AN INTELLIGENT FAULT DIAGNOSER FOR DISTRIBUTED PROCESSING IN TELESCIENCE APPLICATIONS

by

Pamela Marie Kury

A Thesis Submitted to The Faculty of the

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

In Partial Fulfillment of the Requirements

For the Degree of

MASTER OF SCIENCE

WITH A MAJOR IN ELECTRICAL ENGINEERING

In the Graduate College

THE UNIVERSITY OF ARIZONA

1990

STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at the University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

Signed: Daniel M. Kuny

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

Dr. François E. Cellier — August 27, 1990

F. E. Cellier
Professor of Electrical Engineering

Date

ACKNOWLEDGEMENTS

I would like to extend my thanks to all the people who have supported me throughout my education. I cannot thank my parents enough for all the support and encouragement they've given me. To all my friends, I'd also like to extend my gratitude. They often had faith in me when I had lost it. Many times they helped me to put "my world" back in perspective.

I also owe Dr. Cellier a heartfelt thank you for all the time and effort he put into this project. His support throughout my education was most helpful. Hopefully, someday, I will become the writer he tried to make me.

I would also like to thank my committee, Dr. Schooley and Dr. Zeigler, for their evaluations and helpful suggestions on my thesis.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

A system of self maintenance for fault detection and correction in a highly automated system is presented in this thesis. The self maintenance scheme consists of three parts: a watchdog monitor for fault detection, a diagnoser to pinpoint the exact component failure, and a method of repair. The second part of the scheme, the diagnoser, is developed in detail and applied to the teleoperated THAW telescope. The THAW telescope is an Earth-bound prerunner of the Astrometric Telescope Facility (ATF), an approved payload of the forthcoming International Space Station Freedom (SSF). The simulation of the ATF is modified to produce permanent hardware failures. A diagnoser is created using shallow reasoning and a rule base which pinpoints the most likely failure(s) with reasonable success. Implementation of the watchdog monitor and the repair system is left for future work.

CHAPTER 1

INTRODUCTION

Recently, a variety of systems have been developed for use in space. Trends indicate space technologies will continue to be needed in the future. The cost of space systems is always a major factor in deciding their feasibility. If the system requires humans on site for maintenance and guidance, the cost increases considerably. Therefore, automated processes are the most desirable. Unfortunately, components still tend to break in an automated system and then need repair. To truly make an automated system cost effective, it should include some self maintenance functions.

The scheme for such maintenance basically includes three parts. The first part needs to detect that an error condition exists. The second part needs to determine exactly what system component has failed. Finally, some means of repairing the system is necessary. Figure 1.1 illustrates the proposed self maintenance scheme.

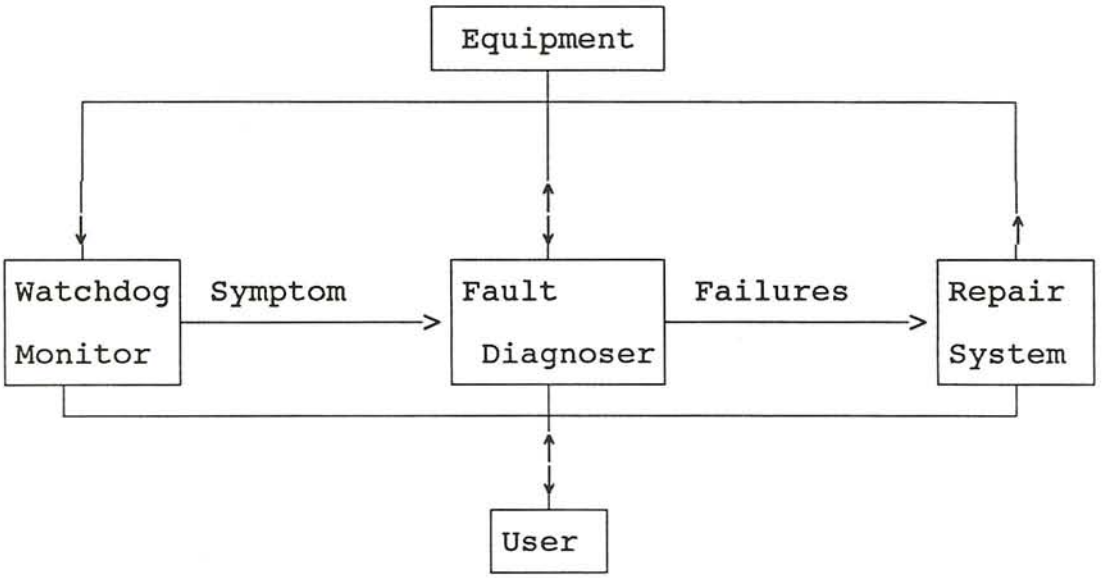

Figure 1.1 - Proposed Self Maintenance Scheme

One possible method of accomplishing the first part of self maintenance, error detection, is a watchdog monitor. The watchdog monitor is an intelligent agent that would supervise the operation of the system and 'reason' about its integrity. The operation of such a monitor could be switched on and off without interfering with the operation of the original process. The monitor would only enhance the existing system. If the monitor found an unexpected result in the system operation, it would only have the power to notify the user or other intelligent agents for further action.

The second logical step in a self maintenance scheme, after detecting something amiss, is to actually pinpoint the fault. Some form of a rule base could be used to diagnose the exact cause of the error condition. The operation of this 'diagnoser' would interact with the original process and might even have the power to issue new commands

to the process for testing. After testing, the diagnoser would then notify the human user or another process of the exact cause of the fault at which point repairs could be initiated or postponed to a more appropriate time.

Finally, some type of automated repair system is needed for a truly autonomous and independent system. Perhaps simple redundancy of components that are likely to fail would be sufficient. Ideally, automated 'repairmen' would be available to correct the error condition or replace faulty components. After repairs were made, the diagnoser test could be run once more to ensure that the fault was fixed and that nothing else was damaged in the process.

This thesis is but one step in creating a high autonomy system for space applications with a focus on the second step of the scheme described above. A method of pinpointing a specific error, given that a defect in the system exists, is presented. Many different methods of fault diagnosis exist. Discussion of these methods is postponed to section 4.1 where all the relevant terms are defined. The diagnoser in this thesis uses shallow reasoning and cause and effect tables (as defined in section 4.1) to determine the exact component malfunction. The diagnoser interacts with the original process and tests a variety of its functions depending upon the probable location of the fault. The watchdog monitor, not developed presently, is assumed to give the location information. Repair of the actual error is also left up to further research. The diagnoser scheme is then applied to an actual system with promising results.

CHAPTER 2

TELESCOPE DESCRIPTION

The concept of self maintenance can be applied to many systems. However, an actual system is needed to test the theory. The error diagnoser part of the self maintenance scheme is applied to a simulation of the THAW telescope at the Allegheny Observatory. Teleoperation of the THAW telescope has been studied as a first step in developing a controller for the Astrometric Telescope Facility (ATF), an approved payload of the forthcoming International Space Station Freedom (SSF). A simulation of the THAW telescope currently exists on a MicroVax at the University of Arizona.[6] It was developed as part of the Telescience Testbed Pilot Program supported by NASA.[1]

2.1 The THAW Telescope

The THAW telescope is located at the Allegheny Observatory in Pittsburgh, PA. The telescope can be considered as consisting of three parts: the observatory, the telescope itself and the measurement instrument which is attached to the telescope. Each of these parts have distinct components and specific operation requirements.

2.1.1 The Observatory

The observatory is defined as the telescope's operating environment, consisting of the dome, the shutter and the observatory room. All of these components should be

operated remotely which will be mandatory in space operations.[1] Of course, the space telescope won't have a "dome" and an "observatory room". These will be replaced by equivalent boundary conditions imposed by the SSF to which the ATF telescope will be attached.

The dome can be rotated 360 degrees for viewing any portion of the sky which is located higher than 30 degrees above the horizon. The dome must have its power and its motor on for rotation. The dome has a predefined starting position. An open position is also predefined and the dome must be in this position to open or close the shutter. The shutter motor must also be on when opening or closing the shutter.

The observatory floor can also move. The floor motor must be on and the telescope must be horizontal for the floor to be raised or lowered. The floor must be in the down position before the telescope is moved or any observations are attempted.

The observatory is also equipped with two room cameras and a set of flood lights. Obviously the flood lights must be on for the room cameras to return any useful information. The flood lights must be off during observations to avoid burning out the photon multipliers on the measurement instrument.

2.1.2 The Telescope

The THAW telescope must be completely automated for teleoperation. Declination and hour angle (or right ascension) measurements define the telescope position. A variety of motors exist for positioning the telescope. Focus and tracking are

also mechanized. Cameras mounted on the telescope itself and on a guider scope are used to select the desired star field.

Declination is defined as the distance from the primary axis ranging from -90.0 degrees to +90.0 degrees. The celestial North Pole corresponds to +90.0 degrees and the celestial equator corresponds to 0 degrees.

The hour angle is the telescope's distance in degrees from the Meridian, a line connecting the celestial North Pole with the Zenith. The hour angle is defined between plus and minus 180 degrees, increasing in the counterclockwise direction. The hour angle is fixed to the location of the observer and changes proportionally to the sidereal time when the telescope is operated in a star tracking mode.

The right ascension uses a coordinate system that is fixed in the sky rather than to the observer as in the case of the hour angle.

<hour angle> = <sidereal time> - <right ascension>

Right ascension increases in the clockwise direction. The telescope's position can be defined by the declination and either the hour angle or the right ascension. The hour angle is more practical when computing the position of the telescope relative to the observatory, while the right ascension is more useful when computing the position of the telescope relative to a celestial object.

The slew motors, step motors and guide motors position the telescope. The slew motors, one for declination and one for hour angle, make the largest adjustments. Then, the step motors are used to make a finer adjustment. Finally, the guide motors are used

to position the scope on the exact star field.

The finder scope is a mirror attached to the telescope to show the telescope's field of view. The finder camera looks through the finder scope to report the same information as the measurement instrument. Thus, there is no need for removing the measurement instrument and no risk of damaging the photon multipliers with hazardous sightings when positioning the scope for a specific observation.

The guider scope and its associated camera enable higher magnification readings with lower precision. The guider system only views one star. It is useful in finding the apparent radius of a star, and to determine current viewing conditions. Centering a desired star in the cross hairs of the guider scope camera also supports accurate tracking of the target star.

The scope is equipped with a tracking system. The tracking motor changes the hour angle as needed to follow an object through the sky as time passes. Unfortunately, this system is not sufficiently accurate, so the guiding system is also needed for keeping the viewing field constant.

The focusing system should also be automated. Once the focus is set, the focus motor automatically adjusts the focus for changes in temperature. Thus, the measurements should always be properly focused.

Notice that automation and control of the THAW simulator reflect the conceptualized system equipped with a teleoperation capability, whereas the simulation of the actual plant reflects current reality. Several of the sensors and controls necessary

for teleoperation of the instrument are currently implemented in the simulator only, and do not reflect actual sensors and controls mounted on the real instrument. The real telescope is not currently equipped for complete teleoperation without a local operator (e.g., the guider system is currently implemented as a screw that must be manually adjusted (no motor) and the same holds true for the focus system).

2.1.3 The Measurement Instrument

The measurement instrument used by the THAW telescope is called the map. The instrument consist of twelve fiber optic probes, so-called photon multipliers. Each probe will be held by a robot arm that can position and orient the probe. The map robot is called "medusa". Currently, the real telescope is not equipped with a medusa system, and the probes must thus be adjusted manually. One probe is used to observe the target star and the others monitor eleven reference stars in the field of view. The multi-arm robot which holds the probes moves the probes into their appropriate configuration for each observation. Each photon multiplier must be brought to a thermal equilibrium before any accurate observations can be taken. Full power is applied only when there exists no risk of overload.

The main purpose of the ATF is to determine if planetary masses exist around selected neighboring stars. The horizontal and vertical positions of a target star from the centroid of the reference star positions in that field of view are determined. The centroid is assumed to be stationary since the reference stars are all much farther away from Sol.

Also, small aberrations of the positions of individual reference stars are further reduced in the process of computing the centroid. By observing the movement of the target star with respect to the centroid over a period of several years, the presence of planetary masses can be determined since the gravitational forces exerted from the planets on their sun will lead to a sinusoidal modulation of the interstellar path of the target star through the galaxy (for further explanation see [1] and [2]).

A device consisting of 1000 alternating transparent and non-transparent lines called a ruler is the only piece of equipment, in addition to the probes themselves, needed for these observations. The ruler is moved slowly across the field of view in both the horizontal and vertical direction for each measurement. Again the ruling process is completely automated for remote operation. Each line has approximately the thickness of a star disk. Thus, when a non-transparent line covers a star no light is registered by the photon multiplier which is positioned on that star. However, when a transparent line covers the star, the full star light is registered by the photon multiplier. Thus, during an observation, each photon multiplier goes through periods of high light intensity followed by periods of minimal light intensity. Due to the linear motion of the ruler, the light intensity follows a sine wave pattern over time. The relative position of the target star can then be extracted from the differences between the phases of the measured sine waves.

2.2 Operation Constraints

The operation of the THAW telescope simulator is completely automated and can be done from a remote site. Many operation constraints exist that are too numerous to mention in this paper (for further information see [1]). Some constraints have already been discussed such as needing the dome in the open position to open or close the shutter or needing the floor to be down before moving the telescope.

If a command is given to the telescope and an unsafe or inappropriate operating conditions exist, then the command is rejected. For example, if the dome is not in the open position and the user sends an open shutter command, then an error message will be returned and the shutter will remain closed. Commands must be within the allowed "transaction envelope" in order to be executed. The user must follow the proper procedures to satisfy these constraints if he expects the telescope to operate properly.

The error diagnoser must also follow these guidelines. In chapter four, the discussion of the diagnoser shows how these constraints are both a help and a hindrance in pinpointing a specific fault. In some cases, the initial conditions for a specific operation cannot be satisfied. When this happens, the integrity of certain components can't be checked. Yet these same operating constraints help to eliminate unnecessary testing and pinpoint the exact fault much faster.

2.3 The Simulated Telescope

The THAW telescope is currently simulated on a MicroVax digital computer. The simulator, which has been coded in Ada, was developed as part of the NASA Telescience Testbed program at the University of Arizona.[6] A commanding MicroVax using OASIS[9] as its command, control, and communication protocol sends instructions to the simulated telescope. Developing this simulation was part of another master's thesis.[6] Consequently, only a brief overview of the simulation will be provided here.

Communication with the telescope simulation can be decomposed into three major tasks. Each task will run simultaneously using the multitasking capabilities of Ada. These tasks are the Command_Receiver, the Command_Processor and a Data_Transmission task. These tasks can be further broken down into subtasks and procedures. Figure 2.1 shows the basic structure of the telescope simulation.

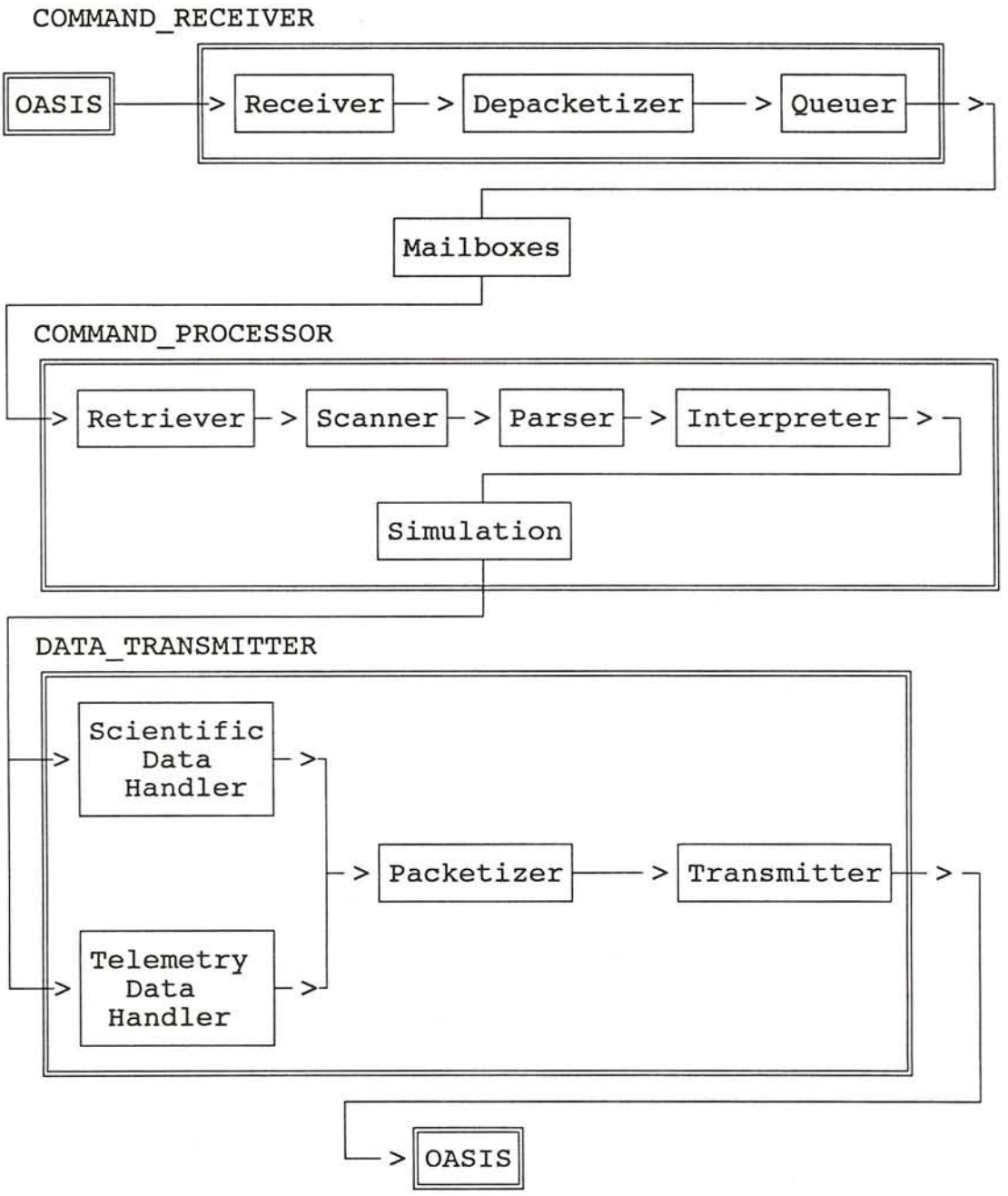

Figure 2.1 - Program Organization

2.3.1 Command Receiver

The Command_Receiver continuously monitors the OASIS communication line. An instruction is sent from the remote controlling computer to the telescope simulator by OASIS. After detecting the command, the Depacketizer part of the Command_Receiver 'unpacks' the instruction and puts it in a form that the Command_Processor understands. The unpacked command is then placed into an appropriate mailbox to await service of the Command_Processor. Selection of mailboxes depends on the type of instruction. There are three types of commands: priority commands, real-time commands, and time-tagged commands.

2.3.2 Command Processor

The Command_Processor takes a command out of a non-empty mailbox. Mailbox selection starts with the priority command mailbox, followed by the time-tagged mailbox and finally the real-time mailbox. The instruction is then scanned for validity. If the command is invalid, an error message is returned to the user, otherwise the command is passed to the parser. The parser recodes the commands to match those in the Command_Code package and passes the new version of the command to the interpreter. The interpreter determines what action needs to be performed and accesses the Tele_Simulation procedure. The Tele_Simulation procedure actually 'does' the operation by modifying the appropriate telescope parameters.

Another part of the Command_Processor is a failure simulator. The failure

simulator simulates transient errors. These errors can be of three types. The first type simulates the local controlling computer detecting a hardware error in the telescope. An error message is returned to the user. The second type of simulated error represents communication errors between the two computers. In this case, the next command is just removed from the mailbox and the Command_Processor continues. The third and final error represents a sensor error when a show status command is issued. All of these errors are transient, meaning they only exist for execution of that command. These errors occur randomly with a frequency depending upon a threshold set by the user. The threshold can be set such that no random errors occur on execution of any command or such that they occur for every attempted command.

2.3.3 Data Transmission

Data_Transmission consists of four main parts: the Status_Data_Handler, the Scientific_Data_Handler, the Packetizer, and the Transmitter. The Status_Data_Handler sends telemetry packets back to the commanding computer through the Packetizer and Transmitter. The four different types of telemetry packets are observatory telemetry, telescope telemetry, MAP telemetry, and clock telemetry. A general telemetry report would include all of these packets. The Scientific_Data_Handler reports camera and measurement instrument information. Both data handlers use the Packetizer and Transmitter to return information to the commanding computer. The Packetizer performs the opposite function of the Depacketizer in the Command_Receiver. The Transmitter

then sends the packet through an Ethernet link to the commanding computer where it is received and processed by the OASIS software.

CHAPTER 3

HARDWARE ERROR SIMULATION

The self maintenance scheme is designed to detect, pinpoint and repair faults in a highly automated system. The error diagnoser section of the self maintenance scheme, which is the main focus of this thesis, identifies faulty system components. In order to test the error diagnoser, the system must have hardware errors to detect. The telescope simulation developed by Lew[6] only implemented transient random errors. The purpose of the newly developed error diagnoser is to detect permanent failures of hardware components in the telescope. Therefore, hardware error simulations must be added to the THAW simulation.

3.1 Theory

Theoretically, each part of the telescope, observatory, and measurement instrument could be disabled by a hardware error. The failure of any component would have a number of noticeable effects on the telescope's operation. The scope of a failure could be confined to its own specific area of the telescope or it could propagate to affect other sections as well.

For example, if the declination sensor failed, all commands that requested the declination position or tried to move the telescope to a new declination would be affected. On the other hand, commands involving the measurement instrument, the observatory or

other parts of the telescope would not be affected at all. However, if the telescope's power supply failed, then all of the telescope functions would be affected and the measurement instrument would be disabled as well.

For the actual telescope, it is not critical to know the exact nature of the hardware error. The effects will become apparent in due course while operating the telescope. However, when simulating the telescope and the hardware errors, it is very important to know the exact effects of an error. If the error is not simulated correctly, then it is unlikely that the error diagnoser will perform correctly on both the simulation and on the actual telescope. Therefore, accurate modeling of the hardware errors is essential.

3.2 Implementation

To actually implement a simulation of selected hardware errors, additional commands and program components were needed. Commands were created to allow selection of the errors randomly, or as chosen by the user. Commands to clear the existing error and to display the current error at the local controlling computer were also added. Various parts of the original simulator had to be modified accordingly. Fifteen hardware errors were selected and simulated. Other errors could be added in the future, but for testing the error diagnoser, these fifteen faults seemed sufficient.

3.2.1 Added Commands

Adding commands to the THAW simulator requires modifying a variety of procedures and packages. A complete explanation of the procedure is found in Lew[6]. A brief explanation of the modifications is presented here. The necessary new commands and their functions are shown in the figure below.

Set error	-	Set a randomly selected hardware error.
Set herror = #	-	Set a specific (#) hardware error.
Show error	-	Show the current hardware error.
Clear error	-	Erase the current hardware error.

Figure 3.1 - List of Added Commands

The words *Herror* and *Clear* are added to the Word_Code package. Each new command requires the addition of a new value in the Command_Code package. New variables were also added to the global variable package. A new variable type *Hard_type* has values ranging from H0 to H15. *Hard_error* is a variable of *Hard_type* and H0 is the default value indicating that all hardware components are working perfectly. When *Hard_error* has any other value, a hardware error is present. Different values of *Hard_error* indicate different component failures. Three flags are also added to simulate broken cameras.

They are: *Fi_blank*, for a broken finder camera, *G_blank*, for a broken guider camera, and *Rc_blank*, for broken room cameras. If a flag is true and a bitmap is requested nothing will be returned, simulating a blank viewing field and a broken camera. For each command record, a boolean value, *Disabled*, is included. The default value of *Disabled* is false and the commands will operate as planned. If *Disabled* is set to true in the Hard_Error_Simulator, the command is not executed and the simulator continues with the next command. Finally, instructions for handling these new commands are added in the Interpreter.

To set a hardware error, the user enters one of the two 'set' commands. If the "set error" command is issued, the interpreter performs a rendezvous with the Create_Herror task at the Herror_Selection point. The task calls the error selection procedure, Herror_Sel, and the error execution procedure, Herror_Exec. The user can also select a specific hardware error by issuing the "set herror = # " command. In this case, the interpreter performs a rendezvous with the Create_Herror task at the Herror_Execution point. The task takes the user's value for the error and then runs the Herror_Exec procedure.

The Herror_Sel procedure randomly selects a value for *Hard_error*. A random number is generated with a seed value and the external VAX-VMS function. The random number is then scaled into fifteen regions and the corresponding Hard_Error is set.

Execution of the hardware error is done in the Herror_Exec procedure. Some of the existing hardware errors simulate component failure. In the simulation, the variables

representing the broken component are set to the 'off' value, even if the user had previously turned that part 'on'. For other errors, nothing happens in the execution routine.

A "show error" command displays the value of *Hard_error* on the screen of the local controlling computer. When testing the diagnoser, it is often necessary to be able to verify its results. After receiving the "show error" command, the interpreter executes the Put_Hard_Error procedure.

The "clear error" command simulates fixing the hardware error. Clearing an existing hardware error is a separate procedure called by the interpreter. The procedure resets the value of *Hard_error* to H0. It also enables any bitmap reports from a formerly broken camera by resetting Blank to false.

3.2.2 Hardware Error Simulation

The Hard_Error_Simulator is a task that runs continuously during the telescope simulation (see figure 3.1 for a full listing). Before a command is executed, the execution task, either Tele_Simulation, the Scientific_Data_Handler, or the Status_Data_Handler, performs a rendezvous with Hard_Error_Simulator at Disable_Check. Hard_Error_Simulator checks the current value of *Hard_error*. If *Hard_error* assumes a value other than H0, then the task proceeds to check the current command code against those that are disabled by that error. When the current command code matches one that is disabled by the hardware error, the *Disabled* flag is set to 'true'. The calling task,

either Tele_Simulation, the Scientific_Data_Handler, or the Status_Data_Handler, checks *Disabled* and executes the command if the flag is false.

For example, if the command is "set dome = open" to move the dome to the open position, the command code is 220160200. Assuming the error is H1, indicating that the dome is frozen in place (see section 3.2.3 for a full explanation of the errors), and the dome is not in the open position, the simulator will try to perform the command. The interpreter performs a rendezvous with the Tele_Simulation task at Do_Dome, which in turn rendezvous at Disable_Check with the Hard_Error_Simulator. The Hard_Error_Simulator checks the current command against those disabled by the error H1. The command code falls within the parameters of the disabled commands and the *Disabled* flag is set to true. Program control is then returned to the Tele_Simulation procedure, which checks for the appropriate initial states. If one of the states is incorrect for moving the dome to the open position, then Unsafe is set to true and the command is ignored. Assuming the initial conditions are met, *Disabled* is also checked and the command is ignored since *Disabled* is true. Finally, the simulator executes the next command or waits for further instruction.

3.2.3 Selected Errors

Fifteen errors are simulated, representing a variety of possible hardware failures. For purposes of testing the error diagnoser, it is assumed fifteen errors are sufficient. Figure 3.2 provides a concise list of the error codes and what they simulate.

Error Code	Simulated Failure
H1	Dome and Shutter Frozen in Place
H2	Telescope Power Off - fuse blown
H3	Observatory Flood Lights Burned Out
H4	Slew Motor Broken
H5	Guide Motor Broken
H6	Step Motor Broken
H7	Ruling Translational Motor Broken
H8	Tracking Motor Broken
H9	Finder Sensor/Camera Broken
H10	Cover Motors Broken
H11	Declination Sensor Broken
H12	Hour Angle Sensor Broken
H13	Guide Clamp Unable to Disengage
H14	Hour Angle Sensor Not Calibrated
H15	Declination Sensor Not Calibrated

Figure 3.2 - Listing of Simulated Hardware Failures

The first error (H1) simulates the dome of the telescope and the shutter being frozen in place. The Herror_Exec procedure sets the outside temperature to a very low value. The following commands are disabled by the Hard_Error_Simulation task: "set dome = open", "set dome = start", "set dome = (angle)", "set shutter = open", and "set shutter = close". Obviously, the dome can't be moved if it is frozen in place. The shutter commands are affected too since ice or snow tends to affect all moving parts exposed to it.

Hard_error (H2) has many effects since it simulates the fuse being blown on the telescope's power supply or some other malfunction that has turned off the supply. The telescope power, ruling power, guide power and motor, focus power and motor, and

tracking power and motor are all turned off by Herror_Exec. Any command that sets these values on again is disabled by the Hard_Error_Simulator. All commands that attempt to slew or step the telescope are also disabled.

The third error (H3) simulates the flood lights in the observatory burning out. The light sensor parameter is set to off and *Rc_blank* is set true by Herror_Exec. The flood lights themselves still appear to be on if they were on previous to setting the hardware error. When a bitmap report from either room camera is requested, a blank value is returned since the room cameras would have nothing to report without lighting.

The next five commands (H4 - H8) all simulate various motors breaking. The slew motor is broken if the *Hard_error* equals H4. All slewing functions are disabled. Calibration functions are also disabled since these functions use slewing. The fifth error (H5) simulates the guide motor breaking. The guide motor is set off in Herror_Exec. All guiding commands are disabled. When *Hard_error* equals H6, the step motor is burned out. All stepping functions are disabled. The seventh hardware error (H7) simulates a broken translational motor for the ruling apparatus. Commands to move the ruler in the away or toward direction are both disabled. The tracking motor is broken when *Hard_error* equals H8. The tracking motor is set to the off state and the command to turn it on is disabled. The tracking control is also turned off so the telescope ceases to track the object and update the hour angle.

The ninth error (H9) simulates the finder camera breaking. The camera state is turned off and *Fi_blank* is set to true. Thus the finder camera will report no information

whenever a "display finder on" command is issued.

The motors that automate removal of the telescope's covers are broken when *Hard_error* is H10. None of the commands to remove or replace the covers will function.

The next two errors (H11 and H12) have far reaching effects. When *Hard_error* is H11, the declination sensor is assumed to be broken. Any command to move the declination is disabled (i.e. slewing, stepping, guiding and calibrating the declination). In addition, the command to report the declination to the user is also disabled. Similarly, hardware error H12 indicates a broken hour angle sensor. All commands to move the hour angle or to report it's value are disabled. It should be mentioned that the hour angle and right ascension are found separately. One value is a mathematical function of the other. However, the hour angle could be correct while the right ascension is incorrect if the clock used to compute the right ascension is broken. In this case, since the hour angle sensor is broken, both the right ascension and the hour angle will be incorrect.

The next error (H13) simulates the guide clamp being stuck in the "on" position. All guiding functions are disabled.

The last two errors (H14 and H15) don't simulate a component failing totally, but rather that the part no longer functions as expected. H14 and H15 represent respectively the right ascension and the declination sensors being out of calibration. Whenever a command is sent to change the position, an offset error value is included in the new final position. Calibration functions are also disabled. The Hard_Error_Simulator also checks

that the requested move with the offset is still in the acceptable range. If the command would exceed the boundaries, then an error message is returned to the user. The user could become quite frustrated when these hardware errors are present if he sends a command that should work and receives a message indicating that the parameters are out of range.

3.3 Future Expansion

Fifteen hardware errors seem sufficient to verify the operation of the error diagnoser, but more errors can be added at any time. The procedures Herror_Sel, and Herror_Exec, the package Global_Variable and the task Hard_Error_Simulator would all need to be modified accordingly.

Presently, only one hardware error can exist at a time. Obviously, in a real system, many faults can occur at once. To simulate more than one error, the variable *Hard_error* could be made into an array instead of a single value. The task Hard_Error_Simulator would also have to be modified to check the current command against those disabled by each of the errors. The procedures Clear_Herror and Put_Hard_Error would also have to be changed. Commands for showing or clearing the whole error array as opposed to a single value could also be added. While the addition of an error array better simulates reality, it is unnecessary for testing the fault diagnoser.

In an actual system, the user never knows when an error will occur. The simulator, however, must be told to create an error. The simulation could be changed to

allow the creation of a hardware error at some random time. Yet, for purposes of testing the error diagnoser, there seems to be no need for randomly occurring errors. It is much easier for the user to set an error and test the diagnoser with the assurance that a fault actually exists.

CHAPTER 4

FAULT DIAGNOSER

The fault diagnoser is the second part of the self maintenance scheme. Once an error condition has been detected by the watchdog monitor, the diagnoser is activated to pinpoint the exact fault. The methods used in creating the diagnoser can be applied to a variety of systems. The Atmospheric Telescope Facility (ATF) is the system used to demonstrate the feasibility of such an approach. The ATF is easily decomposed into three subsystems, the observatory, the telescope and the measurement instrument. A diagnoser or self test is created for each one. At the end of a self test, a listing of the possible error conditions is returned to the user with the most likely one having the greatest weight. The next step in the self maintenance scheme would be the repair of the most likely error. The self test could then be redone to diagnose any further errors, or the user could continue operation of the system under the assumption that the problem had been fixed.

4.1 Theory

The need for accurate automated fault diagnosers is becoming more apparent as systems grow in complexity and become more automated. The reasons for automating a system also justify the creation of automated maintenance systems as well. Many different approaches have been applied with varying degrees of success. The following taxonomy outlines the main techniques presently available.

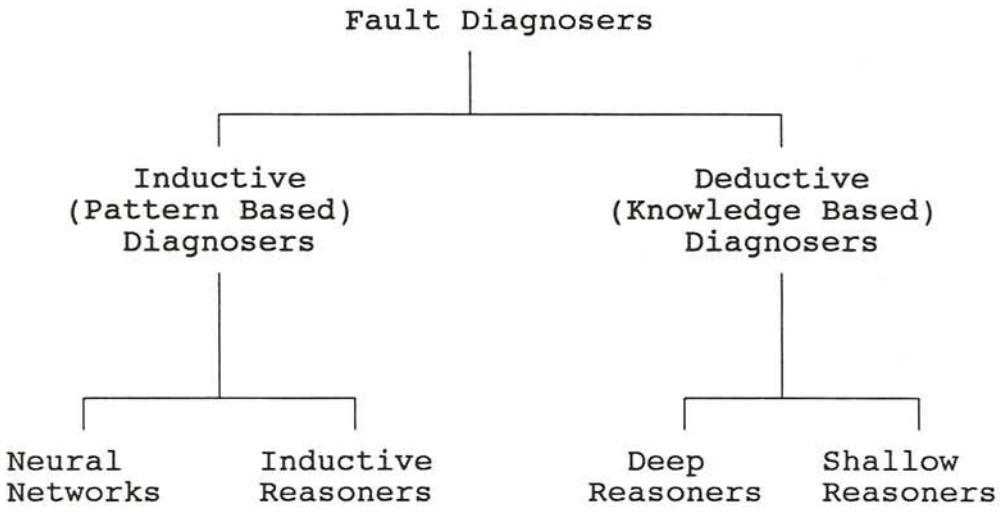

Figure 4.1 - Methods of Fault Diagnosis

All of these methods have been applied to a variety of automated systems.

Inductive diagnostic systems seem to be better suited to systems for which parameter values or even the structure are partially unknown than deductive methods. Inductive diagnosers easily adapt themselves, and they can learn new situations during operation. This capability may be important in the context of long-term fault diagnosis of space-based systems which alter their "correct" behavior over time due to effects of aging and due to technology-related system reconfigurations. A detailed or complete understanding of the system is unnecessary. The inductive method also possesses some drawbacks. A priori knowledge may be difficult to incorporate into the system. The success of these methods is not always guaranteed and the responses of the diagnoser may be difficult to interpret and assess.

A neural network is one method for implementing an inductive diagnoser. Neural networks are able to learn and adapt to dynamic systems which is difficult for the knowledge based deductive system. Neural networks are composed of many layers of interconnected nodes. Paths lead through the network from the input nodes to the output nodes. The paths are established and weighted appropriately during a training phase in which known input and output combinations are presented to the network. The network is trained by comparing its own outputs to the correct output values. The weights are adjusted using an optimization scheme until the discrepancy between the outputs produced by the neural network and the known (correct) outputs is minimized. After the training period has been completed, the network is used to predict outputs even for input combinations not previously encountered. Neural networks will soon be implemented on parallel computer architectures, and they execute rapidly after the learning and testing phases. Neural network fault diagnosis has been applied to an F-16 flight line diagnosis with promising results.[8] Another application of neural networks is fault diagnosis of the fluidized catalytic cracking process in an oil refinery. The methodology worked quite well for single faults, and it still performed respectably when encountering multiple failures.[13] Neural networks do have drawbacks. The learning and testing phases are often painfully slow, and there is no guarantee that the optimization will converge, i.e., it cannot be guaranteed that the neural network is able to learn a particular fault, or that it will still remember a previously learned fault after it was taught another fault.

Inductive reasoners have also been applied to inductive fault diagnosis. The

inductive reasoner will always converge, and the speed of learning can be assessed ahead of time. Inductive reasoning has been applied to classification of faults and switching events in a power system.[5] It has also been applied to the detection of faults in the high altitude horizontal flight of a B747 aircraft.[14] Unfortunately, the inductive reasoner doesn't work well for complex systems with many signals since the time of computation grows exponentially with the number of signals. Also, currently available inductive reasoners don't take advantage of parallel processor architectures.

The deductive reasoner, which is knowledge based, works well for completely defined systems. A priori knowledge can be easily incorporated into this type of diagnoser. Also, deductive diagnosers can be refined in an iterative process. A first crude version of the diagnoser can be built on a few data values only. As more knowledge is gathered about the system, this knowledge can be gradually incorporated into the diagnoser. The results of deductive reasoners are also easy to assess. However, the deductive reasoners cannot easily adapt as a system changes. Complete knowledge of the system is necessary for the deductive reasoner to work correctly.

Many definitions exist for the terms 'deep reasoning' and 'shallow reasoning'. The following definitions are used in this paper. A shallow reasoning system operates on current knowledge only. Alternatively, the deep reasoning system acts on stored as well as on current knowledge. For example, suppose a system is required to forecast tomorrow's high temperature. The shallow reasoner would base its prediction on today's high temperature and possibly on a reading of the current temperature rate of change with

time. If it knew the temperature today was 100 degrees and the gradient was zero, the shallow reasoner would predict the high for tomorrow to be 100 degrees. The deep reasoner would have the high temperature for today, along with readings for many previous days. The deep reasoner would also have a model of weather patterns for comparison when making its prediction. The facts that a stationary high pressure system is over the area and the effects of such a system are also used by the deep reasoner. Again the predicted high temperature for tomorrow is 100 degrees. Both reasoners may reach the same conclusion, but from different information. However, the deep reasoner has more information available, and therefore, it will generally produce more accurate predictions than the shallow reasoner.

A shallow reasoning system is the easiest form of a deductive reasoner to produce. For example, a relatively simple implementation of shallow reasoning can be done using a series of hierarchically grouped if-then-else commands. However, as the size of the system grows, this method can become increasingly cumbersome and redundant. Many statements are needed for tracking down each possible result. Whole program segments may have to be repeated many times, since they occur in various branches of the decision tree. This makes it difficult to maintain such a shallow reasoner. It is difficult to guarantee the integrity and coherence of such a reasoner since a single modification of an if-then-else clause may call for many modifications in various branches of the shallow reasoner. For a more complex system, shallow reasoning can be obtained using a rule base. A rule base is an unstructured linear list of if-then-else clauses which are visited

by the system repetitively and non-sequentially. Rule based systems are more difficult to build, since digital computers are inherently sequential machines, but they are easier to maintain than hierarchically structured if-then-else decision trees, and the size of the rule base grows less rapidly with the complexity of the analyzed situation. Shallow reasoning has been applied to numerous systems. An expert system for fault detection in a CO_2 removal system designed for the forthcoming Space Station Freedom uses this approach.[7] Shallow reasoning is also used in a diagnoser for the loading of liquid oxygen into the Space Shuttle.[2] Unfortunately, shallow reasoning doesn't always provide a good resolution in the case of multiple errors. Even with the rule based approach, the size of a shallow reasoning system grows exponentially with the number of simultaneous faults to be considered. This is a distinct disadvantage of the shallow reasoning approach.

Deep reasoners usually contain a model that runs in parallel with the given system. With the current information, the stored information and the model, the deep reasoner compares its results to the actual system and then reasons about any discrepancies. The size of the deep reasoner grows linearly with the complexity of the system (including simultaneous faults), which gives it a distinct advantage over the shallow reasoning approach. However, deep reasoners are more difficult to build and maintain than shallow reasoners. Also, the deep reasoner may require a longer period of time than the shallow reasoner from the moment when the fault is detected until a decision about its nature will be reached. However, it will report the cause of the fault with a higher accuracy and with

less ambiguity. A qualitative model and a variety of state variables have been used by a deep reasoner applied to a jet-engine oil system.[12] Deep reasoners have also been applied to fault diagnosis in combinatorial and sequential circuits.[10]

Some deductive knowledge based systems have been created that use a combination of deep and shallow reasoning. By incorporating the two methods, the authors have overcome limitations inherent in each. The integrated diagnostic model (IDM) integrates the shallow and deep reasoning methods for fault diagnosis of a simple mechanical system.[3] The Westinghouse Expert Diagnostic System also integrates a variety of knowledge into one conceptual system.[4] In another conceptual study, a shallow reasoner and a deep reasoner were combined to diagnose faults in a Space based robot controlled fluid handling system. In this study, a shallow reasoner is employed to determine quickly and cheaply the general domain of a fault. A deep reasoner, specialized for the suspected domain, then takes over to determine the detailed cause of the fault within the suspected domain.[11]

The shallow reasoning method has been chosen for implementing a fault diagnoser for the Atmospheric Telescope Facility (ATF). The system was fully defined and unlikely to change except by a component failure which the diagnoser is designed to detect. Therefore, inductive approaches were not necessary. The system was not extremely complex, so a deep reasoner was not required. A rule base of error causes and their effects was created for the system. Given the observed effects, the diagnoser should pinpoint the most likely cause. An example is shown in figure 4.2.

Cause	Effect
A	X
B	X
A	Y
C	Y

Figure 4.2 - Cause and Effect Example

If the effects X and Y are discovered, then the possible causes are A, B, or C. However, cause A is more plausible in this case since it is more likely that one error cause, A, has occurred than two, B and C. The diagnoser should return cause A as the most likely error.

4.2 Implementation

The first step in creating an error diagnoser for the Atmospheric Telescope Facility is the creation of the rule base. As mentioned previously, the ATF is decomposed into three sections, therefore three diagnosers are created.

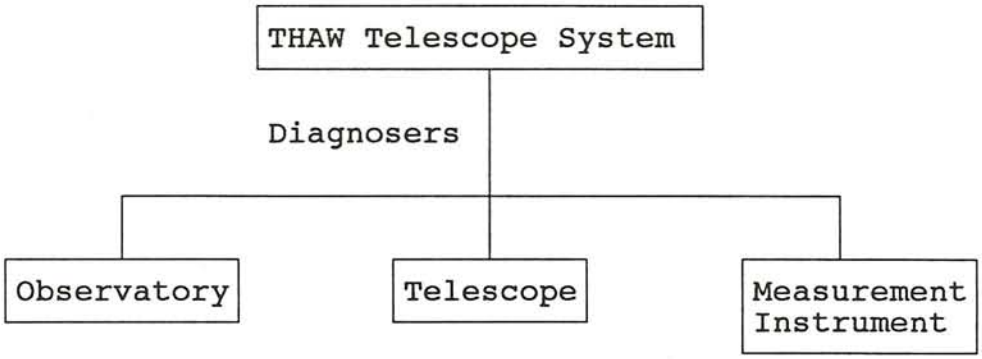

Figure 4.3 - THAW Diagnosers

All of the possible error causes are listed, and the effects of these faults are determined. The observatory cause and effect table is shown below in figure 4.4.

Cause	Effect
Dome power off	Dome fails to move
Dome motor broken	"
Dome frozen in place	"
Shutter power off	Shutter fails to open/close
Shutter motor broken	"
Shutter position frozen	"
Dome not in 'open' position	"
Floor power off	Floor fails to move
Floor motor broken	"
Light switch broken	Lights don't turn on/off
Light bulbs burned out	Lights don't turn on
Lights are off	Room cameras report blank
Light bulbs burned out	"
Finder camera is on	Room cameras don't report
Guider camera is on	"
Room camera 1 broken	Room camera 1 reports blank
Room camera 2 broken	Room camera 2 reports blank

Figure 4.4 - Cause and Effect Table for the Observatory

The telescope rule base is somewhat longer but still basically has the same format as seen in figure 4.5.

Cause	Effect
Floor position not 0	Fail to slew RA
Voltages >= 750	"
Telescope power off	"
RA clamp off	"
Tracking motor on	"
Declination < 30	"
Slew motor broken	"
RA sensor broken	RA doesn't seem to slew
RA sensor not calibrated	Slew RA incorrectly
RA sensor broken	Calibrate RA fails
Meridian sensor broken	"
Telescope power off	Fail to step RA
Tracking motor on	"
Step motor broken	"
RA sensor broken	RA doesn't seem to step
Telescope power off	Fail to guide RA
Guide clamps off	"
Guide power off	"
Guide motor off	"
Guide motor broken	"
RA sensor broken	RA doesn't seem to guide
Floor position not 0	Fail to slew DEC
Voltages >= 750	"
Telescope power off	"
DEC clamp off	"
Tracking motor on	"
Slew motor broken	"
DEC sensor broken	DEC doesn't seem to slew
DEC sensor not calibrated	Slew DEC incorrectly
DEC sensor broken	Calibrate DEC fails
Pole sensor broken	"
Telescope power off	Fail to step DEC
Tracking motor on	"
Step motor broken	"
DEC sensor broken	DEC doesn't seem to slew

Telescope power off	Fail to guide DEC
Guide clamps off	"
Guide power off	"
Guide motor off	"
Guide motor broken	"
DEC sensor broken	"
Floor position not 0	DEC doesn't guide
Voltages >= 750	Fail to slew HA
Telescope power off	"
RA clamp off	"
Tracking motor on	"
Declination < 30	"
Slew motor broken	"
HA sensor broken	HA doesn't seem to slew
HA sensor not calibrated	Slew HA incorrectly
Telescope power off	Fail to step HA
Tracking motor on	"
Step motor broken	"
HA sensor broken	HA doesn't seem to step
Telescope power off	Fail to guide HA
Guide clamps off	"
Guide power off	"
Guide motor off	"
Guide motor broken	"
HA sensor broken	HA doesn't seem to guide
Cover (1-8) stuck	Cover (1-8) fails on/off
Cover motor broken	"
Focus power off	Display not focused
Focus motor off	"
Focus motor broken	"
Focus motor on	Unable to set focus value
Tracking power off	Tracking failure
Tracking motor off	"
Tracking motor broken	"
Telescope power off	"
Declination = 0	"
Finder camera off	Finder information blank
Finder camera broken	"
Guider info. displayed	Finder info. not reported
Light on	"
Telemetry on	"

Guide camera off	Guider information blank
Guide camera broken	"
Finder info. displayed	Guider info. not reported
Light on	"
Telemetry on	"
Telescope power broken	Telescope power fails on
( ) sensor broken	Show ( ) gives wrong info.
DEC clamp stuck	DEC clamp fails on/off
RA clamp stuck	RA clamp fails on/off
Guide clamp stuck	Guide clamp fails on/off
Clutch stuck	Clutch fails on/off

Figure 4.5 - Cause and Effect Table for the Telescope

The cause and effect table for the measurement instrument is shown in figure 4.6.

Cause	Effect
Medusa power off	Medusa fails to move
MAP power off	"
Medusa position sensors broken	"
Translational motor broken	Ruler fails away/home
Rotational motor broken	Ruler fails move x/y
Ruling power off	Ruler fails to move
MAP power off	"
Tracking motor on	"
Probes not at full power	Display ruling blank
Probes' power off	"
Medusa broken	Probes fail to move
Probe sensor broken	Probe fails to orient
Probes broken	Probes fail to report
Supplies broken	Supplies fail to turn on
Supplies broken	Supplies fail to idle
Supplies off	"
Supplies broken	Can't set supplies to max
Supplies off	"
Light on	"

Figure 4.6 - Cause and Effect Table for the Measurement Instrument

As the tables above show, some causes can have more than one effect and some effects can have more than one cause. For example, in the observatory rule base (see figure 4.4), the dome failing to move has a number of causes. The dome power could be off, the dome motor could be broken or the dome could be frozen in place.

The job of the diagnoser is to pinpoint as closely as possible the most likely cause given the observed effects. The diagnoser, or self test, sends a variety of commands directly to the specified subsystem of the simulated ATF and observes the results. Each subsystem (the observatory, the telescope and the measurement instrument) has a variety of functions associated with it. When possible, these functions are separated into non-interacting groups. Each time an effect is detected, the weight of all its possible causes is incremented. If an error condition arises while testing a specific group, the most plausible cause (the one with the highest weight) for the error in that group is stored. The test continues in a similar manner for every group in the selected subsystem. At the end of the self test, the errors from each group are presented to the user. The error with the greatest weight at the end of the test is diagnosed as the most likely cause.

Figure 4.7 illustrates the basic diagnoser approach. Each cause has certain effects and correspondingly each effect can have more than one cause. The asterisks indicate the relationships between the causes and effects. The underlines show the ones encountered in the following example. If effects 1, 2, 4, and 7 were observed, then cause A would have the weight of 2 and cause C would have weight 4. Therefore, the most likely cause is reported as cause C after taking the one with the maximum weight for the tested group

of functions. The other causes with lower weights are not reported to the user.

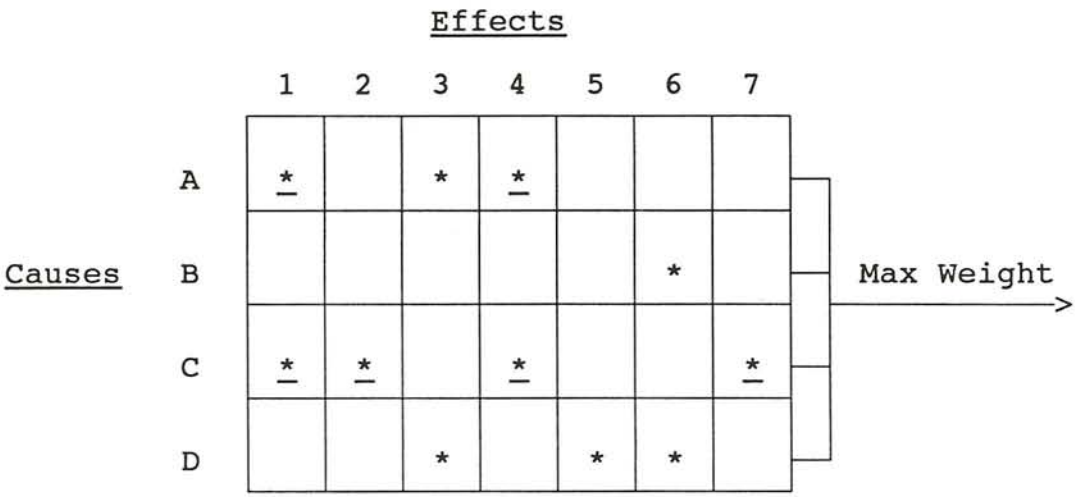

Figure 4.7 - Cause and Effect Mapping Matrix

A new variable record type *Cause* is created for recording the causes of the error conditions. The record has four parameters associated with it. *MAP_t, Observ_t,* and *Tele_t* are all positive integer variables, initialized to zero. *Name* is a character string with a maximum length of 50 specifying a description of the error's cause. Arrays of the *Cause* records are then created. The array *Observ_error* is composed of 20 causes for error conditions determined by the observatory test. *Tele_error* is an array of 50 *Causes* as specified in the telescope test. *MAP_error* is also of length 50 and consists of the causes for errors found in the instrument (MAP) test. Finally the array *Faults*, length 150 contains *Causes* from all of the tests. *Causes* 1-30 in the *Fault* array are reserved for error causes discovered by the observatory test. Similarly, *Causes* 31-90 are filled by the

telescope test, and 91-150 by the MAP test. The extra length of the *Fault* array is needed for shared faults and duplications as explained in the description of the Fault_array procedure.

To be as nonintrusive as possible, the tests have few initial condition requirements. The initial states of the observatory, telescope, or measurement instrument components are saved during testing. At the end of a test, the components' states are restored, barring any debilitating error conditions. Thus, the fault diagnoser interferes as little as possible with the operation of the ATF from the user's perspective.

4.2.1 Self Test

A task called Self_test runs in parallel with the telescope simulation. The task calls the procedures for the diagnoser. Three commands exist for fault diagnosis. "Test observatory" commands the Interpreter to perform a rendezvous with Self_test and start the Test_observ procedure. "Test telescope", operating in a similar manner, performs a test of the telescope functions using the Test_tele procedure. "Test map" calls Self_test, initiating a test of the measurement instrument with the Test_MAP procedure. The *Name* strings in the *Cause* records of the arrays *Observ_error*, *Tele_error* and *MAP_error* are all defined in the Self_test task. Presently, only part of the arrays are filled, allowing new causes to be added easily.

4.2.2 Fault Array

Fault_array is a procedure called by all of the diagnostic procedures. After the functions of a specific group have been tested, the Fault_array procedure sorts out the most likely cause(s) for that group, storing it (them) in the *Fault* array. The procedure stores only the *Cause* with the highest integer value for the specified test (*Observ_t*,*Tele_t*, or *MAP_t*). If more than one *Cause* has the same weight and no *Cause* has a greater one, then all of these *Causes* are included in the Fault array. If no *Cause* has a value greater than zero, nothing is added to the *Fault* array since no recognized error has occurred.

The following example illustrates the Fault_array procedure. Assume that the user is testing the observatory for faults. The Test_observ procedure would be started by the Self_test task. The first group of functions tested by the observatory test are the flood lights and the room cameras. Fault_array is called with the following Ada commands:

```
Last_blank := 1;
First_error := 11;
Last_error := 16;
Numb := 1;
Fault_array(Last_blank, First_error, Last_error, Numb, T_name,
Error_name);
```

The *Causes* for any errors in the group one operations are numbered 11-16 in the *Observ_error* array, as initialized by the Self_test task. Therefore, *First_error* is defined as 11 and *Last_error* is 16. *Last_blank* specifies the first location to fill in the *Fault* array. For the observatory test, the first location is number one. *Numb* indicates the

group number of the functions just tested, in this case number one. *T_name* refers to the diagnoser test name, *Observ*. *Error_name*, also *Observ*, selects the array, *Observ_error* in this case, of the possible error causes.

The Fault_array procedure then checks the value of *Observ_t* in each *Cause* record in the *Observ_error* array. If room camera one is broken, then *Observ_error(13).observ_t* has the value one. This *Cause* record is stored in the *Fault* array and presented to the user as the best possible cause for error in group one. The value of *Last_blank* is also incremented accordingly to point at the next open location in the *Fault* array for the given test.

The Fault_array procedure is called again in this example before the observatory test begins testing the next group of functions. The test of group one turns the flood lights on and off. If the flood lights are still "on", after a command attempted to turn them off, the *Observ_t* parameter of the *Cause* named "Flood Lights Not Off" in the *MAP_error* array is incremented. The *Cause* is included in the *MAP_error* array because it is most often used in the MAP test. For the example, *First_error* and *Last_error* are both set to 37, the number of the specific *Cause*. *Error_name* is changed to *T_map* to indicate that the *Cause* is found in the *MAP_error* array. The Fault_array procedure functions in the same manner as before. After the procedure call, Test_observ resets the value of *Error_name* to *Observ* for future use. The example shows that some *Causes* are shared by more than one test, instead of repeating the record in each test's array. These shared *Causes* are one reason for the extra length in the *Fault* array sections.

Another reason for the extra length is that the Fault_array procedure could store the same *Cause* in the *Fault* array more than once. For example, in the telescope test, a broken declination sensor would be detected by every group testing declination motions. In each case, the *Cause* would be stored in the *Fault* array with its appropriate weight.

At the end of any test, the *Causes* stored in the *Fault* array by that test are sorted. If the same *Cause* occurs more than once, only the one with the highest weight is saved in the array. The other instances are dropped. Finally, all the *Causes* determined by the test and stored in the *Fault* array are presented to the user.

4.2.3 Clear Faults

The Clear_faults procedure clears the specified locations in the *Fault* array and the parameter of the *Observ_error*, *Tele_error*, and *MAP_error* arrays indicated by the calling procedure. Whenever a diagnoser test is run, the appropriate variables in the *Cause* records must first be reset to zero. An example illustrates the operation of the procedure. If the user is testing the observatory, the Test_observ procedure first reinitializes the appropriate *Causes* by calling Clear_faults. The Ada commands are:

```
First_error := 1;
Last_error := 30;
Clear_faults(First_error, Last_Error, T_name);
```

First_error and *Last_error* are the beginning and ending values of the section of the *Fault* array assigned to the selected test. For the observatory test, the values are 1 and 30

respectively. The values for the telescope test are 31 and 90, and those for the MAP test are 91 and 150. In this example, *T_name* is Observ. Clear_faults sets all of the *Observ_t* values of the *Cause* records in the first 30 locations of the *Fault* array to zero. Clear_faults also sets the *Observ_t* values to zero in the *Observ_error*, *Tele_error* and *MAP_error* arrays. Now, no previously diagnosed causes can affect the determination of the exact error cause using the observatory test.

4.2.4 Observatory Test

The user command "test observatory" starts the Ada procedure Test_observ. All functions associated with the observatory are tested by this procedure. The most plausible causes for any detected error conditions are returned to the user. The test is divided into five groups. The first two groups test the room cameras and the flood lights. The third group tests the floor movement. The fourth group tests the observatory dome and shutter. The fifth group checks the operation of the thermometers. The one initial condition for running the error diagnoser on the observatory is that the telescope should be completely horizontal, declination zero. If the declination is not zero, then the testing of the floor motions will be skipped. After the test, the observatory is returned to its pretest state, barring any error conditions that interfere with achieving that state. Warning messages inform the user of any deviations in the final state.

After clearing the old faults as discussed in 4.2.3, the observatory test begins to diagnose any causes for component failure in groups one and two. The flood lights are

turned "on" and their status is checked. Room camera one and two are tested. During the test of these cameras, the finder camera and guider camera systems are disabled. After testing, the finder camera, guider camera and flood light states are restored.

The floor positioning system composes group three. As stated previously, the telescope must be horizontal to test this group, otherwise these tests will be skipped. The floor is moved to position two, if the initial position is not two. If the initial position of the floor is two, the floor is moved to the down, zero, position. Finally, the floor is returned to its original state. Obviously, moving the floor does not make much sense in fully automated remote operation of the equipment. Yet, this test has been included to ensure that the floor is out of the way when the telescope is moved, and to be able to use the self test also as a general test for the correct operation of all system components.

The testing of group four, the dome and the shutter, is much more detailed. The current states of the dome power, dome position, shutter power and shutter position are stored. The dome power is turned on, and the dome position is checked. If the dome is in the "open" position, 20.0, then the dome is moved to 25.0, to verify that the dome can be moved. Next, the dome is moved to the "open" position so the shutter functions can be tested. The shutter is opened and closed. If the dome doesn't seem to be in the "open" position, the observatory test still attempts to open and close the shutter. Of course, the shutter must never be opened during rainfall or dust storms. However, the current simulator does not contain these qualities, and thus, this condition was ignored in

the testbed. This could be added at a later date. If successful in moving the shutter, a error in the dome position sensor is pinpointed. If the dome fails to move from its initial open position and the shutter also fails to move, there is a strong indication that the dome and shutter are frozen in place, especially if the outside temperature supports this assumption. A statement checking the outside temperature verifies the diagnosis of frozen components. Additionally, frozen components are a more likely causes for error than the assumption that both motors have failed simultaneously.

Group five consists of the indoor and outdoor thermometers. No possibility of checking their operation presents itself presently without the use of redundant systems. Redundant systems are not specified for the ATF simulator at this time.

This is a general problem with using the ATF simulator for fault diagnosis. Whenever a fault is reported, two possibilities should be considered: either the equipment has failed or the sensor is broken. For example, if someone calls for an elevator and the indicator light doesn't come on, either the elevator is switched off or the light bulb has burned out. It is impossible to distinguish between these two alternatives except by waiting to see whether the elevator arrives within a prescribed time period.[15] Thus, a deep reasoner may be able to resolve this ambiguity, whereas a shallow reasoner cannot, except through the use of backup sensors. The ATF simulator does not contain provisions for backup sensors. Therefore, component failure and sensor failure can be differentiated only in the case of causes with multiple effects. If all expected effects of a suspected cause are observed, it is more likely that the equipment has failed than all reporting

sensors failing simultaneously. However, in the case of single causes showing single effects measured by a single sensor, it is not currently feasible to distinguish between a malfunction of the component and a malfunction of the sensor.

The observatory test concludes by sending to the user all the *Causes* stored by this test in the *Fault* array. Duplicate occurrences of a *Cause* are dropped, saving only the one with the greatest weight. The *Name* and the value of *Observ_t* are both presented. When more than one *Cause* for error is presented, the *Cause* with the highest weight, *Observ_t* value, is the most likely one. More than one *Cause*, or component failure, could be realistic in some instances. For example, if "Floor motor broken" is given as one possible *Cause* and "Dome motor broken" is given as another, both could be true since their effects are mutually exclusive. The next step in the self maintenance scheme is to repair the component failures. The observatory test could then be used once more to verify correct repairs.

The observatory test detected the following hardware failures accurately (see figure 3.2 for descriptions) : H1 and H3.

4.2.5 Measurement Instrument Test

To diagnose error causes in the Measurement instrument, the user issues the command "test MAP". The Test_MAP procedure is invoked and testing begins. The test is divided into nine groups. There are no necessary initial conditions for the MAP test, but certain conditions, mentioned in the discussion of the relevant group, speed the test's

execution. At the end of the MAP test, all components are restored to their original states, except the probe orientation and the Medusa position. The probes are oriented in the "northeast" direction and the Medusa is in the "home" position at the test's conclusion. Assuming no malfunction in the ruling apparatus, the ruler's final position will be (0,0). Warning messages inform the user of any deviations in the final states. It isn't meaningful to reset the MAP to its original values since an interrupted observation would be ruined anyway.

The measurement instrument error diagnoser begins in the same way as the observatory test. All of the old *Causes* in the designated section of the *Fault* array are cleared, and the *T_map* parameter in all other *Cause* arrays is set to zero. Testing begins by switching the MAP power and Medusa power "on". Group one *Causes* result from any failures detected while doing this. If the MAP power cannot be turned "on", the ruling tests in group five will not be executed.

Group two consists of testing the twelve probes. The probes are oriented first to "southwest" and then to "northeast". After an orientation, the position of each probe is verified, and the appropriate *Cause* is incremented in case of an error.

As the MAP test continues, each of the four voltage supplies is turned "on". The original states of the supplies and their voltage levels are stored for later use. The voltage levels for the probes associated with each supply are then set to "idle", 700 volts. Any discovered faults increment *Causes* in group three.

The initial states of the guider motor, tracking motor, tracking power, and

telescope power are all stored for future use. Group four establishes the necessary initial conditions for testing the ruling apparatus. If any of the initial conditions are not met, the ruling motions cannot be checked. First, the telescope power is turned "on". The declination angle of the telescope is checked. Ideally, the declination would be greater than or equal to 30 degrees at the start of the measurement instrument test. If the declination is inappropriate, the test attempts to slew the value to thirty degrees. With the declination thirty degrees or greater, the tracking power and motor are turned "on".

Group five consists of the error Causes for the ruling apparatus. An explanation of the ruler operation is provided in section 2.1.3. The ruling power is turned "on". Initial conditions for the position of the ruler are not mandatory, but the test proceeds much more quickly if the ruler starts out in the (0,0) position. When the ruler position is not (0,0), the test first attempts to place the ruler there. If the test cannot start the ruler at (0,0) due to a component malfunction, the final ruler position cannot be guaranteed to be (0,0). The ruling motion is first tested in the "X" and "away" directions. Thereafter, it is switched to "Y" and "towards". Finally, the "X" motion is performed again to return the ruler to the (0,0) position.

Testing for groups six and seven does not depend on the ruling functions. The voltage supplies for the fiber optic probes are tested by these groups. The flood lights are turned "off" to avoid damaging the fiber optic probes. The voltage supplies for the probes are set to their "power" state, 1700 volts. Error *Causes* are incremented whenever a malfunction is detected. Group seven contains the *Causes* for setting the voltage levels

back to zero and turning off the supplies.

Finally, group eight establishes the final telescope conditions at the end of the measurement instrument test. The ruler remains at the (0,0) position, assuming no malfunctions. All other components are reset to their states prior to testing. If the declination angle was moved, it is restored to its original position. As in the observatory test, the *Causes* from the appropriate *Fault* array section are reported to the user.

The measurement instrument test detected the following hardware failures accurately (see figure 3.2 for descriptions) : H2, H6, H7, H8, and H11.

4.2.6 Telescope Test

To test functions directly related to the telescope, the user issues the "test telescope" command. The Ada procedure Test_tele tests these functions in eleven groups. The only necessary initial condition for the telescope test is that the floor must be in the "down", 0.0, position. If the declination is greater than 30.0 degrees, testing proceeds more quickly. Assuming no malfunctions, the telescope will have the same final states as those preceding the test.

Like the previous tests, the telescope test clears its section of the *Fault* array and the *Tele_t* parameter in the *Observ_error*, *MAP_error* and *Tele_error* arrays. The first group tests the initial floor condition and sets an error flag accordingly. If the floor is not in the down position, the telescope cannot be moved. The second group sets the level of the voltage supplies to idle, 700 volts, if their value was greater than that initially. If

the voltage levels cannot be lowered the *Bad_vcond* flag is set and the slewing motions of the telescope are not tested. Error causes for group two are sorted and stored in the *Fault* array.

The cover functions are the third group tested. The covers are removed or replaced depending on their previous position. If a cover cannot be returned to its original state, a warning message is sent to the user and the appropriate *Cause* is incremented. Error causes for group three are then sorted, and the most likely ones are stored in the *Fault* array and are reported to the user.

Group four tests the function of the finder and guider cameras. The flood lights are turned "off" for these tests. If the lights fail to turn off, the cameras are not tested. The guider camera is tested first. The states of both cameras are restored at the end of this group's test.

The next group, number five, sets initial conditions necessary for further tests. The tracking motor and tracking power are turned off. The guider motor and guider power are also turned off. If any of these functions fail, the appropriate *Cause* parameter is incremented, and a flag is set. The flags are used by further tests to pinpoint errors. For example, if the guide motor is "on", the boolean variable *Bad_gcond* is set to true. The slewing function does not operate if the guide motor is "on". When *Bad_gcond* is true and the slewing function fails to operate correctly, the cause of the error is not in the slewing apparatus. The fault lies with the guiding equipment. Therefore, before testing the slewing, *Bad_gcond* must be false. Similar flags apply to the tracking motor and

power.

The telescope power is turned "on" for the remaining tests. Group six tests the focus system. The focus power and motor are turned "on". The focus is set to one and then set to zero. The focus motor and power are then returned to their original states.

The next four testing groups all involve moving the telescope. As mentioned previously, the floor must be in the "down" position and the voltage supplies must be less than 750 volts. Guiding functions are tested in group seven, and tracking errors are pinpointed in group eight. The causes for stepping malfunctions are found in group nine. Group ten compiles the *Causes* for errors in the slewing functions.

The declination and right ascension clamps are engaged at the start of the movement tests. The flag *Bad_cond* is set if the clamps cannot be engaged. The slewing functions are not tested if *Bad_cond* is true, since the clamps are necessary for slewing. If it is less than 30 initially, the declination angle is slewed to the start position of 30 degrees. The *Bad_dcond* flag prevents testing the hour angle movement when the declination is less than 30 degrees. At this point in the telescope test, the declination should be 30 degrees or greater, assuming no errors have occurred. Group seven tests the guiding motions. The guide clamp must be "off" to guide the position of the telescope. The direction of motion is positive unless the screw position limits are exceeded. The same procedure is also executed for guiding the hour angle. The guide clamp and motor are also tested. The tracking system is tested next, by activating it and checking that the hour angle changes. The tracking is then disabled, and tracking errors are stored in group

eight. The right ascension should change when the tracking system is off, and this is tested. The stepping functions are tested in group nine. Stepping is only possible if the tracking motor and guider motor are off, indicated by a false value of the *Bad_trcond* and *Bad_gcond* flags. The declination is stepped first. The step value is one if the limits of the position screw will not be exceeded. Otherwise, the value is negative one. The hour angle is stepped in the same way. The hour angle and the declination are now slewed to the home, 0.0, positions in group ten and then returned to their original positions. The user is notified should any of slewing functions fail.

The errors for groups seven through eleven are compiled only if the flag *F_cond* is true. The flag indicates that an error has occurred in the testing of that group's functions. In the observatory test and the measurement instrument test, the segments of the *Observ_error* or *MAP_error* arrays for each group didn't overlap. So each *Cause* could only be incremented by one group. Some *Causes* in Test_tele are incremented by more than one group as figure 4.8 illustrates.

Group	Functions	Cause Array Section	Causes Actually Used
1	Floor	13	13
2	Volts	29-32 (T_map)	29-32 (T_map)
3	Covers	41-48	41-48
4	Cameras	34-40	34-40
5	Init. Cond.	5-25	5,6,8,25
6	Focus	31-33	31-33
7	Guide	9-30	9,21-30
8	Tracking	8-11	8-11
9	Step	20-24	20-24
10	Slew	8-24	all except 14,20
11	Final Cond.	1-26	1-7,10,11,26

Figure 4.8 - Fault Array Sections

For the first six groups, the sections of the *Cause* array do not overlap, making *F_cond* unnecessary. However, group seven contains *Cause* number 25 which could have been incremented in group five. While compiling the faults for group seven, *Cause* 25 should only be stored in the *Fault* array if it was incremented by the tests of group seven. The *F_cond* flag makes this possible. *F_cond* is set to true when an error is detected, while testing a specific group. *Causes* are added to the *Fault* array only if *F_cond* is true. The *F_cond* flag is used in the same manner for all of the following groups.

The last section of the telescope test, group eleven, involves resetting all the components to their initial pretest positions. If error conditions prevent the restoration of the telescope states, a warning message is returned to the user. Finally, the most plausible causes for errors, detected in the telescope test, are presented to the user.

The telescope test accurately pinpointed the following hardware failures (see figure 3.2 for a description) : H2, H4, H5, H6, H8, H9, H10, H11, H12, H13, H14, and H15.

4.3 Operation Example

To actually run the fault diagnoser, the user must first activate the telescope simulation with a "run simu" command. The initial state of the telescope is all systems off, and all components positioned at zero. Next a hardware error should be set. This can be done in two ways as mentioned in section 3.2.1. In the example shown in Appendix A, the hardware error was specified by issuing the command "set herror=1". The dome and shutter are now frozen in place. The observatory fault diagnoser will be run. The initial condition for the test, the declination being zero is met. The user starts the diagnoser with the "test observatory" command. The test begins and messages inform the user of test's progress. All commands issued by the user are shown in boldface type in Appendix A. All messages from the telescope simulation are in capital letters. Messages from the diagnoser are mostly in lower case letters with only occasional use of capitalization. When the test reaches group five, a message is returned to the user stating that the dome may not be in the open position, but the diagnoser is trying to open the shutter anyway. This test helps to distinguish dome sensor error from actual position errors. At the end of the test, the error "Dome position frozen in place" with the weight of three is returned as the most likely fault. The user can then clear the error, to simulate repairs, with the "clear error" command, or continue operations.

CHAPTER 5

RESULTS AND CONCLUSIONS

In order to guarantee high reliability (mean time between unrecoverable failures) in a highly automated system, it is essential to automatically detect and repair most faults in that system. This feature becomes even more critical when the access time for manual repair or replacement of a faulty component is large, such as in the case of a space mission. The cost of space systems is always a major factor in deciding their feasibility, and on site human supervision adds considerably to the cost. Therefore, to be practical and reliable, an automated system must have some way of maintaining itself.

Theoretically, the scheme for such self maintenance should include three parts. The first part needs to detect that an error condition exists. The second part needs to determine exactly what system component has failed. Finally, the third part implements means of repairing the system if necessary.

A watchdog monitor, the first part of the self maintenance scheme, detects erroneous system conditions by comparing its model to the actual system and by reasoning about any discrepancies. The watchdog monitor is not part of the original system, and the monitor's operation has no effect on that system. Upon detecting a fault, the monitor notifies the user of the condition. The user has the option of proceeding with fault diagnosis or continuing operations.

The fault diagnoser is activated by the user. The diagnoser sends commands to

the system and, based on the results, provides the user with a list of the most likely component failures. The list consists of the *Causes* in each test group with the highest weight. The user can issue commands to initiate repair, or proceed with operations while aware of the possible difficulties. After repair, the diagnoser could be run again to verify the repair.

In this thesis, the three part self maintenance scheme is applied to the simulation of the Atmospheric Telescope Facility (ATF). The implementation of the watchdog monitor and repair functions are left for future work. The fault diagnoser is developed and applied with success to the system. Simulated component failures were added to the ATF simulation for testing the diagnoser.

The ATF can be decomposed into three parts: the observatory, the telescope and the measurement instrument. Three diagnostic programs were created to pinpoint the most plausible cause(s) for failure in each section. The watchdog monitor would provide information to indicate which diagnoser to run.

Many methods of fault diagnosis exist. Since the system is completely defined, inductive diagnostic methods were not needed. The shallow reasoning approach was chosen from the deductive methods based on the relative simplicity of the system and the ease of implementation. The diagnoser in this thesis uses a rule based system in the form of cause and effect lists to perform shallow reasoning. Whenever an effect is encountered, its corresponding cause(s) is(are) incremented accordingly. Basically, this method worked well, but in some cases changes were necessary. Since some causes were

so wide ranging, their weight could become too high, overshadowing the actual cause. Therefore, the weights of certain causes were increased by two in order to balance the effect of the wide ranging causes. The decision for additional weighting is design specific and cannot be easily automated.

Consistency checks could be added to make the system even more robust. Presently, only the causes are incremented when effects are detected. If the occurrences of effects were also noted, then the consistency of the diagnosis could be verified. When a cause is reported as the most likely, a check could verify that all of its associated effects were detected as well. A matrix representation as in figure 4.7 would be useful for this. As shown in this example, diagnosis cause A is not consistent with the observed effects, since effect 3 was not encountered.

The proper operation of each diagnoser was verified through exhaustive testing. Each of the fifteen hardware errors were set, and the appropriate diagnosers were run to pinpoint the exact fault. Detection of appropriate initial conditions was also tested by setting faulty conditions and running the test. The test returns messages to the user indicating the faulty initial condition. Preservation of the systems original state was verified in a similar manner. The diagnosers were able to pinpoint the correct fault when the failure occurred within the scope of the test.

Operation constraints of the THAW telescope were quite helpful in pinpointing the exact fault. Operations were ordered within each test to take full advantage of these constraints. The constraints also helped to speed testing by eliminating the need for

testing of certain components.

The time required for running one diagnoser procedure varied depending on the initial state of the system and the nature of the component failure. Certain initial system states enable the diagnoser to perform more quickly as mentioned in chapter four. Also, some failures with wide ranging effects prevent testing of other components and speed the diagnoser. However, the main factor in determining the time required for testing is the system itself. The simulated telescope contains delays which represent the actual time needed by the real system. The diagnoser spends much of its operation time waiting for the simulation.

The use of Ada presented some difficulties as well as some advantages. Ada is considerably different from most programming languages, and becoming familiar with all of its features takes some time. While the number of features was a bit overwhelming, they proved quite useful for the project. The multitasking capabilities of Ada made the entire project possible. The modular format simplified programming and debugging. Ada's exception handling was another useful feature. The ability to define variable types was extremely useful for error simulation and for recording faults.

The proposed self maintenance scheme proved feasible and operates reliably for the investigated high autonomy system. Future work could develop a watchdog monitor system for use with the ATF. Future expansions to the diagnoser itself are also possible. Currently, the diagnoser works correctly for single component failures only. Multiple simultaneous failures could be simulated, and the diagnoser's operation could be tested.

The diagnoser might require slight modification to differentiate these failures. As automation of systems continues, self maintenance methods and especially fault diagnosers will become more sophisticated. This thesis is one step along the path to make high autonomy systems a reality.

APPENDIX A

DIAGNOSER EXAMPLE

run simu
Enter your command
set herror=1
scanner command is set herror=1
The hardware error is
H1
The telescope is free to accept intrusive commands
Enter your command
test observatory
scanner command is test observatory
Beginning Observatory Test
setting initial conditions
Testing Light
LIGHT IS ON
Testing Rcamera1
Testing Rcamera 2
LIGHT IS OFF
no error causes in Group 2
no error causes in Group 3
Testing Floor
FLOOR_POWER IS ON
FLOOR IS POSITIONED AT 0.00
FLOOR_POWER IS ON
FLOOR IS POSITIONED AT 1.00
FLOOR_POWER IS ON
FLOOR IS POSITIONED AT 2.00
FLOOR_POWER IS ON
FLOOR IS POSITIONED AT 1.00
FLOOR_POWER IS ON
FLOOR IS POSITIONED AT 0.00
FLOOR_POWER IS OFF
FLOOR IS POSITIONED AT 0.00
no error causes in Group 4
Testing Dome
DOME IS POSITIONED AT 300.00
DOME POWER IS ON
Checking Dome Power

```
Trying Move Dome to Open Position
Testing Shutter
SHUTTER-POWER IS ON
Dome may NOT be in Open Position
Trying to Move Shutter Anyway
SHUTTER-POWER IS OFF
DOME IS POSITIONED AT 300.00
DOME POWER IS OFF
best possible cause(s) group          5
Dome Frozen In Place          3
no error causes in Group          6
Creating Fault Array
Best Possible Cause(s) is/are
Dome Frozen In Place          3
End Observatory Test
Enter your command
```

REFERENCES

1. Cellier, F. E. (December 1987), "Teleoperation of the Thaw Telescope at the Allegheny Observatory: A Case Study", Telescience Technical Report TSL-004/87, University of Arizona.

2. Delaune, C. I., Scarl, E. A. and Jamieson, J. R. (June 1985), "A Monitor and Diagnosis Program for the Shuttle Liquid Oxygen Loading Operation", Robotics and Expert Systems: Proceedings of the ROBEXS 1st Annual Workshop on Robotics, FL. pp. 167-75.

3. Fink, P. K., Lusth, J. C. and Duran, J. W. (1985), "A General Expert System Design for Diagnostic Problem Solving", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-7, no. 5, pp. 553-60.

4. Havlicsek, B. L. (November 1989), "Integrating Diagnostic Knowledge", IEEE Aerospace Electronic System Magazine, vol. 4, no. 11, pp. 54-9.

5. Kim, C. J. and Russell, B. D. (July 1989), "Classification of Faults and Switching Events by Inductive Reasoning and Expert System Methodology", IEEE Transactions on Power Delivery, vol. 4, no. 3, pp. 1631-7.

6. Lew, A. K. (1988), "Astrometric Telescope Simulator for the Design and Development of Telescope Teleoperation", MS Thesis, University of Arizona.

7. Malin, J. T., Lance, N. (June, 1985), "An Expert System for Fault Management and Automatic Shutdown Avoidance in a Regenerative Life Support System", Robotics and Expert Systems: Proceedings of the ROBEXS 1st Annual Workshop on Robotics, FL, pp. 185-93.

8. McDuff, R. J., Simpson, P. K. and Gunning, D. (1989), "An Investigation of Neural Networks for F-16 Fault Diagnosis", AUTOTESTCON '89 Conference Record: 'The Systems Readiness Technology Conference. Automatic Testing in the Next Decade and the 21st Century', Philadelphia, PA, pp. 351-7.

9. "OASIS CSTOL Reference Manual" (February 1988), LASP, University of Colorado at Bolder.

10. Rogel-Favila, B. and Cheung, P. Y. K. (1989), "Combinational and Sequential

Circuit Fault Diagnosis Using AI Techniques", International Test Conference 1989 Proceedings, p. 950.

11. Sarjoughian, H. S. (1989), "Intelligent Agents and Hierarchical Constraint Driven Diagnostic Units for a Teleoperated Fluid Handling Laboratory", MS Thesis, University of Arizona.

12. Sykes, D. J. and Cochran, J. K. (February 1988), "Development of Diagnostic Expert Systems Using Qualitative Simulation", Artificial Intelligence and Simulation: Proceedings of the SCS Multiconference on Artificial Intelligence and Simulation: The Diversity of Applications, pp. 32-8.

13. Venkatasubramanian, V. and Chan, K. (December 1989), "A Neural Network Methodology for Process Fault Diagnosis", AIChE Journal, vol. 35, no. 12, pp. 1993-2002.

14. Vesantera, P. J. and Cellier, F. E. (1989), "Building Intelligence into an Autopilot - Using Qualitative Simulation to Support Global Decision Making", Simulation, vol. 52, no. 3, pp. 111-21.

15. Wang, Q. (1989), "Management of Continuous System Models in DEVS-Scheme: Time Windows for Event-Based Control", MS Thesis, University of Arizona.