NASA Technical Memorandum 103953

Fault Management for Data Systems

Mark A. Boyd, David L. Iverson, and
F. Ann Patterson-Hine

September 1993

National Aeronautics and
Space Administration

NASA Technical Memorandum 103953

Fault Management for Data Systems

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

September 1993

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035-1000

SUMMARY

We consider issues related to automating the process of fault-management (fault diagnosis and response) for data management systems. Substantial benefits are to be gained by successful automation of this process, particularly for large complex systems. We advocate the use of graph-based models to develop a computer-assisted fault-management system. We begin by describing the general problem and outlining the motivation behind choosing graph-based models over other approaches for developing fault-diagnosis computer programs. We review some existing work in the area of graph-based fault-diagnosis, and offer a new fault-management method which we have developed from existing methods. We apply our method to an automatic telescope system intended as a prototype for future lunar telescope programs. Finally, we describe the application of our method to general data-management systems.

1 INTRODUCTION

Fault management is an important issue that must be addressed in the design and operation of large data systems. Typically a complex system is overseen by one or more human operators who are responsible for the correct operation of the system. Human operators are responsible for diagnosing any failures within the system and for formulating courses of action in response to the failures. The system often exhibits a set of symptoms that characterize a specific failure. From these symptoms the human operators must infer causes of the failure, consequences to the system, and what operational adjustment is necessary to compensate for the inferred failures. As stated, this process relies heavily upon human expertise and can be prone to errors in judgment. Systems that exceed a certain size and complexity invariably exceed the capacity of human diagnosis and fault management. Therefore, a substantial benefit may be obtained by automating the process. Such automation could be achieved by developing a computer program capable of analyzing sets of symptoms exhibited by a system. The computer program would be capable of inferring specific component failures most likely to be causes of the symptoms, and optionally providing recommendations to human operators for effective courses of action. The corrective recommendations could range from performing diagnostics and gathering further failure information, to suggesting repair, reconfiguration, or altering operating procedures to compensate for diagnosed failure. If the diagnostic program is integrated with a hardware system, the program can implement any of the courses of action discussed and would not require human intervention. Such a system would constitute a fully automated recovery system, which could significantly improve reliability and performance.

There are several issues involved in the design and development of a fault management computer advisor program, such as that described above. The program should be able to accommodate large, complex systems. The program should be reasonably easy for non-computer-specialists to use. Results of the analysis performed by the program must be provided for display in a manner understandable to the human operator. Ideally, the display should operate in an intuitive manner, otherwise human operators are likely distrust the computer's diagnosis or recommendations. Since the intended subject of the analysis is a data system in operation, the analysis must be capable of being completed and available for display in a timely manner because human operators will need to respond quickly to whatever failures have occurred. A major issue to consider is the type of system model the advisor program is to use for the analysis process. The type of model can have a major impact on the resources required for the design and implementation of both the advisor program and the model itself. It can also have a major

effect on how well the program and model address and resolve the other issues mentioned above. We discuss some options for underlying system models in the next section.

2 MODEL TECHNIQUES FOR FAULT MANAGEMENT

There are several different approaches available for addressing the above fault management problem. An obvious possibility is the use of a traditional rule-based expert system. It has long been established that such systems are well-suited to the type of inferential reasoning required by a computer-based fault management advisor. However, rule-based expert systems have a number of drawbacks. One of the most important of these is the fact that these systems tend to be large, complex, and difficult to maintain. In addition, the rule base can be difficult to construct and debug, often requiring the services of artificial intelligence (AI) specialists to initially develop and sometimes to maintain a system. Another factor specific to this particular application is the potential for integration of inferential diagnosis reasoning with other system analysis measures, such as reliability. There is no natural method for integrating reliability calculation with inferential reasoning in rule-based expert systems.

The Dependable Multiprocessing group in the Computational Systems Research Branch, Information Sciences Division at NASA Ames Research Center has been working with an alternate approach to developing a fault management advisor. Specifically, we have been investigating the use of *digraphs* and *AND-OR fault trees* to build models of the target system. The resulting analytical models can then be analyzed to produce the desired inferential reasoning to map a set of symptoms into a set of candidate faults and to evaluate the effect of both the faults and any proposed workaround procedures on the system as a whole. Both digraphs and fault trees are *static failure space models* and do not involve simulation. They model the system by depicting the effect of faults and failures on the system. They both belong to a class of models called *combinatorial analytic models*, which express system behavior in terms of the ways that individual component/subsystem failures combine together to affect the overall system. We offer a brief description of both digraphs and AND-OR fault trees in the paragraphs below.

Digraphs

The term digraph stands for *directed graph*. A digraph consists of a number of nodes (depicted by circles) connected by a number of directed arcs. A node usually represents the occurrence of a particular event, such as the failure of an individual component or subsystem. The directed arcs depict how the effects of failures/events propagate through the system. In addition to nodes and directed arcs, digraphs may contain AND gates. An AND gate is represented by a bar, a number of directed arcs terminating on one side of the bar that represent input events to the gate, and a single directed arc emanating from the opposite side of the bar which represents the output event of the gate. The output event occurs only if all of the input events occur. Digraph models usually follow the schematic of a system fairly closely since the effect of failures often propagate along the physical/electrical connections between individual system components, and the schematic is intended to depict precisely these physical/electrical connections. This close structural similarity of the model with the system means that the model is usually much easier to construct and verify than a set of abstract inference rules, such as is required by the rule-based expert systems mentioned above.

The failure of a component or occurrence of an event is represented in the model by the highlighting or *marking* of the digraph node that represents that component/event. The propagation of the effect of the component failure is represented by the subsequent marking of all nodes to which a directed arc exists from the marked node. The overall propagation throughout the system of the component failure is achieved by recursively applying this marking procedure to the newly marked nodes which were directly affected by the initially marked node. The justification for this procedure lies in the assumption that a failed component can, and usually does, produce erroneous output, and that the components that depend on the failed component's output for their input will subsequently produce erroneous output themselves due to the erroneous nature of their input. In this way, the effect of a specific component failure/event on other system components and ultimately on the system as a whole may be traced in the model.

Similarly, the model may be used to identify all the singleton and sets of combinations of nodes whose failure may lead to the failure of a specific "target" node in the digraph. A digraph model of a system can give a designer, engineer, or diagnostician a pictorial view of how individual or combinations of failures can propagate through the system and affect other components in the system. The model in which the propagation is displayed is likely to be similar to structural diagrams of the system, and this can help the human observer assimilate the effect of individual failures more easily than is possible with other modeling techniques. This type of intuitive presentation of the information is important for facilitating understanding of the problem and its consequences by the human operator who ultimately will be responsible for deciding which actions will address failures that have occurred.

Figure 1 shows an example system and a digraph model for it. The system consists of a spaceborne processor which performs monitoring of the spacecraft's operational status and reports its findings to a ground control station on earth. Within the spacecraft, the data are sent over a bus to a pair of redundant transmitters that both transmit to the ground receiver. The ground control station can receive the data successfully if either one of the transmitters is operational. The bottom portion of figure 1 shows a digraph model of the system. The node labeled "ground receiver" represents the event where

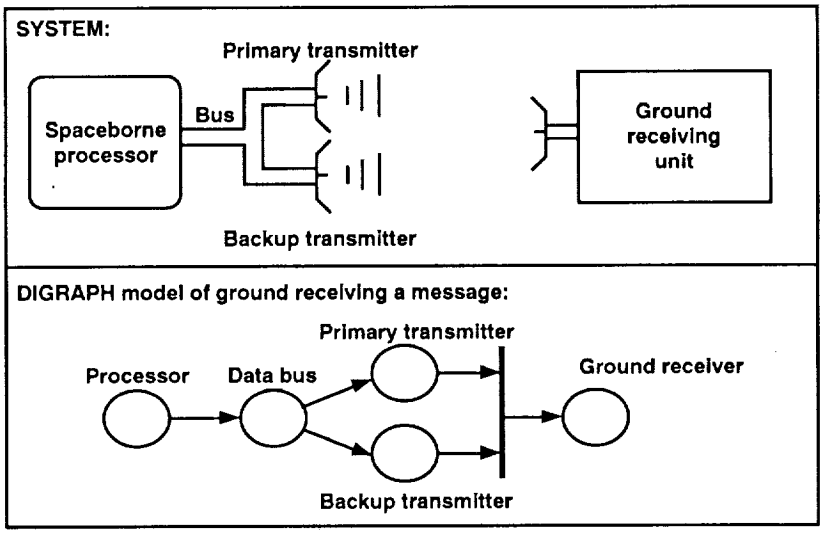

Figure 1. Schematic diagram and digraph model for an example system.

the ground control station is unable to receive the spacecraft's data successfully. The digraph indicates that a failure of the processor could prevent a correct transfer of data over the bus from being initiated. Either a data transfer failure or a bus failure could cause the data to be incorrectly received from the bus by either of the two transmitters. Either a data reception failure by the transmitter, or a failure within the transmitter itself could cause the transmitter to incorrectly send the data to the ground receiver. A failure to correctly receive the data might be caused either by both transmitters being unable to transmit the data or by the failure of the ground receiver itself. Thus a failure in any of the nodes of the digraph (indicated by marking the node) can and probably will propagate through the system as indicated by the directed arcs through the digraph.

Fault Trees

Fault trees are another type of graph-based model of system behavior. A fault tree model of a system consists of a number of basic events depicted by circles, a number of intermediate events depicted by boxes, a number of logic gates that express various combinations of basic and intermediate events, and a number of arcs which connect the basic and intermediate events to gates. Basic events represent fundamental events such as the failure of a single system component. Intermediate events usually represent events like the failure of a subsystem. For our purposes here we are primarily interested in a class of fault trees which contain only two types of gates: AND gates and OR gates. Each gate has a number of input arcs which originate at an event (basic or intermediate) or at the output of some other gate. Each gate also has one output arc which goes to either an intermediate event or to one or more other gates. An AND gate produces an output only if all of its input events occur. An OR gate produces an output if any of its input events occur. Any event or gate output may serve as an input to more than one gate, and such an event is termed a repeated event.

Fault trees model a system by depicting how various combinations of individual component/ subsystem failures act together to affect the system as a whole. The fault tree usually has a single root event at the top of the tree, called the undesired or top event, which represents the primary failure event of interest. Often the top event represents catastrophic failure of the entire system. The fault tree model is developed by examining all the basic or intermediate events, which individually or in combination with other events, may cause the top event to occur. The top event then becomes the output event of an OR gate whose inputs are all the intermediate events which by themselves could cause the top event to occur. If the top event can only be caused by the combined occurrence of some set of intermediate events, the top event will instead be the output event of an AND gate whose inputs are those intermediate events. The fault tree is then developed by recursively examining each intermediate event in turn and determining whether it is the output of an AND gate or an OR gate. This recursive development continues until all intermediate events have been decomposed and all of the most fundamental events in the fault tree are basic events. The development of a fault tree model is consequently a "top-down" decomposition process. This is in contrast to the development process for a digraph model which is a "bottom-up" forward-failure propagation process from the individual components to the more complex subsystems and ultimately the system as a whole. The perspectives of the two model types are complementary.

Figure 2 shows a fault tree model for the example system of figure 1. The top event of the fault tree denotes the failure of the ground control station to successfully receive the spacecraft's data. The

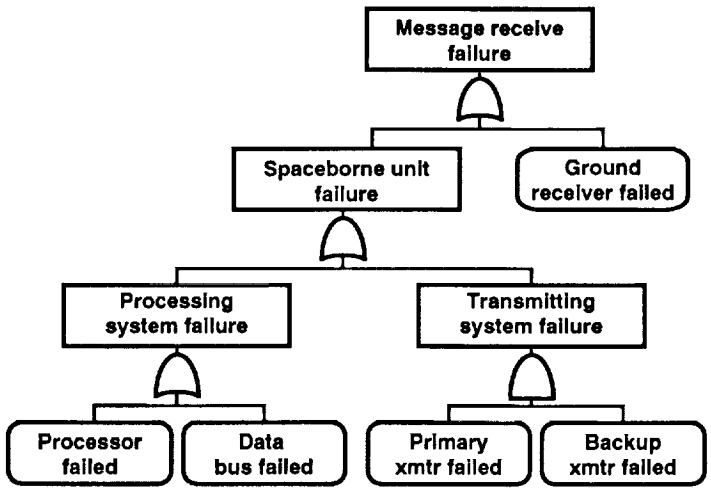

Figure 2. Fault tree model for the example system.

failure to successfully receive the data might be due either to a failure of the spaceborne subsystem or to a failure of the ground-based subsystem (the receiver). A failure of the spaceborne subsystem could be due either to a failure of the transmitter subsystem or to a failure of the processing subsystem. A failure of the transmitter subsystem requires the failure of both transmitters. A failure of the processing subsystem could be due to a failure of either the processor or the bus.

3 FDIR USING GRAPH MODELS

In this section we review some existing work which has been done on the subject of fault management using graph-based models. In the previous section we described two graph-based modeling methods: digraphs and fault trees. The fault management methods we are about to describe are all procedures for analyzing a model of the system which is either a digraph or a fault tree. In order to begin the diagnosis process, all of the methods below require identification of components whose status is observable (for example sensors) and information about which sensors are operating correctly and which are not. This information may be obtained through user inputs to the model or by automated real-time monitoring of sensor outputs. This information is presented to the graph-based models in the following form: sensors that are known to be "good" (output within operational limits) are said to be *normal alarms*; sensors that are known to be "bad" (output outside of operational limits) are said to be *abnormal alarms*. Some of the diagnosis methods described below have the capability to take into account the propagation time of failures from one component to another. For these methods the normal and abnormal alarms include the latest time at which the sensor is known to be good or bad. Once the normal and abnormal alarms have been specified, the result of the analysis of the model is the identification of a set of components (both sensor and non-sensor) whose failure could account for a set of observed symptoms exhibited by the system. At the end of this section we present our most recent fault management method, which is based on the use of both digraphs and fault trees.

Digraph Fault Management Using FEAT

A digraph modeling software tool called Failure Environment Analysis Tool (FEAT) was developed by Lockheed Engineering and Sciences together with NASA Johnson Space Center (JSC) (ref. 1). FEAT allows a user to build and analyze a digraph model of a complex system. The program has a graphical user interface which displays a digraph and uses color highlighting of digraph nodes and AND gates to pictorially display the results of digraph analyses. Among the major features of FEAT are the *source* and *target* operations. When the user selects or marks a digraph node and performs a source operation, FEAT identifies all nodes in the digraph that would become marked as a result of the propagation of the marking of the selected node. The marking of a digraph node corresponds to a failure of the component represented by the node. When a digraph node is selected and a target operation is performed, FEAT identifies all digraph nodes from which a marking could propagate to the selected node. Where the marking of a single digraph node could propagate to a target node, the single node is called a *singleton* and is highlighted in red on the digraph diagram. Where the coincident marking of two distinct nodes is required to cause the target node to be marked, the two nodes together are called a *doubleton* and are highlighted in blue. The user may therefore identify which digraph nodes affect or are affected by a selected individual node through the use of the target and source operations, respectively.

Using this tool, researchers at JSC and McDonnell Douglas Space Systems Company have developed a method of identifying a set of digraph nodes (representing failure of individual system components), which could potentially account for a symptom exhibited by the system in the form of a marked target node in the system model (ref. 2). The method starts by identifying sensors that fall into three categories: sensors that are known to be good (the normal alarms), sensors that are known to be bad (the abnormal alarms), and sensors whose status is unknown due to corruption of their output data or which are not relevant to the current operational mode. Using these sets of observable components, the failure propagation relationships expressed in the digraph model are used to determine possible causes for bad sensor readings and components that must be working to provide the good sensor readings. This information is used to successively prune the set of candidate components (both sensor and non-sensor components), which might be responsible for the symptom represented by the target node. The pruning process makes use of the digraph analysis operations available in FEAT and may be followed pictorially by the analyst. It proceeds by successively identifying subsets from the initial set of candidate components that may be inferred to be working correctly from the failure propagation information in the digraph. At this time the user must manually perform the identification steps and prune the set of candidate components as new subsets of correctly operating components are identified. Future plans call for this manual procedure to be automated.

Interactive Digraph Analysis (IDA)

David Iverson has developed a computer program that implements a variation of the digraph fault diagnosis method described above. The primary difference between the two methods lies in the fact that Iverson's method uses cut sets of digraph nodes to identify and prune sets of candidate components which could be responsible for causing the system's failure symptoms, whereas the JSC/McDonnell Douglas method uses the singleton/doubleton identification features of FEAT to identify and prune the sets of candidate components. A cut set for a digraph node is a set of nodes which, when all marked, cause the digraph node itself to be marked. We note here that neither method takes into account the

time required for failures to propagate through individual digraph nodes, so no timing information is given as part of the normal/abnormal alarm specifications, and propagation time is not considered in the diagnosis process. Iverson's method works in the following way: for each digraph node that represents a normal alarm (a sensor operating correctly) all of the cut sets of that node are found. Next, all nodes that represent abnormal alarms (sensors whose output is outside of operational limits) are identified and all made inputs into a single new AND gate. The output node of this AND gate denotes the observed failure symptoms exhibited by the system. The cut sets for the output node of this new AND gate are then found. These cut sets represent all the possible causes of the symptoms. The actual cause(s) of the observed symptoms will be some subset of these cut sets. The AND gate cut sets are then compared with the combined cut sets for the normal alarm nodes, and any cut set of the AND gate for which one of the normal alarm cut sets is a subset is thrown out. This last step is justified by the following reasoning: if a cut set of the AND gate is a true cause of the failure symptoms, then all the events in the cut set must have occurred in order for the AND gate output event to occur. However, if a subset of the events in the cut set are also a cut set for a normal alarm, then those events can not have occurred.

Therefore, any cut set of the AND gate which contains as a subset a normal alarm cut set cannot have occurred and hence cannot be a cause of the system failure symptoms. Such a cut set can be pruned from the set of candidate causes of the failure. Any AND gate cut sets remaining after this pruning process identify the possible true causes of the system failure symptoms. This method is implemented as part of a general program for analyzing digraphs called the Interactive Digraph Analysis program (IDA). IDA differs from FEAT in that it is based on cut sets instead of matrix methods and lacks the graphical user interface that FEAT offers.

Method of Narayanan and Viswanadham

Narayanan and Viswanadham describe an approach to fault management which involves an integrated use of digraphs, fault trees, and an AI production rule inference engine (ref. 3). They advocate dividing the diagnosis process into two phases: a *failure source location phase* and a *failure cause identification phase*. Each of these phases are analyzed separately using different underlying system models. The failure source location phase involves constructing a hierarchicial model of the system using multi-level digraphs. A symptom of abnormal system behavior is then expressed by the marking of a target node (failure of a subsystem) in the highest order level of the digraph model. The digraph on the highest level is then analyzed to find the subsystems (represented by digraph nodes on that level) whose failure could lead to the marking of the target node. Each such subsystem is then optionally modeled in greater detail by another digraph at a lower level of detail resolution. The digraph analysis is then recursively performed on these lower level digraphs. This recursive decomposition/analysis process is continued down until eventually the lowest level of modeling resolution that is of interest to the user is reached. Once digraph nodes whose marking could lead to the marking of the top level target node are identified on that lowest level, cause-consequence models of failure are constructed to capture the failure modes of the identified candidate components/subsystems. The cause-consequence models are *augmented fault trees*. The fault trees are augmented to contain heuristic and systematic knowledge about temporal, probabilistic, and causal characteristics of faults that occur in the component/subsystem. Once constructed, the augmented fault trees are used to generate production rule knowledge bases for the components/subsystems. As a final step, the failure cause identification process is invoked to analyze the production rule knowledge bases to infer what the causes of failure might be. The inferential

reasoning proceeds through forward and backward chaining operations on the rules. This reasoning process eliminates from consideration potential causes of failure based on the normal alarms, the abnormal alarms, the time required for individual failures to propagate through the subsystems, and information on the importance and probability of propagation of failures.

Fault Tree Diagnosis System

David Iverson and Ann Patterson-Hine have implemented a modified version of Narayanan and Viswanadham's approach to modeling cause-effect relations for failure diagnoses (refs. 4–6). Narayanan and Viswanadham advocate the use of augmented fault trees to generate a production rule knowledge base. Inferential reasoning about failure causes is then performed through forward and backward chaining on the production rules in the knowledge base. This approach of representing the knowledge base with production rules has some drawbacks. One performance-related drawback is the relatively large amount of overhead processing needed to search through the rule base in the course of forward and backward chaining. Another drawback is the increased computational resources needed to generate and store a second representation (the production rules) of the knowledge base that is already completely represented in the augmented fault tree. Iverson and Patterson-Hine addressed these two drawbacks by developing methods to perform the inferential reasoning processes on the augmented fault tree directly, thus obviating the need to generate the production rules. They implemented the augmented fault tree using an object-oriented fault tree paradigm developed by Patterson-Hine (ref. 7). In addition to saving the effort needed to generate and store the production rules, they found that accessing the information in the knowledge base was more efficient than searching an indexed rule base. This follows from the fact that the structure of the fault tree itself encodes the production rules of the knowledge base. Each gate in the fault tree represents a production rule in that the inputs and output of the gate are the antecedents and consequent, respectively, of a production rule in the knowledge base. During a forward or backward chaining operation, the next rule(s) needed are immediately available by virtue of the implicit links between rules embodied in the structure of the fault tree. Separate lookup operations on a table of rules are not required. The resulting diagnosis program is called the Fault Tree Diagnosis System (FTDS) and permits the modeling and diagnosis of system failure causes with a user-entered fault tree model. As FEAT does for digraphs, FTDS uses a graphical user interface and displays the course and results of a diagnosis analysis with color highlighting of the fault tree model. FTDS is implemented under X Windows in Unix(TM) computing environments.

An Integrated Graph-Based Diagnosis Method

We now describe an integrated graph-based diagnosis approach (IGBDM) to the fault management problem which is under development by our group at NASA Ames Research Center. Our approach differs from the integrated approach of Narayanan and Viswanadham in the role played by the digraph. Narayanan and Viswanadham advocated using an analysis of the digraph to determine fault location within the system, then feeding the results of that analysis to production rule knowledge bases to determine the cause of the failure(s) identified by the digraph analysis. In our approach we advocate a more modest role for the digraph. Like Narayanan and Viswanadham, we rely on an augmented fault tree knowledge base to provide determinations of causes of failures through appropriate analysis of the fault tree model. However, we assume that inputs to the augmented fault tree analysis process are available from the user in the form of a specification of the failure symptoms exhibited by the system. In

other words, we rely on monitoring of sensors and/or user input instead of a precursory digraph analysis to provide the temporal status information about system components needed as input by the fault tree analysis process. The digraph model of the system then takes on a new role under our approach. We use a digraph model of the system to aid in the construction of the augmented fault tree model rather than as a subject for analysis. The motivation behind this stems from concerns related to human activity in building the system model. The process of building a fault tree model of a system is readily learned, but is not always an intrinsically natural process and so may require the development of some expertise on the part of the modeler. Digraphs, because of their tendency to be very similar to the schematic of a system, are often more easily developed. This is particularly true when the modeler is someone whose training has not included extensive emphasis in systems modeling. A computer program was recently developed to convert a digraph model into a fault tree model (ref. 8). This program permits the formulation of a system model in the more easily specified digraph format. The digraph model is then automatically converted to a fault tree model and augmented with the information needed for analysis to determine a diagnosis. If the digraph and augmented fault tree models are linked together, the fault management advisor program can perform all analysis on an internal fault tree model while providing the user the opportunity to enter the system model and monitor the results of the analyses with a digraph. This permits the program to interact with the user via a model type which is more easily accommodated by the user while performing the analysis functions with a model type more suited to the computer's inferential reasoning processes. The implementation of our diagnosis approach described here is under development and will be implemented in a Unix(TM) environment under X Windows.

4 EXAMPLE PROBLEM

We now describe the application of our fault management/diagnosis method to an example problem. We model a real system which is being designed and built. The system is an automated telescope being built by the Autoscope Corporation. Autoscope builds automated telescopes intended for unattended operation for extended periods of time in remote areas, such as on secluded mountaintops in the southwestern desert where astronomical observation opportunities are among the best in the United States. The technology that Autoscope is developing has potential application for future NASA programs using telescopes on orbiting platforms or on the moon. We begin by giving an overview of how this current fault management project could fit into a larger NASA program using automated telescopes. We then describe the specific Autoscope automated telescope used for our study. We next develop the digraph model of the telescope subsystem chosen for analysis for this study. Finally, we describe the application of our fault diagnosis method to determine which specific component failures account for certain specified symptoms of abnormal system behavior.

Lunar Telescopes

The automated telescope used for our study is a proposed prototype for the Steerable Automatic Lunar Ultraviolet Telescope Explorer (SALUTE) project. The motivation behind the SALUTE project is the assertion that it is only a matter of time before humans return to take up permanent presence on the moon. When they do, it is certain (given the ideal conditions for astronomical observation the moon offers) that they will bring telescopes with them and initiate major astronomical research projects. Because relatively little is known about the maintenance and operation of precision mechanical devices

in general and telescopes in particular in the lunar environment, it is logical to gain some engineering experience in this area before making substantial investments in large complex telescopes and installing them on the moon. The most obvious way of achieving this engineering experience is to place small, relatively inexpensive automated telescopes on the moon in order to gain information about how the lunar environment will affect the optical, mechanical, and electronic equipment used in lunar telescopes. The role of these telescopes is therefore two-fold: to evaluate the technology required to construct, maintain, and operate lunar telescopes, and secondarily to provide working (albeit relatively small) telescopes to allow lunar astronomical research to begin immediately rather than have to wait until humans are ready to establish a permanent presence on the moon.

Under the SALUTE project a number of small automated telescopes would be soft-landed at various locations on the moon. Each telescope would communicate with a control center on earth through two communications links (a high-bandwidth link primarily for data and a low-bandwidth link primarily for commands). Researchers on earth would direct the observations made by the telescopes by uploading commands to the telescope over the communications links, and the telescopes would transmit the gathered data back to earth over the same links. The telescopes include several computers on board that are capable both of determining the individual steps required to make the observations (such as how to locate and acquire a star, or how to make appropriate background and reference measurements) and also of directing the telescope movements needed to perform the observations. The commands uploaded from earth can therefore be of a very high-level nature (for example: observe this specific star every X hours for Y minutes to obtain photometric measurements). The intent is to keep the operational expense of the system to a minimum, and the project will rely on the use of computer-assisted automated management systems (for fault diagnosis and management, AI-based planning and scheduling) to keep earth-based ground control staff and resource requirements very low. The system also is intended to be widely accessible to researchers from small colleges and possibly even selected high school students. This will be practical only if the system has a very low overhead cost.

The Autoscope Telescope

The system we chose to analyze for our study is a model AT-16 automated telescope manufactured by Autoscope Corporation. The telescope is designed as part of an integrated automated observatory which is capable of performing (without human intervention) all functions required to operate all electrical and mechanical equipment needed for astronomical observation. The goal is to permit astronomers to direct nightly observations and analyze data obtained from those observations from the comfort of their offices rather than personally traveling to observatories located in remote areas. This is accomplished by enabling the astronomers to issue commands to the observatory over ordinary phone lines using a PC with a modem in their office. These commands specify what observations to make and how to make them. The observatory is responsible for making the observations and gathering the resulting data. The process of carrying out this responsibility requires that the observatory be able to perform the following functions: scheduling the observations in an appropriate way following whatever priorities are specified, ensuring that observations are made only between sundown and sunrise, monitoring the weather and closing down if bad weather would damage the telescope or equipment, monitoring and reporting equipment failures to appropriate human maintenance personnel, and taking appropriate corrective action in response to equipment failures. Following a night of observations, the astronomer

retrieves the collected data for analysis, again over ordinary phone lines using a PC and a modem. The presence of humans in the operation is required only in the event of equipment failures.

Figure 3 shows a diagram of the automated observatory. Two modems are available for reception of instructions and transmission of data. A master observatory computer interfaces the observatory with the outside world by mediating and responding to communications to/from the modems. The master computer communicates with the controllers for the astronomical instruments (photometer and charge coupled device camera) used for making measurements. These instruments also provide feedback data to the control computer. The master computer also communicates with a telescope/observatory control computer which is responsible for most of the functional operation of the observatory. The telescope/observatory control computer communicates with the weather station to monitor the weather conditions and close down the observatory if damaging weather conditions develop. Through intermediary telescope and observatory controllers, the control computer also directs the movements and functions of the telescope itself and observatory enclosure, power supplies, and environmental control devices. Humans (when present) can communicate with the master computer through a keyboard and monitor.

Figure 4 shows a picture of the telescope which is the subject of our study. Figure 5 shows a simplified diagram of the telescope. We now offer a very brief description of the major features of the telescope that are relevant to the model we are using for our study. The main components are a large primary mirror and a smaller secondary mirror, both contained in a protective tube. Light enters the tube at the top and travels to the bottom of the tube where it reflects off the primary mirror. The light next travels back to the top of the tube where it is reflected back yet again off the secondary mirror. It then travels back to the bottom of the tube and through the middle of the primary mirror to measuring instruments located behind the primary mirror. Generally only one instrument receives the incoming light at a time; however the light may be directed to as many as four different instruments by an instrument selection device. The directional orientation of the tube may be moved, allowing the telescope to be aimed anywhere in a large expanse of the sky. Directional pointing of the telescope is accomplished by three stepper motors (X, Y, and Z) which are under computer control. Three additional motors (T, U, and V) control movements of the secondary mirror, which may be adjusted independently of the primary mirror and the rest of the telescope body. The stepper motors are supplied by two power supplies, a 5-Volt and a 24-Volt power supply. The T, U, and V motors are supplied by both the 5-Volt and the 24-Volt power supply, providing for a limited amount of redundancy (if the 24-Volt power supply fails, these motors can still operate, but will move more slowly when supplied by only the 5-Volt power supply). The X, Y, and Z motors are supplied only by the 24-Volt power supply and do not have the benefit of any redundancy with respect to power.

For the purposes of this segment of the study we decided to concentrate our attention on the TCS-200 telescope controller subsystem of the overall telescope. The telescope controller is the most complex subsystem in the telescope, and is responsible for directing telescope operations and movements in response to commands from the control computer. It is represented by the box labeled "Telescope Controller" shown in figure 3. Figure 6 shows a simplified schematic of the telescope controller subsystem. The controller subsystem consists of three parts: a control card which plugs into an expansion slot within the telescope control computer (an IBM 386 class PC), the controller unit itself, and a number of peripheral sensor and motor devices which actually perform the operations on the telescope. The

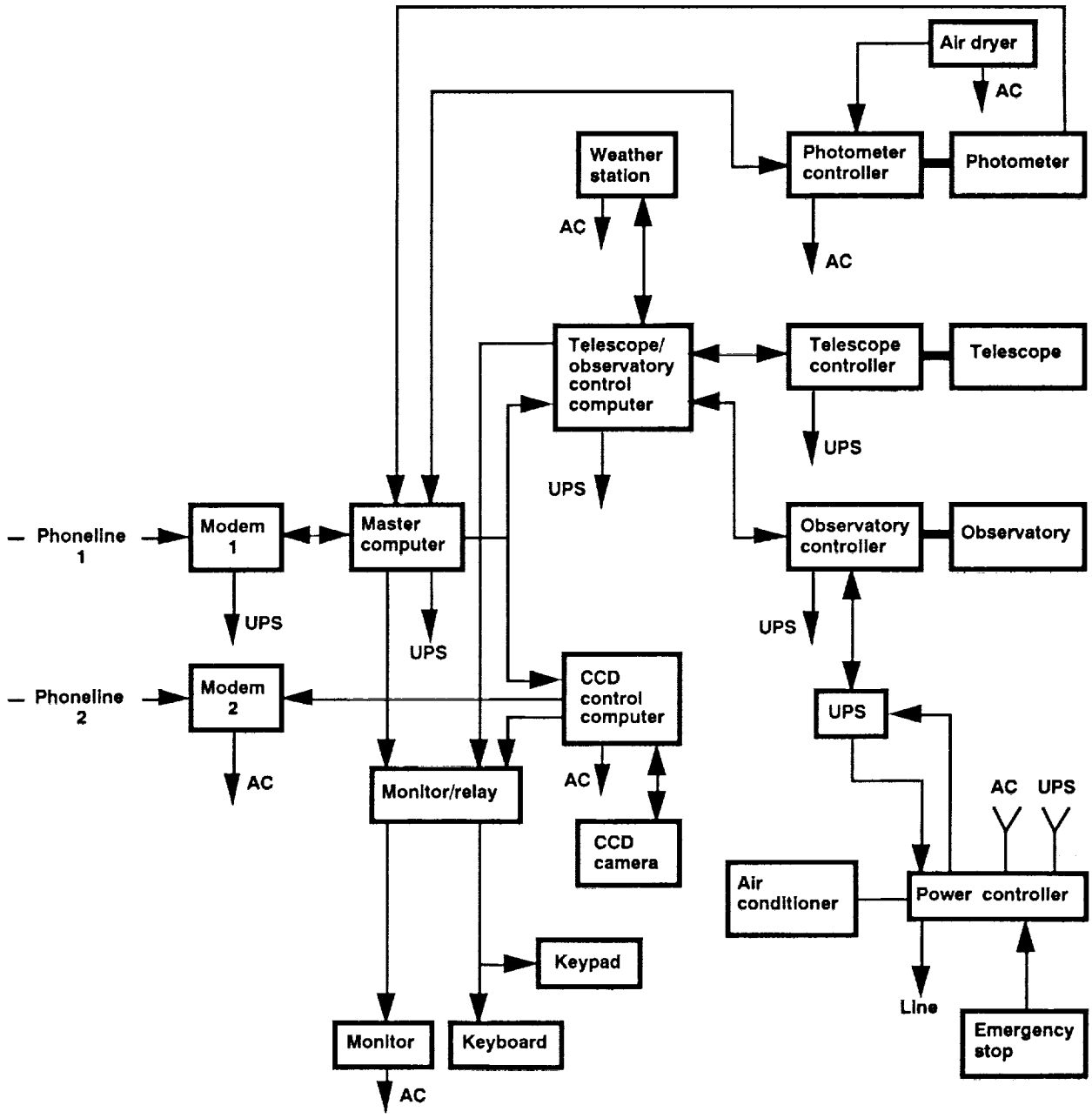

Figure 3. Schematic diagram for an automated observatory system.

telescope control card (left side of (fig. 6)) provides the controller subsystem's interface with the control computer. The controller itself (see center of (fig. 6)) contains a 24-Volt and a 5-Volt power supply, a number of various drivers for the peripheral sensors and motors, and a junction board which routes signals received from the telescope control card to the individual drivers. The drivers are connected through ports to the sensors and motors located on the telescope frame shown in the right side of Figure 6. There are a number of probe points located throughout the controller junction board at which signals may be monitored by a diagnostics subsystem (one is currently under development by Autoscope) or by a human technician. The motors include devices like the various stepper motors which move the

Figure 4. Automated telescope.

telescope frame along specific directions and are necessary for positioning the telescope so that the lens is centered on the specific object to be observed.

System Model

Figure 7 displays the digraph model for the simplified telescope controller subsystem. The general structure of the digraph parallels the topological structure of the schematic of figure 6 fairly closely. The digraph indicates that failures can propagate from the control computer digraph node (labeled "TelCntrlComp") to the telescope control card (digraph node labeled "TelCntrlCard"), through the various ports to the controller junction board (digraph node labeled "JB"). From the junction board, failures can propagate through the various ports to individual drivers, then on to the individual peripheral devices. The propagation of failures therefore can be seen to follow the structural and electrical connections within the subsystem as depicted in the schematic. There are a number of disconnected digraph segments that depict the effect of propagation of failures in the power supplies. The various drivers and peripheral devices are dependent on one or the other of the two power supplies, so the failure of one

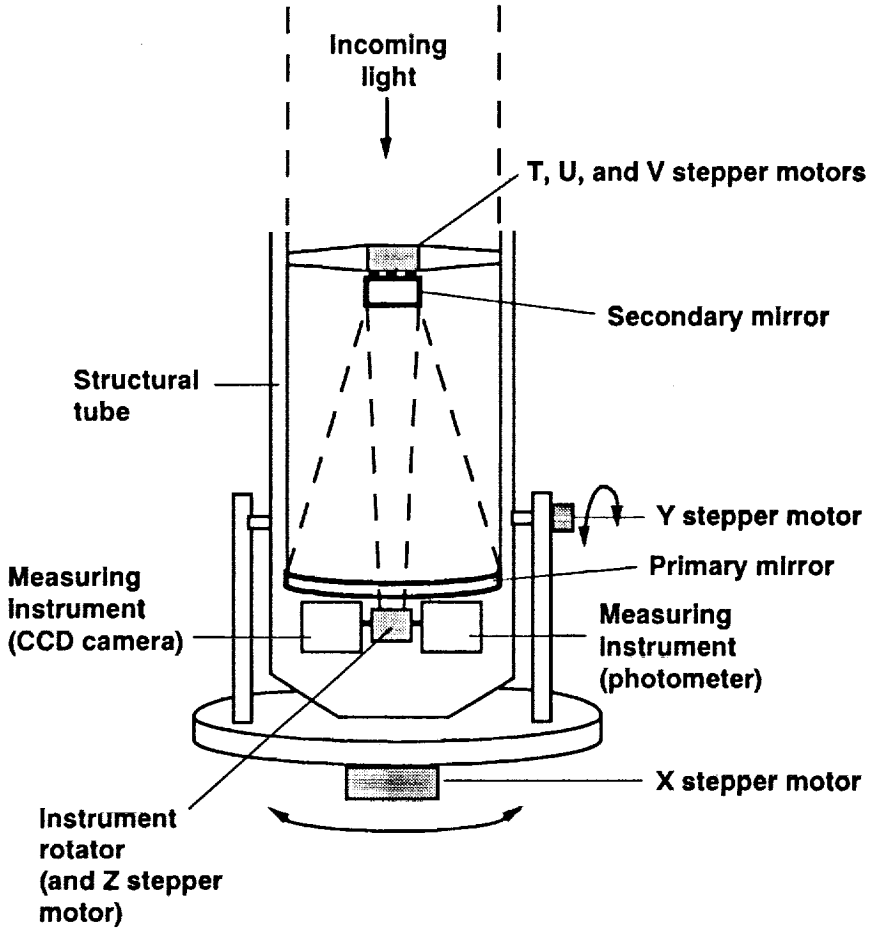

Figure 5. Diagram of the automated telescope.

power supply will determine which subset of the drivers and/or devices will be affected. Since the telescope controller subsystem used for this example has been simplified for the purposes of illustration, not all failure modes of the telescope controller are reflected in the system model presented here.

Fault Diagnosis Using the Model

The digraph model shown in figure 7 was developed with the aid of the FEAT digraph modeling tool. In order to perform diagnosis processing, the digraph model is first converted to a fault tree. This is accomplished by a conversion program (ref. 8) which automatically generates an equivalent fault tree model from a digraph. Figure 8 shows the fault tree obtained from the conversion of the digraph in figure 7. Briefly, the conversion process proceeds in the following way: for each node in the digraph, an OR gate is created in the fault tree which contains the same label as the digraph node. The OR gate has one basic event input which denotes an independent failure of the component represented by the digraph node. The label of the basic event is the label of the digraph node with "–F" appended. The OR gate also has one input for each incoming directed arc to the digraph node being converted. Each of these fault tree input events represents a propagated failure or event corresponding to the individual

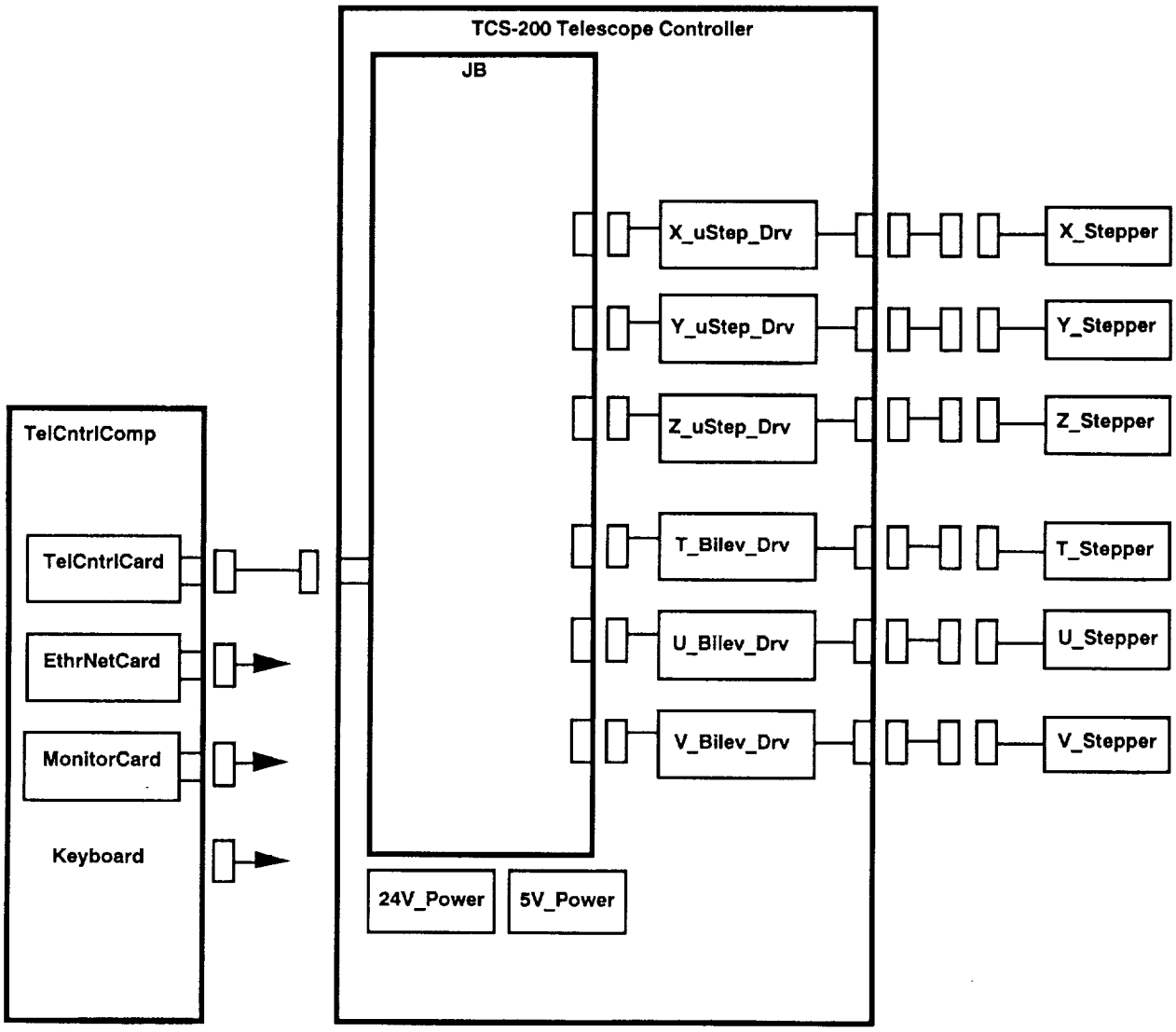

Figure 6. Schematic diagram for the TCS 200 telescope controller subsystem.

incoming digraph arc and the digraph node from which it originates. This procedure reflects the fact that each node of the digraph represents a failure that may be due either to a failure of the component itself or a failure propagated along one of the inputs to the digraph node. The AND gates in the digraph are converted directly into AND gates in the fault tree. The fault tree in Figure 8 has been slightly modified to remove some redundant event nodes that are produced by the automatic conversion process, but which have no meaning in the diagnosis process.

Once in the form of the fault tree, the FTDS program may be used to perform diagnostic analyses. FTDS displays the fault tree on the user's screen and permits the user to identify components/subsystems which are known to be working at specific times (normal alarms) and also components/subsystems which are known to be failed at specific times (abnormal alarms, or the symptoms). From this information,

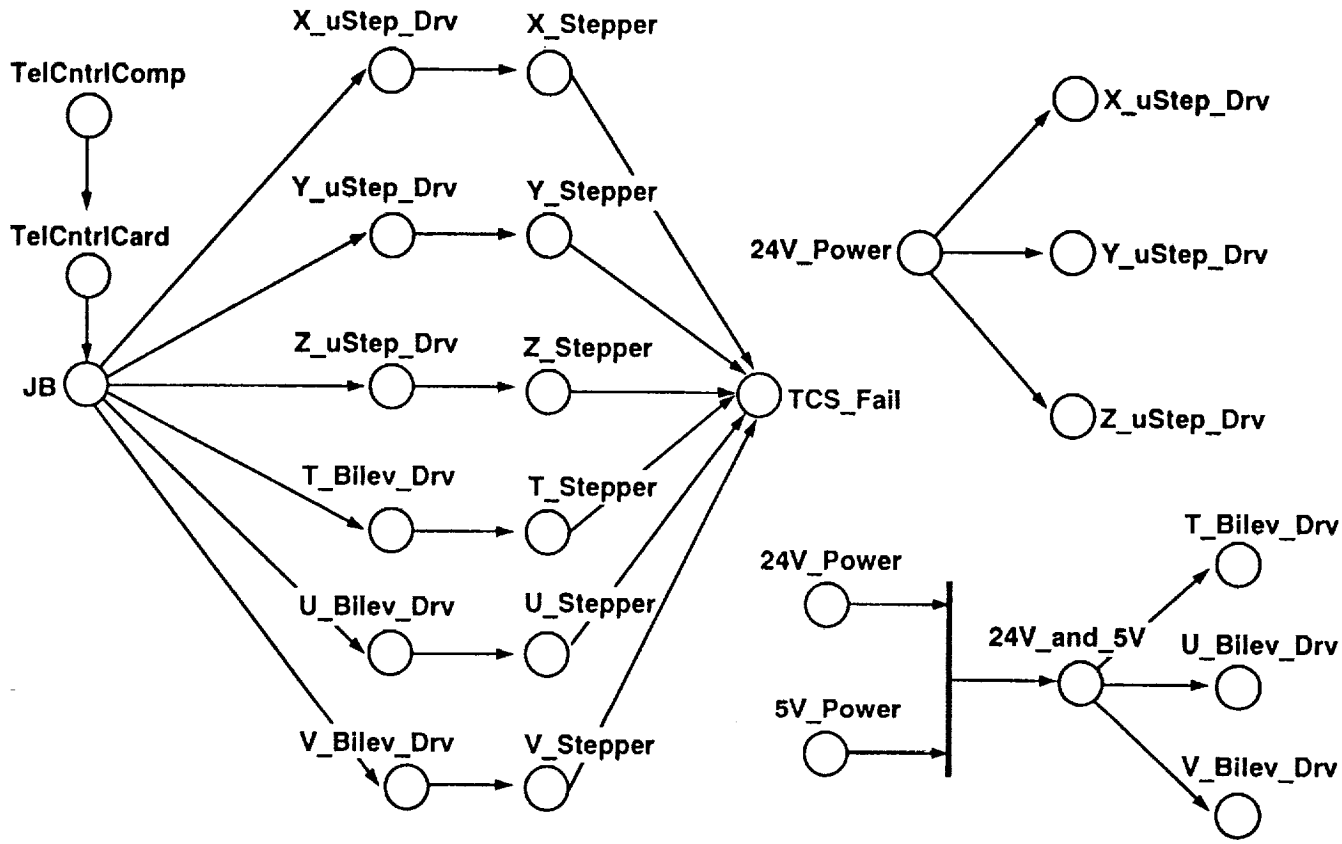

Figure 7. Digraph model for the TCS 200 telescope controller subsystem.

FTDS first uses forward and backward-chaining through the production rules implicit in the structure of the fault tree to determine all the components and subsystems that must be working and the latest times at which they must be working in order for the normal alarms to be satisfied. FTDS then uses backward- and forward-chaining through the fault tree to determine which must have failed (and the latest times at which they must have failed) in order for the abnormal alarms to be accounted for. The result is a set of candidate components whose failure could be the cause of the observed abnormal alarms. If more than one cause is identified, further investigation in the form of diagnostics may be needed to identify the component(s) whose failure was the actual cause of the symptoms (abnormal alarms). The conversion process produces a fault tree whose basic event nodes have a one-to-one correspondence with the nodes of the digraph. As a consequence of this, the results of the diagnosis may be transferred back to the digraph and displayed pictorially to the user. The user may then see, through color highlighting, the results of the diagnosis on both the digraph and the system schematic.

As a simple example, consider the digraph and fault tree models of the TCS 200 controller subsystem shown in figures 7 and 8. Suppose it has been determined (either through monitoring of sensors or human observation) that at time interval 5 the T Stepper motor (digraph and fault tree nodes labeled T_Stepper) is not operating correctly. Suppose further that it is known through monitoring that at time interval 4 the junction board (digraph and fault tree nodes labeled JB) *is* operating correctly. Our diagnosis will proceed under the assumption that it takes one time unit for the effects of a failure to propagate through each node in the digraph. In terms of the fault tree this means that a failure takes one time unit

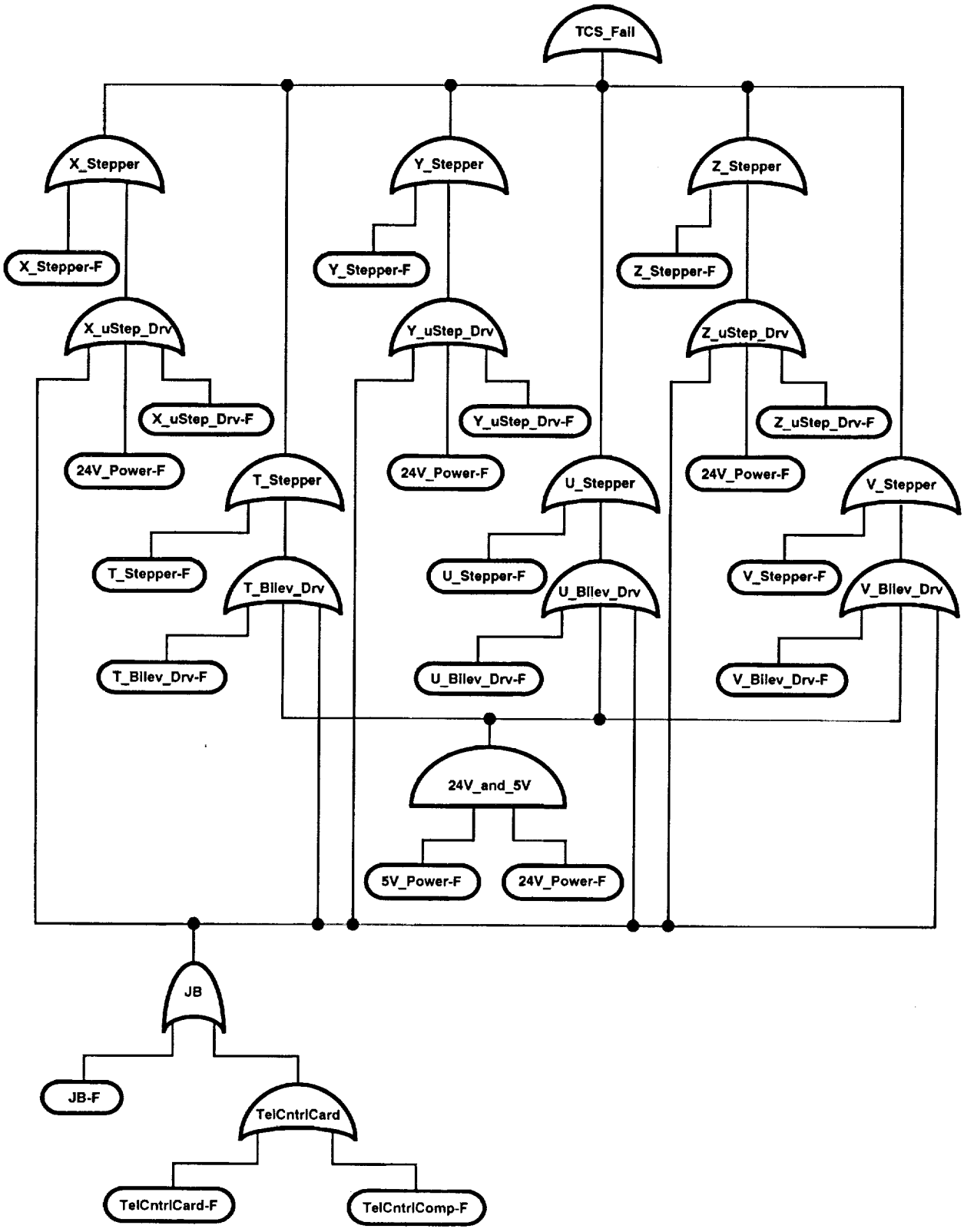

Figure 8. Fault tree model for the TCS 200 telescope controller subsystem.

to propagate from a gate to the gate immediately above it, with the exception that propagation from basic events labeled "<node>-F" to an OR gate labeled "<node>" immediately above the basic event, takes zero time units. The assumption of a one-unit propagation time is arbitrary. The propagation times may in fact be individually specified for each node in the fault tree to more accurately predict actual propagation conditions. The diagnosis process is begun by telling FTDS that the system/subsystem represented by the top node of the fault tree was observed to have failed at a specified time (time 10). FTDS begins by examining all of the normal alarms that have been set and inferring which components must have been working at what latest time in order for the normal alarms to have been able to exist as specified. If the junction board was known to be operating correctly at time interval 4, the telescope control computer (labeled TelCntrlComp) and the telescope control card (labeled TelCntrlCard) had to be working correctly at (and before) time intervals 2 and 3, respectively. These are the only inferences that can be made based on the normal alarms. FTDS next considers the abnormal alarms, which here includes only the failure of the T_Stepper at time interval 5. This failure might be caused either by the failure of the stepper motor itself (node labeled T_Stepper-F) or by the failure of the T-Bilevel drive (node labeled T_Bilev_Drv). The failure of the T-Bilevel drive might be caused by the failure of the drive itself (node labeled T_Bilev_Drv-F) or by the failure of both power supplies (node labeled 24V_and_5V). The event represented by the node labeled 24V_and_5V will only occur if the events labeled 5V_Power-F and 24V_Power-F both occur. Hence, the set of candidate causes for the abnormal alarms is determined to be { (T_Stepper-F at time 5 or before), (T_Bilev_Drv-F at time 4 or before), (5V_Power-F at time 3 or before), (24V_Power-F at time 3 or before) }. This set can be pruned further if more information is known in terms of normal alarms. For example, if it is known that the V Stepper (node labeled V_Stepper) is operating correctly at time interval 5, then the two power supplies can be eliminated as candidates (they must be working at time 3 or the V stepper motor could not be working at time 5).

As illustrated in the above discussion, the results of the diagnosis is a set of component failures that are possible causes of the observed symptoms as expressed by normal and abnormal alarms. Generally, the availability of more detailed normal and abnormal alarm information makes possible more specific or accurate diagnoses in that smaller sets of candidate component failures can be identified. Ideally, the normal and abnormal alarm information is obtained from continuous, automated monitoring performed on the observable system components to provide the maximum amount of information on which the diagnosis can operate. Once a set of possible causes is identified, the human operator is able to perform diagnostics on the candidates to further identify the exact cause of the system's failure symptoms. Alternatively, the operator can alter the operation of the system to bypass or minimize the effect of the potentially faulty components identified by the diagnosis.

5 APPLICATION OF IGBDM TO A GENERAL DATA SYSTEM

The initial role we envision for our automated fault management computer program (Integrated Graph-Based Diagnosis Method) applied to general data management systems is one of an advisor to human operators of the system. The system digraph model will have been built by the engineers responsible for designing the system itself. Displays of the digraph model, the system schematic(s), and optionally the fault tree model would be available to system operators on workstation monitor screens. The outputs of the sensors in the system would be connected to monitoring devices that could signal the diagnosis program if any out-of-tolerance conditions arise. Automated monitoring of this type would

be especially useful for large complex data management systems. Upon the occurrence of one or more component failures, the diagnosis program would read the symptom data (supplied by the sensor monitors in terms of normal and abnormal alarms) into the digraph/fault tree model and initiate the diagnosis process. The results of the diagnosis could be displayed simultaneously on the fault tree, digraph, and schematic views of the system for the operator by using color highlighting of the components causing the failure. The diagnosis system could then optionally display suggestions to the operators on such matters as what further diagnostic tests (if any) would be appropriate to run, how the system could be reconfigured to circumvent the failed components, and how operating procedures should be modified to work around the effect of the failures. This information may be simply retrieved from a previously established operating procedures database by keying on the remaining operational configuration of the system. Further enhancements of the advisory function may be provided by connecting the fault tree model with a hypertext system containing information about component/subsystem specifications and characteristics, and operating procedures. This would provide the operators with an interactive information resource for exploring ways to evaluate the severity of the failures and their effects and to define or identify potential work-arounds to them.

The eventual intended role of IGBDM is as an automated diagnosis and recovery system. In this capacity IGBDM will be capable of monitoring of system sensors and detection of incorrect sensor outputs, diagnosing the causes of failure symptoms observed through sensor monitoring, reconfiguring the system to bypass failed components and/or altering the operating processes of the system to mitigate the effect of the failures. Such a capability would increase both the reliability and the performance of systems, particularly systems for which human maintenance and repair are available only rarely, or not at all. To achieve this level of automation, our diagnosis and recovery computer program must have some additional capabilities beyond those already described. We discuss the most important of these additional capabilities in the next section.

6 FUTURE WORK

We have described in a previous section the scope of the fault management problem, which is addressed by our current implementation of our fault diagnosis/management approach. In this section we identify some additional aspects of the fault management problem, which we hope to address in future development of our current work.

The implementation of our approach to fault diagnosis and management so far has concentrated on primarily passive processing of input data from sensors and/or human operators. In many cases this alone may be sufficient to arrive at a satisfactory diagnosis of the component/subsystem failures that are affecting the system. However, there may be situations in which a more active process is useful. The passive analysis will identify a number of components/subsystems that *may* have failed. Once these candidate subsystems are identified, further diagnostics may be required to positively confirm which subsystems have failed and which have not. Because the structure of the fault tree encodes the "production rules" in the knowledge base that are necessary for inferential diagnostic reasoning, the identification of a failed subsystem (consequent of a rule) corresponds to the selection of an intermediate event node in the fault tree which represents the subsystem of interest. The object-oriented representation for the fault tree we are using lends itself very well to incorporating active procedures into the analysis process. For example, the object which denotes the subsystem of interest may contain a pointer to a subroutine

that is to be executed whenever the subsystem is identified as potentially failed. The subroutine might implement diagnostic routines specifically designed to ascertain the operational status of the subsystem. Alternatively, the subroutine might initiate an automatic reconfiguration intended to bypass the failed subsystem or modify its operation so that it is more consistent with an impaired operational status. The subroutine might even serve both functions—running an appropriate diagnostic suite on the candidate subsystem, then initiating reconfiguration only upon confirmation that the subsystem has experienced a failure. Alternatively, the subroutine's function may be limited to printing recommendations for corrective action to the human operators of the system. In this way, the diagnostic analysis process can include active components that can be incorporated in a natural way due to the nature of the augmented fault tree model of the system.

There is another aspect of potential interest that may be readily integrated into the diagnosis analysis process, largely because of the nature of augmented fault tree models. There may be situations in which it would be advantageous to be able to integrate reliability evaluation with fault diagnosis. This capability might assist in deciding which set of candidate faults to investigate first, or which set of competing faults should receive priority in devising work-around procedures. Fault trees are a well-established reliability modeling technique. The diagnosis method we advocate here utilizes fault trees in a capacity other than their traditional reliability modeling role. The object-oriented structure we use for the augmented fault tree makes it easy to accommodate both diagnosis and reliability modeling simultaneously within the same fault tree model. Each object that represents an event in the fault tree, whether basic or intermediate, needs only to include a field to contain information about the component's failure rate or the event's probability of occurrence. As a consequence, reliability evaluation and fault diagnosis can easily be integrated together in a natural way. We hope to be able to extend our fault diagnosis program to incorporate both reliability modeling and active diagnosis processing in future development of our fault management/diagnosis program.

7 CONCLUSION

We have described a new method for automated fault management using graph-based models. Our method uses both digraphs and fault trees as underlying system models for the analysis process. This method is one that has evolved from a number of previously existing methods which use graph-based models. Our method is suitable for application to general data management systems. It is intended to assist the operators of such systems in assessing the causes of system failure symptoms and to decide on appropriate corrective action to take in response to the system impairment(s). We have presented an example application of our fault management method to illustrate its use. We are in the process of implementing this method in the form of a computer program running under the Unix(TM) operating system and the X Windows user environment. We are also exploring the development of more expanded roles for this methodology, including topics such as the integration of our fault diagnosis method with reliability modeling methods.

8 REFERENCES

1. Miller, Jim: FEAT User's Manual (Draft). Lockheed Engineering and Sciences Rep., 1991.

2. Clark, Colin; Jowers, Steven; McNenny, Robert; Culbert, Chris; Kirby, Sarah; and Lauritsen, Janet: Fault Management for the Space Station Freedom Control Center. In Proceedings of the 30th Aerospace Sciences Meeting and Exhibit, Reno, NV, January 6–9, 1992.

3. Narayanan, N. Hari; and Viswanadham, N.: A Methodology for Knowledge Acquisition and Reasoning in Failure Analysis of Systems. IEEE Trans. Systems, Man, and Cybernetics, vol. 17, no. 2, 1987, pp. 274–288.

4. Iverson, David L.; and Patterson-Hine, Francis A.: A Diagnosis System Using Object-Oriented Fault Tree Models. In Proceedings of the Fifth Conference on Artificial Intelligence for Space Applications, Huntsville, AL, May 22–23, 1990.

5. Iverson, David L.; and Patterson-Hine, Francis A.: Object-Oriented Fault Tree Models Applied to System Diagnosis. AAIC Paper, April 1990.

6. Patterson-Hine, Francis A.; and Iverson, David L.: An Integrated Approach to System Design, Reliability, and Diagnosis. NASA TM-102861, 1990.

7. Patterson-Hine, Francis A.: Object-Oriented Programming Applied to the Evaluation of Reliability Fault Trees. Ph.D. Thesis, Univ. of Texas at Austin, 1988.

8. Iverson, David L.: Automatic Translation of Digraph to Fault Tree Models. In Proceedings of the Reliability and Maintainability Symposium. Las Vegas, NV, January 21–24, 1992.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1993	Technical Memorandum

4. TITLE AND SUBTITLE

Fault Management for Data Systems

5. FUNDING NUMBERS

476-14-03

6. AUTHOR(S)

Mark A. Boyd, David L. Iverson, and F. Ann Patterson-Hine

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Ames Research Center
Moffett Field, CA 94035-1000

8. PERFORMING ORGANIZATION REPORT NUMBER

A-92147

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

NASA TM-103953

11. SUPPLEMENTARY NOTES

Point of Contact: Mark A. Boyd, Ames Research Center, MS 269-3, Moffett Field, CA 94035-1000
(415) 604-3678

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified-Unlimited
Subject Category – 51

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

We consider issues related to automating the process of fault management (fault diagnosis and response) for data management systems. Substantial benefits are to be gained by successful automation of this process, particularly for large, complex systems. We advocate the use of graph-based models to develop a computer-assisted fault management system. We begin by describing the general problem and outlining the motivation behind choosing graph-based models over other approaches for developing fault diagnosis computer programs. We review some existing work in the area of graph-based fault diagnosis, and offer a new fault management method which we have developed from existing methods. We apply our method to an automatic telescope system intended as a prototype for future lunar telescope programs. Finally, we describe an application of our method to general data management systems.

14. SUBJECT TERMS

Fault management, Graph based models, Automated diagnosis

15. NUMBER OF PAGES

22

16. PRICE CODE

A02

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified		