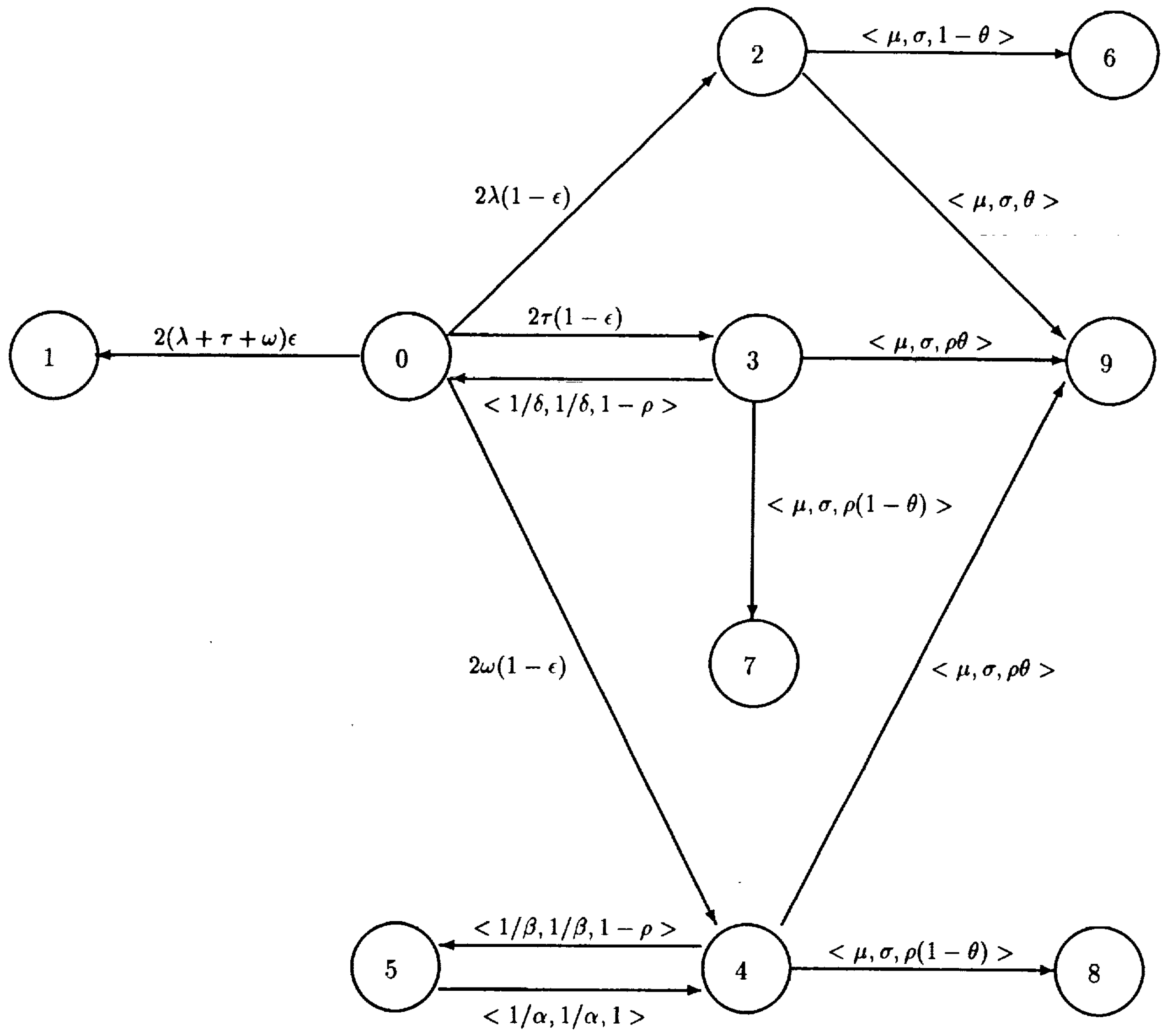
● Exponential ○ General

Rate: 4e3

Mean: 2.5e-4

Standard Deviation: 2.5e-4

Figure 2.20: Subsystem recovery parameters.


$< \mu, \sigma, 1-\theta >$
$2\lambda(1-\epsilon)$
$< \mu, \sigma, \theta >$
$2(\lambda+\tau+\omega)\epsilon$
$2\tau(1-\epsilon)$
$< \mu, \sigma, \rho\theta >$
$< 1/\delta, 1/\delta, 1-\rho >$
$< \mu, \sigma, \rho(1-\theta) >$
$2\omega(1-\epsilon)$
$< \mu, \sigma, \rho\theta >$
$< 1/\beta, 1/\beta, 1-\rho >$
$< \mu, \sigma, \rho(1-\theta) >$
$< 1/\alpha, 1/\alpha, 1 >$
0
1
2
3
4
5
6
7
8
9

Key: State Description

0	no faults, n spares
1	system failed
2	1 permanent fault, n spares
3	1 transient fault, n spares
4	1 active intermittent fault, n spares
5	1 benign intermittent fault, n spares
6	1 permanent fault, $n-1$ spares
7	1 transient fault, $n-1$ spares
8	1 active intermittent fault, $n-1$ spares
9	no faults, $n-1$ spares

Figure 2.21: Partial Markov model of a processor duplex with n cold spares and no repair.

represents its rate. A transition that follows a general time distribution is labeled with three expressions that represent the mean, the standard deviation, and the probability this transition will take place.

2.2.5 System Reconfiguration

The system reconfiguration parameters are shown in Figure 2.22 with example values. First is the specification name. The second parameter indicates whether this reconfiguration is triggered by a logical subsystem failure, a mission phase change, or a component in a logical subsystem failing without a spare. Third is the name of the component type whose failure without a spare will trigger the reconfiguration. Fourth is the probability a soft fault can be isolated such that the system can be reconfigured. The rest of the parameters specify the combined time distribution of the reconfiguration and the mission phase change, if any.

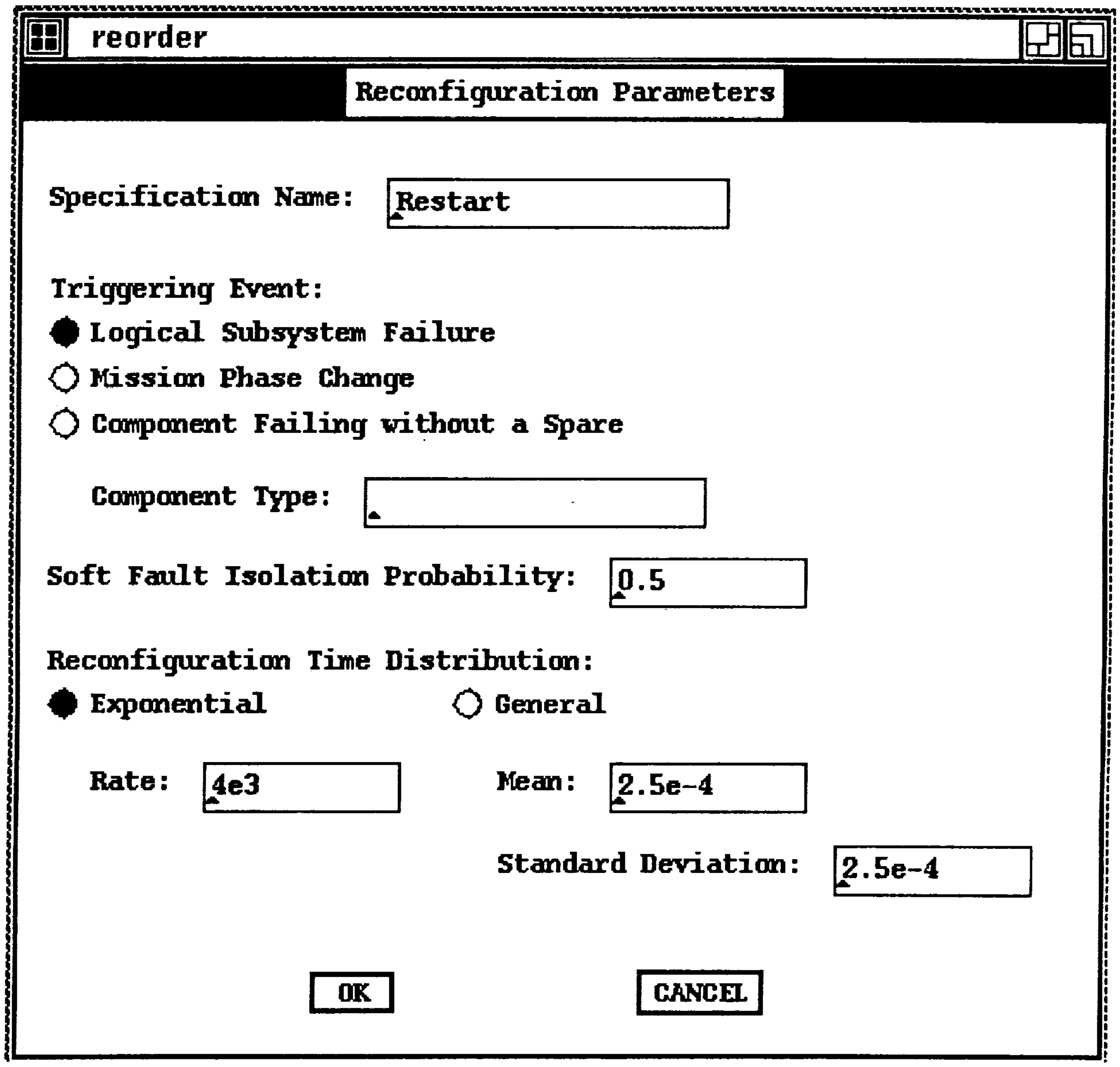
2.2.6 Model Generation

The optional window shown in Figure 2.23 with default values allows a user familiar with the ASSIST program, described in Section 1.2, to select which, if any, state space reduction technique is to be used in generating the model and to specify any associated parameters. First is the optional ASSIST prune condition, which is specified as a Boolean expression of the state variables. The second parameter indicates the optional trimming method to be used. The last two parameters indicate whether the trim rate should be selected automatically or manually, and the trim rate to be used when it is selected manually.

2.2.7 Model Evaluation

The optional window shown in Figure 2.24 with default values allows a user familiar with the SURE program, described in Section 1.2, to specify parameters used in the evaluation of the model. First is the mission time used for calculating the failure probability. Second is the number of times the SURE program will go around a loop in the model before truncating its traversal. Third is the number of digits of accuracy required. The SURE program will issue a warning if pruning and truncation result in an upper bound on the failure probability that does not meet this accuracy requirement.

The next two parameters indicate whether the SURE prune level should be selected automatically or


reorder

Reconfiguration Parameters

Specification Name: Restart

Triggering Event:
[✓ Logical Subsystem Failure]
Mission Phase Change
Component Failing without a Spare

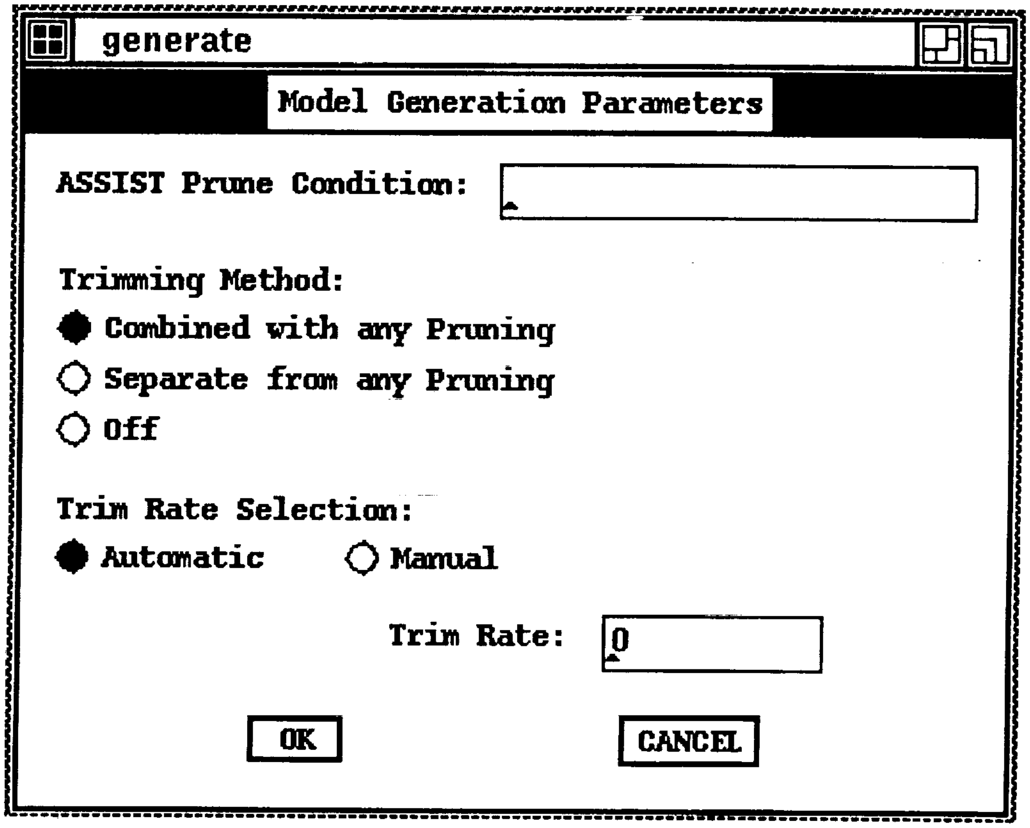
Component Type:

Soft Fault Isolation Probability: 0.5

Reconfiguration Time Distribution:
[✓ Exponential] General

Rate: 4e3
Mean: 2.5e-4
Standard Deviation: 2.5e-4

Figure 2.22: System reconfiguration parameters.



generate

Model Generation Parameters

ASSIST Prune Condition:

Trimming Method:

[✓ Combined with any Pruning]

Separate from any Pruning

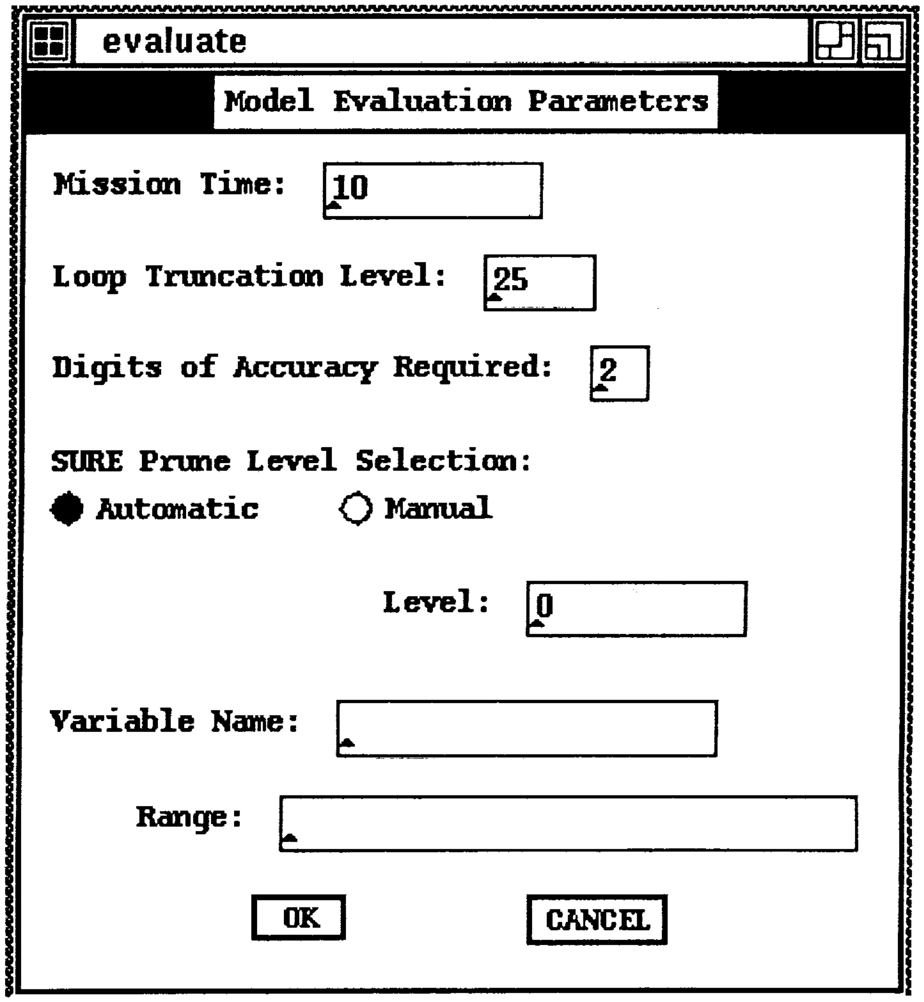
Off

Trim Rate Selection:

[✓ Automatic] Manual

Trim Rate: 0

Figure 2.23: Model generation parameters.



evaluate

Model Evaluation Parameters

Mission Time: 10

Loop Truncation Level: 25

Digits of Accuracy Required: 2

SURE Prune Level Selection:

[✓ Automatic] Manual

Level: 0

Variable Name:

Range:

Figure 2.24: Model evaluation parameters.

manually, and the prune level to be used when it is selected manually. ASSIST pruning affects which states are generated, whereas SURE pruning affects which of the generated states are evaluated. If no SURE pruning is desired, the SURE prune level selection should be manual and the prune level should be left at its default value of zero.

The last two parameters are optional. They are used when the failure probability is to be calculated as a function of a previously defined variable. In that case, the name of the variable must be given along with its range. The range can be specified as:

$$l \text{ to } h \text{ add } i$$

where l and h are the low and high ends of the range and i is the increment added to vary the variable's value over that range. The range can also be specified as:

$$l \text{ to } h \text{ by } f$$

where f is the multiplication factor used to vary the variable's value over the range.

Chapter 3

Example

The FTMP is composed of 10 line replaceable units (LRUs) and five buses that connect them. Each LRU contains a processor (*p*), memory (*m*), clock (*c*), input/output port (*io*), and power supply (*ps*). Each bus has five lines. A transmit bus (*tb*) line is used for a processor to transmit to a memory or input/output port. A receive bus (*rb*) line is used for a processor to receive from a memory or input/output port. The processors and clocks communicate with components of their own type by using the poll bus (*pb*) and clock bus (*cb*), respectively.

The components of the 10 LRUs are initially configured into three processor triads, two memory triads, and one clock quad to provide full performance. The clock bus is initially configured as a quad but degrades to a triad if two bus lines fail. If the clock bus degrades to a triad, the clock quad is also degraded to a triad. The other buses are configured as triads from the start.

The system requires two processor triads, two memory triads, and one clock quad or triad to provide degraded performance. Each triad requires that two of its components be working for it to be operational. Each quad requires that three of its components be working for it to be operational. Each LRU requires that its memory, clock, input/output port, and power supply be working for it to be operational. If two components in a triad or quad fail before one can be replaced by a spare, the system will crash.

When a component in a triad or quad fails it is replaced by a spare if available. Otherwise, quads degrade into triads. If there are three processor triads, they degrade into two triads.

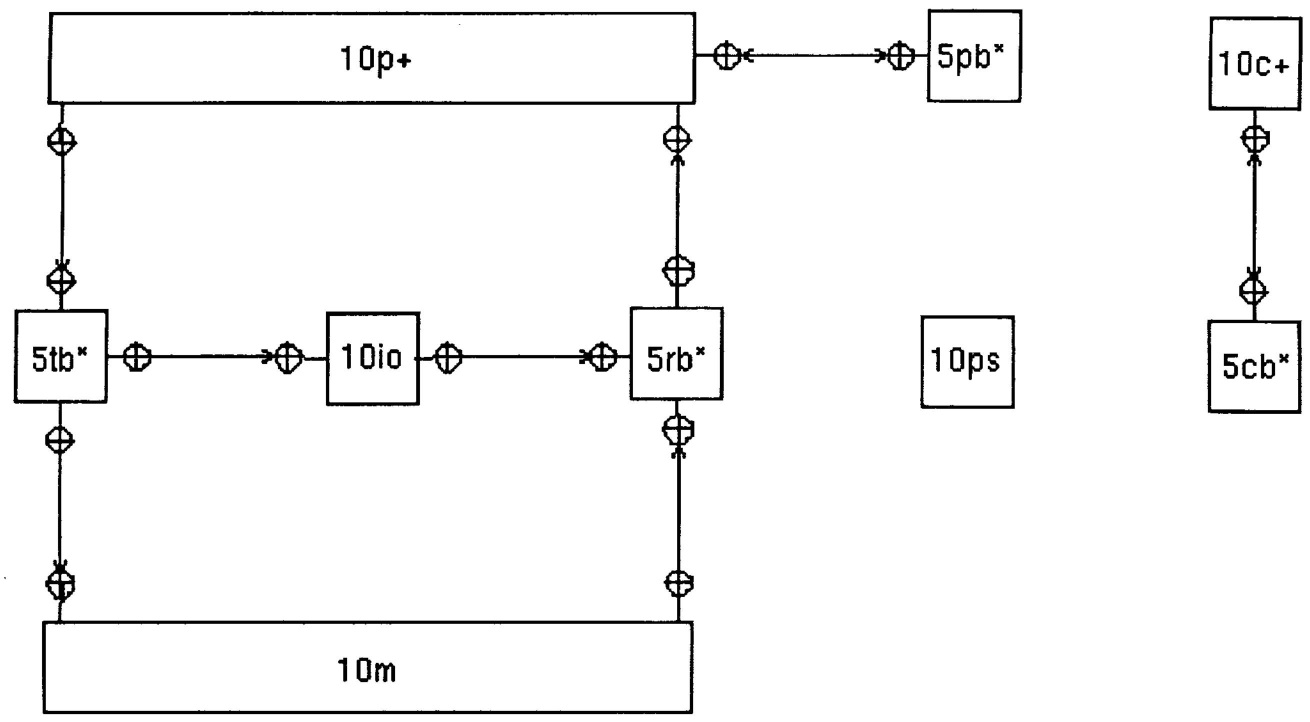


Figure 3.1: External structure of the FTMP.

The architecture and requirements of the FTMP are described by Figures 3.1 through 3.10. The logical hierarchy and requirements of the triad subsystem class are not shown since they are the same as those shown in Figures 2.11 and 2.16, respectively. The parameters of the FTMP are not shown since they would be specified using the same windows as those shown in Section 2.2.

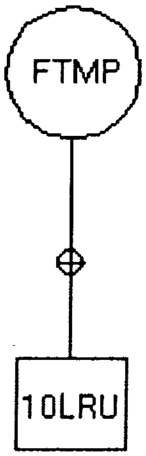


Figure 3.2: Physical hierarchy of the FTMP.

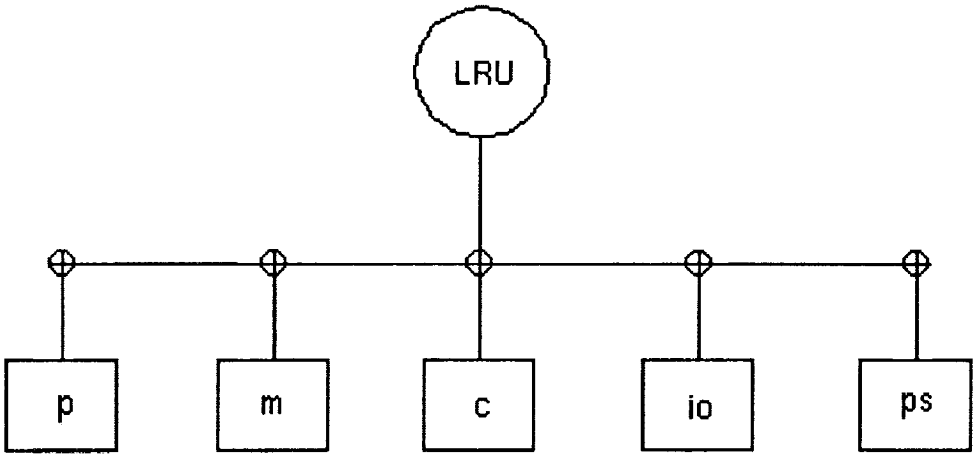


Figure 3.3: Physical hierarchy of the LRU subsystem type.

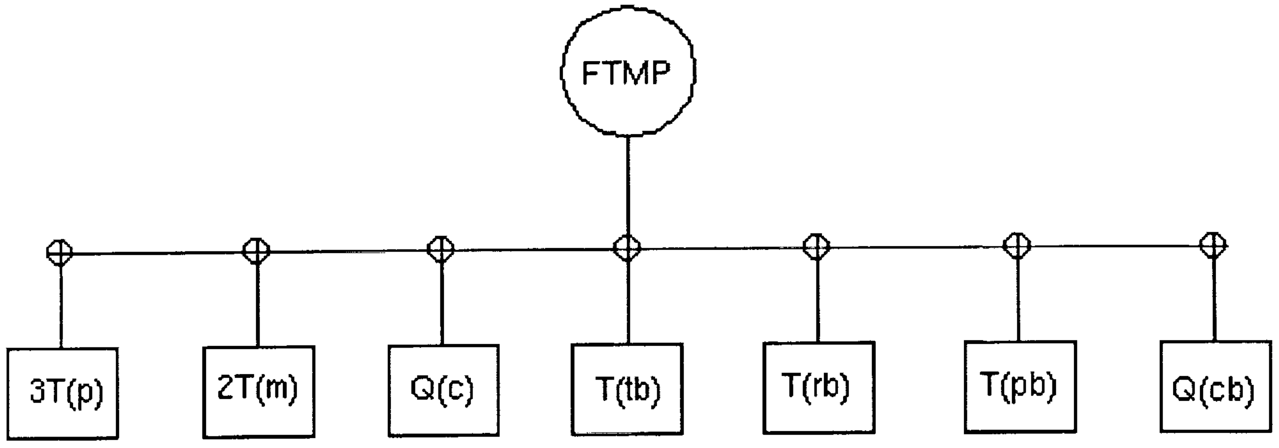


Figure 3.4: Logical hierarchy of the FTMP.

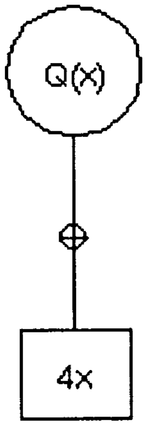


Figure 3.5: Logical hierarchy of the quad subsystem class.

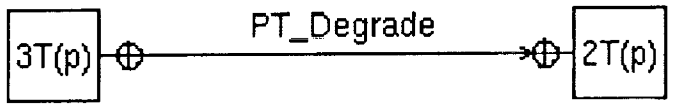


Figure 3.6: Degradation to less processor triads.

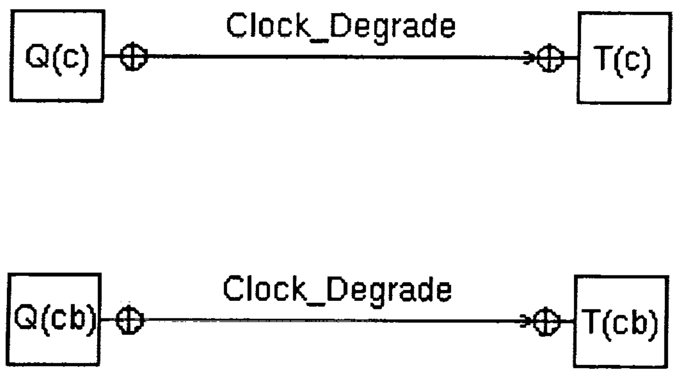


Figure 3.7: Degradation to less clock redundancy.

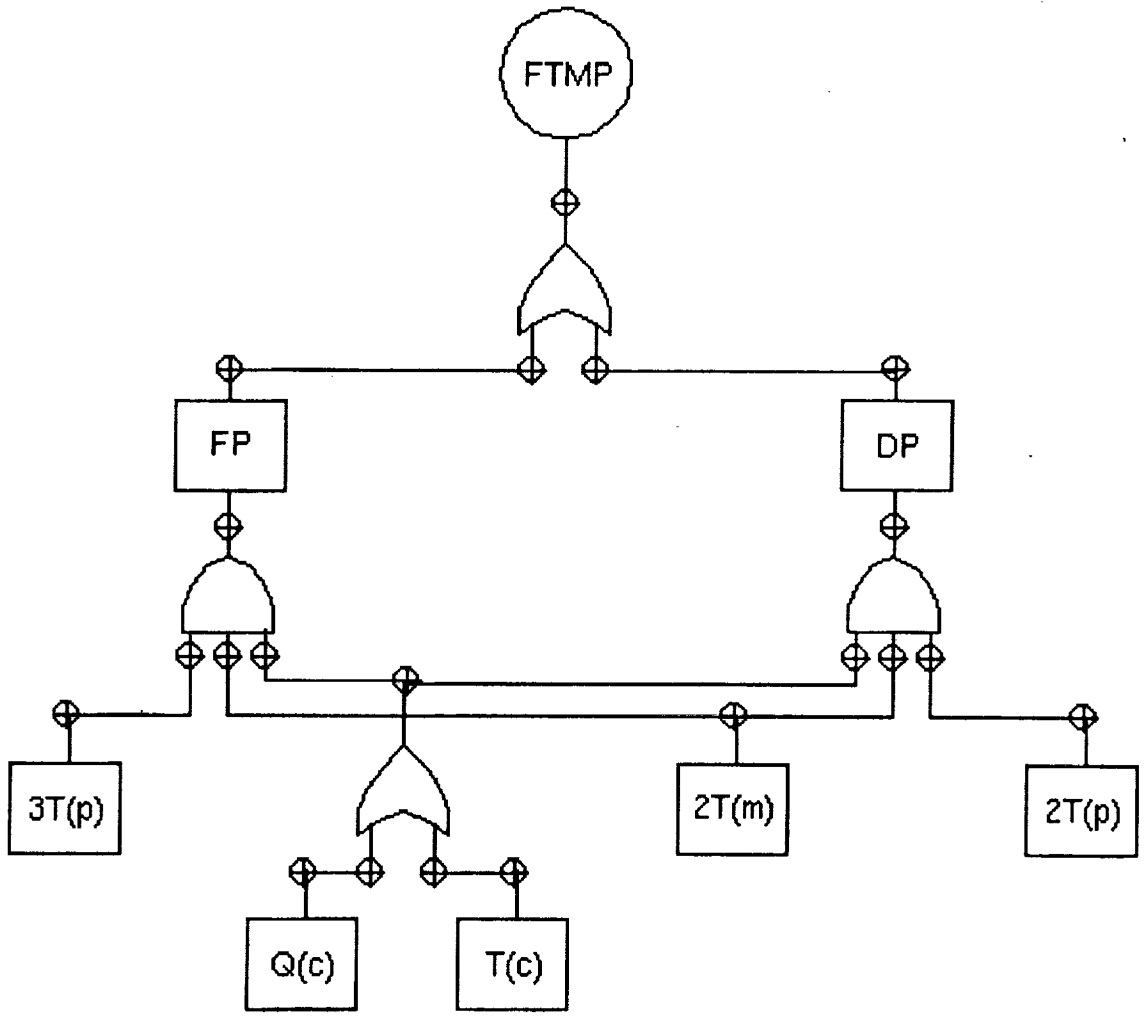


Figure 3.8: Requirements of the FTMP.

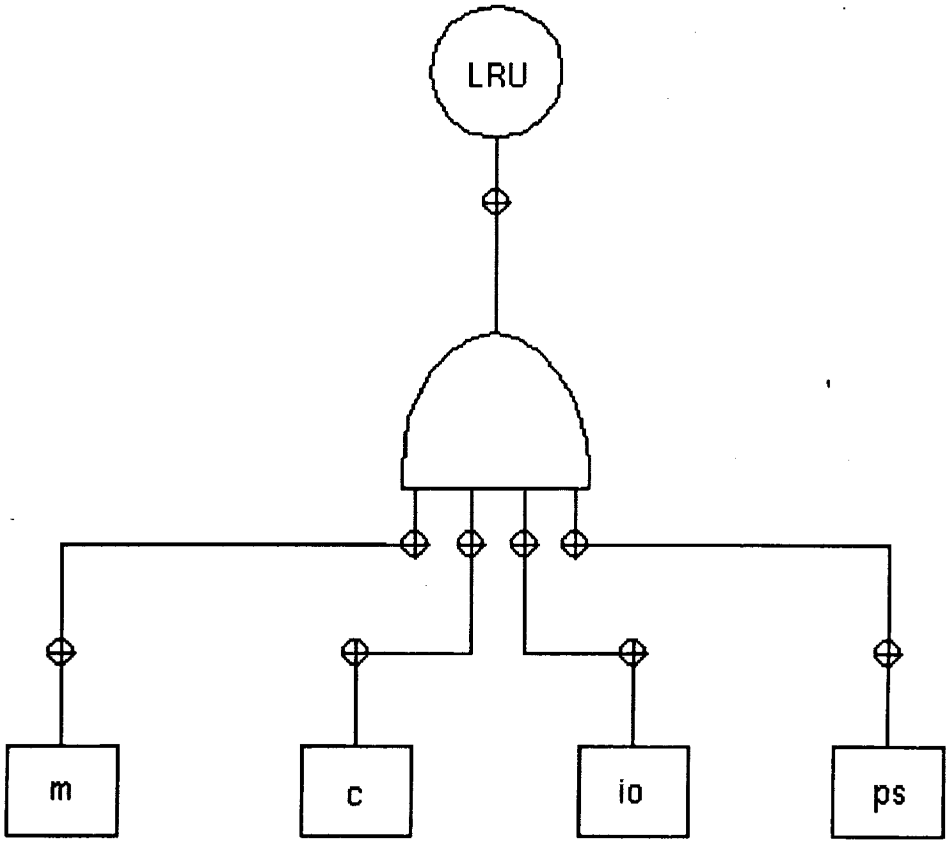


Figure 3.9: Requirements of the LRU subsystem type.

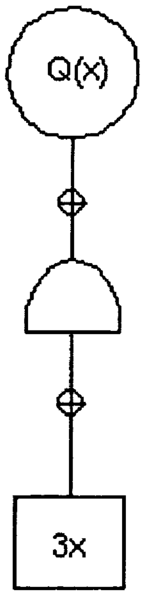


Figure 3.10: Requirements of the quad subsystem class.

Chapter 4

Conclusions

This paper has presented a general graphical user interface (GUI) for automatic reliability modeling of processor-memory-switch structures using a Markov model. This GUI is based on a hierarchy of windows. One window has graphical editing capabilities for specifying the system's communication structure, hierarchy, reconfigurations, and requirements. Other windows have text fields, pop-up menus, and buttons for specifying parameters and selecting actions. The application of the GUI was exemplified using the Fault-Tolerant Multiprocessor described in [LS83].

This GUI is to become the first of four steps in an automated reliability modeling process using a Markov model. The second step, which we are implementing in the ARM program, will be automating the specification of the model using the ASSIST language. The last two steps, automating the generation and evaluation of the model, have already been implemented in the ASSIST and SURE programs.

Since the GUI is not specific to these programs, it could also be used as the front-end for other reliability analysis programs. The advantages of the GUI approach are (a) utility to a larger class of users, not necessarily expert in reliability analysis, and (b) a lower probability of human error in the computation.

Bibliography

[AGM+71] Algirdas Avižienis, George C. Gilley, Francis P. Mathur, David A. Rennels, John A. Rohr, and David K. Rubin. The STAR (self-testing and repairing) computer: An investigation of the theory and practice of fault-tolerant computer design. *IEEE Transactions on Computers*, November 1971.

[BJ90] Ricky W. Butler and Sally C. Johnson. The art of fault-tolerant system reliability modeling. Technical Report TM102623, NASA-Langley Research Center, March 1990.

[BPR84] Salvatore J. Bavuso, Paul L. Peterson, and Dave M. Rose. CARE III model overview and user's guide. Technical Report TM85810, NASA-Langley Research Center, June 1984.

[BS88] Ricky W. Butler and Philip H. Stevenson. The PAWS and STEM reliability analysis programs. Technical Report TM100572, NASA-Langley Research Center, March 1988.

[But85] Ricky W. Butler. An abstract specification language for Markov reliability models. Technical Report TM86423, NASA-Langley Research Center, 1985.

[BW88] Ricky W. Butler and Allan L. White. SURE reliability analysis—Program and mathematics. Technical Report TP2764, NASA-Langley Research Center, March 1988.

[Chu67] Kai L. Chung. *Markov Chains with Stationary Transition Probabilities*. Springer-Verlag, 1967.

[CM65] D. R. Cox and H. D. Miller. *The Theory of Stochastic Processes*. Methuen, 1965.

[CMS82] Xavier Castillo, Stephen R. McConnel, and Daniel P. Siewiorek. Derivation and calibration of a transient error reliability model. *IEEE Transactions on Computers*, July 1982.

[DTGN84] Joanne B. Dugan, Kishor S. Trivedi, Robert M. Geist, and Victor F. Nicola. Extended stochastic Petri nets: Applications and analysis. In E. Gelenbe, editor, *Performance '84*. North-Holland, 1984.

[DTSG86] Joanne B. Dugan, Kishor S. Trivedi, Mark Smotherman, and Robert M. Geist. The hybrid automated reliability predictor. *Journal of Guidance, Control, and Dynamics*, May-June 1986.

[HBH90] Sandra V. Howell, Salvatore J. Bavuso, and Pamela J. Haley. A graphical language for reliability model generation. In *Proceedings of the Reliability and Maintainability Symposium*, 1990.

[Joh86] Sally C. Johnson. ASSIST user's manual. Technical Report TM87735, NASA-Langley Research Center, August 1986.

[Joh88] Sally C. Johnson. Reliability analysis of large, complex systems using ASSIST. In *Proceedings of the Digital Avionics Systems Conference*, October 1988.

[KS82] Vittal Kini and Daniel P. Siewiorek. Automatic generation of symbolic reliability functions for processor-memory-switch structures. *IEEE Transactions on Computers*, August 1982.

[Lap85] J.-C. Laprie. Dependable computing and fault tolerance: Concepts and terminology. In *Proceedings of the Fault-Tolerant Computing Symposium*, 1985.

[Lee85] Larry D. Lee. Reliability bounds for fault-tolerant systems with competing responses to component failures. Technical Report TP2409, NASA-Langley Research Center, 1985.

[LL78] C. Landrault and J.-C. Laprie. SURF—A program for modeling and reliability prediction for fault-tolerant computing systems. In Josef Moneta, editor, *Information Technology*. North-Holland, 1978.

[LS83] Jaynarayan Lala and T. Basil Smith III. Development and evaluation of a fault-tolerant multiprocessor (FTMP) computer. Technical Report CR166071-3, NASA-Langley Research Center, May 1983.

[LS86] Carlos A. Liceaga and Daniel P. Siewiorek. Towards automatic Markov reliability modeling of computer architectures. Technical Report TM89009, NASA-Langley Research Center, August 1986.

[MA82] Srinivas V. Makam and Algirdas Avižienis. ARIES 81: A reliability and life-cycle evaluation tool for fault-tolerant systems. In *Proceedings of the Fault-Tolerant Computing Symposium*, 1982.

[MAG82] Srinivas V. Makam, Algirdas Avižienis, and Gintaras Grusas. ARIES 82 user's guide. Technical Report CSD-82-830, UCLA, August 1982.

[McC81] Stephen R. McConnel. *Analysis and Modeling of Transient Errors in Digital Computers.* PhD thesis, Carnegie-Mellon University, 1981.

[Rom70] V. I. Romanovsky. *Discrete Markov Chains.* Wolters-Noordhoff, 1970.

[SBG79] J. J. Stiffler, L. A. Bryant, and L. Guccione. CARE III final report: Phase one. Technical Report CR159122, NASA-Langley Research Center, 1979.

[SBN82] Daniel P. Siewiorek, C. Gordon Bell, and Allen Newell. *Computer Structures: Principles and Examples.* McGraw Hill, 1982.

[SS82] Daniel P. Siewiorek and Robert S. Swarz. *The Theory and Practice of Reliable System Design.* Digital Press, 1982.

[Szc90] Martha R. Szczur. TAE Plus: Transportable applications environment plus, A user interface development tool for building X window-based applications. In *Proceedings of the MIT X Conference*, Boston, Massachusetts, January 1990.

[TG81] Kishor S. Trivedi and Robert M. Geist. A tutorial on the CARE III approach to reliability modeling. Technical Report CR3488, NASA-Langley Research Center, 1981.

[Vli90] John M. Vlissides. *Generalized Graphical Object Editing.* PhD thesis, Stanford University, June 1990.

[Whi84] Allan L. White. Upper and lower bounds for semi-Markov reliability models of reconfigurable systems. Technical Report CR172340, NASA-Langley Research Center, 1984.

[WP90] Allan L. White and Daniel L. Palumbo. State reduction for semi-Markov reliability models. In *Proceedings of the Reliability and Maintainability Symposium*, 1990.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No. NASA TM-104068	2. Government Accession No.	3. Recipient's Catalog No.

4. Title and Subtitle A General Graphical User Interface for Automatic Reliability Modeling	5. Report Date May 1991
	6. Performing Organization Code
7. Author(s) Carlos A. Liceaga, Daniel P. Siewiorek	8. Performing Organization Report No.
	10. Work Unit No. 505-66-21-02
9. Performing Organization Name and Address Langley Research Center, Hampton, VA 23665-5225	11. Contract or Grant No.
	13. Type of Report and Period Covered Technical Memorandum
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration, Washington, DC 20546-0001	14. Sponsoring Agency Code

15. Supplementary Notes

Carlos A. Liceaga, NASA Langley Research Center, Hampton, VA
Daniel P. Siewiorek, Carnegie-Mellon University, Pittsburgh, PA

16. Abstract

The analysis and evaluation of reliability measures using Markov models is required for Processor-Memory-Switch (PMS) structures that have standby redundancy, or are subject to transient or intermittent faults or repair. The task of generating these models is tedious and prone to human error due to the large number of states and transitions involved in any reasonable system. Therefore, model formulation is a major analysis bottleneck, and model verification is a major validation problem. The general unfamiliarity of computer architects with Markov modeling techniques further increases the necessity of automating the model formulation. Automation requires a general system description language that can accommodate new fault-tolerant techniques and system designs.

This paper presents a general Graphical User Interface (GUI) for automatic reliability modeling of PMS structures using a Markov model. This GUI is based on a hierarchy of windows. One window has graphical editing capabilities for specifying the system's communication structure, hierarchy, reconfiguration capabilities, and requirements. Other windows have text fields, popup menus, and buttons for specifying parameters and selecting actions. An example application of the GUI is given.

17. Key Words (Suggested by Author(s)) Graphical User Interface; Automatic Reliability Modeling; Processor-Memory-Switch (PMS) Structures; Fault Tolerance; Markov Models	18. Distribution Statement Unclassified - Unlimited; Star Category 61

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of pages 56	22. Price A04