NASA CR- 181,645

NASA Contractor Report 181645

NASA-CR-181645
19880016540

SEMI-MARKOV ADJUNCTION TO THE CAME PROGRAM

GENE ROSCH,
MONICA A. HUTCHINS,
FRANK J. LEONG,
and PHILIP S. BABCOCK IV

THE CHARLES STARK DRAPER LABORATORY, INC.
CAMBRIDGE, MA

Contract NAS9-17560
APRIL 1988

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

TABLE OF CONTENTS

ABBREVIATIONS AND ACRONYMS

ASSIST	Abstract Semi-Markov Specification Interface to the SURE Tool
BMAC	Boeing Military Aircraft Company
CAME Program	Computer-Aided Markov Evaluator Program
CARE III	The Computer Aided Reliability Evaluator
CSDL	The Charles Stark Draper Laboratory, Inc.
DIU	Device Interface Unit
FDIR	Fault Detection, Isolation and Reconfiguration
FMEA	Failure Modes and Effects Analysis
FTP	Fault Tolerant Processor
HARP	The Hybrid Automated Reliability Predictor
I/O	Input/Output
IAPSA II	Integrated Airframe/Propulsion Control System Architecture
MARK 1	The Markov Modeling Package
MTTF	Mean-Time-To-Failure
SURE Program	Semi-Markov Unreliability Range Evaluator Program

1. INTRODUCTION

The desire for a reliable product has become a major goal in the design of many systems. Tactical military systems, spacecraft, civilian aircraft, nuclear power plants, and undersea vehicles, to name a few, are all systems which have stringent requirements regarding their reliability. Failure of the system can cause the loss of use of the system until it is repaired, the damage or loss of the system itself, or ultimately the loss of life. Therefore, it follows that the ability to accurately estimate the probability of failure of these types of systems is an important task in the development of these systems and in verifying, in fact, that a particular system satisfies its reliability requirements.

Accurately predicting the reliability of a highly reliable system is often not a simple task. Though the reliability of many of the hardware components comprising these systems continues to improve, the strictness of the reliability requirements imposed on the system frequently leads the design into the realm of fault-tolerant systems. Hardware redundancy and software logic are utilized to achieve a greater reliability than would otherwise be possible without the repetition of hardware. As a result, in predicting the probability of failure of these systems, the repetition of the hardware and the effect of the redundancy management scheme which the system incorporates in utilizing this hardware need to be adequately accounted for.

Life testing to establish the reliability is generally not practical for highly reliable systems. Life testing would consist of selecting a random sample of n systems, testing them under specified environmental conditions, and observing the time to failure of each system. Since, typically, the mean-time-to-failure (MTTF) is measured in years and the cost of individual systems high, life testing generally does not permit the accumulation of a statistically significant sample.

A more practical approach to predicting system reliability is to analytically model the behavior of the system based on the known failure modes of its components and the anticipated effect of the redundancy management scheme. The failure modes of the components which make up the system and the actual rates of occurrence of these modes is frequently known or can be practically established through life testing. From these known failure modes and a thorough understanding of how the system will react to these failure modes, a mathematical model can then be constructed to predict the reliability of the system.

Generally, analytical modeling techniques fall into three categories — simulation techniques, combinatorial techniques, and Markov (and semi-Markov) modeling techniques. Simulation techniques, such as a Monte Carlo approach, tend to require a prohibitively large number of simulation runs to generate statistically significant results for highly reliable systems. With combinatorial techniques, such as a fault tree approach, it is difficult to capture the sequence dependent nature of events inherent in fault tolerant

systems. Markov and semi-Markov models offer a method of incorporating the sequence dependency of events in a fault-tolerant system into the structure of a model which can be realistically solved. Their growing use is indicative of the utility of this technique for modeling fault-tolerant system reliability [1, 2, 3, 4].

A Markov or semi-Markov reliability model consists of a set of states reflecting the operational status of the system and the transition probabilities between these states due to component failures or redundancy management decisions. Written as a set of differential equations with respect to time, the system reliability can be predicted by solving for the state probabilities as a function of time. Usually this is done numerically by propagating the state vector from the initial conditions. The reliability of the system is represented by the probability of the system being in any of the states identified as operational.

The construction of a Markov or semi-Markov model from a well-defined system description is conceptually straightforward, but, in application, it can be a difficult and tedious process. Generally, there are two reasons why this is so. First, the person constructing the reliability model needs to manually perform the system-level failure modes and effects analysis (FMEA). That is, for each possible component failure, or combination of component failures, the modeler must discern the consequences of the failure and the resulting operational status of the system. Second, the number of states required to satisfactorily represent the behavior of the system can easily become unmanageably large.

Currently there exists a number of packaged computer programs available which expedite the process of describing fault-tolerant systems and provide a solution algorithm for the full system model [2]. CARE III [5, 6], HARP [7], MARK 1 [8, 9], and SURE [10] are representative of these packages. CARE III and HARP are computer programs which use behavioral decomposition to separate the modeling of the fault-occurrence processes from the fault-handling processes. MARK 1 is a generic program for solving systems which can be described by Markov models. SURE is a computer program which solves semi-Markov models. Though these programs are extremely useful, in all of these packages it is left up to the analyst to construct the necessary model to predict the reliability of the system.

In the past few years, the C. S. Draper Laboratory, Inc. (CSDL) has attempted to address this problem.. As an Independent Research and Development project, a computer-aided reliability analysis tool called the Computer-Aided Markov Evaluator (CAME) program is currently being developed to automate the model construction process [2, 11, 12]. The objective of this tool is to automatically create an appropriate fault-occurrence model from a top-down system description utilizing a set of rules that reflect Markov modeling techniques.

The CAME program has matured to the level where it is becoming a practical tool. Given a high-level description of a system, the CAME program constructs a strictly Markov model to predict the reliability of the system and provides features to evaluate the

created model. The analyst provides the program with the failure modes of the components which make up the system to be analyzed, how these components relate to one another and the operational properties of the system at large. Then the analyst can select from the available options which influence model construction — that is, the model truncation and aggregation techniques which should be applied during the model construction process. From this information, the CAME program can automatically perform the system-level FMEA and construct an appropriate Markov model. In this way, the CAME program is performing the system-level FMEA — not the analyst.

With the aid of the CAME program, larger systems can be analyzed. The program can methodically apply model truncation and aggregation techniques to ultimately reduce the size of the Markov model constructed. In addition , since the model is now constructed automatically, the practical size of the Markov model is only limited to the physical constraints of the computer the program runs on and the time the analyst is willing to wait for the construction and solution of the model.

One shortcoming of the CAME program is its inability to incorporate detailed modeling of the intricacies of fault detection, isolation and reconfiguration (FDIR) into the constructed reliability model. The user interface of the program allows the analyst to include the details of the redundancy management strategy into the system description and, through coverage parameter, the analyst can direct the program to account for the possibility of FDIR errors. But, these fault-handling processes are assumed to occur instantaneously and the coverage parameters are simply weighting factors in the transition probabilities in the constructed model. So, the model constructed by the CAME program may not accurately model all fault-handling processes. If the details of this aspect of the system being analyzed is a relevant factor in determining its reliability, then ultimately, the results produced from the constructed model will be inaccurate.

The objective of this project is to expand the rule-based CAME program in its ability to incorporate fault-handling processes into a constructed model. Specifically, it is to modify the program so that it can develop an appropriate semi-Markov model when the input system description includes events which have distributions that are locally time dependent — such as those characteristic of the fault-handling processes in many fault-tolerant systems. In addition, the program will be modified so the constructed model can be output in a form which can be directly evaluated by the SURE program. The purpose of this report is to document the changes made to the CAME program to accommodate this new feature and to illustrate its use.

This report is organized so as to provide a clear explanation of the alterations and additions made to the CAME program. Section 2 provides a relatively brief discussion of the basic CAME program. A general overview is given of the features and capabilities of the program with sufficient detail provided for those areas which are impacted by this project. Section 3 documents the alterations and additions made to the program to allow for semi-Markov events to be incorporated into a reliability model. To exemplify the use of

this new feature and evaluate its utility, the CAME program was used to analyze portions of the Integrated Airframe/Propulsion Control System Architecture (IAPSA II) [13]. This is presented in Section 4. Section 5 discusses the conclusions reached regarding the semi-Markov adjunction to the CAME program.

2. THE CAME PROGRAM

2.1 Overview

The objective in creating the Computer-Aided Markov Evaluator (CAME) program is to automate much of the process of modeling and analyzing the reliability of fault-tolerant systems. In this way, an analyst with a basic understanding of Markov models, and their utilization in reliability analysis, can use the CAME program to construct an appropriate Markov model to predict the reliability of the system of interest. The analyst need not be well versed in Markov modeling techniques. He must have a thorough understanding of how the system being analyzed responds to failures of its components, but the expertise needed to construct an accurate and useful model is incorporated into the program.

The CAME program is implemented in the ZETALISP programming language on a Symbolics 3600-series computer. ZETALISP is a dialect of LISP copyrighted by Symbolics, Incorporated. The CAME program is designed to run on any of the Symbolics Systems using Release 6 of the software. Specifically, the program has been developed on the 3640 and 3670 Systems using Release 6.1 of the software. The program makes much use of the flavor feature of ZETALISP and extensively employs the graphical input/output features of the Symbolics machine.

Figure 2-1 is a block diagram of the functional modules of the CAME program. The user interface provides the means through which the user inputs a description of the system to be analyzed, controls the construction of an applicable Markov model, and then, controls the numerical evaluation of the model. The user-interface is designed to be utilized interactively during the analysis process and considerable effort has been invested into making it flexible, consistent and user-friendly. The input from the user interface is stored in the system data base. The actual construction of the Markov model is performed by the model builder using the information in the system data base and the rules for model building in the rule base. The constructed model is stored in the model data base. The model inspector module allows the user to examine and verify, in detail, the model constructed by the model builder. The model evaluator module is the portion of the program containing the implementation of the numerical algorithms which the user can invoke to numerically evaluate the constructed model.

The analyst operates the CAME program through six "windows" — the System Architecture Window, the Reconfigurations Window, the Performance Requirements Window, the Further Specifications Window, the Markov Modeler Window, and the Model Evaluator Window. The first four are used to describe the system being analyzed. This information is stored in the system/application database. The control of the model builder module and the ability to examine and display the model stored in the model data base is done through the Markov Modeler Window. The Model Evaluator Window controls the model evaluator module.

2.2 System Specification

The CAME program models systems which consist of a collection of interconnected components (hardware or software) with known exponential failure rates. The analyst specifies the system to be modeled to the program through four windows. These windows are discussed in detail in the subsections which follow.

To facilitate the discussion of the system specification windows, consider the example system presented in Figure 2-2. This figure illustrates the screen display of a simple, user specified system. The screen is set so all four of the system specification windows can be displayed at once. All of the objects shown in the windows are mouse-sensitive and defined by the user through an interactive menu system. The system being modeled is a trivial example of dual-channel control system. The system consists of three components — two processors (*p1* and *p2*) and a node which interconnects them. The system is operational as long as one of the two processors is operating and is in control. Initially, *p1* is in control. In the event of a failure of *p1*, control can switch to *p2*. But this transfer of control requires the link connecting the two processors (the component *node*) be operating.

2.2.1 System Architecture Window

This window defines the components which comprise the system and how they are interconnected. Each component defined by the user represents an object the program can fail individually to discern the failure modes of the whole system. Associated with each component are a number of properties alterable by the user. These are the name of the component, which "classes" it is a member of, the failure rate of the component, the fraction of failures of this component that are covered, the repair rate of the component, and whether or not the component should be treated as a cold spare. A "class" is a tag which the user can use to specify a group of components which have a common dependency or use to denote the group of components some place within the system specification without individually specifying each component. The interconnections between architecture components can be specified to indicate the possible directions information can flow between the components. This is primarily used in determining whether a "path" exists from one component to another within the system architecture.

In the System Architecture Window of Figure 2-2, the three components of the dual-channel control system (*p1*, *p2* and *node*) are shown. An interconnection is also defined between these components, but in this case it is not pertinent to the definition of the system.

2.2.2 Reconfigurations Window

System reconfigurations in response to specified changes in component functionality can be defined within the Reconfigurations Window. Through state transition

diagrams, the user can define entities which will evaluate to some specified component (or label) dependent on the state they were in before a particular component failure is assumed (during the process of model construction) and the current condition of its components. These entities can be used to reflect the reconfiguration strategies within the system.

Consider the dual-channel controller specified to the CAME program in Figure 2-2. A relevant quantity in defining the system is the processor which is in control. The reconfiguration strategy for which processor is in control is specified to the program with the state transition diagram labeled *p-in-control* in the Reconfigurations Window. The two circles represent the two possible states for *p-in-control* to evaluate to — *p1* or *p2*. Initially, *p1* is in control (the label is located over the initial state). According to the definition of this system, in the event that *p1* is no longer operating (*failed*), control can transfer over to *p2* if *node* is operating (*unfailed*). The arrow specifies the possible transition from state *p1* to state *p2* and the text above it indicates the condition which triggers the transition. The text is written using the syntax of the LISP language. So, in order for the transition to be triggered (during construction of the Markov model) the triggering condition must evaluate to "T" (the logical true of LISP).

The triggering condition

(and (failed p1) (unfailed node))

will evaluate to T only if the function

(failed p1)

and the function

(unfailed node)

both evaluate to T. The functions *failed* and *unfailed* are functions defined within the world of the CAME program. *Failed* evaluates to T when the component within the list is failed and NIL otherwise. Alternately, *unfailed* evaluates to NIL when the component within the list is failed and T otherwise. Therefore, the *p-in-control* transfers from *p1* to *p2* whenever *p1* has failed and *node* is not failed.

2.2.3 Performance Requirements Window

The Performance Requirements Window is where the user specifies the definition of system operation and, if desired, degraded modes of operation. The performance levels of the system being described are specified with logic diagrams using functions defined within the world of the CAME program — such as *and*, *or*, *unfailed*, or *path* — and user defined functions. When multiple performance levels are specified, the program presumes the system will be operating in the highest performance level possible. The achievement of none of the specified performance levels is a system failure or system loss.

For the example of the dual-channel controller, only one performance level is specified. The system is operating as long as one of the two processors is operating and is in control. This is specified by the user-defined object defined as

(unfailed (current p-in-control)).

Again, the syntax of this function is that of LISP and the functions *unfailed* and *current* are defined in Section 2.2.5. If any particular Markov state (as the Markov model is being constructed) cannot satisfy this performance level, then the performance level of the system is assumed to be NIL (system loss).

2.2.4 Further Specifications Window

The Further Specifications Window permits the definitions of user-defined functions used in other parts of the system description and the specification of additional performance requirements for groupings of components (i. e. classes). This is the window where the user defines all the symbols which could be used to specify a failure rate, a coverage value, or a repair rate within the properties of the architecture components.

Symbols exist within the CAME program to clearly define to the user which parameters within a constructed Markov model can be changed in value without effecting the integrity of the model. With the exception of a coverage value of one, two parameters (where a parameter may be a failure rate, coverage value, or repair rate specified for each architecture component) are assumed to be the same only if they are the same symbol. This is an important definition when the program performs aggregation in the construction of a model.

In Figure 2-2, the Further Specifications Window shows the failure symbol *fp* and the coverage symbol *cp* defined by the user. Though not shown in the figure, the components *p1* and *p2* are both assigned a failure rate of *fp* and a coverage value of *cp* through the properties menu of the System Architecture Window. Therefore, these symbols need to be defined in the Further Specifications Window and respectively assigned values.

2.2.5 Functions Defined Within the World of the CAME Program

Figure 2-2 illustrates how functions are used to define the dual-channel controller example to the CAME program. Essentially, any of the primitives of LISP can be used by the analyst in the Reconfigurations, Performance Requirements, and Further Specification Windows. Very few prove to be of great use directly. The *and* and *or* logical functions are commonly utilized in specifying systems. In fact, to facilitate the clarity of the system description, the *and* and *or* functions can be displayed as graphical objects within the Performance Requirements and Further Specifications Windows. In addition, a number of functions are defined which are particularly useful in defining fault-tolerant systems and

have rigid definitions within the CAME program. Some of the most useful are listed as follows:

- *(failed comp)* This returns T if the architecture component *comp* is failed and NIL otherwise.

- *(unfailed comp)* This returns T if the architecture component *comp* is unfailed and NIL otherwise.

- *(functional comp-or-class)* If *comp-or-class* is an architecture component, then this function returns T if *comp-or-class* is functional and NIL otherwise. If *comp-or-class* is a class, then this function returns T if at least one component which is a member of the class *comp-or-class* is functional and NIL otherwise. (The functionality of an architecture component can be defined by the user the same way performance levels are defined for a system. Therefore, whether or not a component is functional can depend on other factors besides whether or not the component has failed. If the functionality of a component is not explicitly defined within the system specification, the functions unfailed and functional are synonymous.)

- *(at-least-n-functional n class-or-list-of-comps)* This returns T if at least *n* components with the class or the list of components *class-or-list-of-comps* are functional and NIL otherwise.

- *(current std)* This returns the label of the current state of the state transition diagram *std* in the Reconfigurations Window.

- *(path from-comp-or-list-of-comps to-comp-or-list-of-comps)* This returns T if a path exists from the component (or list of components) *from-comp-or-list-of-comps* to the component (or list of components) *to-comp-or-list-of-comps* and NIL otherwise.

2.3 Model Construction

After a system is properly defined within the four system specification windows, the user can direct the CAME program to construct an appropriate Markov model through the Markov Modeler Window. Using the menus of this window, the user would first select the model truncation and aggregation rules which should be applied in the construction of the model, then initiate model construction, and finally, he could inspect the model which was built.

The CAME program provides several options, available to the user, to reduce the size of the Markov model which is ultimately constructed. These are:

1. Model truncation

2. Performance-level aggregation at the next-to-final failure level

3. Aggregation of the next-to-final failure level into one performing state

4. System loss state aggregation

5. Outward aggregation

6. Backward-sweep aggregation

Though the evaluation of the model has been automated and included as part of the program, for many real systems the constructed model can grow very quickly to a size which overburdens the computational power and/or memory capacity of the machine the program is run on. By carefully combining states in accordance with the options selected, the CAME program can, if possible, reduce the ultimate size of the model without sacrificing its integrity. This reduces the time and memory required for constructing and solving the model.

The CAME program builds a Markov model failure level by failure level. Model construction starts with an initial state in which the system is assumed to have no failures. For each state within the present failure level, all functional components are failed one at a time and the appropriate state (or states) is created in the next failure level. Therefore, all of the possible transitions from each state within a particular failure level will be defined before expanding the next higher failure level.

As each state of the Markov model is created, the performance and reconfiguration definitions are reviewed and the appropriate performance level and reconfiguration status is assigned. The failed, functional, and in-use lists are also determined and assigned to the state. The failed list contains the actual components which were failed to reach this state. In contrast, the functional list contains all the components which are functional in this state. Note that whether or not a component is functional can depend on other factors besides whether or not the component has failed. The in-use list contains those components which satisfy the highest performance definition possible. Therefore, for this particular state they can be thought of as "in use" by the system.

Excepting backward sweep aggregation, the selected truncation and aggregation rules are applied as the model is being constructed. The truncation option can be used to truncate the constructed model at a specified failure level or at a specified number of failure levels past the appearance of the first system loss state. Performance-level aggregation at the next-to-final failure level aggregates the states at the next to final failure level with the same performance level into a single state for each performance level. Aggregation of the next-to-final failure level into one performing state has a similar effect except all states which are not system loss states would be aggregated into a single state. System loss

aggregation causes the system loss states at each failure level to be aggregated into a single system loss state for that failure level. Outward aggregation compares all of the states at each failure level as each failure level is constructed and if any two states both have the same in-use list and the same functional list, then these two states can be aggregated into a single state.

After the complete model is constructed, backward sweep aggregation will take place if this option was selected by the user[1]. Starting with the failure level two failure levels before the last failure level and working toward the initial failure level, backward sweep aggregation is performed on each failure level. Two exit transitions, emanating from different states, are said to match if both have the same transition rate and both terminate at the same state. If two states within the same failure level have the same exit transitions (meaning that there is a one-to-one match between each exit transition of the first state to that of the second state) and they both have the same performance level, then they can be aggregated during backward-sweep aggregation.

The Markov Modeler Window also provides features which allow the user to inspect a Markov model. Once a model has been constructed from the system description in the system specification windows, the model can be examined in a textual format or interactively through the graphical interface. The textual description can be displayed directly in the Markov Modeler window or stored in a file and subsequently printed through an output device. The textual description describes the states of the model and the transitions between them to a degree in which it is possible for the user to validate the model. This same information can be examined interactively through the Markov Modeler Window. The user can direct the program to graphically display the structure of the constructed model and then interactively probe the individual states and transitions of the model.

2.4 Model Evaluation

Through the Model Evaluator Window, the user can invoke one of the available algorithms to numerically solve the state transition matrix represented by the constructed Markov model. The solution can then be displayed as a plot of the probability of system loss as a function of time or simply as a value (or the calculated bounds) at the final time specified. The state vector at the final time or at the calculated intermediate times can also be displayed.

Presently, three solution algorithms are available in the CAME program. The first is an implicit Euler integration algorithm that evolves the state vector iteratively by repeated

---

1 Actually, in order for backward sweep aggregation to take place, the user must also select the model truncation option and either the performance-level aggregation at the next-to-final failure level option or the aggregation of the next-to-final failure level into one performing state option. Therefore, the final two failure levels will have been aggregated to the maximum extent possible as the model was constructed.

multiplications of the vector times the transition matrix. If M is the discrete-time transition matrix, P is the state probability vector, and Δt is the integration step time then:

$$\begin{aligned} P(\Delta t) &= M\, P(0) \\ P(2\Delta t) &= M\, P(\Delta t) \\ &\bullet \\ &\bullet \\ &\bullet \\ P(n\Delta t) &= M\, P((n-1)\Delta t) \end{aligned} \qquad (2\text{-}1)$$

and the solution is $P(t) = P(n\Delta t)$.

The second algorithm is a variation of the Euler integration which takes advantage of the time invariance of the transition matrix. Note that since M is time invariant, Equation (2-1) can be rewritten as:

$$P(n\Delta t) = M^n\, P(0) \qquad (2\text{-}2)$$

Therefore M^n can be solved by repeatedly squaring the transition matrix. If Δt is selected such that $n = 2^i$, where i is an integer, then Equation (2-2) is sufficient to get P(t). If, however, $n \neq 2^i$ then products of intermediate squarings of the matrix must be used to get P(t). Options exist in the CAME program to supply a Δt that is either used directly or is used as a guide for selecting a time step such that $n = 2^i$.

The third algorithm is a hybrid of the first two algorithms. While matrix squaring can produce faster solutions and data points logarithmically spaced in time, it is not capable of generating linearly spaced outputs. For these cases a hybrid of matrix squaring and Euler integration is invoked by the CAME program. Matrix squaring is used to generate a "base transition matrix" and Euler integration operates on this matrix to reach the solution P(t).

A more thorough discussion of the latter numerical solution techniques of Markov models can be found in [14].

3. SEMI-MARKOV ADJUNCTION

3.1 Approach

The objective of this project is to modify the CAME program so that it can develop an appropriate semi-Markov model when the input system description includes events which have distributions that are locally time dependent — such as those characteristic of the fault-handling processes in many fault-tolerant systems. In deciding the approach to accomplishing this, a number of assumptions are made at the inception. The first is that the altered CAME program should be able to solve a useful class of problems. The second is that the interest is in demonstrating the feasibility of generating semi-Markov models with the CAME program to model fault-handling processes. Therefore, though a few different approaches are viable, the approach taken is not necessarily the best. In fact, the intention is to add this new ability to the program in the simplest fashion possible. The third assumption is that the models constructed by the CAME program will be solved using the SURE program [10]. Therefore, the approach taken should be compatible with this solution technique.

Currently, the method by which the CAME program incorporates the effect of FDIR errors into the Markov model it builds is through the coverage property associated with each architecture component. When an architecture component is failed which has a coverage parameter other than one, two states are created by the CAME program in the next failure level. The first is the completely covered state. The transition rate to this state from the current state is the coverage value times the failure rate of the component failed. The second state created is the state which results if this component failure is uncovered. The transition rate for this event is the sum of one minus the coverage value times the failure rate of the component. In this way the fault-handling behavior of the system being modeled is reflected in the constructed Markov model as the weighting factor of an event which has an exponential distribution. Therefore, though the outcome of the processes which perform the FDIR are incorporated into the constructed model, the details of the processes themselves are not.

In a reconfigurable fault-tolerant system, the response of the system to fault arrivals occurs at a finite rate and does not necessarily have an exponential distribution. Though this reconfiguration rate is generally much faster than the rate of failure of any of the components, the reconfiguration time can have an impact the system's reliability. The longer this time period, the greater the probability that a second event will occur (such as the failure of another component) while the system is still performing FDIR. This nearly coincidental event could have a critical effect on the performance of the system by confusing the FDIR process — resulting in an uncovered failure. It is this phenomenon that the CAME program is modified to include in the constructed model.

The approach taken to adding this new capability to the program is to associate a new property with each architecture component which, when enabled, would directly cause the creation of an additional state (or states) to account for the effect of the fault-handling processes which are enacted when this component fails. The possible states are:

1. The state which models correct FDIR in response to the component failure.

2. The state which models incorrect FDIR in response to the component failure.

3. The state which models the nearly coincidental failure.

The transitions to the first and second states are semi-Markov since they model the FDIR processes of the system. The transition to the third state is Markov since the event which causes it is the failure of a component. The following three subsections discuss the alterations made to the CAME program to implement this approach.

3.2 Changes to the System Specification

To allow the user to specify a system in which the fault-handling processes are modeled with semi-Markov transitions, two changes are made within the system specification windows. The first is an alteration to the architecture component object of the System Architecture Window; the second is the addition of a semi-Markov transition class object to the Further Specifications Window.

A new property is now associated with each architecture component of the System Architecture Window. This new property will be the semi-Markov transitions which may be triggered when the architecture component is failed. Figure 3-1 illustrates the changes made to the menus used to define the properties of an architecture component. The upper-left menu is the object menu for an architecture component object and the two menus below it are the properties menu and the semi-Markov properties menu. Both are accessed through the architecture component object menu. The user specifies this new property through the question "Are there semi-Markov transitions?" in either the properties menu or the semi-Markov properties menu for an architecture component. "Yes" enables this property for the particular component selected; "no" leaves the component to be treated in the normal manner.

The semi-Markov properties menu permits the user to specify the possible transitions which can result when the selected component fails (see Figure 3-1). This is a new menu added to the CAME program. Since the SURE program will be used to solve the semi-Markov models constructed by the CAME program, the possible fast transitions can be specified with White's method [10]. Therefore, the conditional mean transition time, the conditional standard deviation of the transition time, and the transition probability are the necessary and sufficient parameters which specify the fast transition. These are the parameters the user would input for the "Fast Transition Specification" of the semi-Markov

properties menu. The default values of a mean of 1.0×10^{-6} h, a standard deviation of 1.0×10^{-7} h, and a transition probability of 1.0 are shown in Figure 3-1.

The question "Object must be in use for fast transition to occur?" of the semi-Markov properties menu allows the user to impose the additional condition that the component be "in use" in order for the semi-Markov transitions to be generated. If "yes" is chosen, then the component must be "in use" (that is, on the in-use list) when it is failed in order for the subsequent semi-Markov transition(s) to take place. If "no" is chosen, then this is not a necessary condition.

The final entry to the semi-Markov properties is the specification of the "Critical Component(s)". The critical component(s) would be the components whose near coincident failure would cause a system loss if it occurred before the fast transition event (FDIR) is completed.

The second change to the system specification is the addition of a semi-Markov transition class object. This object is created to conform with the methodology the CAME program has for handling numbers and symbols. A semi-Markov class object is the symbol which could be used in place of the mean, standard deviation and transition probability in the Fast Transition Specification of the semi-Markov properties menu. Its use is analogous to the use of the failure symbol and coverage symbol discussed in Section 2.2.4.

3.3 Changes to the Model Building Process

Figure 3-2 illustrates how the model building process is altered when an architecture component is encountered which has a fast, semi-Markov transition associated with it. State A represents the current state from which transitions will be generated. State A is at the n failure level. As before, all functional components in this state will be failed to create states in the next failure level. But now, if a component whose failure could possibly trigger a semi-Markov transition is encountered, the generation of the state in the next failure level will proceed differently. First, it must be decided whether or not the semi-Markov transition will take place. That is, whether or not the component must be in use to have a semi-Markov transition and, if it must be in use, whether or not it is. If the semi-Markov transition is not be triggered, then the model building will proceed as is currently implemented in the CAME program — a standard Markov transition is constructed.

If the semi-Markov transition is triggered, then the states generated will follow as in Figure 3-2. State B, and the appropriate transition from state A, is created as a result of the failure of the component. The transition, represented by the rate $\lambda_{component}$, is handled as transitions are currently handled in CAME, but state B would have to be treated specially. System performance and reconfigurations are not analyzed in state B. This is to avoid interpreting state B as a system loss state since a subsequent FDIR event may bring the system back to an operational state. Instead, state C and the semi-Markov transition

from state B to state C is generated. The semi-Markov transition from state B to state C is completely specified by the conditional mean transition time (μ), the conditional standard deviation of the transition time (σ), and the transition probability (ρ). Performance and reconfiguration analysis is done and assigned to state C. If state C evaluates to a system loss, then the transition to this state from state B should be assigned a transition probability of one and state D is not generated.

If ρ is less than one and state C does not evaluate to a system loss, then state D and the semi-Markov transition from state B to state D are created. State D represents a system loss state at the (n+1) failure level due to an incorrect reconfiguration.

After states B, C and D are created, the building of the (n+1) failure level continues until completed. Then the (n+1) failure level is expanded to create the (n+2) failure level. When state B is encountered as the (n+1) failure level is expanded, only one transition is created and it will have a destination of state E. State E is the system loss state due to coincident failures. The rate of the transition from state B to state E is the sum of the failure rates of all the components which are still failable and identified as "critical components" by the user for the component whose failure created state B.

State C can be treated as any other state in the (n+1) failure level. Building the transitions which emanate from it to the states at the next failure level can proceed as is presently implemented in the program.

With some modification, the truncation and aggregation rules available in the CAME program can also be utilized with the construction of semi-Markov models. What follows is a summary of how they are altered to function with regard to the states and transitions generated from the semi-Markov property of an architecture component:

1. Model Truncation

Generally, model truncation remains unchanged. In the course of constructing a model, when the truncation rule is triggered[2] only one more failure level will be created and it will consist of a single state. All operational states at the next to last failure level (including any B states) are assigned a single exit transition which goes to the single aggregate state at the final failure level. The rate of this transition is the sum of the failure rates of all the components defined in the architecture window.

2. Performance-Level Aggregation at Next-To-Final Failure Level

This state aggregation technique functions as it had — with the exception that B states are treated specially. When this aggregation technique is triggered, states created at the next-to-final failure level will be aggregated together according

---

2 The user specifies the number of failure levels to be constructed or the number of failure levels to constructed beyond the appearance of the first system loss state.

to performance level. The B states will remain unaggregated since they have no performance level. After the next-to-final failure level is completed, any B states with the same exit transitions will then be aggregated together.

3. Aggregation of the Next-To-Final Failure Level into One Performing State

This state aggregation technique is altered in the same manner with regard to the B states as performance level aggregation at the next-to-final failure level.

4. System Loss State Aggregation

When triggered, all the system loss states — including those generated as the result of the failure of an architecture component with an enabled semi-Markov property — will be aggregated within each failure level.

5. Outward Aggregation

Two states, at the same failure level, can be outwardly aggregated if both the "in use" list and the "functional" list of the first state respectively matches those of the second. When semi-Markov transitions have been included in the constructed model, this process is altered to include the C states. The C states are handled as any of the normal Markov states. This includes the aggregation of C states with the normal Markov states. The B states are excluded from any outward aggregation.

6. Backward-Sweep Aggregation

Two exit transitions, emanating from different states, match if both have the same transition rate and both terminate at the same state. If two states have the same exit transitions (meaning that there is a one-to-one match between each exit transition of the first state to that of the second state) and they both have the same performance level, then they can be aggregated during backward-sweep aggregation.

When semi-Markov transitions have been included in the constructed model, backward sweep aggregation may still take place — but with one modification. Now, when two transitions are compared, they have the same transition rates if one of the following two statements are true:

1. They are both Markov and have the same rate.

2. They are both semi-Markov and have the same conditional mean transition time, conditional standard deviation, and transition probability.

Otherwise, they are not the same and cannot be aggregated.

3.4 Evaluation of the Constructed Model

Once a model is constructed by the CAME program, the CAME program can output it in a form which can be directly evaluated with the SURE reliability analysis program [10]. The solution algorithms within the Model Evaluator Module of the CAME program are not applicable to solving semi-Markov models. Therefore, to numerically solve the models generated with this new feature to the program, commands have been added which allow the user to create a listing of the model in the input language of the SURE program. This listing can be read directly by the SURE program.

Figure 3-3 illustrates the command menus of the Model Evaluator Window and the two commands which are added to create the input listing. The menu in the upper left of the figure is the command menu accessed directly from the window. The second menu is accessed by selecting the "Other operations" command of the first menu. The commands "Describe front end for SURE" and "Save front end for SURE" are the new commands which create the input listing. "Describe front end for SURE" displays the input listing on the screen of the Symbolics machine. "Save front end for SURE" stores the input listing in a file specified by the user.

Two points need to be noted regarding the CAME/SURE interface. The first is that the labeling conventions available to the user in defining a system within the system specification windows of the CAME program are much more liberal than those of the SURE input language. In the process of creating an input listing, the CAME program may alter many of the labels given to components to make them compatible with the character set and length limitations of the SURE program.

The second point regards the interpretation of the numerical results calculated with the SURE program for a truncated model. When the user indicates to the CAME program that he desires the constructed model to be truncated, the effect is that the model will no longer produce a single result. The solution of the model for the probability of system loss yields a lower and an upper bound reflecting the approximation introduced by truncation. If the SURE program is utilized to solve a constructed semi-Markov model, it will compute the lower and upper bounds of the probability of being in each of the death states (the system loss states). If the semi-Markov model made use of the truncation technique, then the ultimate result after using the SURE program is the bounds on the lower and upper bounds for the probability of system loss. Therefore, the sum of the lower bounds calculated by the SURE program for each of the lower bound system loss states produced from the CAME program represents the lower bound of system loss. The sum of the upper bounds calculated by the SURE program of all of the system loss states (including the upper bound aggregate system loss state) represents the upper bound of system loss.

4. AN EXAMPLE: IAPSA II

On June 8, 1987, the Boeing Military Aircraft Company (BMAC) gave a presentation at the NASA Langley Research Center which included their reliability analysis of the Integrated Airframe/Propulsion Control System Architecture (IAPSA II) reference configuration being developed under NASA Contract No. NAS1-18099 [13]. The reliability analysis was performed by C. William Lee. The approach taken by the BMAC to analyzing the reliability of the IAPSA II was to divide the system into a number of smaller groups distinguished by their functions. The intent was to select each of these functional groups with the goal to maximize the independence between them. Individual models were created to predict the reliability of each of the functional groups.

The reliability models necessary to predict the probability of system loss for each of the functional groups were generated utilizing the Abstract Semi-Markov Specification Interface to the Sure Tool (ASSIST) program [15, 16, 17]. The ASSIST program is a program which translates an abstract specification of a semi-Markov model (in the input language of ASSIST) into a format which can be input to the SURE program. The input language of the ASSIST program allows the user to specify the model at a higher level than enumerating each of the possible states and each of the possible transitions between these states. The program generates the actual model from the user's more general specification of the processes which are taking place. In the BMAC reliability analysis of the IAPSA II, the created models were solved using the SURE program.

As a means of evaluating the alterations made to the CAME program to support the construction of semi-Markov models and of evaluating the CAME program itself, the CAME program is used to model the reliability of three of the defined functional groups of the IAPSA II. In this way, the results produced from the models constructed with the CAME program can be directly compared with the results from the reliability model created using the ASSIST program.

In the presentation given by BMAC, several reliability models were used to predict the safety of the flight control section. The ASSIST program was used to create models for the functional groups: Flight Control Computation, Pilot Sensing, Body Motion Sensing, Airflow Sensing, and Pitch Surface Control. Of these five, three functional groups are modeled using the CAME program. They are: Flight Control Computation, Pilot Sensing, and Body Motion Sensing. These three groups best illustrate the strengths and weaknesses in the model approach used by the CAME program.

The intention of this exercise is to evaluate the CAME program, not to provide an analysis of the IAPSA II. The system being modeled with the CAME program is the system defined in BMAC's presentation and ASSIST input listings specifying the reliability models. (The ASSIST input listings for the Flight Control Computation, Pilot Sensing, and Body Motion Sensing Groups are included in Appendix A.) No attempt is made to

independently analyze the IAPSA II directly. Therefore, no claim is made that the results produced from the CAME program generated models represent predictions of the reliability of the IAPSA II. The Flight Control Computation, Pilot Sensing, and Body Motion Sensing models created by the CAME program are compared to the respective models generated with the ASSIST program. The objective is to independently reproduce the results generated through the ASSIST models with comparable models generated with the CAME program.

4.1 Flight Control Computation

The ASSIST listing of BMAC's Flight Control Computation model does not specify any semi-Markov transitions. It is included as one of the groups modeled by the CAME program because it provides a benchmark for comparing the ASSIST and CAME programs which does not invoke the semi-Markov adjunction to the CAME program.

Figure 4-1 illustrates the Flight Control Computation Group. According to BMAC's reliability analysis, the Flight Control Computation Group is a quad fault tolerant processor (FTP) consisting of 10 components — the 4 individual channels of the FTP and the 6 network interfaces. The group is defined to be operational as long as at least 2 of the 4 channels are operational and at least one of these two channels has access to one of the two I/O networks (i.e., one of these two channels has an operating network interface). Otherwise, the Flight Control Computation Group is at system loss. In addition, the following system level FMEA applies:

1. A failure of a FTP channel also brings down the network interfaces within that channel.

2. A failure of a network interface results only in the loss of that network interface; the FTP channel it is attached to may still be operational.

The input system description used to describe the Flight Control Computation Group to the CAME program is shown in Figures 4-2 and 4-3. Figure 4-2 shows the System Architecture Window. Note that *ftp1*, *ftp2*, *ftp3*, and *ftp4* are the four channels of the FTP. The six components labeled *parmn* are the respective network interfaces for each of the channels of the FTP (*m* indicates the network being interfaced with and *n* specifies the channel it is associated with). The specified interconnections between the FTP channels and the network interfaces have no bearing on the description and are included only to be illustrative.

The Performance and Further Specifications Windows are presented in Figure 4-3. With regard to safety, the Flight Control Computation Group has two possible states — operational (p-level 1) or system loss. This is specified to the CAME program by defining the p-level shown in the Performance Requirements Window. The definition for p-level 1 of the system shows that the system is at p-level 1 if at least two of the four channels of the

FTP are functional *and* at least one of the six network interfaces is functional. If p-level 1 is not satisfied, the state of the system is assumed to be at system loss.

The Further Specifications Window specifies the classes and symbols utilized. The class definitions are used to limit the functional network interfaces to those attached to a FTP channel which is functional. Note that the four defined classes *p1*, *p2*, *p3*, and *p4* each depend only on whether or not one particular channel of the FTP is functional. Though not shown in Figure 4-2, the following components are assigned to classes:

1. Component *par11* is a member of the class *p1*.

2. Component *par12* and *par22* are members of the class *p2*.

3. Components *par13* and *par23* are members of the class *p3*.

4. Component *par24* is a member of the class *p4*.

In this way, when a particular channel fails, the program assumes the attached network interfaces will also become non-functional. Therefore, in the specification of p-level 1, it is not necessary to indicate the connection between the functional network interface and the functional channel. The two failure symbol definitions shown in the Further Specifications Window, *channel* and *interface*, are used to specify common failure rates for the channels of the FTP and for the network interfaces, respectively.

From the input system description for the Flight Control Computation Group, the Markov model described by the SURE input listing in Figure 4-4 is generated by the CAME program. The truncation rule is set so that the model is truncated one failure level after the appearance of the first system loss state. The aggregation rules are set to allow the maximum possible aggregation of states to take place. The result is the model described by the SURE input listing of Figure 4-4 which is a 15 state model with 28 transitions that extends to the fourth failure level. The relevant measures of the created model, along with the time it took the CAME program to generate this model, are given in the first entry of Table 4-1 — "Loss of access to both networks".

Table 4-1 presents a comparison of the models generated by the ASSIST and CAME programs for Flight Control Computation, Pilot Sensing, and Body Motion Sensing Groups. The relevant measures for the Flight Control Computation Group are recorded as the first line of this table. The ASSIST generated model for this group is obtained directly from the ASSIST program using the corresponding input listing obtained from BMAC's analysis of the IAPSA II reference configuration[3]. (See Appendix A.) Note that the "Run Time" listed in Table 4-1 is the execution time of the respective program.

---

3 The ASSIST and SURE programs are run on a Digital Equipment Corporation VAX 11/8600 using Version 4.4 of the VMS operating system.

Therefore, the "Run Time" represents the model construction time by the respective program, either the CAME program or the ASSIST program.

For the mission time and failures rates used in BMAC's safety analysis of the IAPSA II, the bounds on the probability of system loss are calculated using the CAME generated model of the Flight Control Computation Group. BMAC utilized a mission time of 3 hours and failure rates of 220.0×10^{-6} and 40.0×10^{-6} failures/hour for the channel and network interfaces of the FTP, respectively. These failure rates are listed in Table 4-2. The results produced with these parameters are recorded as the first entry in Table 4-4 along with those predicted from the ASSIST generated model. Note that bounds calculated for the probability of loss for both models are all 1.15×10^{-9}.

To provide an additional example for the evaluation of the CAME program, a variation of the Flight Control Computation Group is contrived. This variation is identical with the original definition of the Flight Control Computation Group except, in this case, the FTP must be able to access both I/O networks. To use the ASSIST program to generate an appropriate model, all that is necessary is to change the DEATHIF statement in the original input listing to:

```
DEATHIF     (NGFTP1 + NGFTP2 + NGFTP3 + NGFTP4 < 2)
        OR  (NPAR11 + NPAR12 + NPAR13 < 1)
        OR  (NPAR22 + NPAR23 + NPAR24 < 1);
```

To use the CAME program, all that is needed is to alter the input system description shown in Figures 4-2 and 4-3 so the Performance Requirements Window is as presented in Figure 4-5.

For this variation, the relevant measures of the models generated from the CAME and ASSIST programs are reported in Table 4-1 as the "Loss of access to either network" and the respective predictions for the probability of loss at 3 hours is recorded in Table 4-4. Again, the numerical results are in close agreement even though the models are quite different. With the exception of the predicted upper bound calculated from the CAME generated model, the bounds for the models generated from both the ASSIST and CAME programs are all 1.52×10^{-9}. The upper bound calculated from the CAME generated model is slightly higher than this — 1.53×10^{-9}.

4.2 Pilot Sensing

The Pilot Sensing Group is that part of the input/output network which provides the sensor information from the cockpit for flight control to the flight control FTP. It includes the redundant sensors of the pitch stick, roll stick and rudder pedal and the two I/O networks. Each sensor type is quadruplicated with each one assigned to one of the four device interface units (DIUs) of the cockpit. Two of the DIUs interface with one I/O network and the other two with the second I/O network. Figure 4-6 shows the complexity

of Flight Control I/O Network 1 and how the DIUs for the cockpit sensors assigned to this network are attached. Figure 4-7 illustrates Flight Control I/O Network 2 and how the other two DIUs are assigned to this network.

The Pilot Sensing Group is considered operational as long as valid sensor information is available to the flight control computer for each of the three sensor types — pitch stick, roll stick and rudder pedal. Valid sensor information is available for each sensor type if the good values outnumber the bad values to mask faults. Bad sensors can be detected and removed from the voting process, but this takes a finite amount of time. If a failure of one sensor occurs and a second sensor failure of this same type occurs before the first is removed, then the good sensors no longer outnumber the bad since there are only four sensors of each type. So this second nearly coincidental failure causes a system loss. If the system can be reconfigured to remove the first failure from the voting process before the second occurs, then the system continues to function since the good sensors outnumber the bad by two to one.

Failures within the flight control I/O networks effect the ability to access sensor information by the flight control FTP. A failure of a node or link within one of the networks would cause the permanent loss of communications through that component. In addition, this failure may cause the temporary loss of the entire network until the network can be regrown. In BMAC's analysis, a worst case assumption is made to model the I/O networks. All failures within a network are assumed to cause a regrowth of the network. All sensors on the network would be unavailable to the flight control FTP until regrowth is completed. Therefore, there are two ways in which failures within the flight control I/O networks can contribute to a loss of the Pilot Sensing Group. The first is the occurrence of a nearly coincident failure of a sensor on one network while the other network is being regrown because of a failure. For this sensor type, the good sensors no longer outnumber the bad, so a system loss occurs. The second way results when the failure of one of the cockpit sensors on one network occurs and then a failure occurs which causes the other network to be regrown. Again, for at least one of the three sensor types, the good sensors no longer outnumber the bad and a system loss results. Note that this second failure, which can occur anytime after the cockpit sensor failure, causes a temporary exhaustion of the necessary number of sensors of one of the three types of cockpit sensors.

Four categories of fault sequences are identified in BMAC's analysis:

1. Exhaustion of the necessary number of sensors of one of the three types.

2. Nearly coincident sensor failures for sensors of the same type.

3. Nearly coincident sensor failure and a network recovery.

4. Temporary exhaustion of the necessary number of sensors of a particular type.

The first three can all be accounted for with the expanded version of the CAME program. The fourth category cannot be accounted for. As is specified in Section 3.3, the state of the semi-Markov model which directly results from the failure of a component in which the fast transitions will take place (state B in Figure 3-2) does not have its performance evaluated. Therefore, the delay before reconfiguration takes place is, by default, an operational state. This precludes the modeling of the temporary exhaustion failure mode. (It would not be a difficult program change to alter the program to enable it to model this failure mode.)

Figures 4-8 through 4-11 present the four system specification windows of the CAME program with the input system description for the Pilot Sensing Group. The Pilot Sensing Group is the first example which invokes the CAME program's ability to construct an appropriate semi-Markov model.

Figure 4-8 presents the System Architecture Window and Figure 4-9 illustrates the Reconfigurations Window. As in BMAC's analysis of the Pilot Sensing Group, because of the size and complexity of the two flight control I/O networks, it is necessary to use an approximation to model these networks. The architecture components *par11*, *par12*, *par13*, *par14* and *par15* and the reconfiguration diagram *partition-1-in-use* are used to approximate the behavior of I/O Network 1 exclusive of the two nodes the sensors are attached to. The component *par11* is initially in use and the other four components *par12*, *par13*, *par14* and *par15* are specified as cold spares (failable only when in use). All of these components can cause the generation of semi-Markov transitions. These five components will sequentially fail and the next one will be brought into use to approximate the failures within the I/O network which cause regrowth of the network but do not cause isolation of the cockpit nodes. The components *par21*, *par22*, *par23*, *par24* and *par25* and the reconfiguration diagram *partition-2-in-use* are used in the same way to approximate the behavior of I/O Network 2. The components *node1*, *node2*, *node3* and *node4* represent the cockpit network nodes, along with their respective DIUs and primary electrical systems. The *pit*, *roll* and *rud* components respectively represent the pitch stick, roll stick and rudder pedal sensors.

Figure 4-10 shows the Performance Requirements Window for the Pilot Sensing Group. The Pilot Sensing Group is considered operational as long as at least two sensors of each of the three sensor classes are functional. Functional in this context means they are providing good data to the flight control FTP. The *pit* components are the members of the class *pitch-sensor*, the *roll* components are the members of the class *roll-sensor*, and the *rud* components are the members of the class *yaw-sensor*. Each of these sensors is functional only if it has access to the network is is attached to.

The Further Specifications Window is shown in Figure 4-11. The classes *n1*, *n2*, *n3*, and *n4* are utilized to specify how each sensor is attached to the networks. The components *pit1*, *roll1*, and *rud1* are defined as members of class *n1*. So, when class *n1* fails (which it does when either the component *node1* fails or *partition-1* fails), the components *pit1*, *roll1*, and *rud1* will become non-functional. Similarly, the components

pit2, *roll2*, and *rud2* are defined as members of class *n2*; *pit3*, *roll3*, and *rud3* as members of class *n3*; and *pit4*, *roll4*, and *rud4* as members of class *n4*.

Using the input system description for the Pilot Sensing Group, the CAME program generated an appropriate semi-Markov model to predict its reliability. In order to create a model which would calculate sufficiently tight bounds for the prediction of the probability of system loss, the model is built out through the fourth failure level. Figure 4-12 presents the structure of the model as displayed through the Markov Modeler Window of the program. The relevant measures of this model are recorded in Table 4-1 (as the line labeled "Original" under "Pilot Sensing") along with the other groups being analyzed. Note that Table 4-1 includes two entries for the ASSIST generated model. The first entry is the model generated from the original input listing from BMAC's analysis. The second entry is the model which is generated if the temporary exhaustion failure sequences are removed from the ASSIST input listing. This is done by removing the lines of code in Appendix A.2 which are marked with a bar in their left margin.

Using the model generated by the CAME program and the same input parameters used in the BMAC analysis (see Tables 4-2 and 4-3), the bounds for the probability of loss of the Pilot Sensing Group are calculated for a mission time of 3 hours with the SURE program. These are recorded in Table 4-4. Also recorded for the Pilot Sensing entry are the predictions from BMAC's ASSIST generated model and the prediction from BMAC's model with the temporary exhaustion failure sequences removed. Note that the bounds predicted from the CAME generated model do not contradict the results produced from either of the two ASSIST generated models and all three are reasonably close. This is because the temporary exhaustion failure sequences, which are not included in the CAME generated model, only represent about 25 percent contribution to the probability of system loss. So, the predictions of both of the ASSIST generated models fall within the bounds from the CAME generated model.

4.3 Body Motion Sensing

In terms of analyzing its reliability, the Body Motion Sensing Group is very similar to the Pilot Sensing Group. The Body Motion Group is the part of the I/O network which provides sensor information from the body motion sensors to the flight control FTP. It includes the eight gyros, the eight accelerometers, and the two I/O networks. Two gyros and two accelerometers are assigned to each of the four DIUs. Figure 4-6 and 4-7 illustrate how these four DIUs are linked to the two Flight Control I/O networks.

The Body Motion Sensing Group is considered operational as long as valid sensor information is available to the flight control computer for both of the sensor types — the gyros and the accelerometers. Since the gyros and accelerometers are skewed sensors, valid sensor information can be derived for each sensor type as long as the good sensors outnumber the bad sensors by at least 3 or more to mask faults. As in the Pilot Sensing Group, bad sensors can be detected and removed from the voting process, but this takes a

finite amount of time. If one gyro fails and then one of the other three Body Motion Sensing DIUs (i. e., not the DIU this gyro is directly attached to) fails before this gyro is removed from the voting process, then the ratio of good to bad gyros is only 5 to 3. The result is a system loss. If the bad gyro is removed from the voting process before the second failure (the failure of the DIU), then the Body Motion Sensing Group remains operational. The effect of failures within the I/O networks exclusive of the nodes and DIUs directly attached to the body motion sensors is as described for the Pilot Sensing Group. (See Section 4-2 for a discussion of this effect.)

The same four categories of failure sequences exist in the Body Motion Sensing Group as the described for the Pilot Sensing Group:

1. Exhaustion of the necessary number of sensors of one of the two types.

2. Nearly coincident sensor failures for sensors of the same type.

3. Nearly coincident sensor failure and a network recovery.

4. Temporary exhaustion of the necessary number of sensors of a particular type.

BMAC's analysis identified the temporary exhaustion of body motion sensors as the dominant failure sequence with regard to flight safety [13]. Since the CAME program is unable to incorporate this failure sequence into the models it constructs, the Body Motion Sensing Group should highlight this deficiency in the program.

The input system description of the CAME program for the Body Motion Sensing Group is presented in Figures 4-13 through 4-16. Note that the system description for the Body Motion Sensing Group is very similar to that of the Pilot Sensing Group. Figures 4-13 and 4-14 are, respectively, the System Architecture Window and the Reconfigurations Window. The same modeling approximation to model the effect of the two I/O networks as is utilized in the Pilot Sensing Group is employed in the system description for the Body Motion Sensing Group. The architecture components *par11*, *par12*, *par13*, *par14*, *par15*, *par21*, *par22*, *par23*, *par24*, and *par25* and the reconfiguration diagrams *partition-1-in-use* and *partition-2-in-use* mimic the behavior of the I/O Network 1 and I/O Network 2 exclusive of the four nodes the sensors are attached to. These objects are used as the objects with the same names in the system description for the Pilot Sensing Group. The components *node1*, *node2*, *node3* and *node4* represent the network nodes, along with their respective DIUs and primary electrical systems, the gyros and accelerometers are directly attached to. The *gyro* and *acc* architecture components respectively represent the gyros and accelerometers — the body motion sensors.

Figure 4-15 presents the Performance Requirements Window and Figure 4-16 shows the Further Specifications Window. As displayed in the Performance Requirements

Window, the Body Motion Sensing Group is operational (*p-level 1*) as long as at least four gyros and at least four accelerometers are functional. Functional in this context means they are providing good data to the flight control FTP. In the Further Specifications Window, the classes *n1*, *n2*, *n3*, and *n4* are utilized to specify the dependency of the individual body motion sensors on the nodes they are directly interfaced to. The components *gyro1*, *gyro2*, *acc1*, and *acc2* are defined as members of class *n1*. So, when class *n1* fails, the components *gyro1*, *gyro2*, *acc1*, and *acc2* will become non-functional. Similarly, the components *gyro3*, *gyro4*, *acc3*, and *acc4* are defined as members of class *n2*; *gyro5*, *gyro6*, *acc5*, and *acc6* are defined as members of class *n3*; and *gyro7*, *gyro8*, *acc7*, and *acc8* are defined as members of class *n4*.

From the system description, the CAME program generated a semi-Markov model to predict the Body Motion Sensing reliability. In order to create a model with sufficiently tight bounds for the prediction of the probability of system loss, the model is constructed out through the fourth failure level. The relevant measures of this model are recorded in Table 4-1 (as the line labeled "Original" under "Body Motion Sensing") along with the other groups being analyzed. Note that, as with the Pilot Sensing Group entries, two models are created from BMAC's ASSIST input listings for comparison. The first is the model generated directly from the original input listing. The second is the model generated with the temporary exhaustion sequences removed from the input listing[4].

Using the same input parameters as those used in the BMAC analysis (see Table 4-2 and 4-3), the bounds for the probability of system loss of the Body Motion Sensing Group are calculated for a mission time of 3 hours with the SURE program. These are recorded in Table 4-4 along with the predictions for the two models created from the ASSIST input listings. Note the wide divergence between the predictions of the original ASSIST generated model and those of the CAME generated model. This divergence is attributable to the CAME program's inability to incorporate the temporary exhaustion failure sequences into the constructed model. Note that for the ASSIST model with the temporary exhaustion failure sequences removed, there is close agreement with the predictions of the CAME generated model.

4.4 Discussion

The predictions from the CAME program for probability of loss of the Flight Control Computation, Pilot Sensing, and Body Motion Sensing Groups compare well with those produced with the ASSIST program for these groups. As indicated in Table 4-1, the actual models produced for each group from the two programs are quite different. A comparison of the states of the models for each group is difficult because of the size of most of the models and is avoided. But the numerical predictions derived from these models is comparable for each group because both programs are attempting to construct a

---

4 This is done by removing the lines of code in Appendix A.3 which are marked with a bar in their left margin.

semi-Markov model to calculate the probability of system loss for the same physical system.

Care should be taken in comparing the measures recorded in Table 4-1 for the ASSIST and CAME programs. For each of the groups examined, the model construction time ("Run Time") is recorded as being much faster for the ASSIST program. For the Pilot Sensing and the Body Motion Sensing Groups it is orders of magnitude faster. The ASSIST and CAME programs are each run on different machines so a rigid comparison of the model construction times between these two programs is misleading. However, the differences in model construction times are too great to be attributed only to the speed of the host machines. The CAME program attempts to incorporate more of the analysis process into the program itself than the ASSIST program does. Therefore, the results of Table 4-1 do not reflect the total time it takes to construct a model — only the time it takes each of the two programs to construct a model from their respective input specifications.

The measures of the size (number of states and transitions) of the models generated by the ASSIST and CAME programs in Table 4-1 should also be compared with caution. The CAME program provides the user the option of invoking a number of model truncation and state aggregation rules which are applied as the model is being constructed to minimize the ultimate size of the model. These options are utilized in the construction of all of the models generated with the CAME program. In contrast, BMAC's ASSIST input listings show little attempt made at reducing the size of the constructed models. Therefore, though the results of Table 4-1 show the models constructed by the CAME program to be smaller in size than those constructed with the ASSIST program, the reason for including these measures is to highlight the wide divergence between the models created with the two programs.

The predictions for the Flight Control Computation Group from the CAME and ASSIST generated models recorded in Table 4-4 are in very close agreement because both are creating a strictly Markov model for a well defined physical system which falls within the capabilities of both programs. Though the Flight Control Computation Group is a realistic system, its behavior with regard to reliability is conceptually very straightforward. Since both programs are constructing a Markov model to predict the probability of system loss, the fact that the predictions are in such close agreement tends to validate the models produced from the two programs.

The difference between the predictions of the probability of system loss from the CAME and ASSIST generated models for the Pilot Sensing and Body Motion Sensing Groups in Table 4-4 is attributed to the CAME program's inability to incorporate the failure mode of temporary exhaustion of sensors. With this failure mode removed from the ASSIST generated models, the predictions from the models generated from both programs are in close agreement. Since this failure mode represents a major contributor to system loss in the Body Motion Sensing Group, this limitation of the CAME program is highlighted by this example.

5. CONCLUSIONS

The feasibility of utilizing the CAME program to generate appropriate semi-Markov models to model fault-handling processes is successfully demonstrated. The CAME program is still very much a tool in its research and development phase. But the usefulness of such a tool has been illustrated. The functional groups of the IAPSA II represent a realistic physical system in which the creation of analytical reliability models, without some tool to aid in the construction of these models, would be an extremely difficult task. With the exception of the temporary exhaustion of sensors failure mode, the CAME program is able to construct semi-Markov models which accurately predict the reliability of the system (i. e., as compared with the prior analysis done by the BMAC).

With slight modification, the CAME program would be able to model the temporary exhaustion of sensors failure mode in the Pilot Sensing and Body Motion Sensing Groups of BMAC's functional groups. For the system defined by the IAPSA II, a minor change in the program would allow it to recognize the temporary exhaustion of sensors failure mode. However, the program is being developed as a tool for analyzing the reliability of many types of systems — not just the IAPSA II. Further research is required before incorporating such a change into the general program.

The CAME program is a useful tool in the analysis of the IAPSA II . The methodology of the program and the structure of the user interface are conducive to the task of model the reliability of the IAPSA II. In fact, should the need arise to independently analyze a system like the IAPSA II, a modified version of the CAME program could be created which would recognize all of the identified types of failure modes.

The CAME program presents a framework which can be adapted to allow the modeling of many reliability problems. The program automates the model construction process and incorporates many of the techniques available for reducing the size of the ultimate model. At this time, all the capabilities of the program have not been explored nor all its facets defined. Just what features should be included in the program is a current area of research.

6. REFERENCES

[1] Schabowsky, Jr., R. S., E. Gai, B. Walker, J. Lala, and P. Motyka,"Evaluation Methodologies for an Advanced Information Processing System", Proceedings of the 6th IEEE/AIAA Digital Avionics Systems Conference, December 1984.

[2] Babcock, IV, P. S., F. Leong, and E. Gai, On the Next Generation of Reliability Analysis Tools, NASA Contractor Report 178380, The Charles Stark Draper Laboratory, Inc., Cambridge, MA, October 1987.

[3] Walker, B. K., A Semi-Markov Approach to Quantifying Fault-Tolerant System Performance, T-717, The Charles Stark Draper Laboratory, Inc., Cambridge, MA, July 1980.

[4] Geist, R. M. and K. S. Trivedi, "Ultra-high Reliability Prediction for Fault-Tolerant Computer Systems", IEEE Transactions on Computers, Vol. C-32, December 1983, pp. 1118-1127.

[5] Bavuso, S. J., D. L. Peterson, and D. M. Rose, CARE III Model Overview and User's Guide, NASA Technical Memorandum 85810, June 1984.

[6] Trivedi, K. S. and R. M. Geist, A Tutorial on the CARE III Approach to Reliability Modeling, NASA Contractor Report 3488, December 1981.

[7] Dugan, J. B., K. S. Trivedi, M. K. Smotherman, and R. M. Geist, The Hybrid Automated Reliability Predictor, Duke University, 1985.

[8] Lala, J. H., MARK 1 - Markov Modeling Package, The C. S. Draper Laboratory, Inc., Cambridge, Mass., March 1983.

[9] Lala, J. H.,"Interactive Reductions in the Number of States in Markov Reliability Analysis", Proceedings of the AIAA Guidance and Control Conference, August 1983.

[10] Butler, R. W., The SURE Reliability Analysis Program, NASA Technical Memorandum 87593, 1986.

[11] Schabowsky, Jr., R. S., M. A. Radlauer, and J. Brandner, "Automated Markov Reliability Model Formulation", Proceedings of Robotics and Expert Systems Conference, June 1986.

[12] Schabowsky, Jr., R. S., M. A. Luniewicz, and M. A. Radlauer, "Automated Reliability Model Construction", Proceedings of AIAA Guidance, Navigation, and Control Conference, August 1986.

[13] Viewgraphs & ASSIST Listings from BMAC's IAPSA II Presentation, NASA Langley Research Center, June 8, 1987.

[14] Babcock, IV, P. S., An Introduction to Reliability Modeling of Fault-Tolerant Systems, CSDL-R-1899, The Charles Stark Draper Laboratory, Inc., Cambridge, MA, September 1986.

[15] Butler, R. W., An Abstract Specification Language for Markov Reliability Models, NASA Technical Memorandum 86423, April 1985.

[16] Butler, R. W., "An Abstract Specification Language for Markov Reliability Models", IEEE Transactions on Reliability, Vol. R-35, No. 5, December 1986.

[17] Johnson, S. C., ASSIST User's Manual, NASA Technical Memorandum 87735, August 1986.

Table 4-1. Comparison of the Models Generated with the ASSIST and CAME Programs

FUNCTIONAL GROUP	ASSIST GENERATED MODEL			CAME GENERATED MODEL		
	Run Time (s)	Number of States	Number of Transitions	Run Time	Number of States	Number of Transitions
Flight Control Computation						
Loss of access to both networks	22.06	224	1120	48 s	15	28
Loss of access to either network	17.51	220	818	43 s	17	36
Pilot Sensing						
Original	14.63	266	353	17 h 12 m	99	356
Without temporatory exhaustion failure sequences	12.94	249	336			
Body Motion Sensing						
Original	57.11	252	516	13 h 14 m	40	110
Without temporatory exhaustion failure sequences	33.64	235	495			

Table 4-2. Failure Rates Used in Analysis of IAPSA II Reference Configuration

COMPONENT	FAILURE RATE (failures / 10^6 h)
Flight Control Computation	
FTP Channel	220.0
FTP Network Interface	40.0
Pilot Sensing	
Pitch Position Sensor	10.0
Roll Position Sensor	10.0
Yaw Position Sensor	10.0
Network Node (including DIU and Primary Electrical System)	50.0
Network Temporary Outage	561.0
Body Motion Sensing	
Gyro	50.0
Accelerometer	30.0
Network Node (includes DIU and Primary Electrical System)	50.0
Network Temporary Outage	561.0

Table 4-3. Semi-Markov Transition Parameters Used in Analysis of IAPSA II Reference Configuration

TYPE	STATISTICS	
	Mean (hours)	Standard Deviation (hours)
Sensor Recovery Time	3.0×10^{-4}	1.0×10^{-4}
Temporary Network Outage Duration	3.0×10^{-4}	1.0×10^{-4}

Table 4-4. Results Produced with SURE Program

FUNCTIONAL GROUP	PROBABILITY OF LOSS AT 3 h			
	ASSIST GENERATED MODEL		CAME GENERATED MODEL	
	Lower Bound	Upper Bound	Lower Bound	Upper Bound
Flight Control Computation				
Loss of access to both networks	1.15×10^{-9}	1.15×10^{-9}	1.15×10^{-9}	1.15×10^{-9}
Loss of access to either network	1.52×10^{-9}	1.52×10^{-9}	1.52×10^{-9}	1.53×10^{-9}
Pilot Sensing				
Original	2.22×10^{-10}	2.25×10^{-10}	1.64×10^{-10}	2.26×10^{-10}
Without temporatory exhaustion failure sequences	1.64×10^{-10}	1.67×10^{-10}		
Body Motion Sensing				
Original	5.04×10^{-7}	5.06×10^{-7}	7.74×10^{-10}	9.46×10^{-10}
Without temporatory exhaustion failure sequences	7.74×10^{-10}	7.91×10^{-10}		

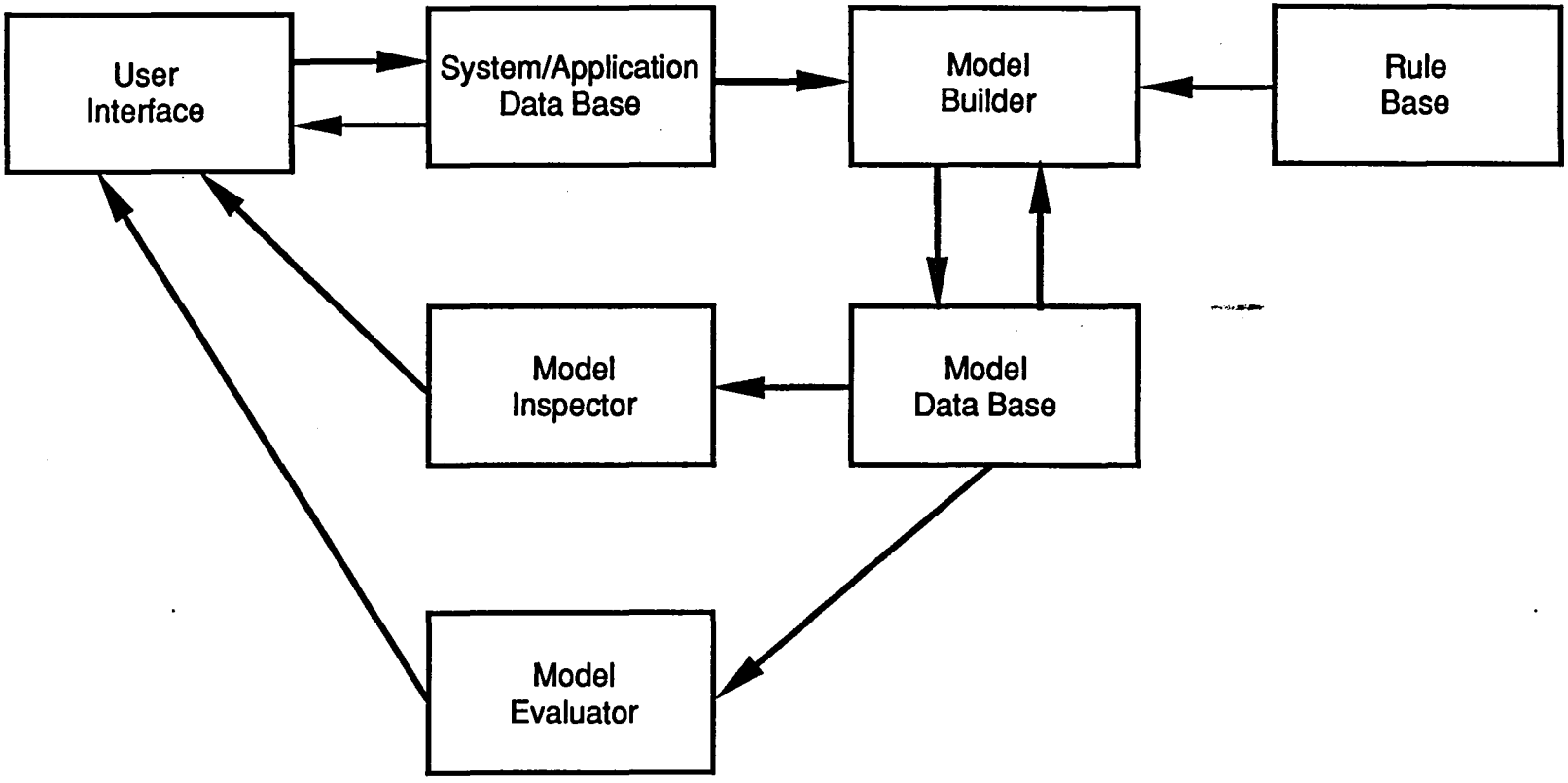

Figure 2-1. Block Diagram of CAME Program

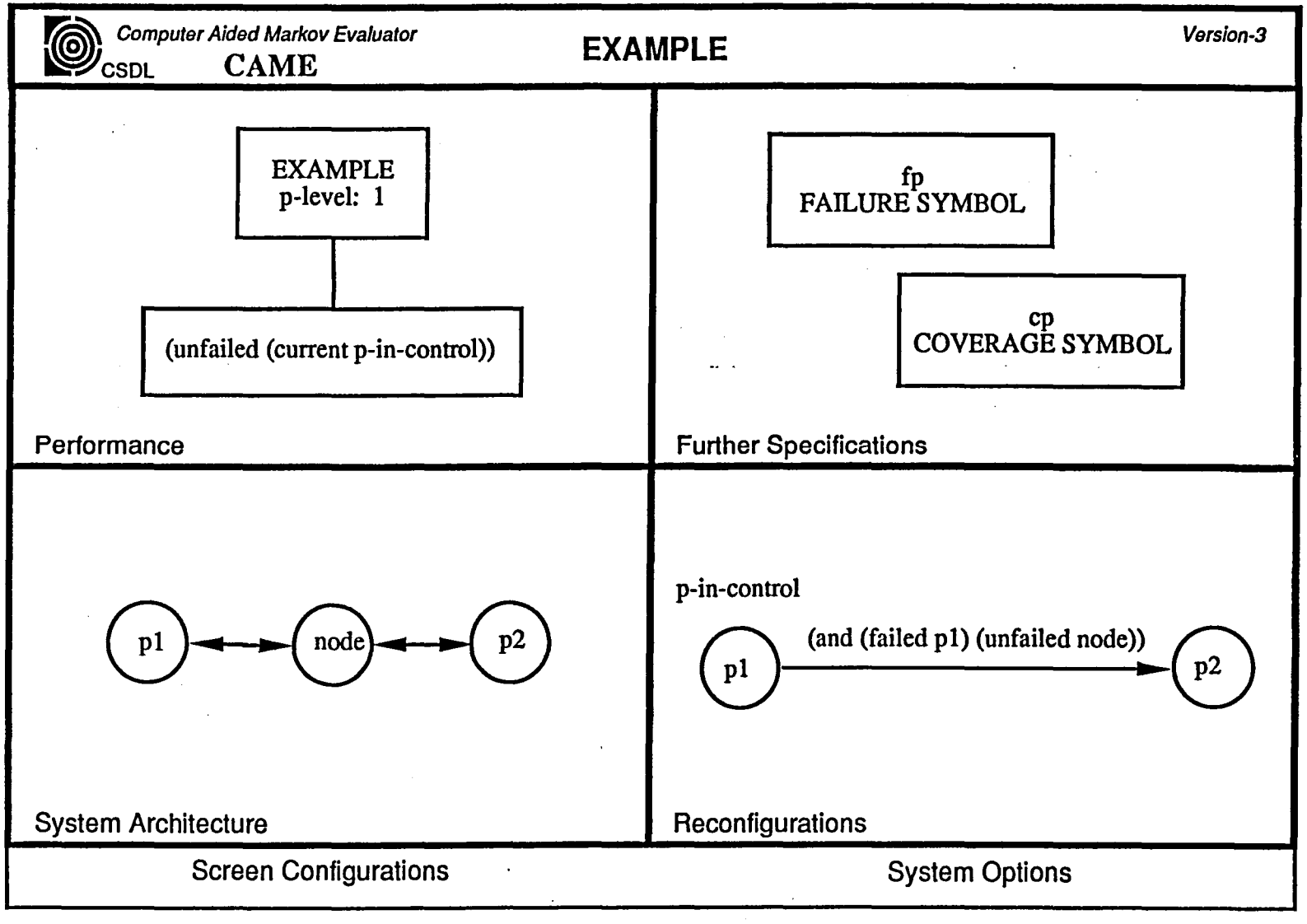

Figure 2-2. Example System Description

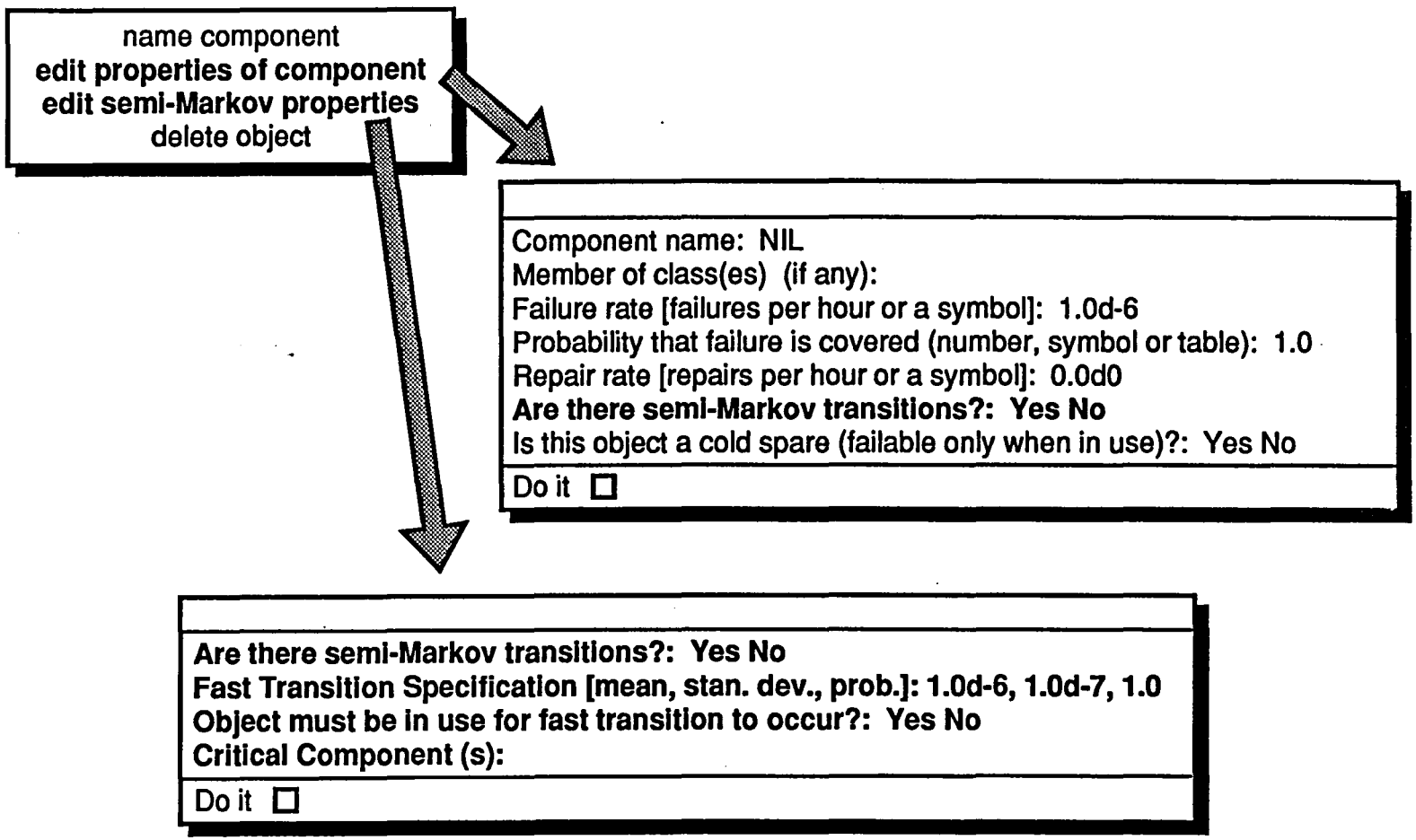

Figure 3-1. Alterations to System Architecture Menu

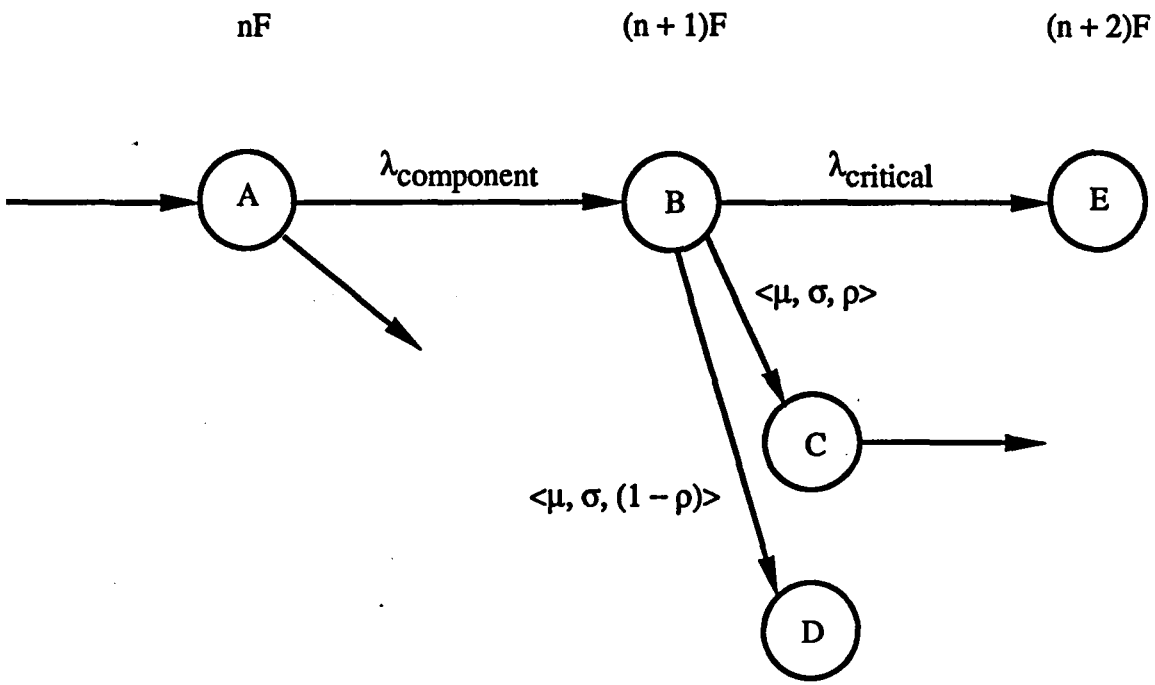

Figure 3-2. Alteration to Model Building Process

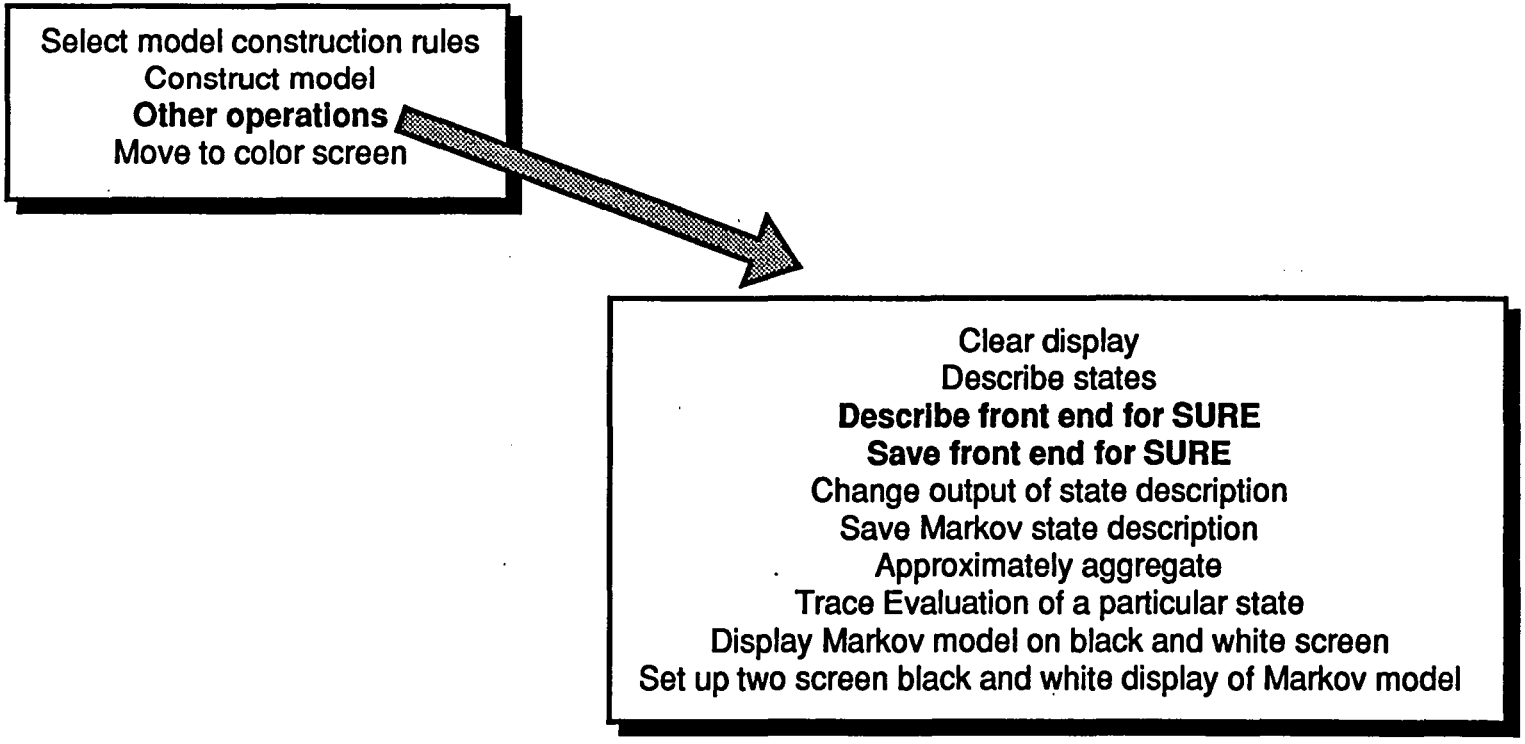

Figure 3-3. Alterations to Model Evaluator Menus

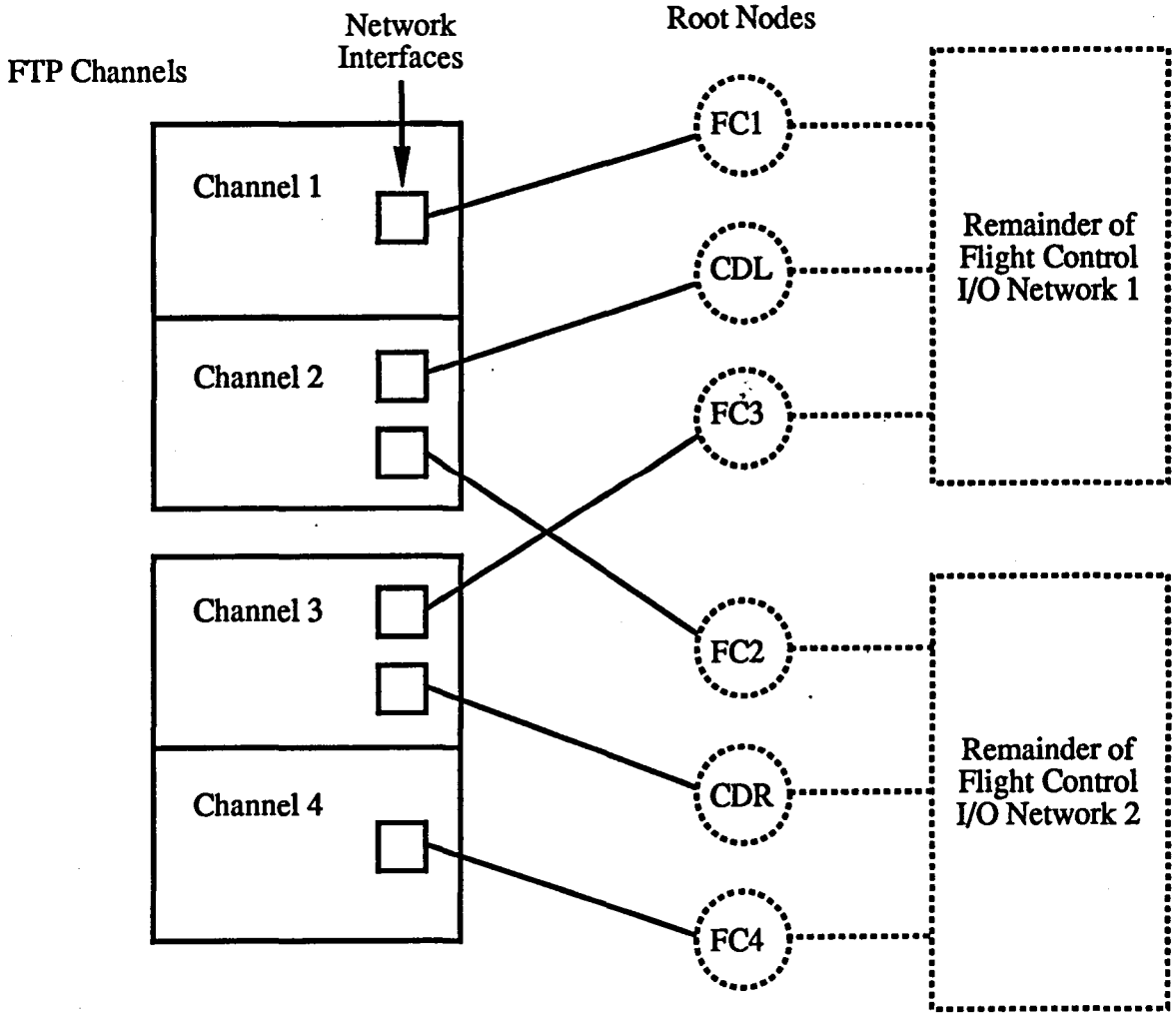

Figure 4-1. Flight Control Computation Group

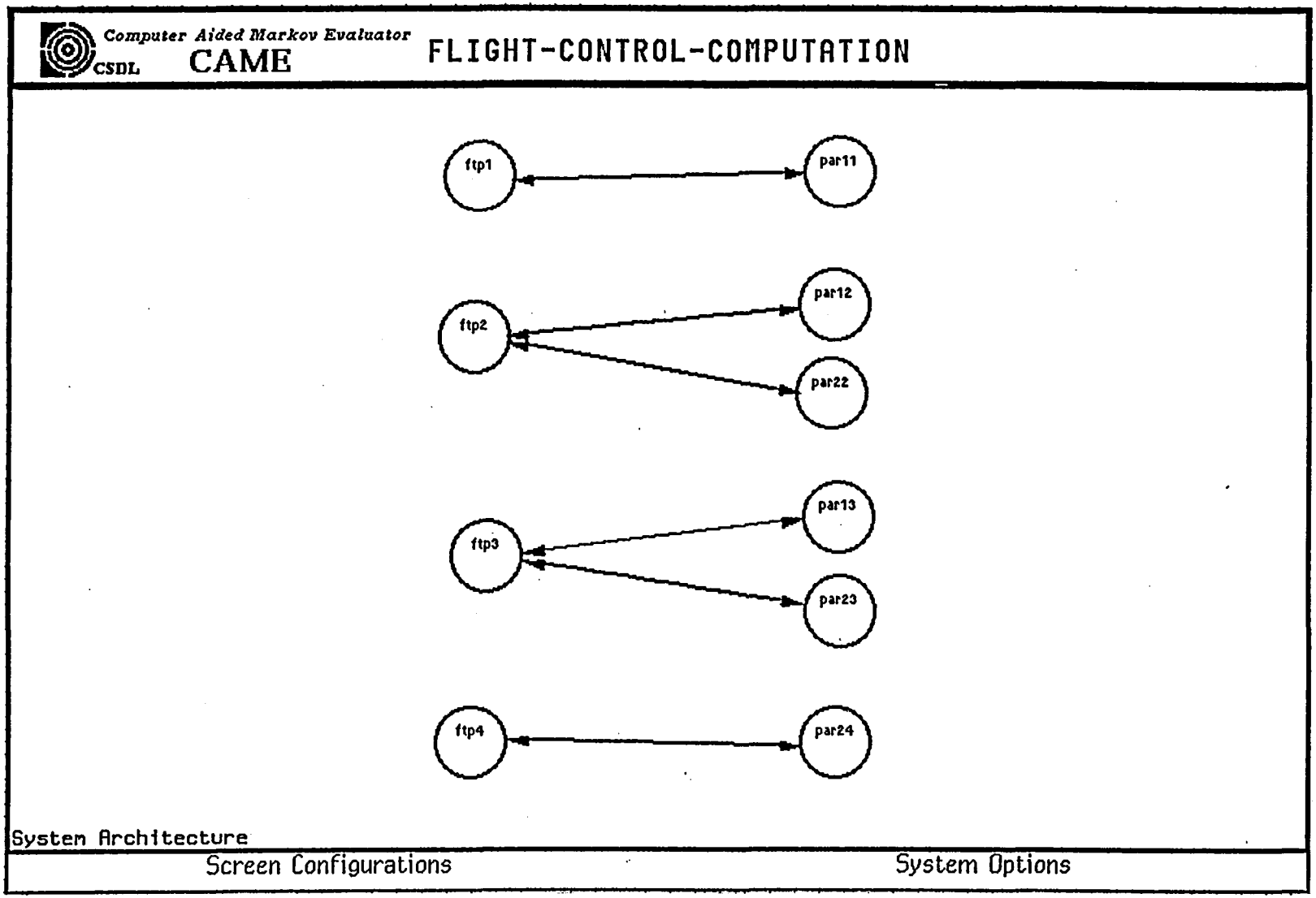

Figure 4-2. System Architecture Window for the Flight Control Computation Group

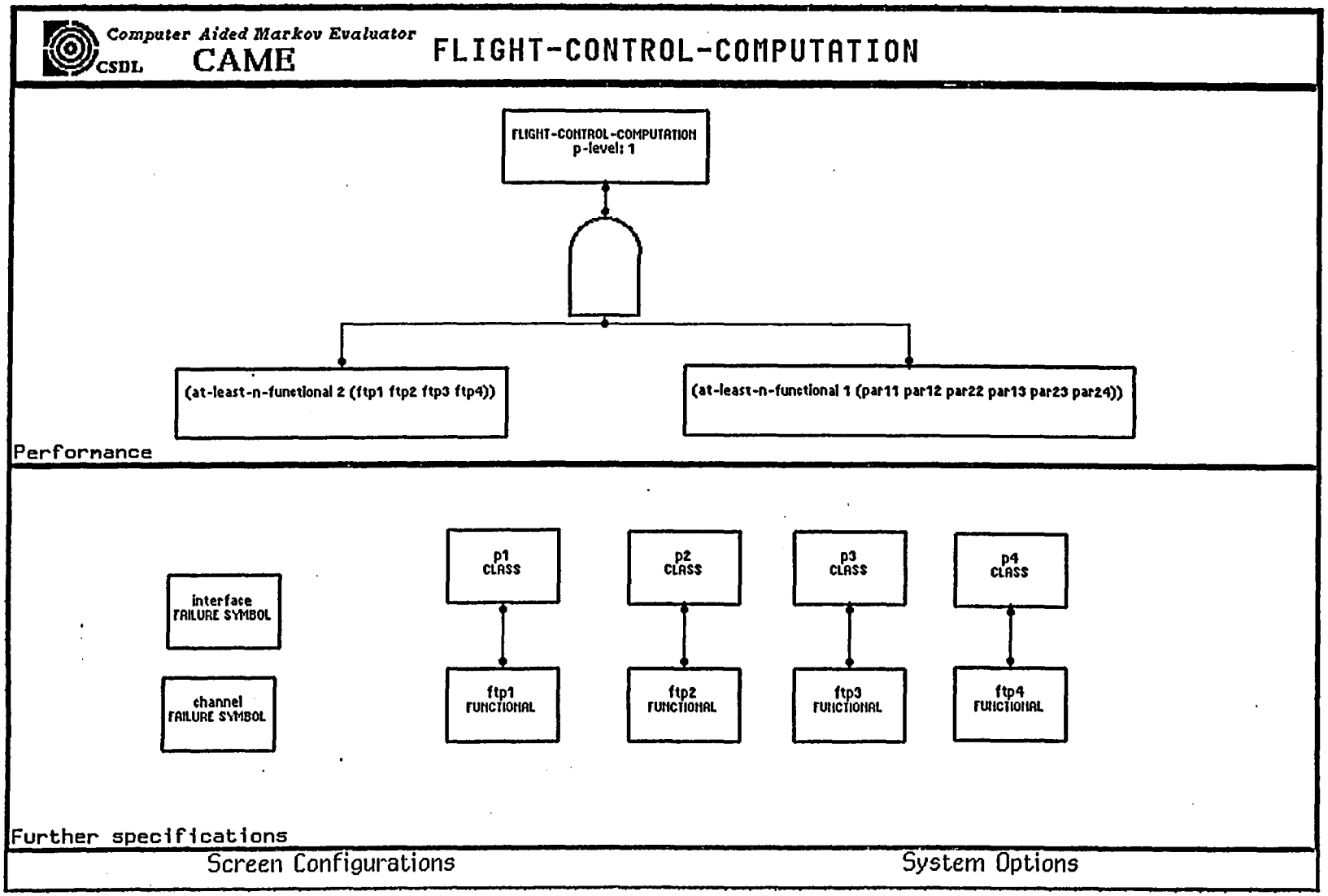

Figure 4-3. Performance Requirements and Further Specification Windows for the Flight Control Computation Group

```
(* The following is a file created by the CAME program
    in the language of the SURE program.

       Name of System:  FLIGHT-CONTROL-COMPUTATION
       DATE CREATED:  12/02/87 14:34:56 *)

(* WARNING!!! Original symbol name interface changed to inter *)
inter = 4.0d-5;
(* WARNING!!! Original symbol name channel changed to chann *)
chann = 2.2d-4;
(* Failure rate for component ftp1 is the symbol chann *)
ftp1 = chann;
(* Failure rate for component ftp2 is the symbol chann *)
ftp2 = chann;
(* Failure rate for component ftp3 is the symbol chann *)
ftp3 = chann;
(* Failure rate for component ftp4 is the symbol chann *)
ftp4 = chann;
(* Failure rate for component par11 is the symbol inter *)
par11 = inter;
(* Failure rate for component par12 is the symbol inter *)
par12 = inter;
(* Failure rate for component par13 is the symbol inter *)
par13 = inter;
(* Failure rate for component par24 is the symbol inter *)
par24 = inter;
(* Failure rate for component par22 is the symbol inter *)
par22 = inter;
(* Failure rate for component par23 is the symbol inter *)
par23 = inter;
(* User should only edit values not symbols after the model is built *)

1,3 = ftp3 + ftp2;
1,2 = ftp4 + ftp1;
1,4 = par24 + par11;
1,5 = par23 + par22 + par13 + par12;
2,7 = chann;
2,6 = ftp3 + ftp2;
2,8 = par23 + par22 + par13 + par12 + inter;
3,9 = chann;
3,6 = ftp4 + ftp1;
3,10 = inter + par24 + inter + par11;
4,8 = chann;
4,10 = ftp3 + ftp2;
4,11 = chann;
4,12 = par23 + par22 + par13 + par12 + inter;
5,10 = chann;
5,8 = chann + ftp4 + ftp1;
5,12 = inter + par24 + inter + inter + par11;
```

Figure 4-4. SURE Input Listing for Flight Control Computation Group

```
6,13 = chann + chann;
6,14 = inter + inter + inter;
7,13 = ftp2 + ftp3;
7,14 = par12 + par13 + par22 + par23;
8,14 = chann + chann + chann + inter + inter + inter +
inter;
9,13 = ftp1 + ftp4;
9,14 = par11 + par24;
10,14 = ftp1 + chann + fpt4 + inter + inter + inter;
11,14 = chann + chann + chann + inter + inter + inter +
par22 + par23;
12,14 = ftp1 + ftp2 + ftp3 + fpt4 + inter + inter +
inter + inter;
14,15 = ftp1 + ftp2 + ftp3 + fpt4 + par11 + par12 +
par13 + par24 + par22 + par23;

(* WARNING!! A truncated model was constructed by the CAME program.
   Therefore, discretion should be used in interpreting the lower and
   upper bounds of the probability of system loss results from SURE. *)

(* WARNING!! Model was aggregated by the CAME program. *)

(* The lower bounds of the probability of system loss of the SURE model
   is the sum of the lower bounds of the states generated by the CAME
   program.  That is, the sum of the lower bounds of states  13 *)

(* The upper bounds of the probability of system loss of the SURE model
   is the sum of the upper bounds calculated by the SURE program
   of all the system loss states generated by the CAME program
   That is, the sum of the upper bounds of states  13 15 *)

(* White's method is assumed *)
(* The following default values are used: *)
list = 2;
time = 10;
run sure.out;
exit
```

Figure 4-4. SURE Input Listing for Flight Control Computation Group (Continued)

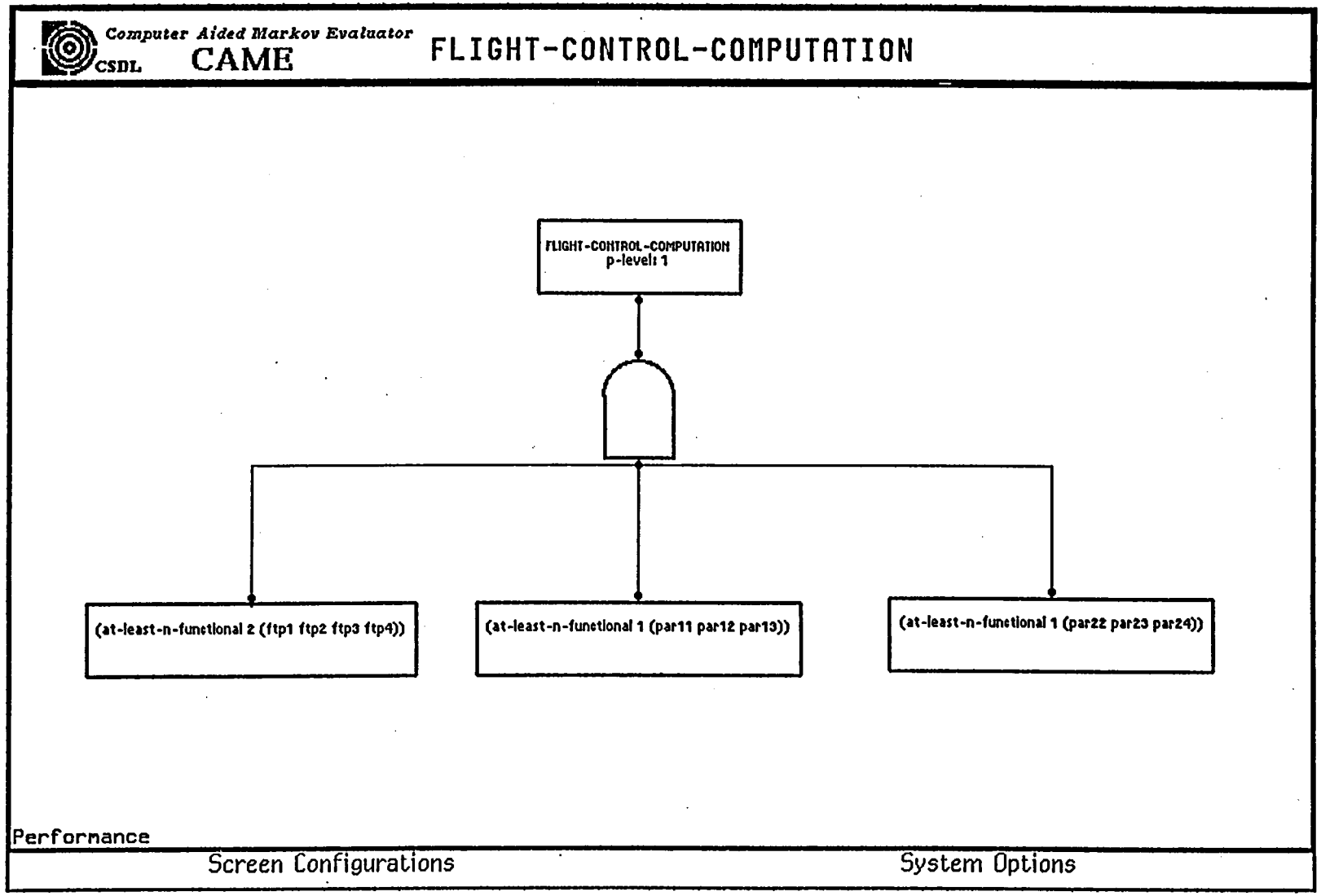

Figure 4-5. Variation of the Flight Control Computation Group

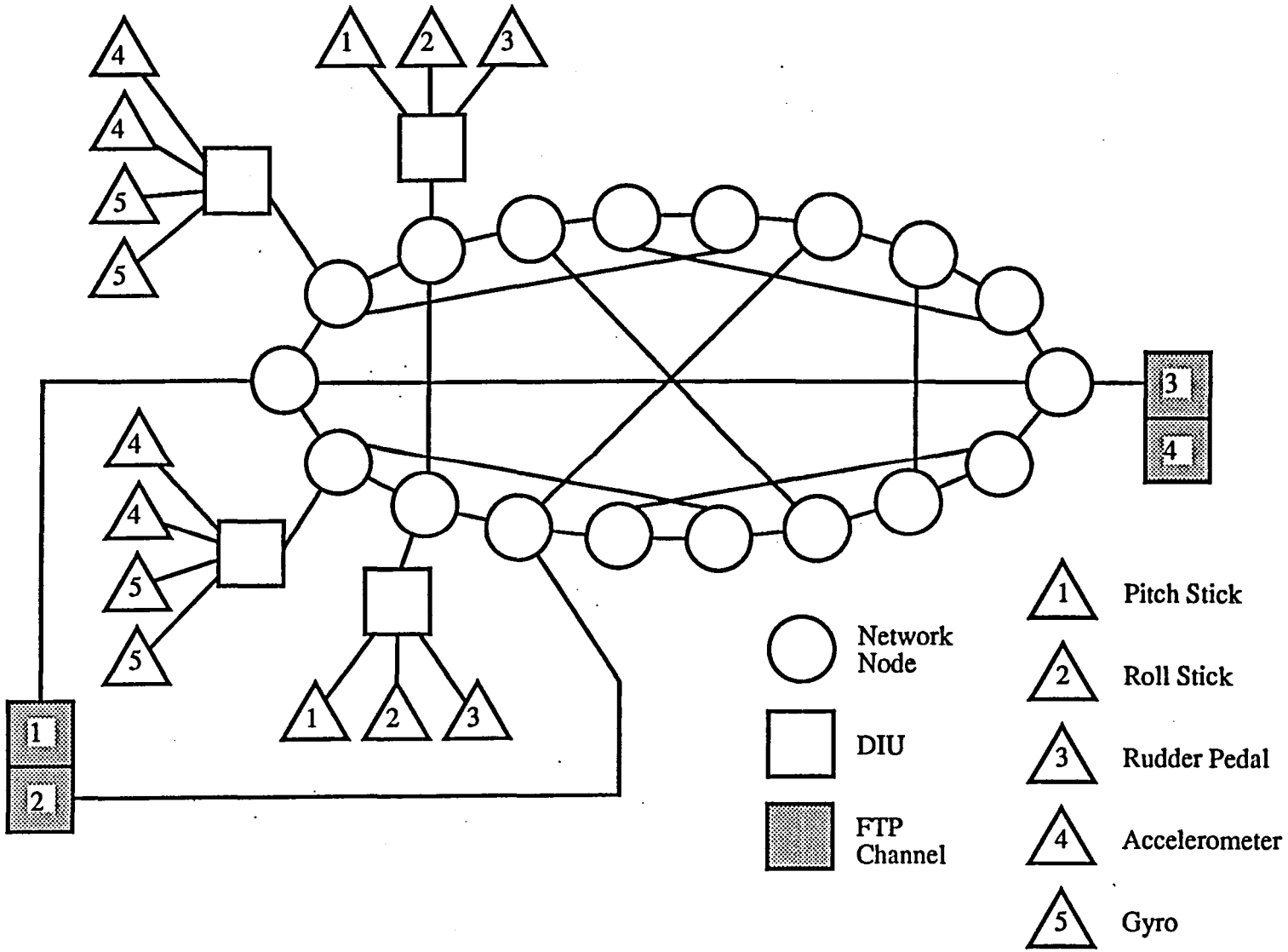

Figure 4-6. Flight Control I/O Network 1

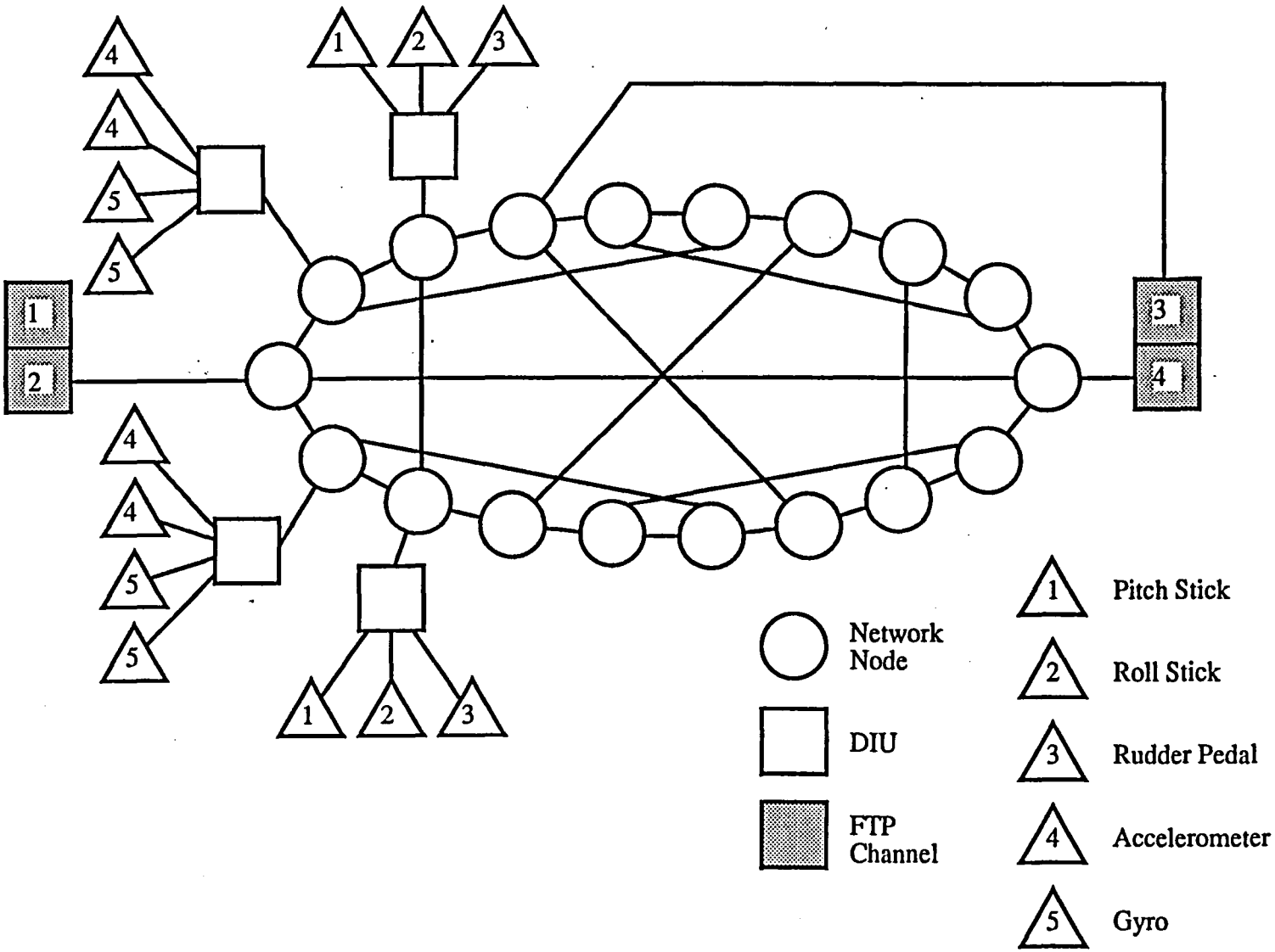

Figure 4-7. Flight Control I/O Network 2

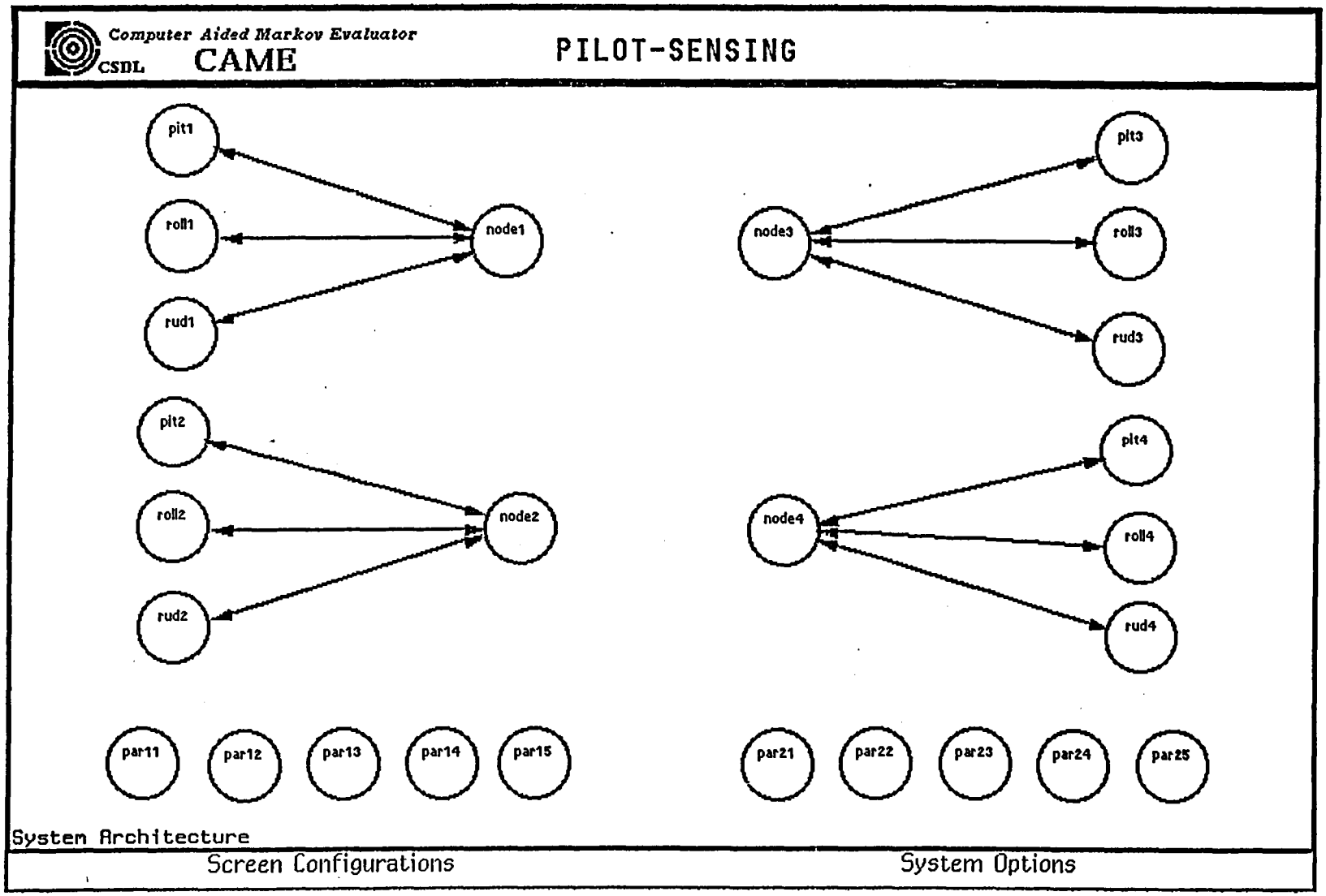

Figure 4-8. System Architecture Window for the Pilot Sensing Group

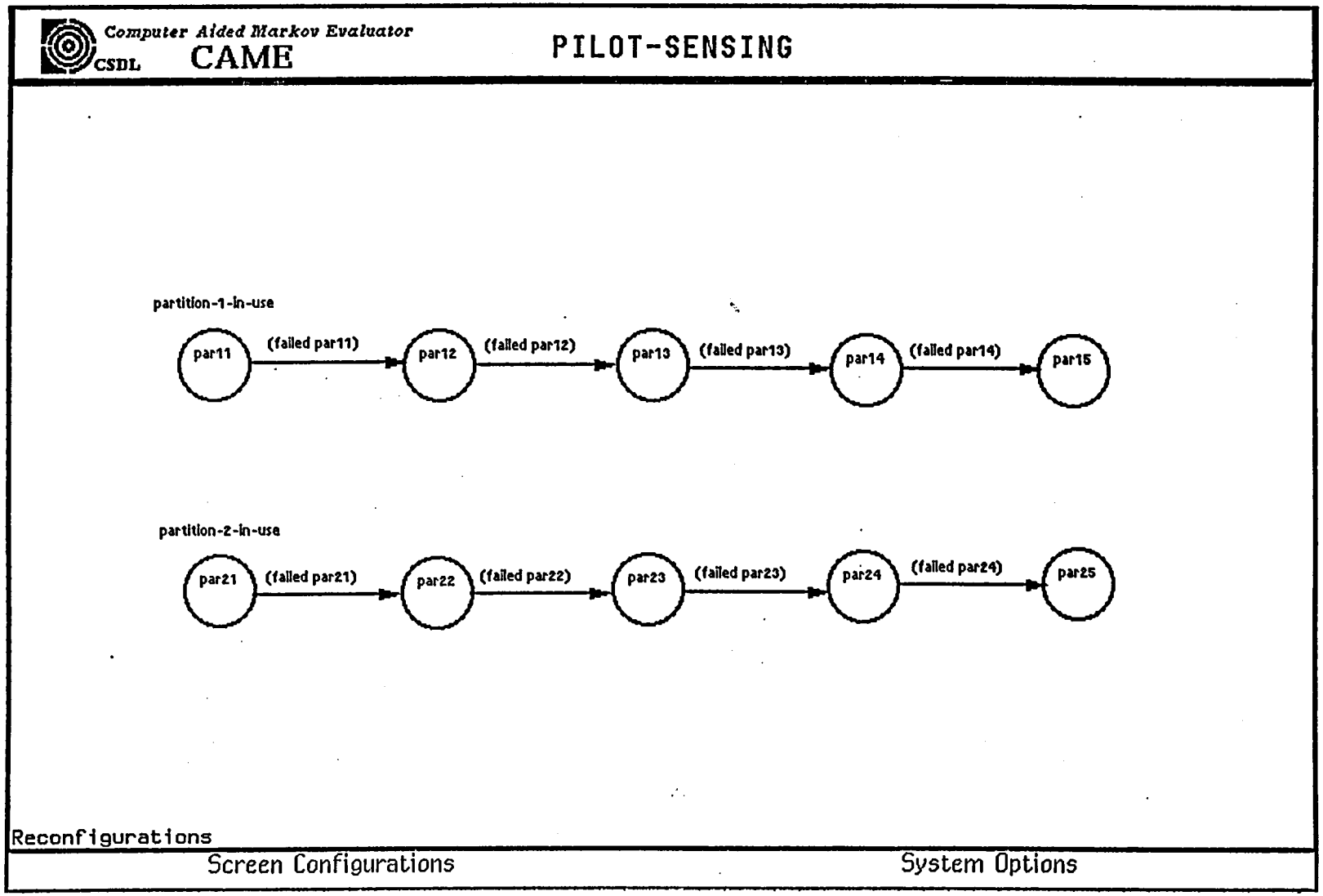

Figure 4-9. Reconfigurations Window for the Pilot Sensing Group

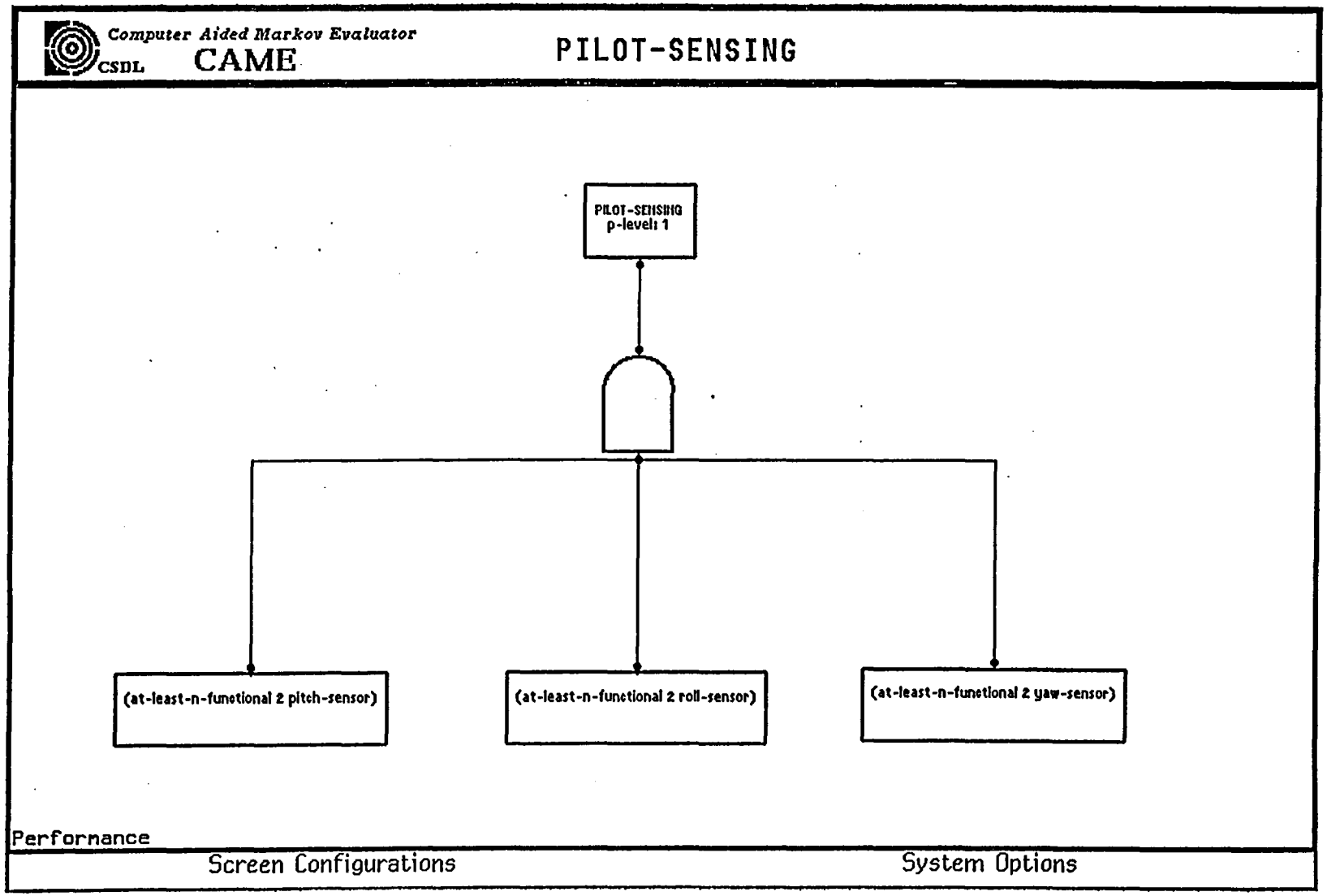

Figure 4-10. Performance Requirements Window for the Pilot Sensing Group

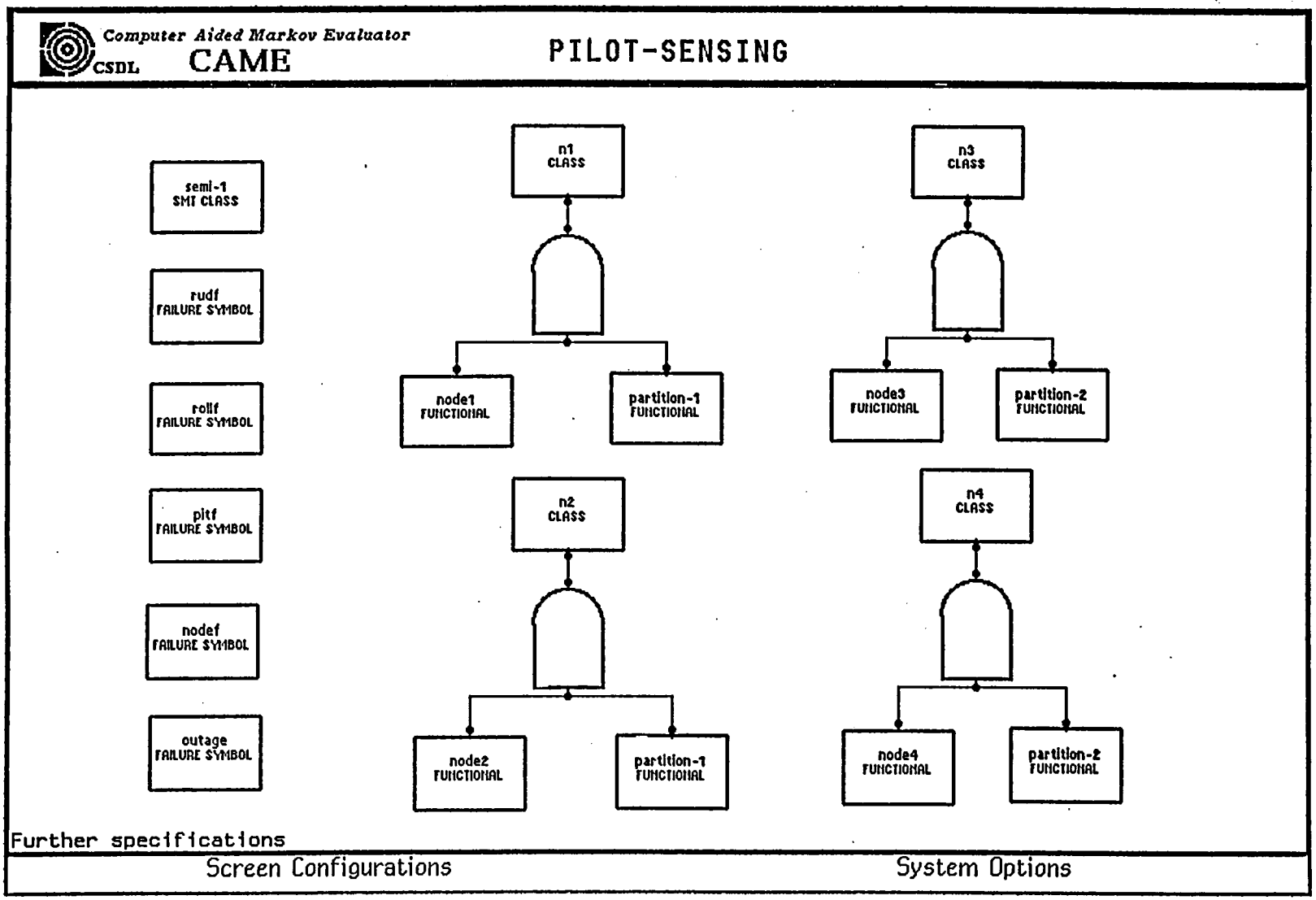

Figure 4-11. Further Specifications Window for the Pilot Sensing Group

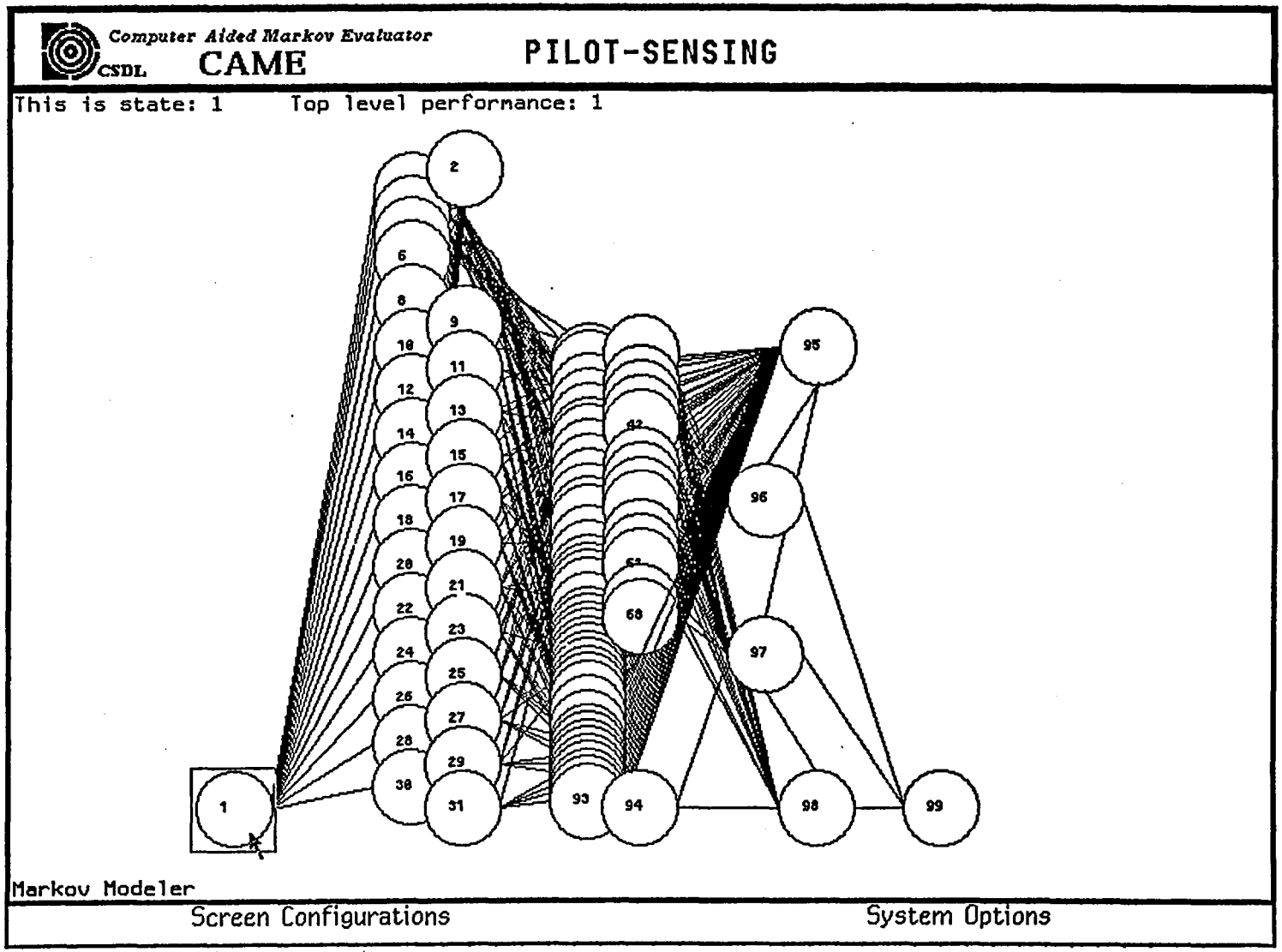

Figure 4-12. Semi-Markov Model for the Pilot Sensing Group

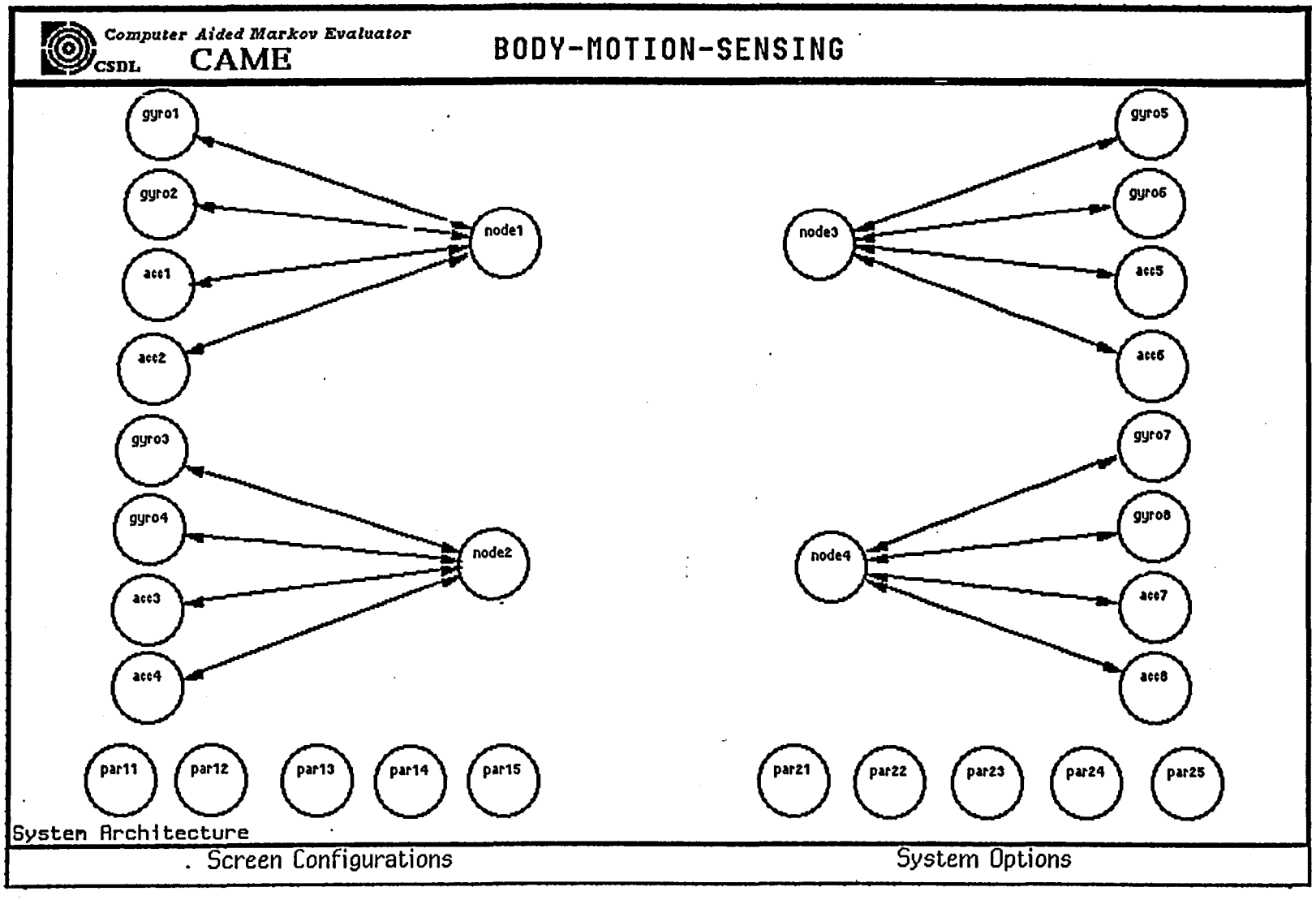

Figure 4-13. System Architecture Window for the Body Motion Sensing Group

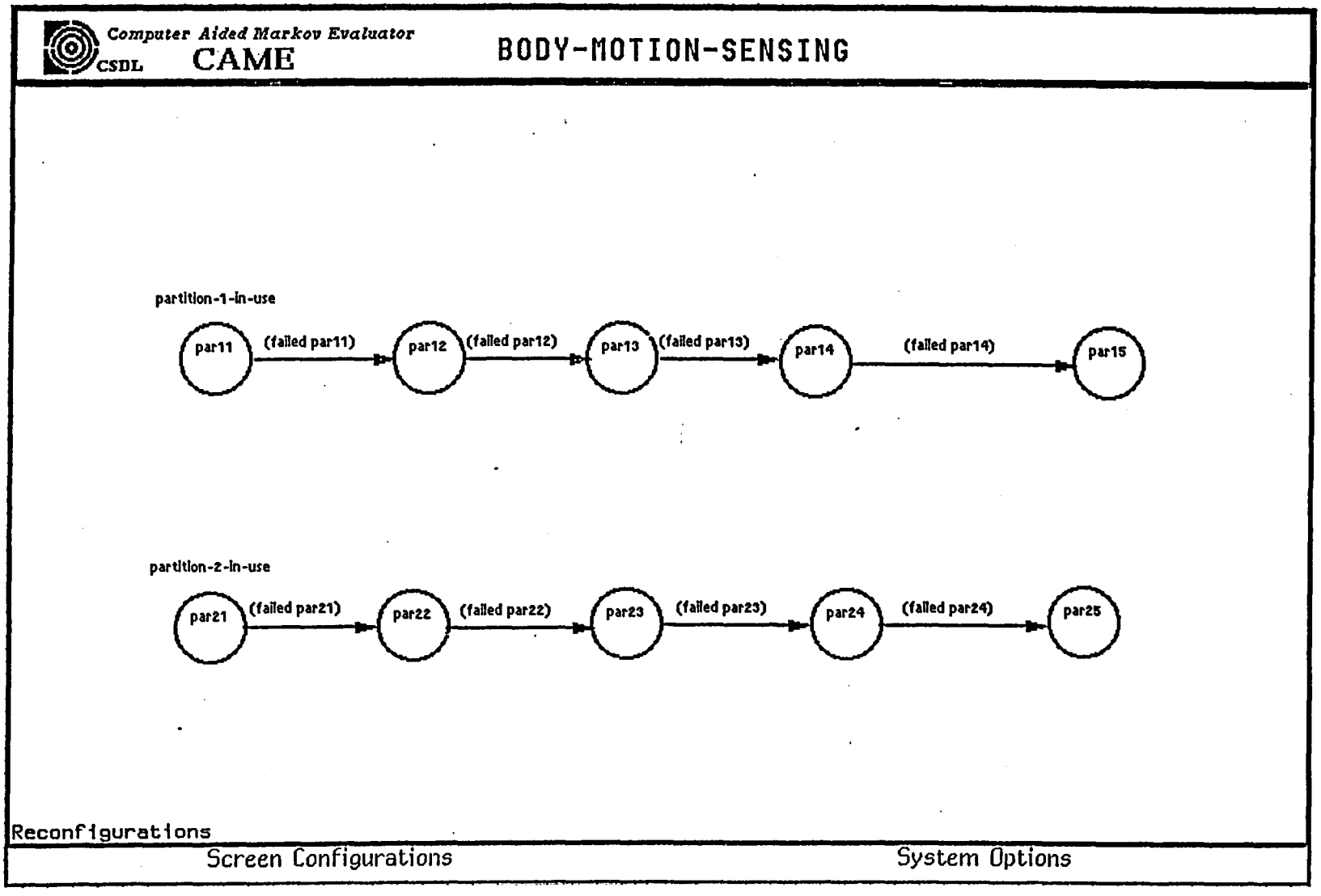

Figure 4-14. Reconfigurations Window for the Body Motion Sensing Group

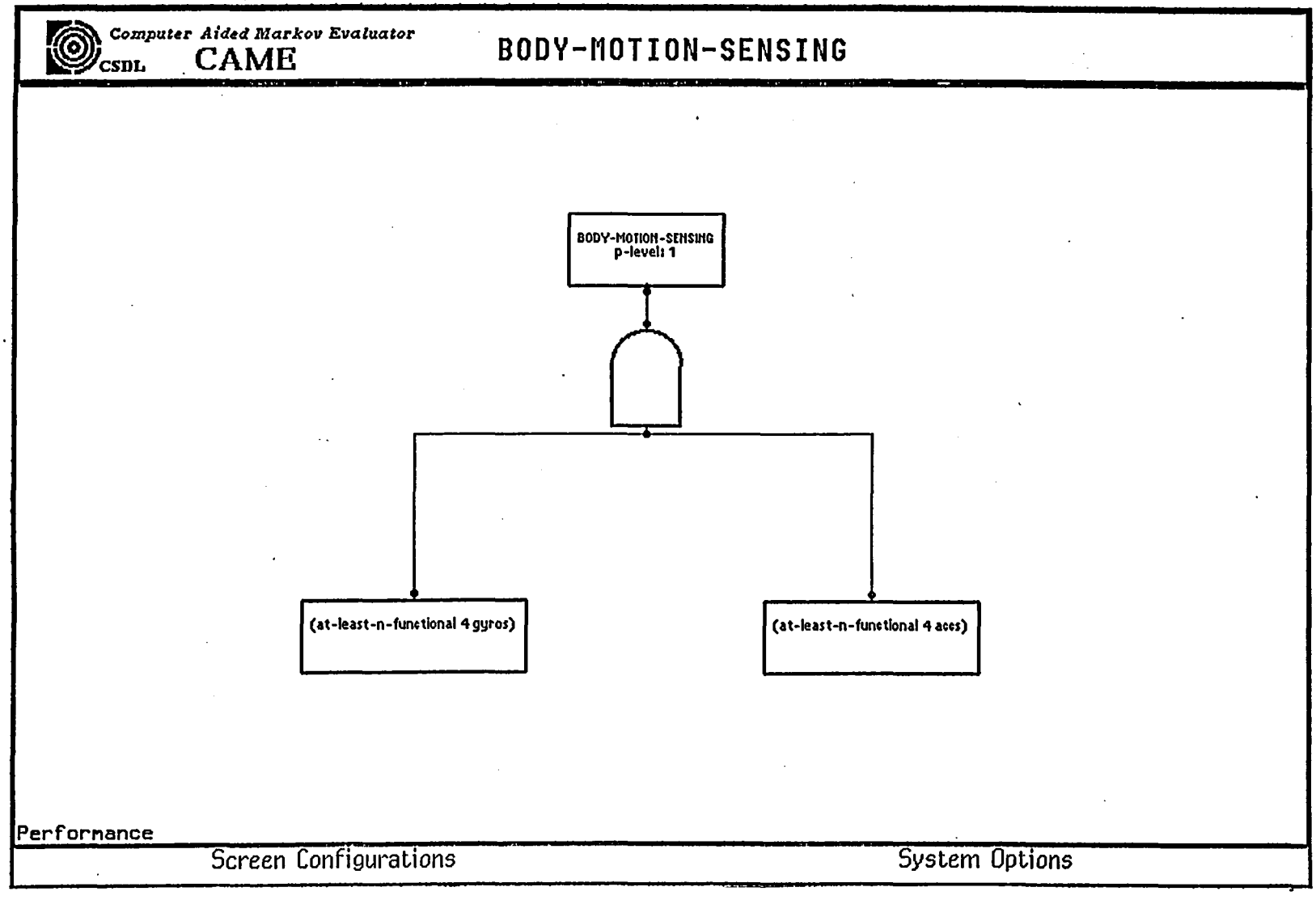

Figure 4-15. Performance Requirements Window for the Body Motion Sensing Group

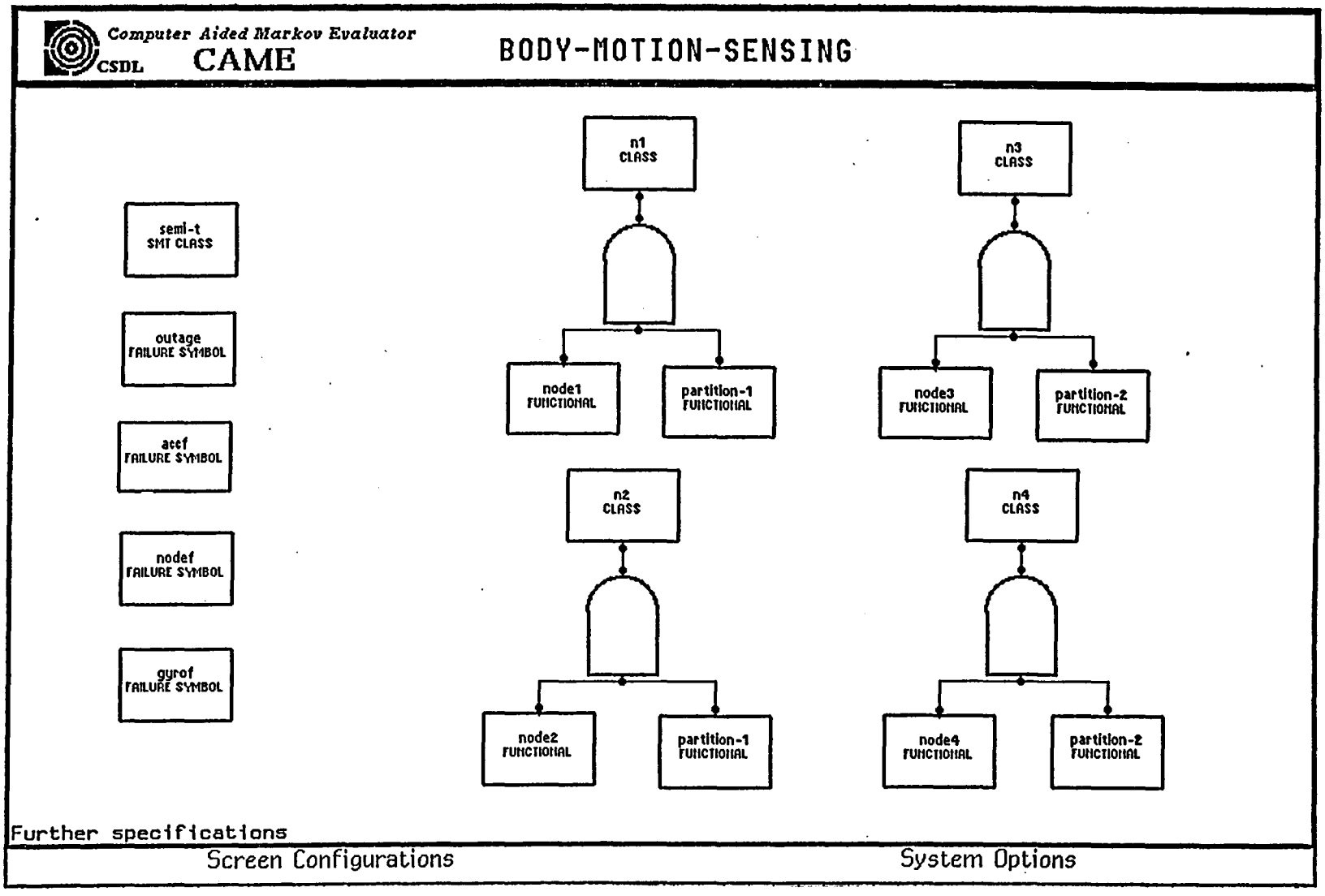

Figure 4-16. Further Specifications Window for the Body Motion Sensing Group

APPENDIX A

A. BMAC'S ASSIST INPUT LISTINGS

A.1 Flight Control Computation

```
SPACE = ( NGFTP1: 0..1,      (* FTP CHANNEL STATUS           *)
          NPAR11: 0..1,      (* PARTITION INTERFACE STATUS   *)
          NGFTP2: 0..1,      (* FTP CHANNEL STATUS           *)
          NPAR12: 0..1,      (* PARTITION INTERFACE STATUS   *)
          NPAR22: 0..1,      (* PARTITION INTERFACE STATUS   *)
          NGFTP3: 0..1,      (* FTP CHANNEL STATUS           *)
          NPAR13: 0..1,      (* PARTITION INTERFACE STATUS   *)
          NPAR23: 0..1,      (* PARTITION INTERFACE STATUS   *)
          NGFTP4: 0..1,      (* FTP CHANNEL STATUS           *)
          NPAR24: 0..1);     (* PARTITION INTERFACE STATUS   *)

START = ( 1,1, 1,1,1, 1,1,1, 1,1 );

DEATHIF     NGFTP1 + NGFTP2 + NGFTP3 + NGFTP4 < 2
        OR NPAR11 + NPAR12 + NPAR13 (* SINGLE PARTITION SUCCESS CASE *)
        +  NPAR22 + NPAR23 + NPAR24 < 1; (* SINGLE PARTITION SUCCESS *)

LAMFTP  = 220.0E-6;          (* FTP CHANNEL FAILURE RATE              *)
                             (*   ---INCLUDES CENTRAL POWER SOURCE--- *)
LAMCOM  = 40.0E-6;           (* FTP NETWORK INTERFACE FAILURE RATE    *)
                             (*   ---INCLUDES ROOT NODE---            *)

IF NGFTP1 > 0 TRANTO NGFTP1 = 0, NPAR11 = 0
                BY LAMFTP;
IF NGFTP2 > 0 TRANTO NGFTP2 = 0, NPAR12 = 0,
                NPAR22 = 0
                BY LAMFTP;
IF NGFTP3 > 0 TRANTO NGFTP3 = 0, NPAR13 = 0,
                NPAR23 = 0
                BY LAMFTP;
IF NGFTP4 > 0 TRANTO NGFTP4 = 0, NPAR24 = 0
                BY LAMFTP;

IF NPAR11 > 0 TRANTO NPAR11 = 0 BY LAMCOM;
IF NPAR12 > 0 TRANTO NPAR12 = 0 BY LAMCOM;
IF NPAR13 > 0 TRANTO NPAR13 = 0 BY LAMCOM;
IF NPAR22 > 0 TRANTO NPAR22 = 0 BY LAMCOM;
IF NPAR23 > 0 TRANTO NPAR23 = 0 BY LAMCOM;
IF NPAR24 > 0 TRANTO NPAR24 = 0 BY LAMCOM;
```

A.2 Pilot Sensing

```
SPACE = ( NGPIT: 1..4,       (* NUMBER OF ACTIVE PITCH STICK SENSORS  *)
          NFPIT: 0..2,       (* NO. OF RECOVERING PITCH SENSORS       *)
          NGROL: 1..4,       (* NO. OF ACTIVE ROLL STICK SENSORS      *)
          NFROL: 0..2,       (* NO. OF RECOVERING ROLL SENSORS        *)
          NGYAW: 1..4,       (* NO. OF ACTIVE RUDDER PEDAL SENSORS    *)
          NFYAW: 0..2,       (* NO. OF RECOVERING YAW SENSORS         *)
          NGCOM: 0..4,       (* NO. OF COCKPIT NETWORK NODES          *)
          PAROUT: 0..2 );    (* PARTION RECOVERY INDICATOR            *)

START = ( 4, 0, 4, 0, 4, 0, 4, 0 );

DEATHIF ( NGCOM = 4 AND NGPIT + NGROL + NGYAW < 9 ) OR  (* SPECIAL *)
        ( NGCOM = 3 AND NGPIT + NGROL + NGYAW < 6 ) OR  (* TRUNCATION *)
        NGPIT-NFPIT < 2 OR NGROL-NFROL < 2 OR NGYAW-NFYAW < 2 OR
        NFPIT + PAROUT > 1 OR NFROL + PAROUT > 1 OR
        NFYAW + PAROUT > 1 OR NGCOM < 2;

LIST  = 3;
TIME  = 3.0;
PRUNE =1.0E-15;
ECHO  = 0;

LAMPIT  = 10.0E-6;           (* POSITION SENSOR FAILURE RATE        *)
LAMROL  = 10.0E-6;
LAMYAW  = 10.0E-6;
LAMCOM  = 15.0E-6;           (* NETWORK NODE FAILURE RATE           *)
LAMOUT  = 561.0E-6;          (* NETWORK TEMPORARY OUTAGE RATE       *)
LAMDIU  = 15.0E-6;           (* DIU FAILURE RATE                    *)
LAMELEC = 20.0E-6;           (* PRIMARY ELECTRICAL SYSTEM FAIL RATE *)
RECMEAN = 3.0E-4;            (* SENSOR RECOVERY TIME MEAN           *)
RECSTD  = 1.0E-4;            (* SENSOR RECOVERY TIME SDT DEV        *)

IF NGPIT > 0 AND NFROL = 0 AND NFYAW = 0
        TRANTO NFPIT = NFPIT+1 BY (NGPIT-NFPIT)*LAMPIT;

IF NGROL > 0 AND NFPIT = 0 AND NFYAW = 0
        TRANTO NFROL = NFROL+1 BY (NGROL-NFROL)*LAMROL;

IF NGYAW > 0 AND NFPIT = 0 AND NFROL = 0
        TRANTO NFYAW = NFYAW+1 BY (NGYAW-NFYAW)*LAMYAW;

        (*** PARTITION OUTAGE FAILURES AND SENSOR RECOVERIES ***)

IF NFPIT > 0 TRANTO NGPIT = NGPIT-1, NFPIT = NFPIT-1 BY
            < RECMEAN, RECSTD >;

IF NFROL > 0 TRANTO NGROL = NGROL-1, NFROL = NFROL-1 BY
            < RECMEAN, RECSTD >;

IF NFYAW > 0 TRANTO NGYAW = NGYAW-1, NFYAW = NFYAW-1 BY
            < RECMEAN, RECSTD >;

IF PAROUT = 0 AND ( NGCOM = 4 AND NGPIT+NGROL+NGYAW > 9 OR
              NGCOM = 3 AND NGPIT+NGROL+NGYAW > 6 )
              TRANTO PAROUT = 1 BY LAMOUT;
```

```
IF PAROUT = 1 TRANTO PAROUT = 0 BY < RECMEAN, RECSTD >;

      (*** TEMPORARY EXHAUSTION DUE TO OUTAGE ***)

IF NFPIT = 0 AND NFROL = 0 AND NFYAW = 0 AND PAROUT = 0 THAN
      IF NGPIT= 2 OR NGROL=2 OR NGYAW=2 TRANTO PAROUT = 2 BY LAMOUT/3.0;
ENDIF;

      (*** NODE FAILURE TRANSITIONS ***)
      (*** ONLY FULL TRANSISTIONS CARRIED ***)

IF NGCOM > 0 AND NFPIT = 0 AND NFROL = 0 AND NFYAW = 0 AND PAROUT = 0
           THEN
           TRANTO  NGPIT=NGPIT-1, NGROL=NGROL-1, NGYAW=NGYAW-1,
                   NGCOM=NGCOM-1 BY NGPIT*NGROL*NGYAW *
                   (LAMCOM + LAMDIU + LAMELEC) / NGCOM**2;

ENDIF;
```

A.3 Body Motion Sensing

```
SPACE = ( NGGYRO: 0..8,      (* NO. OF GYROS                          *)
          NFGYRO: 0..2,      (* NO. OF FAILED GYROS BEING RECOVERED   *)
          NGACC:  0..8,      (* NO. OF ACCS                           *)
          NFACC:  0..2,      (* NO. OF ACC BEING RECOVERED            *)
          NGCOM:  1..4,      (* NO. OF NETWORK NODES                  *)
          PAROUT: 0..2 );    (* PARTITION OUT INDICATOR               *)

START = ( 8, 0, 8, 0, 4, 0 );

DEATHIF   NGGRYO  < 4 OR
          NGACC   < 4 OR
          NFGYRO + PAROUT > 1 OR NFACC + PAROUT > 1 ;

LIST    = 3;
TIME    = 3.0;
PRUNE   = 1.0E-15;
ECHO    = 0;

LAMG     = 50.0E-6;        (* GYRO FAILURE RATE                      *)
LAMA     = 30.0E-6;        (* ACCELEROMETER FAILURE RATE             *)
LAMC     = 50.0E-6;        (* NETWORK NODE AND DIU AND ELECTRICAL    *)
                           (* SYSTEM FAILURE RATE - INCLUDES         *)
                           (* LAMDIU = 15.0E-6   LAMELEC = 20.0E-6   *)
LAMOUT   = 561.0E-6;       (* PARTITION OUTAGE RATE                  *)
RECMEAN = 3.0E-4;          (* SENSOR AND PARTITION MEAN RECOVERY TIME *)
RECSTD   = 1.0E-4;         (* SENSOR AND PARTITION SDT DEV           *)

          (*** GYRO FAILURES ***)

IF NGGYRO > 0 AND NFGYRO = 0 AND NFACC = 0 AND PAROUT = 0 THEN
          TRANTO NGGRYO=NGGRYO-1, NFGYRO = NFGYRO + 1 BY NGGYRO*LAMG;
ENDIF;

          (*** ACCELEROMETER FAILURES ***)

IF NGACC > 0 AND NFGYRO = 0 AND NFACC = 0 AND PAROUT = 0 THEN
          TRANTO NGACC=NGACC-1, NFACC = NFACC + 1 BY NGACC*LAMA;
ENDIF;

          (*** SENSOR RECOVERIES AND PARTITION OUTAGE CASES ***)

IF NFGYRO > 0 THEN
     TRANTO NFGYRO=NFGYRO-1 BY < RECMEAN, RECSTD >;
     TRANTO PAROUT=1 BY LAMOUT;
ENDIF;

IF NFACC > 0 THEN
     TRANTO NFACC=NFACC-1 BY < RECMEAN, RECSTD >;
     TRANTO PAROUT=1 BY LAMOUT;
ENDIF;

          (*** PARTITION TEMPORARY OUTAGE ***)
```

```
IF NFGYRO = 0 AND NFACC = 0 AND PAROUT = 0 THEN
     TRANTO PAROUT = 1 BY 2 * LAMOUT;

          (* PARTITION EXHAUSTION FAILURES *)

     IF NGGYRO < 7 OR NGACC < 7 THEN
          IF NCOM = 4 THEN
               IF NGGYRO = 6 AND NGACC > 6 TRANTO PAROUT = 2
               BY 3*LAMOUT/7;
               IF NGGYRO = 6 AND NGACC = 6 TRANTO PAROUT = 2
               BY 75*LAMOUT/98;
               IF NGGYRO = 6 AND NGACC = 5 TRANTO PAROUT = 2
               BY 17*LAMOUT/14;
               IF NGGYRO = 5 AND NGACC > 6 TRANTO PAROUT = 2
               BY LAMOUT;
               IF NGGYRO = 5 AND NGACC = 6 TRANTO PAROUT = 2
               BY 17*LAMOUT/14;
               IF NGGYRO = 5 AND NGACC = 5 TRANTO PAROUT = 2
               BY 3*LAMOUT/2;
               IF NGACC = 6 AND NGGYRO > 6 TRANTO PAROUT = 2
               BY 3*LAMOUT/7;
               IF NGACC = 5 AND NGGYRO > 6 TRANTO PAROUT = 2
               BY LAMOUT;
          ENDIF;

          IF NCOM = 3
               TRANTO PAROUT = 2 BY LAMOUT;

     ENDIF;

ENDIF;

          (*** PARTITION RECOVERY AND SIMULTANEOUS FAILURES ***)

IF PAROUT > 0 THEN
     TRANTO PAROUT = PAROUT-1 BY < RECMEAN, RECSTD >;
     TRANTO NFGYRO = NFGYRO + 1 BY (NGGYRO*LAMG)/2;
     TRANTO NFACC = NFACC + 1 BY (NGACC*LAMA)/2;
ENDIF;

          (*** NODE FAILURE TRANSITIONS ***)

IF NCOM > 0 AND NFGYRO = 0 AND NFACC = 0 AND PAROUT = 0 THEN
     TRANTO NGGYRO=NGGYRO-2, NGACC=NGACC-2, NCOM=NCOM-1
     BY NGGYRO*(NGGYRO-1)*NGACC*(NGACC-1)*LAMC/
     (4*(2*NCOM-1)*NCOM*(2*NCOM-1));

     IF 2*NCOM-NGGYRO > 0 THEN
          TRANTO NGGYRO=NGGYRO-1, NGACC=NGACC-2, NCOM=NCOM-1
          BY 2*NGGYRO*(2*NCOM-NGGYRO)*NGACC*(NGACC-1)*LAMC/
          (4*(2*NCOM-1)*NCOM*(2*NCOM-1));

          IF 2*NCOM-NGGYRO > 1 THEN
               TRANTO NGACC=NGACC-2, NCOM=NCOM-1 BY (2*NCOM-1-NGGYRO)*
               (2*NCOM-NGGYRO)*  NGACC*(NGACC-1)*LAMC/
               (4*(2*NCOM-1)*NCOM*(2*NCOM-1));

               IF 2*NCOM-NGACC > 0 THEN
```

```
                TRANTO NGACC=NGACC-1, NCOM=NCOM-1
                BY (2*NCOM-1-NGGYRO)*
                (2*NCOM-NGGYRO)* 2*NGACC*(2*NCOM-NGACC)*LAMC
                / (4*(2*NCOM-1)*NCOM*(2*NCOM-1));

                IF 2*NCOM-NGACC > 1 THEN
                     TRANTO NCOM=NCOM-1 BY (2*NCOM-1-NGGYRO)*
                     (2*NCOM-NGGYRO)*
                     (2*NCOM-1-NGACC)*(2*NGCOM-NGACC)*LAMC
                     / (4*(2*NCOM-1)*NCOM*(2*NCOM-1));
                ENDIF;

           ENDIF;

      ENDIF;

      IF 2*NCOM-NGACC > 0 THEN
           TRANTO NGGYRO=NGGYRO-1, NGACC=NGACC-1, NCOM=NCOM-1 BY
           2*NGGYRO* (2*NCOM-NGGYRO)* 2*NGACC*(2*NCOM-NGACC)*
           LAMC/ (4*(2*NCOM-1)*NCOM*(2*NCOM-1));

           IF 2*NCOM-NGACC > 1 THEN
                TRANTO NGGYRO=NGGYRO-1, NCOM=NCOM-1 BY
                2*NGGYRO* (2*NCOM-NGGYRO)*
                (2*NCOM-1-NGACC)*(2*NCOM-NGACC)*LAMC/
                (4*(2*NCOM-1)*NCOM*(2*NCOM-1));

           ENDIF;

      ENDIF;

 ENDIF;

 IF 2*NCOM-NGACC > 0 THEN
      TRANTO NGGYRO=NGGYRO-2, NGACC=NGACC-1, NCOM=NCOM-1 BY
      NGGYRO* (NGGYRO-1)* 2*NGACC*(2*NCOM-NGACC)*LAMC/
      (4*(2*NCOM-1)*NCOM*(2*NCOM-1));

      IF 2*NCOM-NGACC > 1 THEN
           TRANTO NGGYRO=NGGYRO-2, NCOM=NCOM-1 BY
           NGGYRO*(NGGYRO-1)*  (2*NCOM-1-NGACC)*(2*NCOM-NGACC)*
           LAMC/(4*(2*NCOM-1)*NCOM*(2*NCOM-1));

      ENDIF;

 ENDIF;

ENDIF;
```

Standard Bibliographic Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-181645		

4. Title and Subtitle	5. Report Date
Semi-Markov Adjunction to the CAME Program	April 1988
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Gene Rosch, Monica A. Hutchins, Frank J. Leong, and Philip S. Babcock IV	

9. Performing Organization Name and Address	10. Work Unit No.
The Charles Stark Draper Laboratory, Inc. 555 Technology Square Cambridge, MA 02139	505-66-71-02
	11. Contract or Grant No. NAS9-17560
	13. Type of Report and Period Covered Contractor Report
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Langley Research Center Hampton, VA 23665-5225	14. Sponsoring Agency Code

15. Supplementary Notes

Langley Technical Monitor: Daniel L. Palumbo

This report was prepared for Langley under Task 87-50 of Johnson Space Center contract NAS9-17560.

16. Abstract

In recent years, the utilization of Markov (and semi-Markov) models has been established as a useful technique in analyzing the reliability of fault-tolerant systems. However, the process of constructing an appropriate model for a complex system can be an overwhelmingly difficult and tedious task. As an attempt to address this problem, the C. S. Draper Laboratory, Inc. is currently developing a computer-aided reliability analysis tool called the Computer-Aided Markov Evaluator (CAME) program. The goal of the tool is to automatically create an appropriate fault-occurrence model from a top-down system description utilizing a set of rules that reflect Markov modeling techniques. As the objective of this project, the rule-based CAME program is expanded in its ability to incorporate the effect of fault-handling processes into the construction of a reliability model.

This new capability is added to the CAME program by modeling the fault-handling processes as semi-Markov events. When this new capability is invoked, the CAME program constructs an appropriate semi-Markov model. To solve a constructed semi-Markov model, the CAME program outputs it in a form which can be directly solved with the Semi-Markov Unreliability Range Evaluator (SURE) program.

As a means of evaluating the alterations made to the CAME program to support the construction of semi-Markov models and of evaluating the CAME program itself, the CAME program is utilized to model the reliability of portions of the Integrated Airframe/Propulsion Control System Architecture (IAPSA II) reference configuration being developed under a NASA contract. The reliability predictions are compared with a previous analysis done by the Boeing Military Aircraft Company (BMAC). The results bear out the feasibility of utilizing the CAME program to generate appropriate semi-Markov models to model fault-handling processes.

17. Key Words Suggested by Author	18. Distribution Statement
Reliability Analysis, Markov Models, Semi-Markov Models, Automated Model Construction	Unclassified-Unlimited Subject Category 38

19 Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	66	A04

End of Document