View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by NASA Technical Reports Server

NASA Technical Memorandum 102861

An Integrated Approach to System Design, Reliability, and Diagnosis

F. A. Patterson-Hine and David L. Iverson

(NASA-TM-102861) AN INTEGRATED APPROACH TO SYSTEM DESIGN, RELIABILITY, AND DIAGNOSIS (NASA) 17 p CSCL 13B

N91-15426

Unclas
G3/31 0329087

December 1990

National Aeronautics and
Space Administration

NASA Technical Memorandum 102861

An Integrated Approach to System Design, Reliability, and Diagnosis

F. A. Patterson-Hine and David L. Iverson, Ames Research Center, Moffett Field, California

December 1990

NASA
National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035-1000

SUMMARY

The requirement for ultradependability of computer systems in future avionics and space applications necessitates a top-down, integrated systems engineering approach for design, implementation, testing, and operation. The functional analyses of hardware and software systems must be combined by models that are flexible enough to represent their interactions and behavior. The information contained in these models must be accessible throughout all phases of the system life cycle in order to maintain consistency and accuracy in design and operational decisions. One approach being taken by researchers at Ames Research Center is the creation of an object-oriented environment that integrates information about system components required in the reliability evaluation with behavioral information useful for diagnostic algorithms. Procedures have been developed at Ames that perform reliability evaluations during design and failure diagnoses during system operation. These procedures utilize information from a central source, structured as object-oriented fault trees. Fault trees were selected because they are a flexible model widely used in aerospace applications and because they give a concise, structured representation of system behavior. The utility of this integrated environment for aerospace applications in light of our experiences during its development and use is described. The techniques for reliability evaluation and failure diagnosis are discussed, and current extensions of the environment and areas requiring further development are summarized.

INTRODUCTION

The presentation of information about the design and operation of complex systems is a central issue in the development of information systems that support all phases of a system's life cycle. Engineers use various models of the system's configuration and behavior as they move through requirements specification, design, manufacturing, assembly, and integration, and on to operation of the system. Some models support the analysis of subsystems, whereas others facilitate the assessment of system-level interactions and interfaces. Typically, data obtained during early phases of the life cycle are needed in a different format as input to other models later in the life cycle. A flexible, adaptable information representation can ease the transitions between the various models.

Computer-based information systems are being developed at NASA that focus initially on design automation as a foundation for the development of design knowledge capture systems (ref. 1). Design automation can be achieved by integrating currently available software packages such as computer-aided design (CAD) tools, data base packages, and computer-aided manufacturing (CAM) tools. Many of these products are implemented in an object-oriented environment because that environment promotes the development of software systems that are modular, extendable, flexible, and concise.

These design automation products can capture the substance of a complex system; however, further development of integrated tool sets is required to capture how and why a design satisfies functional requirements. Results of engineering analyses such as reliability assessment and design studies of fault detection, isolation, and recovery (FDIR) systems characterize a subset of these requirements. Two tools, OBREL (ref. 2) and the Fault Tree Diagnosis System (FTDS) (refs. 3,4) perform such analyses. These tools access a central, object-oriented knowledge base, and graphic interfaces enable engineers to interactively modify their designs and get immediate feedback on the effects of the changes. Reliability models are useful throughout the lifetime of the system to identify critical components and functions, but are often not updated as the system is modified. When the tools access a central knowledge base, the latest information is automatically used in each analysis. For instance, modifications made as a result of a reliability study are immediately incorporated in the FDIR system. The tools are developed for use on workstations which are commonly found on engineers' desks, where they can be used just as a hand calculator would be.

OBREL and FTDS are composed of algorithms for system reliability evaluation and fault diagnosis that are based on an object-oriented fault tree representation. Fault trees are powerful tools for characterizing the behavior of complex systems. They are straightforward, concise models of the relationships between system components. Fault trees, although traditionally used as tools in design and safety assurance, are also used for fault diagnosis (refs. 5-9). Several of these diagnostic applications pre-process the fault tree to reduce it to a more manageable form. In other cases, the fault trees are used to aid knowledge acquisition for rule-based fault diagnostic systems. Our system enhances these implementations by replacing the knowledge base of if-then rules with the object-oriented fault tree representation.

The integrated reliability analysis and fault diagnosis environment is described here, first by presenting the object-oriented fault tree representation and reliability evaluation procedures, and then by incorporating additional information required for fault diagnosis and describing the diagnostic routines. Final sections discuss future work in each of these areas and summarize our experiences with this system.

The graphical interfaces described herein were developed by Jack Liao of Ames Research Center.

RELIABILITY CALCULATIONS

A fault tree is a graphical representation of the logical relationships between the components in a system. The structure of the tree describes how the system can reach an undesired state. The tree is constructed by defining an undesired state and the events that would have to occur to put the system in that state. Primary or basic events in the tree represent fault events such as component failures. Logic gates are used to represent combinations of events that would cause the next-level event to occur. Logical AND gates indicate that output from an event occurs only if all of the input events occur; OR gates indicate that output occurs if at least one of the input events occurs. An example fault tree is shown in figure 1.

The object-oriented paradigm is well suited for representing the hierarchical structure of a fault tree. Fault tree evaluation can produce quantitative results including probabilities of the occurrence of system failures and quantitative rankings of component contributions to system failure. Most techniques for evaluating fault trees underutilize the information about system structure and are, therefore, inefficient and needlessly complex. Object-oriented representations provide the flexibility that is needed in defining structures to represent fault tree logic gates and basic event data, and the complex interrelationships that exist between them. An object-oriented representation of the fault tree enables access of tree structural information by evaluation algorithms, greatly reducing the time required for solution. These fault tree models are also easily updated to reflect system composition or configuration modifications, and can be reevaluated quickly after updates by using recursive tree evaluation procedures.

Fault Tree Object Descriptions

An example fault tree object, corresponding to the top event of the fault tree in figure 1, is shown in figure 2. The object contains slots for information such as the name of the event, the type of event, event inputs and outputs (children and parent(s), respectively), indications of the locations of repeated tree events, and the probability of occurrence for each event. Alternative solution algorithms may require different parameters, but the object-oriented descriptions are easy to modify and expand.

The fault tree objects are created using a graphical tree editor. This allows the object definitions to be transparent to the user so that a thorough understanding of the principles of object-oriented programming is not necessary. The system designer can choose between logic gate and basic event definition by clicking on the appropriate icon displayed beside the editor window. The parent–child links are stored in each event object automatically by using a mouse to graphically connect events as a tree. Other parameters are defined and entered via pop-up windows.

Fault Tree Evaluation

OBREL performs a quantitative evaluation of the fault tree using a combination of standard fault tree reduction techniques modified for the object-oriented implementation. A bottom-up procedure is used for subtrees that do not contain repeated events, and a recursive, top-down algorithm is used for subtrees that contain repeated events. The major difficulty in solving fault trees quantitatively involves events that are repeated in more than one location in the tree. In those cases, the tree as a whole is not statistically independent, and special algorithms are required for the solution. Conventional fault tree codes apply these algorithms, which usually have much larger computer resource requirements than solution algorithms for statistically independent trees, to the entire tree. Modularization procedures can be used to locate and simplify statistically independent subtrees prior to tree evaluation to reduce the tree to a simpler form; however, conventional procedures only simplify subtrees that are statistically independent of every other event in the tree. Modularization is more effective when it can be tailored to fit the specific algorithm. For instance, in the case of the top-down recursive algorithm utilized in this implementation, subtrees need only be statistically independent of a specific subset of tree events, not of the entire tree. Using this algorithm, the tree is evaluated by recursively simplifying sets of events, and the independence of an event is analyzed with respect to the set of events being simplified at that point in the evaluation. The tree is effectively modularized dynamically at each point in the evaluation.

The locations of repeated events are propagated up the tree and stored in each event object. This information is used by the modularization procedure; it is also used to determine which evaluation procedure is required for the particular event. Unlike many conventional fault tree evaluation codes, which calculate the probability of occurrence of the top event only, OBREL produces results for every event in the tree. The intermediate results are stored in each event object as they are calculated and, along with the specialized modularization procedure, are used to improve the performance of the top-down algorithm. The details of this approach are presented in reference 2 and will not be repeated here, but the following example will illustrate key features of the evaluation process.

Example 1

The fault tree shown in figure 1 is a moderate-size tree with several repeated basic events and a repeated subtree. Events 28 and 19 appear more than once in the tree, and the entire subtree with OR-GATE 20 as the top node appears below both AND-GATE 2 and AND-GATE 3. The triangular symbol shown in the diagram is a transfer gate, in fault tree terminology, and is used to indicate a continuation in the tree structure. The repeated subtree is not statistically independent of the rest of the tree structure because event 19 appears outside the subtree structure as well. The combination of algorithms utilized by OBREL presents a unique solution process, combining the efficient bottom-up solution for most of the tree with the top-down solution that is required for the evaluation of only a few of the events.

The modularization procedure propagates information about the location of events 19, 20, and 28 throughout the tree objects. As an example, NODE 28 is stored in the :dependent slot of OR-GATE 34, OR-GATE 32, OR-GATE 30, OR-GATE 26, OR-GATE 25 (from the second occurrence), and finally AND-GATE 24. The :dependent-eval slot of AND-GATE 24 contains NODE 28, indicating that the top-down algorithm must be used to evaluate this gate because of the multiple occurrences of NODE 28 in its structure. As the repeated event information is propagated through the rest of the tree, it becomes apparent that only one other gate, OR-GATE 1, requires the top-down solution. The number of recursive calls required to evaluate a fault tree is a performance indicator of the solution technique. The solution of this tree, including quantitative results for every event in the tree, requires 91 recursive calls and can be calculated in just a few seconds on a Macintosh II using Allegro Common LISP. The solution of the top event using a simple top-down procedure that does not utilize independence checks or intermediate result information (since there are no intermediate results calculated) requires over 580,000 recursive calls.

An additional benefit of using an object-oriented environment is the ability to modify the tree and to reuse intermediate results in the evaluation of the new tree from branches that are not affected by the modifications. For example, the unavailability of NODE 5 in the example tree was changed, and the tree was recalculated in only 41 recursive calls. The number of previous results that can be reused depends, of course, on the location of the modifications. In general this capability will result in significant savings during parameter variation studies, and it is not available in other commonly used fault tree evaluation codes. This capability greatly increases the productivity of design engineers in comparing the reliability characteristics of similar systems.

FAULT DIAGNOSIS

FTDS is based on the failure–cause identification process of the diagnostic system described by Narayanan and Viswanadham (ref. 9). Their system has been enhanced in the present implementation by replacing the knowledge base of if-then rules with an object-oriented fault tree representation. This allows the system to perform its task much faster and facilitates dynamic updating of the knowledge base in a changing diagnostic environment. Accessing the information contained in the objects is more efficient than performing a lookup operation on an indexed rule base. Additionally, the object-oriented fault trees can be easily updated to represent the current system status.

Rules As Objects

Narayanan and Viswanadham suggested that the rule base for the diagnostic system be constructed directly from a fault tree representation of the system to be diagnosed. In their system each fault tree gate is converted to an if-then rule. An AND gate becomes a rule with a conjunction of the child events of the gate as an antecedent and the output event of the gate as a consequent. An OR gate has a disjunction rather than a conjunction in the antecedent. The rules are stored as text in a data base and when the system needs a rule it must perform a rule base lookup by using a failure event as the lookup key. This involves a significant amount of processing overhead as the system performs data base access and pattern matching. FTDS reduces this overhead considerably by representing rules as objects.

FTDS objects currently contain the same information as the if-then rules used by Narayanan and Viswanadham, but can easily be expanded for additional capability. The event name stored in the object is the consequent of the rule. The rule antecedent is found by following the children pointers in the object to the antecedent events. New slots added to the fault tree objects to hold additional parameters that are needed by the diagnostic routine are described in the next section. In the current implementation of FTDS the objects are stored in a hash table and referenced by event name. This scheme allows very quick reference to a rule, given its consequent event, and easy retrieval of rules containing a given event in their antecedent. To find the rule object with failure event E as its consequent, all that is required is a single hash table lookup. To find rules with event E in their antecedent, the system only needs to look up the object for E and follow the pointers contained in that object's parent slot.

Augmented Object Descriptions

The information used by the diagnostic system that needs to be added to the fault tree objects includes contributory factors, or C-factors, and time intervals. The C-factor associated with a failure event in a fault tree is an heuristic measure of the likelihood that the occurrence of the parent fault of that event was caused by that event rather than by one of its siblings. The time interval of a failure event under an OR gate is an estimate of how much time will elapse from the moment that event occurs until its parent failure event occurs. In the case of a child event under an AND gate, the time interval is the time between the moment when all of the child events have occurred and the occurrence of the parent event. Narayanan and Viswanadham call the resulting model an augmented fault tree (ref. 9). Figure 3 illustrates the additional slots needed to represent this information in the fault tree objects.

This information can be entered into the environment using the graphical fault tree editor described above, which generates fault tree object descriptions.

Diagnosis Process

The diagnostic system is initially given information about the system being diagnosed in the form of normal and abnormal alarms (the nomenclature used by Narayanan and Viswanadham). Each possible alarm corresponds to a system failure event and is referred to by the name of that event. A normal alarm indicates that the failure event it is monitoring has not occurred, and an abnormal alarm indicates that the failure event has occurred. In addition, each abnormal alarm includes the suspected time at which the failure event occurred, and each normal alarm includes the latest time the specified event was known to have not occurred.

The diagnostic process is initiated by specifying the estimated time of occurrence of a failure, the current set of normal alarms and the time that each normal alarm was last confirmed, and a set of abnormal alarms with estimated failure times. The diagnosis begins by inferring the relevant failure events that must have occurred and those that could not have occurred based on the information in the normal and abnormal alarm sets. The alarm sets are updated accordingly. The system uses the alarm sets to guide its search of the diagnosis space. It does not consider those portions of the diagnosis space with diagnoses containing sets of basic failure events that would cause the occurrence of a failure in the normal alarms set. Also, those portions of the search space with diagnoses containing abnormal alarms are searched early in the diagnosis process. The system also checks possible diagnoses for temporal and causal consistency. The time-of-occurrence information provided for each alarm is used to propagate temporal constraints throughout the fault tree.

Using the abnormal alarm information, the system selects starting points for the diagnostic process, and builds constraint sets that help to narrow the diagnosis search space. After this information has been gathered, the system uses heuristically driven backward chaining to find a set of basic failure events that are a likely cause of the failure being diagnosed. More information about this approach can be found in the description of the failure–cause identification process in Narayanan and Viswanadham (ref. 9) and in the system description by Iverson and Patterson-Hine (refs. 3,4).

The diagnostic process can be examined using a graphical displayer which allows the user to enter alarm information interactively and then view the alarm-propagation and backward-chaining procedures that produce the likely cause of system failure. The effectiveness of the fault tree model of system behavior under various failure situations can be studied. Fault trees are developed in varying levels of detail (determined by the particular application), and the displayer can be used to locate events in the tree that need greater resolution. The effects of component dependencies can also be observed. The displayer could also be used to train operators to expect certain failure propagation paths, given certain classes of alarms, and to become familiar with fault isolation procedures.

Example 2

This example considers a representative subset of the Space Station Freedom Data Management System (DMS) which consists of three subsystems connected by a token ring network. In this system, shown in figure 4, subsystems A and C consist of three processors each and subsystem B consists of two processors. Each processor cluster is connected to the network through a pair of network interface units (NIUs). In order to simplify the fault tree model, several assumptions are made about the operating requirements of the system. First, all three subsystems must be operational as defined in the following three assumptions for the system to be considered operational: (1) all processors in each subsystem must be operational; (2) cluster A requires information periodically from cluster B, and cluster C must receive information from both clusters A and B; and (3) one of the two NIUs in each subsystem must be operational for a processor cluster to communicate with the other subsystems. Second, the token ring network has the capability to bypass a single faulty link between clusters, so segments of cable on both sides of a cluster must be faulty to prevent the cluster from communicating with the other clusters on the network.

A fault tree model of this system is shown in figure 5. Events that contribute directly to the top event, representing system failure, are inputs to an OR gate. These events include failure of subsystem A, failure of subsystem B, and failure of subsystem C. If any of those events occur, the system is considered failed. The failure of subsystem A can be attributed to failure of the processor cluster, failure of both of the NIUs, or failure of subsystem A to receive its required information from subsystem B. Subsystem B is considered failed if either processor in the cluster fails. Cluster B requires no communication from the other subsystems for successful operation, so failure of the network does not cause subsystem failure in this case. The failure of subsystem C can be attributed to failure of the processor cluster, failure of one of the NIUs, or failure of subsystem C to receive its required information from subsystems A and B. Failure of a subsystem to receive required information can be attributed to failure of any of the transmitting cluster's processors, failure of the two links in the network surrounding either the transmitting cluster or the receiving cluster, or failure of one of the transmitting cluster's NIUs.

Suppose that at time 10 the DMS system goes down. A record of sensor data provides the information that there was a failure in cluster C at time 8, and it was known that the cluster-C NIUs were functioning at time 8 and that cluster A was functioning at time 9.5. The initial normal alarms set is {(Cluster A, 9.5) (NIU Cluster C, 8)}, and the initial abnormal alarms set is {(Cluster C, 8)}. With this information the diagnostician reasons from the fault tree that the failure of cluster C was sufficient to cause the failure of the entire DMS system. This conclusion is reached by considering the object representing the DMS system failure. That object is an OR gate with a child pointer to the object representing the failure of cluster C. Since it is known that cluster C failed, it is assumed that its failure is the cause of the DMS failure. The accuracy of this assumption depends, of course, on the completeness of the fault tree in representing all possible causes of system failure. Any failures inferred in this way from information in the abnormal alarms set will be added to the abnormal alarms set along with their estimated failure times.

The diagnostic system then uses the information in the normal alarms set to determine which other failure events have not occurred. Since the cluster-C NIU system is represented by an AND-gate and the cluster-C NIUs were known to be functioning at time 8, it reasons that all the child events of the object representing that gate could not have occurred at a time before time 8 minus the error-propagation time recorded in the time-interval slot in the cluster-C object. In other words, at time 8 – t, where t is the error propagation time, the system knows that both NIU-C1 and NIU-C2 were working properly. Similar reasoning is done based on the fact that cluster A is in the normal alarms set. By backward chaining from this fact, the system infers that the cluster-A NIUs are functioning properly, that the processors in cluster A are all running correctly, and that the information from cluster B is reaching cluster A. This backward chaining continues until all relevant failure events in the fault tree that could not have occurred are recognized. These events are added to the normal alarms set, and the information in that set is used to guide the diagnosis. Notice that the information obtained in backward chaining from a normal alarm is not necessarily restricted to the branch of the fault tree under the original normal alarm. When a given failure event can contribute to the cause of more than one other failure event it will appear in several places in the fault tree. For instance, in this case it is determined that cluster B is operating correctly since cluster A is receiving information from it. This inference provides the additional information that the failure in cluster C was not caused by a failure to receive the needed information from cluster B, because cluster B was down. Such repeated events can help narrow the diagnosis search space considerably.

Now that the diagnostic system has determined a high-level cause of the DMS failure, as well as which failures definitely have and have not occurred, it goes on to find a set of basic events that were a likely cause of the top-level failure. In this case it starts by backward chaining from the cluster-C failure event. The first event it considers as a cause for the cluster-C failure is the failure of the cluster-C processors. This is because the cluster-C processors node has the highest C factor of all of cluster C's children that are not contained in the normal alarms set. Continuing the reasoning from there, the diagnostic system reaches the conclusion that at least one of the cluster-C processors must have failed sometime before time 6, and that this failure propagated through the system and caused the entire DMS system to fail at time 10.

FUTURE WORK

Current efforts are being focused on specific aspects of the evaluation of the reliability of integrated hardware and software systems and on more robust diagnostic procedures for real-time systems. In particular, techniques are being developed to model the functional interactions of hardware and software components in very large systems such as those on Space Station Freedom. Traditional approaches such as Markov chains, used to assess the reliability of safety-critical systems, are incapable of handling the size and complexity of these highly integrated designs, because the models grow quite large. A new hybrid modeling technique is being developed which augments conventional fault trees, themselves incapable of modeling dynamic behavior, with Markovian models that describe dynamic behavior quite naturally. Modularization procedures are being developed so that the models can be partitioned into smaller pieces that can be solved using conventional algorithms. Techniques for integrating models of software functionality are also being investigated. It is common practice to include events in a fault tree that represent modules of software critical to the operation of the system. If the modules of software are designed to be fault tolerant, utilizing recovery blocks or n-version programming, the fault tree representation can be used to depict the structure of the modules, but it is not suitable for quantitative evaluation (ref. 10). The presence of undetected failures and correlated faults must be taken into account, and the appropriate model in that case is the Markov model (refs. 10,11). Methods for including Markov models for the software components in the fault tree model of the overall system are also being studied.

A major limitation of diagnostic systems based solely on component connectivity and failure mode information is their inability to diagnose failures that the designers did not anticipate. Model-based diagnostic systems overcome this limitation by utilizing analytical models of the system, which can be used to simulate actual system behavior. As current data on system state is fed through the model, anomalous behavior is detected by comparing actual behavior trends with expected values. One drawback of these systems is the length of time it takes to run the simulations. Methods are being explored to integrate graph-based and model-based techniques so that the advantages of each can be exploited (graph models offer concise representations of system behavior in failure space, and model-based approaches are flexible and robust). The Thermal Expert System (TEXSYS) project at Ames integrates a range of techniques for real-time control and fault diagnosis of a thermal bus prototype for Space Station Freedom, demonstrating that integrated approaches may be more powerful than any single technique alone (ref. 12). Current studies include the design of interacting on-board and ground-based diagnostic systems as well as extensions to diagnostic algorithms for multiple-fault diagnosis.

CONCLUDING REMARKS

Two tools for engineering analyses of highly reliable systems, one for quantitative reliability evaluation and the other for fault diagnosis, have been developed based on an object-oriented representation of fault trees. The fault tree serves as a central knowledge base for the integrated tool set, ensuring that consistent design information is used in both procedures. The tools have a graphical interface for data entry and the display of results, thus enabling the engineer to modify system models easily and to understand the effects of the changes quickly. The availability of the models in an accessible form improves the design process by eliminating redundant model development in various stages of the life cycle. The object-oriented models are particularly useful since they are easily modified to characterize various aspects of system behavior, thereby promoting the development of additional analysis tools that will access the same knowledge base.

REFERENCES

1. Erickson, Jon D.; Crouse, Kenneth H.; Wechsler, Donald B.; and Flaherty, Douglas R.: Considerations for a Design and Operations Knowledge Support System for Space Station Freedom. NASA TM-102156, 1989.

2. Patterson-Hine, F. A.; and Koen, B. V.: Direct Evaluation of Fault Trees Using Object-Oriented Programming Techniques. IEEE Trans. Reliability, vol. 38, no. 2, June 1989, pp. 186-192.

3. Iverson, David L.; and Patterson-Hine, F. A.: A Diagnosis System Using Object-Oriented Fault Tree Models. Fifth Conference on Artificial Intelligence for Space Applications, Huntsville, AL, May 1990.

4. Iverson, David L.; and Patterson-Hine, F. A.: Object-Oriented Fault Tree Models Applied to System Diagnosis. Applications of Artificial Intelligence VIII, Orlando, FL, Apr. 1990.

5. Rodriguez, G.; and Rivera, P.: A Practical Approach to Expert Systems for Safety and Diagnostics. In Tech, vol. 33, no. 7, 1986, pp. 53-57.

6. Pipitone, F.: The FIS Electronics Troubleshooting System. Computer, vol. 19, no. 7, 1986, pp. 68-76.

7. Weisbin, C. R.; de Saussure, G.; Barhen, J.; Oblow, E. M.; and White, J. C.: Minimal Cut-Set Methodology for Artificial Intelligence Applications. IEEE 1st Conference on Artificial Intelligence Applications, 1984.

8. Schwarzblat, M.; and Arellano, J.: An Expert Diagnostic and Prediction System Based on Minimal Cut Sets Techniques. IEEE 2nd Conference on Artificial Intelligence Applications, 1985.

9. Narayanan, N. H.; and Viswanadham, N.: A Methodology for Knowledge Acquisition and Reasoning in Failure Analysis of Systems. IEEE Trans. Systems, Man, Cybernetics, vol. SMC-17, Mar./Apr. 1987, pp. 274-288.

10. Hecht, Herbert; and Hecht, Myron: Fault-Tolerant Software. Fault Tolerant Computing: Theory and Techniques. Vol. 11. D. K. Pradham, ed., Prentice-Hall, 1986.

11. Arlat, Jean; Kanoun, Karama; and Laprie, Jean-Claude: Dependability Modeling and Evaluation of Software Fault-Tolerant Systems. IEEE Trans. Computers, Apr. 1990, pp. 504-513.

12. Glass, B. J.: A Model-Based Approach to the Symbolic Control of Space Subsystems. AIAA Paper 90-3430, Portland, OR, 1990.

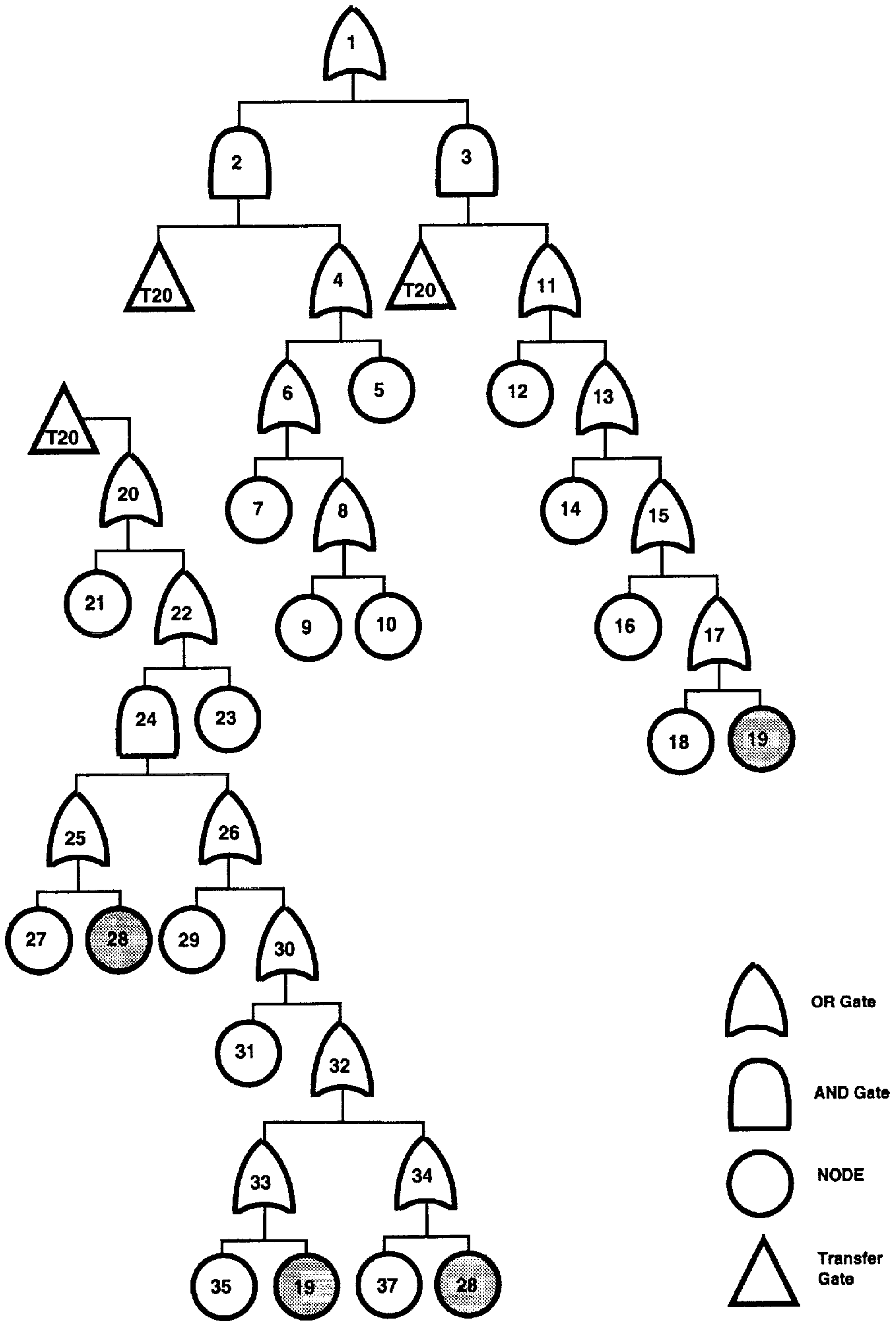

Figure 1. Sample fault tree.

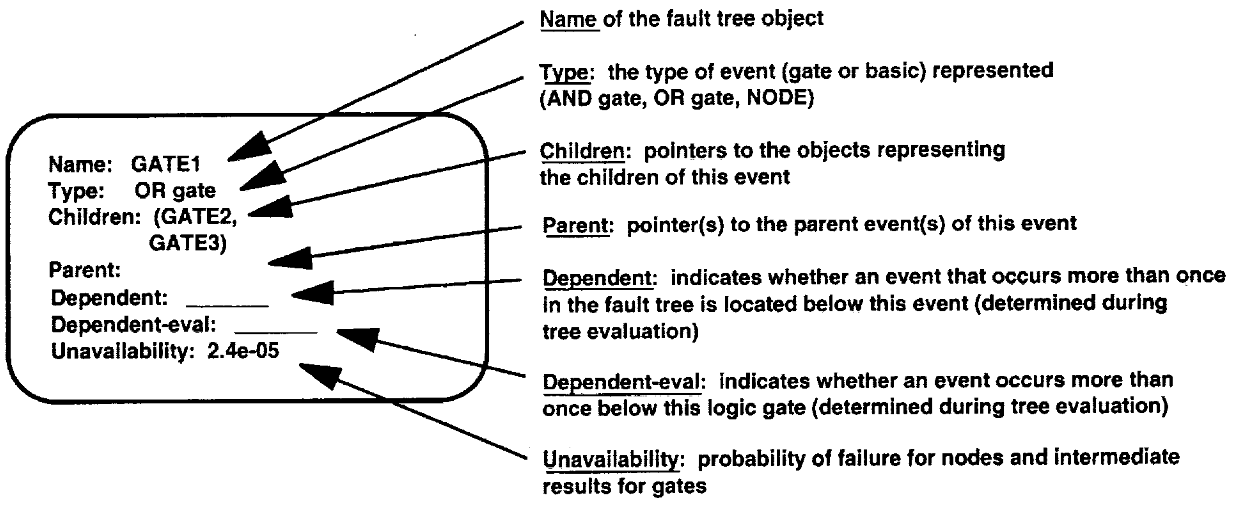
Name: GATE1
Type: OR gate
Children: (GATE2, GATE3)
Parent:
Dependent:
Dependent-eval:
Unavailability: 2.4e-05
Name of the fault tree object
Type: the type of event (gate or basic) represented (AND gate, OR gate, NODE)
Children: pointers to the objects representing the children of this event
Parent: pointer(s) to the parent event(s) of this event
Dependent: indicates whether an event that occurs more than once in the fault tree is located below this event (determined during tree evaluation)
Dependent-eval: indicates whether an event occurs more than once below this logic gate (determined during tree evaluation)
Unavailability: probability of failure for nodes and intermediate results for gates

Figure 2. Example fault tree object.

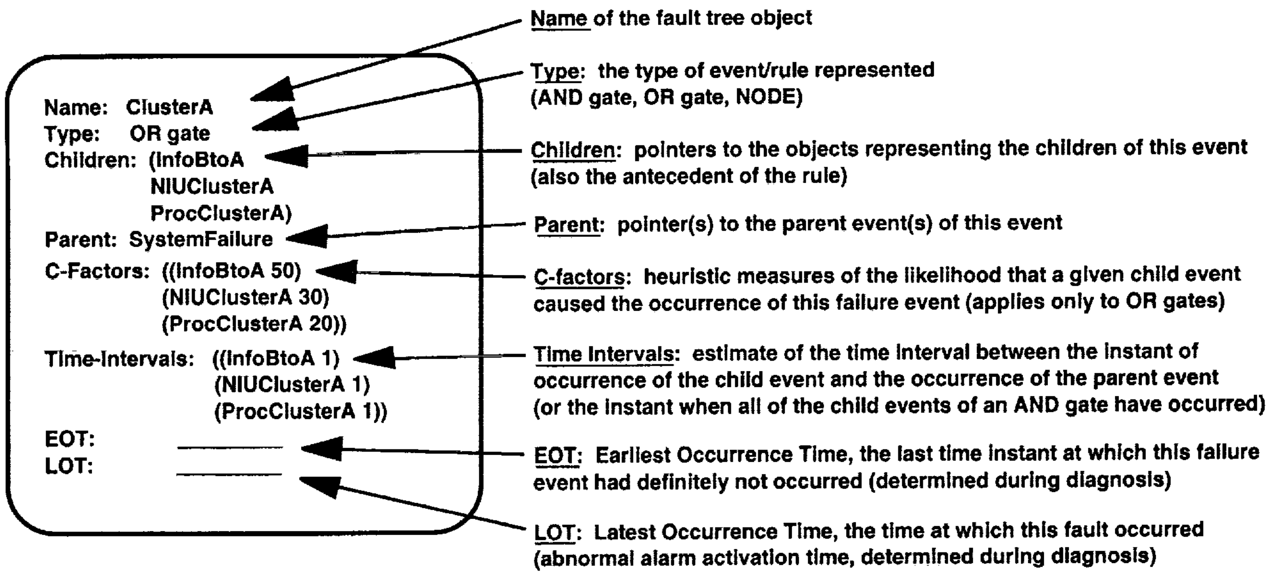
Name: ClusterA
Type: OR gate
Children: (InfoBtoA NIUClusterA ProcClusterA)
Parent: SystemFailure
C-Factors: ((InfoBtoA 50) (NIUClusterA 30) (ProcClusterA 20))
Time-Intervals: ((InfoBtoA 1) (NIUClusterA 1) (ProcClusterA 1))
EOT:
LOT:
Name of the fault tree object
Type: the type of event/rule represented (AND gate, OR gate, NODE)
Children: pointers to the objects representing the children of this event (also the antecedent of the rule)
Parent: pointer(s) to the parent event(s) of this event
C-factors: heuristic measures of the likelihood that a given child event caused the occurrence of this failure event (applies only to OR gates)
Time Intervals: estimate of the time interval between the instant of occurrence of the child event and the occurrence of the parent event (or the instant when all of the child events of an AND gate have occurred)
EOT: Earliest Occurrence Time, the last time instant at which this failure event had definitely not occurred (determined during diagnosis)
LOT: Latest Occurrence Time, the time at which this fault occurred (abnormal alarm activation time, determined during diagnosis)

Figure 3. Augmented fault tree object.

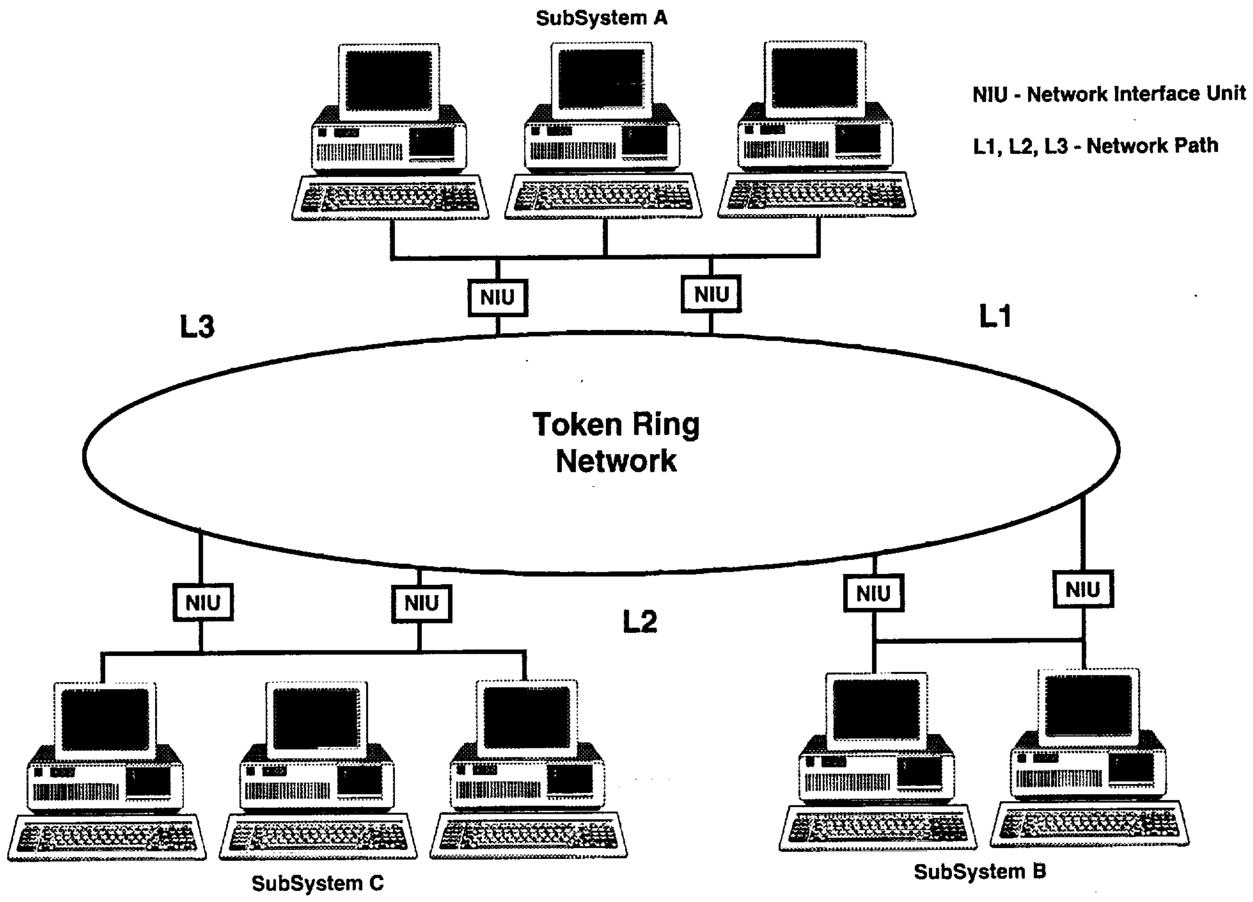

Figure 4. Sample DMS.

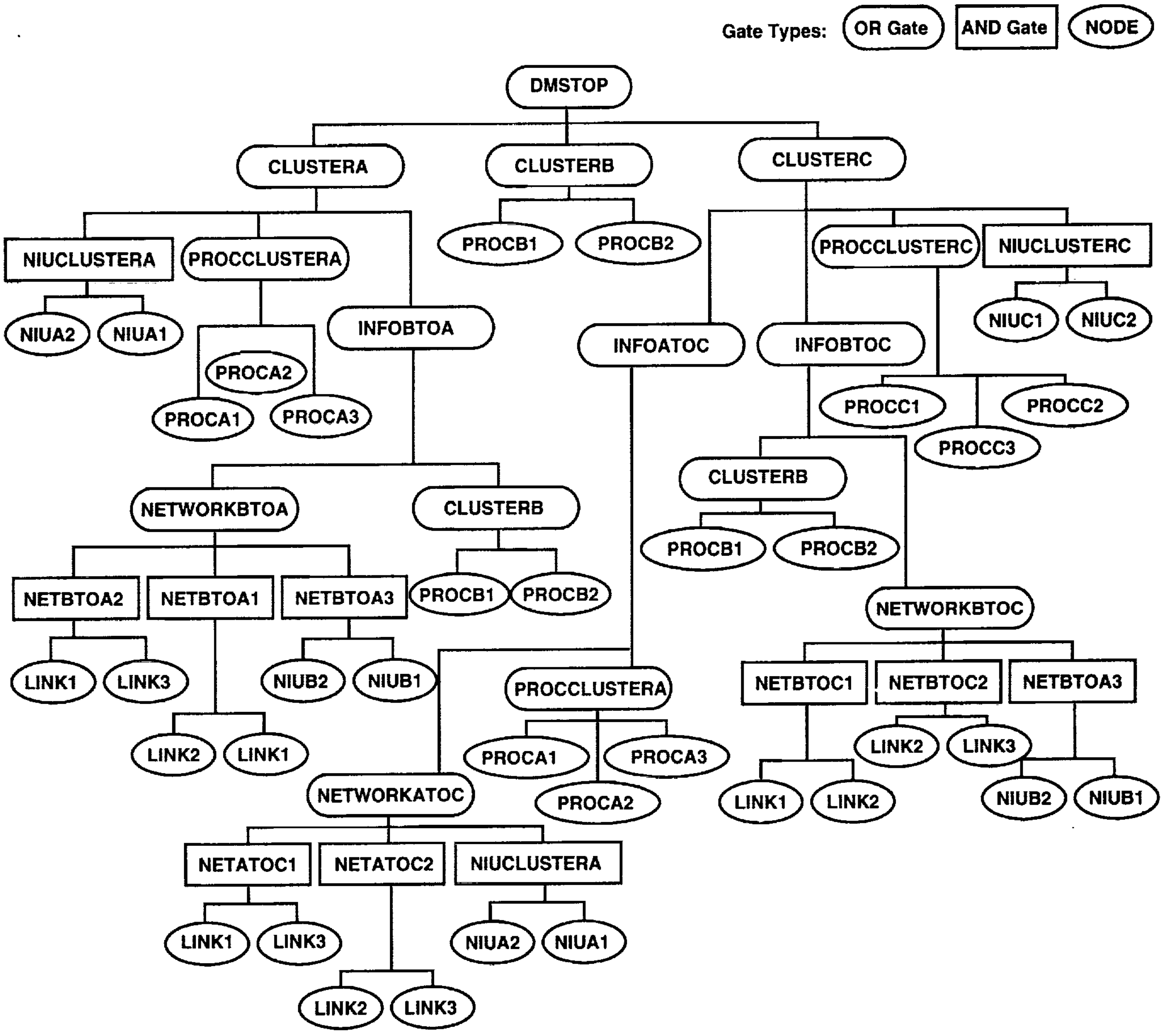

Figure 5. DMS fault tree.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No. NASA TM-102861	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle An Integrated Approach to System Design, Reliability, and Diagnosis		5. Report Date December 1990
		6. Performing Organization Code
7. Author(s) F. A. Patterson-Hine and David L. Iverson		8. Performing Organization Report No. A-90272
		10. Work Unit No. 476-84-03
9. Performing Organization Name and Address Ames Research Center Moffett Field, CA 94035-1000		11. Contract or Grant No.
		13. Type of Report and Period Covered Technical Memorandum
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546-0001		14. Sponsoring Agency Code

15. Supplementary Notes

Point of Contact: F. A. Patterson-Hine, Ames Research Center, MS 244-4,
Moffett Field, CA 94035-1000
(415) 604-4178 or FTS 464-4178

16. Abstract

The requirement for ultradependability of computer systems in future avionics and space applications necessitates a top-down, integrated systems engineering approach for design, implementation, testing, and operation. The functional analyses of hardware and software systems must be combined by models that are flexible enough to represent their interactions and behavior. The information contained in these models must be accessible throughout all phases of the system life cycle in order to maintain consistency and accuracy in design and operational decisions. One approach being taken by researchers at Ames Research Center is the creation of an object-oriented environment that integrates information about system components required in the reliability evaluation with behavioral information useful for diagnostic algorithms. Procedures have been developed at Ames that perform reliability evaluations during design and failure diagnoses during system operation. These procedures utilize information from a central source, structured as object-oriented fault trees. Fault trees were selected because they are a flexible model widely used in aerospace applications and because they give a concise, structured representation of system behavior. The utility of this integrated environment for aerospace applications in light of our experiences during its development and use is described. The techniques for reliability evaluation and failure diagnosis are discussed, and current extensions of the environment and areas requiring further development are summarized.

17. Key Words (Suggested by Author(s)) Fault tree, Systems engineering, Fault diagnosis, Object-oriented programming, System reliability	18. Distribution Statement Unclassified-Unlimited Subject Category - 31

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 16	22. Price A02

NASA FORM 1626 OCT 86

For sale by the National Technical Information Service, Springfield, Virginia 22161