NASA Contractor Report 4317

Model Authoring System for Fail Safe Analysis

Scott E. Sikora

CONTRACT NAS2-12451
AUGUST 1990

NASA

NASA Contractor Report 4317

Model Authoring System for Fail Safe Analysis

Scott E. Sikora
Charles Stark Draper Laboratory, Inc.
Cambridge, Massachusetts

Prepared for
Ames Research Center
Dryden Flight Research Facility
under Contract NAS2-12451

NASA

National Aeronautics and
Space Administration

Office of Management

Scientific and Technical
Information Division

1990

Table of Contents

Chapter 1:

Introduction to the Modeling Authoring System

Before any type of life-critical equipment can be certified, it must undergo an extensive type of reliability testing called *fail safe analysis*. In this type of examination, the equipment design plans are exhaustively checked for critical elements whose error will cause a catastrophic system failure. However, this type of safety analysis is very expensive, and often done only on early design iterations. Thus, design errors introduced through subsequent design improvements may remain undetected, and thereby present serious hazards to life and equipment. However, if the safety analysis procedure could be performed more quickly, more easily and more often, the number of design errors that would reach the final design stage could be significantly reduced. Exploring this automation of fail safe analysis is the focus of the Model Authoring System (MAS).

MAS is a prototype software application that allows a safety engineer to model circuit designs quickly and easily. From these models, MAS can automatically generate either a Fault Tree Analysis (FTA) or a Failure Modes and Effects Analysis (FMEA) for any component in the system. By taking the burden of producing the analysis off the engineer, MAS allows system designers to spend more of their time designing and inspecting the safety analyses, and less time in actual analysis generation.

Before continuing, it is useful to note the basic requirements that a reliability tool should meet. These requirements are listed as follows:

- MAS must be easy enough to use so that it can be utilized on many design iterations with many types of designs.

- MAS must automate the laborious accounting involved in tracing the causes and effects of component failures, while allowing the safety engineer to effectively employ his own knowledge and experience.

- Because the FTA and FMEA techniques are used on existing designs to analyze modifications and improvements, MAS should utilize a bottom-up approach to knowledge capture.

- MAS must output the type of graphical representations and worksheets familiar to safety engineers.

- Because the information entered and created in an analysis is often very useful as reference data, MAS must organize and retain this information for documentation purposes.

This list outlines most of the requirements of an application system that would be ready for immediate, real world use. However, since MAS has been fabricated only as a prototype, many of the goals have been satisfied only in part. By analyzing the strengths and weaknesses of the prototype version of MAS, as will be done in the following chapters, it will be shown that a system fulfilling all of the above requirements could be designed. Many of the problems such a production system will encounter have been addressed by MAS.

1.1. System Overview

The Model Authoring System is a software system utilizing established artificial intelligence and expert system techniques to analyze circuit designs. The engineer generates analyses by first entering a circuit in a graphical format. Next, the behavior of each circuit object is defined, and finally, the particular component to beanalyzed is specified. Thus, there are three separate parts to the system; *schematic capture*, *behavior definition/library objects*, and *analyses generation*. Each of these parts will be briefly outlined below, and then discussed more completely in following chapters.

MAS allows an engineer to enter the circuit design as a frame-based knowledge base in an expert system shell. Each circuit component becomes an object in a knowledge base, which brings the power of object oriented programming and inheritance to the knowledge representation. The circuit is entered with a technique called *schematic capture.* The system begins with a simple bitmap picture of the circuit schematic. The engineer then *captures* the information from the schematic by using a mouse and menu interface to draw boxes around the circuit components. These boxes then represent a circuit object, which is entered into the knowledge base. The boxes are mouse sensitive, which allows the user to directly manipulate the objects in the knowledge base.

From this point, MAS utilizes a method called *behavior definition.* In order to generate the fail safe analyses the behavior for each object must be known. MAS relies on modeling component behavior in the form of IF-THEN rules. Again, MAS utilizes a mouse and menu interface in guiding the user towards writing these rules. For example, a rule defining the behavior of a wire might be written to the effect of,

IF THE STATUS OF WIRE.A IS FAILED
THEN THE STATUS OF SWITCH IS NO-SIGNAL.

For some objects, MAS requires that the engineer write his own behavior definition rules. However, there exists a feature within MAS that automatically defines the behavior for many types of components. These objects, called *library objects*, consist of wires, grounds, batteries, relays, and terminal strips. Since these objects are relatively common in the type of circuit diagrams that will be analyzed, having their behavior defined automatically can greatly reduce the modeling time.

Another feature that exists in MAS is an *abstraction* technique. Circuit components are entered on the hardware level, but this level of detail in a model is often prohibitively complicated. MAS allows an engineer to group objects into a new parent object, which incorporates the behavior of all the original objects. By defining different levels of a model, all of which have the capability to generate a fail safe analysis, MAS can allow the engineer great flexibility in the types of analyses he can create.

Once the circuit is fully modeled, fail safe analyses can be generated literally at the touch of a button. Because of the knowledge representation chosen, an inference engine can be utilized to easily produce the fail safe analyses. Simply by selecting the object and the failure mode, either a Fault Tree Analysis or a Failure Mode and

Effects Analysis can be graphically produced, which the engineer can then inspect for design flaws.

1.2. Organization

This document is organized in six chapters, including this short introduction. Chapter 2 consists of research and background, both on the techniques of traditional fail safe analysis, and of previous work on testing systems. Chapter 3 consists of a detailed overview of the MAS system, specifically describing schematic capture and behavior definition, along with how the fail safe analyses are generated. Chapter 4 presents the theory behind library objects and default behavior in detail. Chapter 5 represents MAS analyses on a simplified real world example. A section of the Emergency Hydraulic System in an experimental F-18 has been isolated, and using MAS the system was modeled and a Fault Tree Analyses was generated with the system. Finally, Chapter 6 presents the results and conclusions of experiences in building and running MAS. It is hoped that the experiences gained from creating this system will provide a proof-of-concept that this type of system could be a useful tool for reliability engineering.

Chapter 2:

Background

Before continuing with an in depth analysis of the Model Authoring System (MAS), it is necessary to take a closer look at a few of the issues involved. This chapter is divided into three separate parts, all of which are vital to the development of MAS. The first is a brief overview of the methodology of fail safe analysis, particularly Fault Tree Analysis and Failure Mode and Effects Analysis. The second section examines the KEE expert system environment on which MAS is based. Finally, the third and final section will be a brief overview of past research in this field, and its relevance to MAS. This background work should give a clearer understanding of the goals and limitations of MAS, which will become evident in later chapters.

2.1. Fail Safe Analysis

In most engineering applications, especially those which are life critical, system safety is of great importance. Therefore, a large methodology has been developed to review safety in all phases of the project life cycle, starting with concept and design. The discipline of system safety analysis has been developed in order to verify the safety of large, complex, redundant systems. Usually, standard reliability testing on the system level is almost impossible, due to the complex nature of the system. Often accidents occur because of complicated interconnection of environmental conditions, component behavior and human error.

Thus, in order to analyze how failures are likely to occur in such a situation, a number of safety techniques have been developed. The list of possible techniques for the evaluation of system safety is quite large and includes, but is not limited to, such techniques as

Preliminary Hazard Analysis, Fault Hazard Analysis, Failure Mode and Effects Analysis, Fault Tree Analysis, Common Cause Failure Analysis, Sneak Circuit Analysis, Software Hazard Analysis, Operating and Support Hazard Analysis, and Management Oversight Risk Trees. All of these techniques demand a thorough understanding of the functioning of the system and an intimate knowledge of possible failures. In the next two sections, two of these analyses are detailed, namely Fault Tree Analysis (FTA) and Failure Mode and Effects Analysis (FMEA). Both of these analyses are widely used, and represent two sides of the analysis scale. The FTA can be considered a deductive analysis, which attempts to explain the causes of system failures. On the other hand, the FMEA is more of an inductive approach, and asks the type of "What if...?" questions vital to engineering design.

2.1.1. Failure Mode and Effects Analysis (FMEA)

The Failure Mode and Effects Analysis is one of the most widely employed techniques for system safety analysis. Basically, the FMEA enumerates possible failure modes for system components, and then traces through the characteristics and consequences of these failures. The FMEA is usually very qualitative, however certain quantitative techniques can be employed.

Most of the major features of a FMEA are illustrated in Figure 2-1. In the first column, the system component is enumerated. In the second column, the possible failure modes for this component are listed. This is followed by the possible causes of failure in the third column and the possible effects of these failures in the fourth. The next column then details the probability of occurrence, followed by the criticality, which tells how vital the component is to system safety and performance. Finally, the last column lists possible actions to reduce the probability of occurrence. Note, in preparing the FMEA, the safety engineer must be very familiar with the workings of the system, as well as with the detailed performance of each of the components.

Obviously, the basic FMEA can be improved on in many ways. Notably, more columns can be added to the table, which could specify such features as possible symptoms and methods of detection for various failure modes. However, this type of analysis is best left to the next section, in which Fault Tree Analysis is described.

2.1.2. Fault Tree Analysis (FTA)

The Fault Tree Analysis is a graphical, deductive, analytical technique for rigorously examining the causes of component failure. The output of a fault tree analysis is a logic tree of the type shown in Figure 2-3. FTA's can be used for either qualitatively studying hazards, or when failure probabilities exist, quantitatively determining risk assessment.

As you can see in Figure 2-3, the Fault Tree is often done on a relatively significant event in system performance. Then, the event is broken down into a number of preceding events, which are organized with logic gates. Because of this organization, the Fault Tree Analysis works best on systems that can be broken down in this logical manner.

The basic principles of Fault Tree Analyses can be shown in a simple example. A sample FTA for a circuit schematic is shown in Figure 2-3, while the circuit this analysis is based on appears in Figure 2-2. The first event is the fault that the motor does not work. The causes of this event can be either a primary motor failure, or a secondary failure, which is no current to the motor. Notice the differences between primary failures and secondary failures. Primary failures, represented by circles, represent independent failures, while secondary failures, represented by rectangles, have underlying causes. The rest of the fault tree is constructed in this way, with each secondary failure requiring more expansion. The remaining part of the tree is fairly self-explanatory, and the table in Figure 2-4 shows some of the more common fault tree symbols and their meanings.

When completed, the Fault Tree Analysis can be an extremely useful safety aid. Not only does it help engineers find trouble spots within the design, but it also all provides exact causes and combinations of these trouble spots. This is why the fault tree is recognized as one of the most useful quantitative and qualitative system analysis aids.

2.2. The KEE Expert System Environment

The purpose of this section is to give a background to the terminology and concepts behind the Expert System Building Tool (ESBT) that was used in this project. In general, ESBT's have three major parts, the *knowledge base*, the *inference engine*, and the *user interface*. The ESBT that was chosen in this project was the Knowledge Engineering Environment (KEE), because of strengths in all three of these areas. The following sections are devoted to explaining each of these three areas.

2.2.1. The Knowledge Base

The knowledge base within KEE is dedicated to representing modeling knowledge, and consists of two separate aspects, *object descriptions* and *rules*. The object descriptions are achieved through the use of *frames*. Frames are tabular data structures which contain *slots* which are filled with data which describes the particular object. In KEE, frames have *inheritance*, which allows a frame to inherit slots and their values from parent frames. Special types of frames, called *instances* can have no children, and are used to represent specific objects. Normal frames are used to describe *classes* of objects.

The second type of knowledge representation used within KEE are rules. Rules are used to change a situation and/or modify the frames, and are represented by IF-THEN statements. IF-THEN statements have a list of antecedents, which make up the IF part of the rule, and a list of conclusions, which make up the THEN part of the rule. These rules are then utilized by the *inference engine* to modify frames in a useful way.

2.2.2. The Inference Engine

The inference engine has two main techniques to support reasoning, *backward chaining* and *forward chaining*. Backward chaining starts with a hypothesized conclusion and determines if there is enough knowledge, both in frames and in rules, to determine if the conclusion is true, or *solved*. This is performed by searching through all the rules which have the hypothesized conclusion as their conclusion. Then, the antecedents for the matching rules become the new hypothesized conclusions, and the process continues until there are no more matching rules or until a given set of conclusions is matched by the data already present in the frame. This type of *goal-driven* technique was popularized by the original MYCIN expert system.

The second technique for inference is forward chaining. In forward chaining, a given fact is added to the knowledge base. Then, the conclusions of the rules which have been newly satisfied are entered into the knowledge base through frame slots. This process continues until all of the conclusions have been traced.

2.2.3. The User Interface

One of the greatest strengths of KEE is the user interface. KEE has a wide variety of graphic interfaces available to the developer, including menus, windows, figures, bitmaps, active images, etc. These graphical capabilities make the system particularly well suited for the types of graphical safety analysis the Model Authoring System is attempting to develop.

KEE offers a large number of other capabilities, including sophisticated framing and inheritance, hypothetical reasoning, object-oriented programming, predicate-logic language, demons, and support for user defined methods, inheritance roles, logic operators, functions and graphics. However, the Model Authoring System deals only with the basic issues described above. Further development of MAS would certainly take advantage of many of the other capabilities, but as the system now stands, the basic system suffices.

2.3. Other Research in This Field

There is a large body of research devoted to the development of analyzing, testing and diagnosing systems. One realm of research that is particularly relevant to MAS is a technique called *model based reasoning*. Model based reasoning concerns itself with the basic task of determining which component failures could result in a *discrepancy* between actual behavior and predicted behavior. For example, if a circuit should output a voltage of X and actually has a voltage of Y, what failures in the circuit components would cause the observed output? This has been called model based reasoning because it is based on knowledge of the circuits' structure and behavior, as well as on observations.

The basic technique for model based reasoning is well established, despite a number of different approaches. When troubleshooting a system, the approach can be broken down into three sections, *hypothesis generation*, *hypothesis testing*, and *hypothesis discrimination*.

Hypothesis generation deals with generating a list of possible failures that could cause the observed outcome. Generating the list could be as simple as an exhaustive enumeration, or could utilize sophisticated observations about behavior and causality. Most modern day approaches use a technique which involves recording justifications along with values during the simulation, then using the justification premises as possible candidates.

The next phase, hypothesis testing, attempts to verify that a generated hypothesis can indeed cause the observed behavior. Again, there are many solutions to this problem. One simple approach is to enumerate all the failure modes of an object, and then use simulation to test all of the possible outcomes. However this has the disadvantage of requiring that the actual failure consist of one of the modeled failure modes. More sophisticated techniques, such as *constraint suspension* or *generate and test integration* have also been used. Constraint suspension involves modeling the system with constraints, and then testing suspects by determining if all other components other than the suspect is working properly. On the other hand, the generate and test systems involve integrating testing knowledge into hypothesis generation.

The final task is hypothesis discrimination, which involves distinguishing between two hypotheses which both cause the predicted output. Again, there are a variety of techniques which have been used. Namely, more information must be gathered, in order to uniquely isolate a hypothesis. This can be done in either of two ways, *probing*, which probes the circuit for additional information, or *testing*, which applies new inputs to the circuit. In both of these cases, the goal is to obtain the correct hypothesis with the minimum of cost.

This three part structure of generation, testing and discrimination matches a large number of troubleshooting systems, including INTER [1976], WATSON [1976], ABEL [1981], SOPHIE [1982], HT [1982], LOCALIZE [1982], IDS [1984], DART [1984], LES/LOX [1985], GDE [1987], DEDALE [1987]. It is important to understand this is a well established technology which is almost ready for application.

However, the Model Authoring System attempts to provide a solution to a related, but different problem. The systems described above attempt to *troubleshoot* the circuit, while MAS deals strictly with *analysis*. While many of the same techniques are applicable, the problem is posed in a slightly different way. While the elements of hypothesis generation, testing and determination are still present, they are quite different.

First, note that instead of asking for the exact cause of an actual problem, MAS is asking for all the causes of a potential problem. Thus, the problem of hypothesis generation and testing become integrated, since MAS must generate all the correct hypotheses that will cause a specified output. The task of hypothesis determination then becomes a non-problem, since the safety engineer is actually concerned with all of the possible hypotheses that are generated, rather than isolating one in particular.

In conclusion, it seems that MAS is addressing a new kind of problem altogether. Although many of the techniques of past research are applicable, namely those of hypothesis generation and testing, MAS organizes them in a different way. It is exactly this new twist to an old problem that makes the problem of automating fail safe analysis so interesting.

ITEM	FAILURE MODES	CAUSE OF FAILURE	POSSIBLE EFFECTS	PROB.	CRITICALITY	POSSIBLE ACTION TO REDUCE FAILURE RATE OR EFFECTS
Motor Case	Rupture	a. Poor Workmanship b. Defective materials c. Damage during transportation d. Damage during handling e. Overpressurization	Destruction of missile	0.0006	Critical	Close control of manufacturing processes to ensure that workmanship meets prescribed standards. Rigid quality control of basic materials to eliminate defectives. Inspection and pressure testing of completed cases. Provision of suitable packaging to protect motor during transportation.
Propellent grain	a. Cracking b. Voids c. Bond Separation	a. Abnormal stresses from cure b. Excessively low temperatures c. Ageing effects	Excessive burning rate; overpressurization; motor case rupture during otherwise normal operation	0.0001	Critical	Carefully controlled production. Storage and operation only within prescribed temperature limits. Suitable formulation to resist effects of aging.
Liner	a. Separation from motor case b. Separation from motor grain or insulation	a. Inadequate cleaning of motor case after fabrication b. Use of unsuitable bonding material c. Failure to control bonding process properly	Excessive burning rate Overpressurization Case rupture during operation	0.0001	Critical	Strict observance of proper cleaning procedures. Strict inspection after cleaning of motor case to ensure that all contaminants have been removed.

Figure 2-1: A Sample FMEA for a Rocket Subsystem

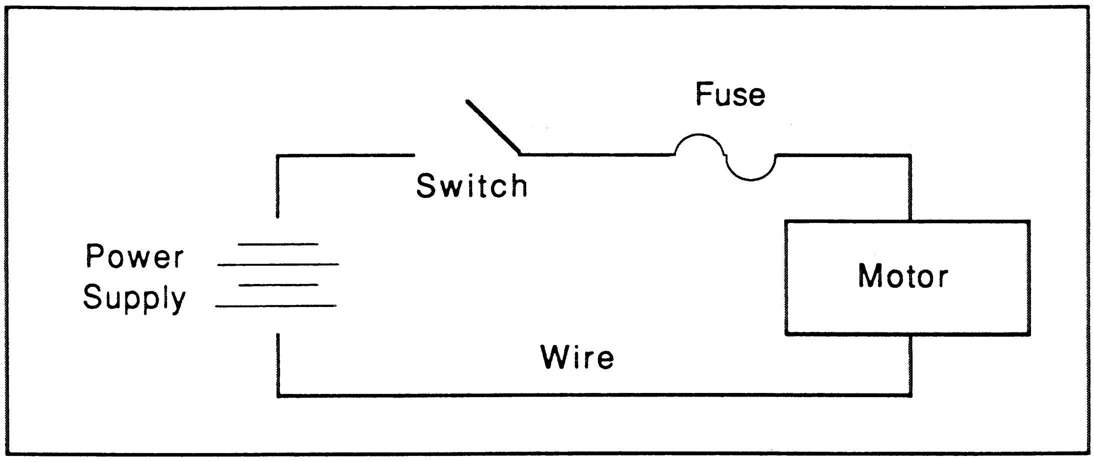


Figure 2-2: Electrical Motor Circuit

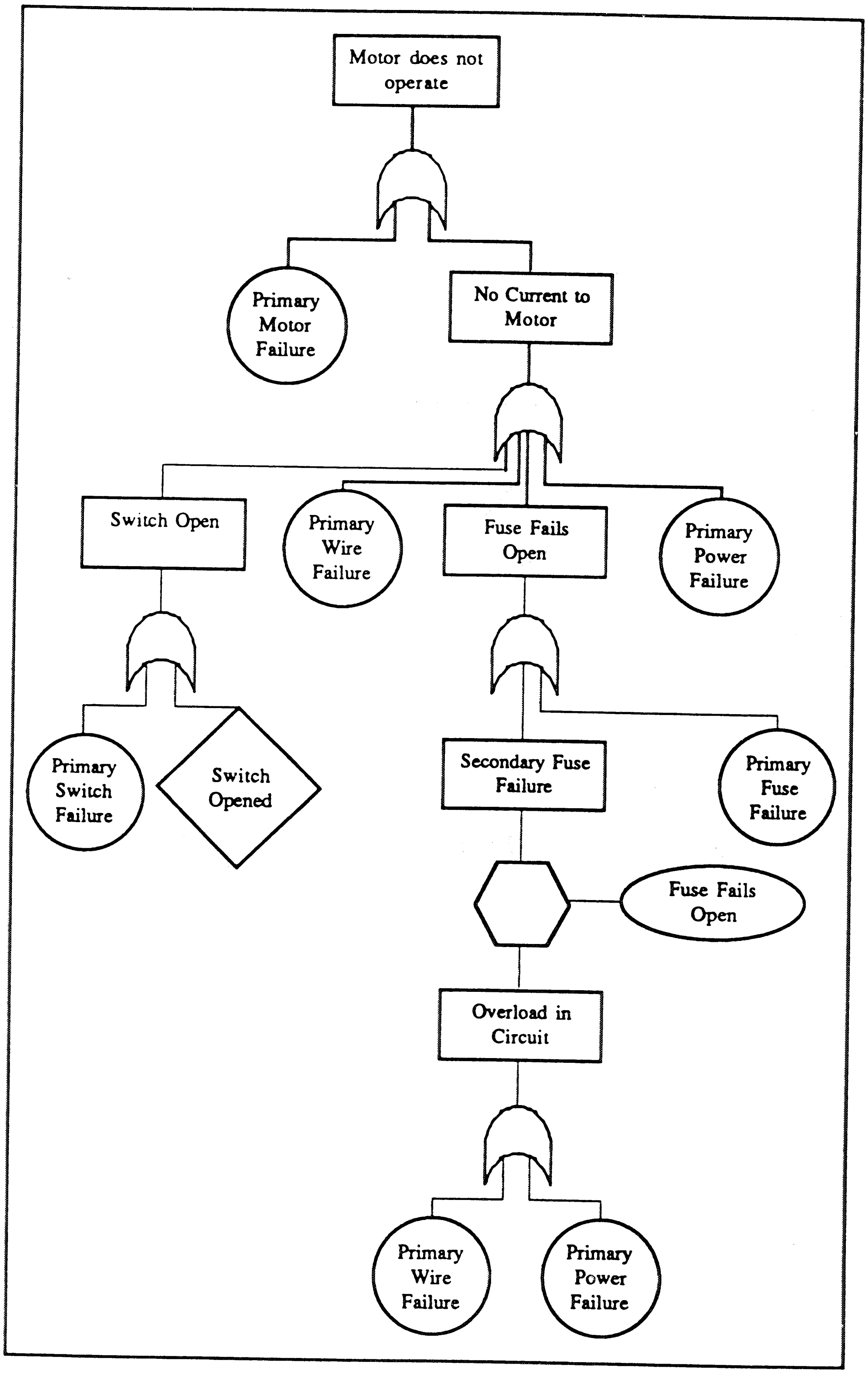


Figure 2-3: Fault Tree for Electrical Motor Circuit

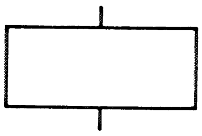
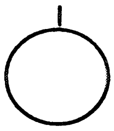
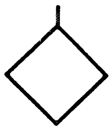
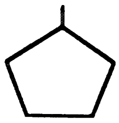
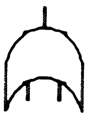
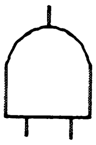
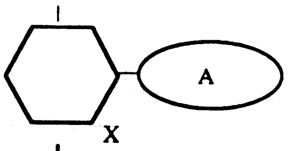
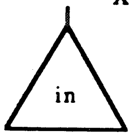
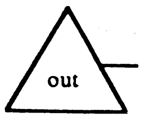
	Fault event; it is usually the result of the logical combination of other events.
	Independent primary fault event.
	Fault event not fully developed, for its causes are not known; it is only an assumed primary fault event.
	Normally occuring basic event; it is not a fault event.
	The union operation of events; i.e., the output event occurs if one or more of the inputs occur.
	The intersection operation of events; i.e., the output event occurs if and only if all the inputs occur.
	Output exists when X exists and condition A is present.
 	Triangle symbols provide a tool to avoid repeating sections of a fault tree or to transfer the tree construction from one sheet to the next. The triangle-in appears at the bottom of a tree and represents the branch of the tree shown somplace else. The triangle out appears at the top of a tree and denotes that the tree is a subtree shown someplace else.

Figure 2-4: Commonly Used Fault Tree Symbols

Chapter 3:

Model Authoring System Overview

It is useful to think of the Model Authoring System (MAS) as consisting of four distinct aspects: *schematic capture*, *behavior definition*, *library objects* and *analyses generation*. It is best to deal with these issues one at a time, illustrating the concepts with a relatively simple example. In this way, the the system can be presented in a logical and orderly way. This chapter deals with three of these issues, namely schematic capture, behavior definition and analyses generation. The next chapter is devoted to the definition and specification of the library objects.

3.1. Structure Modeling

For the engineer, the starting point in circuit safety analysis is the standard engineering circuit diagrams. The information contained in these circuit diagrams is complete enough to allow a safety engineer to perform many types of analyses, including the Fault Tree Analyses and the Failure Mode and Effects Analyses generated by MAS. The first challenge to MAS is to represent the information found in the schematic in a format that is suited to the problem.

3.1.1. Internal Representation

Since MAS is designed as a rule-based deduction system, two things are needed to develop a useful model, rules and objects. In MAS, rules are used to define the behavior of the model, and therefore generate the actual fail safe analysis. However, rules refer to the state of data structures, or objects, that must already exist. Thus, the first step in building an expert system is to define a *knowledge base* containing the objects to which the rules will refer. The next step is to write the rules which define the way the objects behave. Finally, the rules will be searched in a particular way in order to generate conclusions about the objects.

In MAS, the knowledge base objects represent circuit components, the rules describe the behavior of these objects, and searching these rules generates the fail safe analyses. Capturing the circuit object information and placing it into the knowledge base is the first step in generating an analysis. In MAS, this definition step is referred to as *schematic capture*, since the information is entered directly from the circuit schematic.

In order best to understand schematic capture, it is necessary first to understand the target knowledge representation. Because of the original design decision to use the Knowledge Engineering Environment (KEE) expert system tool, the choice of representation has actually already been decided. As described in the previous chapter, KEE stores objects as object-oriented frames with inheritance. More specifically, KEE uses data *units*, each containing a collection of *slots*, which are characteristics which may be inherited from the object's parent unit, or are local to the data unit. Figure 3-1 shows a group of sample KEE units, along with their inheritance roles and some of their slots. Also, certain types of units may be *instances* of a parent unit, as also illustrated in the figure. Instances are specific realizations of a particular unit, which contain all of the unit slots. Many specific instances of a parent unit can be specified, each inheriting the slots of the parent unit. These instances are unique however, in that they can not spawn children.

Some of the slots that are common to all data units are:

- *composite-objects* - If this object is an abstraction object appearing on a higher level, then the grouped components it represents are stored in this slot.
- *levels* - This slot contains all of the abstraction levels which contain this object.
- *failure-modes* - This slot contains the failure modes of the object.

- *hotspots* - This slot contains the highlighted schematic box associated with the object.

- *my-images* - This slot contains the abstract images that represent this object in abstraction levels.

- *status* - This is the current status of the object, either *OK* or one of the failure modes.

- *connections* - This object contains the data units connected to this object.

- *rules* - This slot contains the rules associated with that object.

- *type* - This slot contains the object type. Currently, the type may be one of *generic, terminal strip, battery, ground, wire* or *relay*.

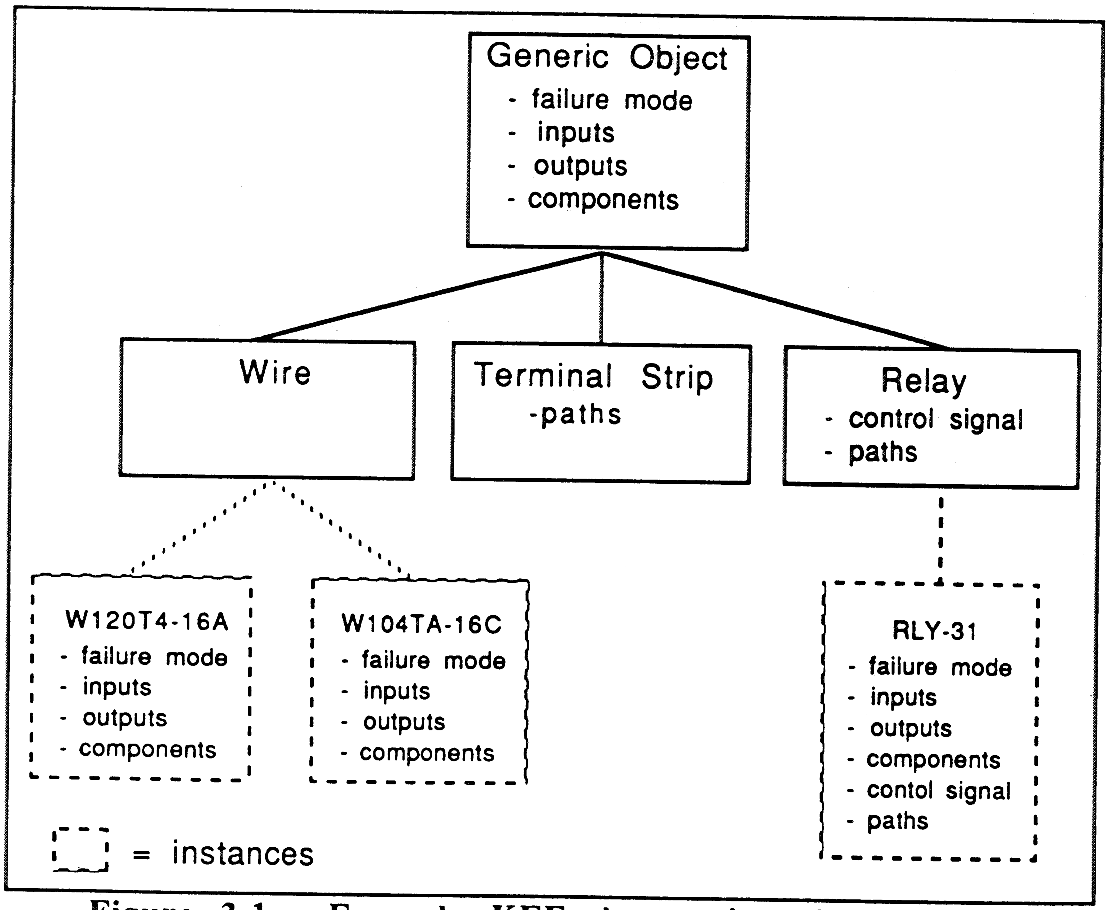


Figure 3-1: Example KEE data units with instances

MAS contains a library of these KEE data structures, which consists of wires, relays, grounds, batteries, terminal strips and generic objects. These library objects act as parents for each individual instance of circuit data units. These data units correspond one to one for each circuit component in the design and thus, make up the knowledge base. Once the circuit knowledge base has been defined by representing components as data units, rules can be written which allow the behavior of the component to be represented.

3.1.2. Schematic Capture

Now that the knowledge representation is established, the next step is to transform the raw bitmap circuit schematics into that representation. In MAS, this is performed through a simple, intuitive mouse and menu technique, which is designed for rapid modeling. The circuit schematic is first called to the screen by opening the corresponding bitmap file. Next, the user uses the mouse to draw a highlighted box around a particular component on the circuit diagram. This component is then modeled as a KEE unit. The system then asks the user to name and identify the component, which MAS will use to transform the information into an instance unit in the circuit knowledge base. This process is continued until all the relevant components on the schematic have been modeled. Note, once the user has highlighted an appropriate area on the diagram, the schematic then becomes "smart", in the sense that the highlighted box is mouse sensitive, allowing the corresponding KEE instance unit to be manipulated by the user.

Once the objects are defined on the schematic, the next step for the user is to specify the connections between the components. Again, this is performed with mouse and menu techniques, which utilize the "smart" spots defined on the schematic. The user first selects a "smart" spot (which corresponds to a component) that must have a connection. Then, according to the component type, the system then guides the user to selecting other "smart" spots (or components) to connect. For example, if the object to be connected is a battery, MAS asks the user to first identify the positive connections, and then the negative connections. In this way, not only the connection information but also certain component specific information is captured into the KEE unit.

As an example, this process can be detailed with a very simple circuit. The schematic for the circuit is shown in Figure 3-2. When the schematic is first entered into MAS, the system knows nothing about the circuit or the types of components it contains. However, once objects have been defined by the user, the knowledge base contains a detailed picture of the structure of the schematic. Figure 3-3 shows the circuit after the user has defined all of the circuit components. Each highlighted box is now mouse sensitive, allowing the box to be directly correlated to the KEE unit in the knowledge base. With the circuit shown, the knowledge base contains six KEE units: *battery* of type battery, *wire-a, wire-b* and *wire-c* of type wire, and finally, *switch* and *motor* of type generic.

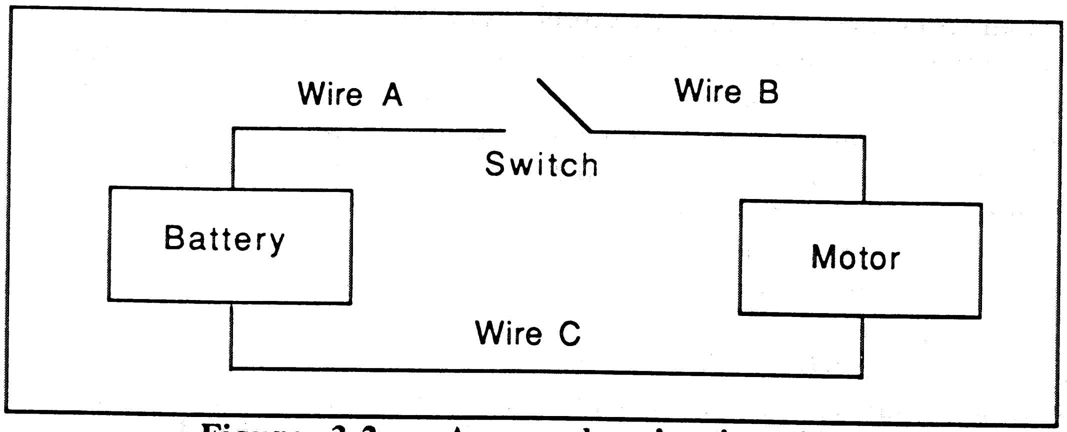


Figure 3-2: A sample circuit schematic

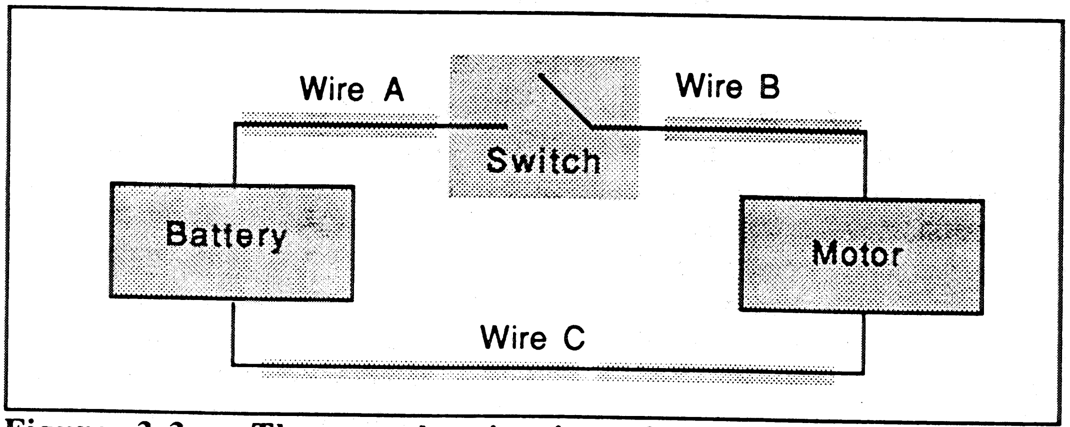


Figure 3-3: The sample circuit defined with "smart" spots

There is one final aspect to schematic capture that should be noted. Once the diagram and the connections have been defined, an abstract view can be engaged. Figure 3-4 shows the example circuit in the abstract view. The abstract view of the circuit replaces the schematic highlighted boxes with generic rectangles that represent the circuit components and connections contained in the knowledge base. Lines represent KEE units of type wire, and the connections are those specified during schematic capture.

Once in abstract view, the user has the option to implement a specific abstraction technique. With this feature, the user can copy the abstract view onto a new "level". Once the new level has been created, objects can be collected into groups which form new data objects. These new data objects can then have their behavior defined with rules, and a completely new FTA or FMEA can be performed in the new level. In turn, this procedure can be repeated many different times creating different abstract levels of the circuit. This feature can be utilized to hide many levels of detail that are not critical to an analysis. Thus, the safety engineer can perform fail safe analysis on the circuit on any level of detail, simply by selecting an appropriate abstraction level. For example, figure 3-5 shows the sample circuit on a new abstraction level. Note, the switch and the battery have been compressed into a new "power" component, which hides some of the details of the original circuit.

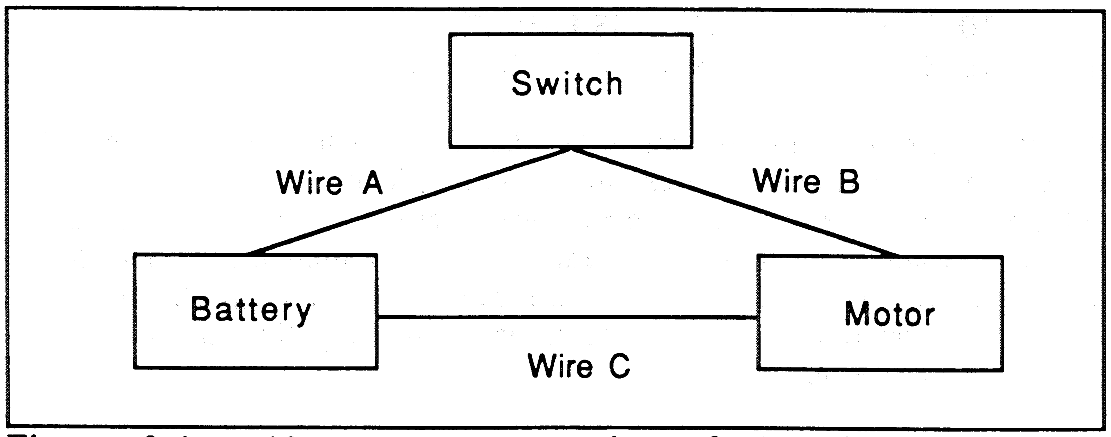


Figure 3-4: Abstract representation of the simple circuit

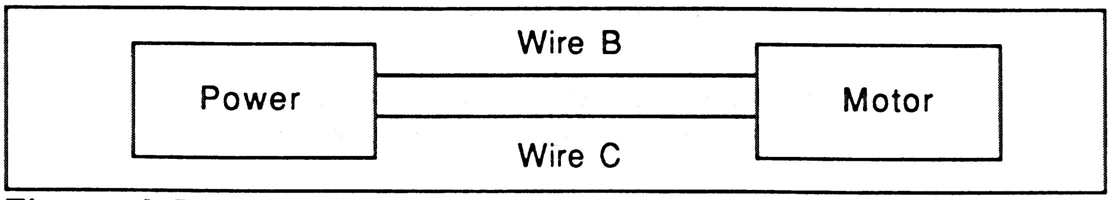


Figure 3-5: Abstract representation of the simple circuit on a new level.

3.2. Behavior Modeling

Once the objects and connections have been defined and stored in the circuit knowledge base, there remains one more step until Fail Safe Analyses can be performed. Namely, the behavior of each circuit component must be defined. In MAS, the behavior is defined in terms of *rules*. More specifically, for each object a series of IF-THEN rules is written, which describe the consequences and conditions of each of the failure modes of the object. Writing these rules completes the circuit model specification.

3.2.1. Rule Capture

Just as MAS guides the user through defining objects and connections, the system also guides the user through rule definitions. Two types of behavior rules are utilized by MAS, *single premise* rules and *multiple premise* rules. Single premise rules describe the behavior of the object during a single failure. For example, if a wire is failed, one consequence might be no signal through the wire. Thus, a single premise rule might be:

```
IF    THE STATUS OF WIRE.A IS FAILED
THEN  THE STATUS OF WIRE.A IS NO-SIGNAL
```

Multiple premise rules, on the other hand, describe the behavior of the system if multiple failures are present. Thus, these rules have multiple IF statements, which match each failure. This type of rule is necessary when describing the behavior of a redundant object that might only fail if multiple failures are present. For example, if a motor could operate under any of three power supplies, a rule for the motor failure might be:

```
IF    THE STATUS OF WIRE.A IS NO-SIGNAL
      THE STATUS OF WIRE.B IS NO-SIGNAL
      THE STATUS OF WIRE.C IS NO-SIGNAL
THEN  THE STATUS OF MOTOR IS FAILED
```

This rule makes the simplifying assumption, that if no current is supplied to the motor, the motor will no longer work.

The two types of rules, multiple and single premise, are used to define the behavior of each of the circuit components. Each component has a group of rules specifying both the causes of each component failure mode and the effects of each component failure mode. In this way, the behavior of the component is specified for all conditions.

MAS is designed to allow both single and multiple premise rules to be written for each component. By exploiting one simple fact about the type of rules that must be defined, MAS can use a mouse and menu technique to write rules. This one consistency is simply that for each statement in the rule, (either the premise or the conclusion) only two things must be known, the name of the object to which the statement refers and the failure mode of the object. Thus, when specifying the rule premises and conclusions, the system only needs to know the object and failure mode. These two bits of data lend themselves well to a mouse and menu interface specification. Namely, the object can be chosen by selecting the appropriate "smart" spot, and the failure mode can then be selected from a menu.

For example, to define a single premise rule MAS first needs to know the object for which the rule is being defined. This type of rule is written by the system in the following way. For each object, there is a menu option called *Define Behavior*. This option guides the user through defining the behavior of the object using the mouse and menu techniques. Basically, the system loops through all of the failure modes of the object. For each failure mode, the premise of the rule is already written, since the current failure mode and the object are known. Thus, the premise of the rule takes on the form IF THE STATUS OF *object* IS *failure mode*. Now, the conclusions of the rule must be specified. In order to get the conclusions, the user clicks on the objects that are affected by this type of failure. For each object the user clicks on, a menu is brought up displaying all of the failure modes of the object. From that point, the user needs only to click on the appropriate failure mode, and the rule conclusion is fully specified. This is performed for all the objects that are affected by the particular failure until the entire rule is written. In this way, the behavior for each component is captured in rules.

MAS defines multiple premise rules in much the same way. First, the user clicks on an option to *Define Multiple Premise Rule*. Next, the rule premises are written by clicking on the appropriate objects and selecting the appropriate failure modes. Once all of the premises are finished, then the conclusions for the rule are defined in exactly the same way: i.e. clicking on the object and selecting the failure mode.

The advantages of this technique are quite obvious. First, although the explanation is a bit complicated, the ease of use of the mouse and menu interface is quite friendly, and allows rapid modeling. Second, by avoiding having the user's typing the rules himself, a much more rigorous and intuitive error control system can be utilized. Finally, the user needs to know very little about expert systems and rule writing. Instead, he simply defines the behavior for each individual object by selecting the components and failure modes which that object affects.

As an illustration, the behavior rules for the simple circuit of Figure 3-2 should be examined. This will serve to clarify how behavior is modeled for each individual component. To keep the example simple, the failure modes for each component have been limited to two, namely *no-signal* and *failed*. While this limitation does remove some of the validity of the model, the basic concepts remain the same.

In the circuit, the first component to be modeled is the **Battery**. The data unit that represents the battery contains all of the information that is needed to write the behavior. The first failure mode for the battery is the primary failure mode, *failed*. If the battery is failed, then the current, or signal, on **Wire A** would disappear, while the status of **Wire C** would remain unaffected. Thus, the first rule describing the battery behavior would be,

```
IF    THE STATUS OF BATTERY IS FAILED
THEN  THE STATUS OF WIRE.A IS NO-SIGNAL
```

The second failure mode for the battery is *no-signal*. Since we have made the limiting statement that all components have two failure modes, some compromises in the modeling process must be made. The case of a battery suffering a failure mode of *no-signal*, does not quite make sense, so this case is ignored.

The second component to be examined is **Wire A**. Again, **Wire A** has two failure modes, *failed* and *no-signal*. If the status of **Wire A** is failed, it is assumed that the wire has broken, therefore no current would reach the switch. Thus, the rule associated with a Wire A failure is,

IF THE STATUS OF WIRE.A IS FAILED
THEN THE STATUS OF SWITCH IS NO-SIGNAL

If Wire A suffers a *no-signal* failure, then obviously the switch will also have no current. Therefore, the rule describing the behavior of Wire A with a no-signal failure mode is,

IF THE STATUS OF WIRE.A IS NO-SIGNAL
THEN THE STATUS OF SWITCH IS NO-SIGNAL

	Failed	No-Signal
Battery	IF THE STATUS OF BATTERY IS FAILED THEN THE STATUS OF WIRE.A IS NO-SIGNAL	
Wire A	IF THE STATUS OF WIRE.A IS FAILED THEN THE STATUS OF SWITCH IS NO-SIGNAL	IF THE STATUS OF WIRE.A IS NO-SIGNAL THEN THE STATUS OF SWITCH IS NO-SIGNAL
Switch	IF THE STATUS OF SWITCH IS FAILED THEN THE STATUS OF WIRE.B IS NO-SIGNAL	IF THE STATUS OF SWITCH IS NO-SIGNAL THEN THE STATUS OF WIRE.B IS NO-SIGNAL
Wire B	IF THE STATUS OF WIRE.B IS FAILED THEN THE STATUS OF MOTOR IS NO-SIGNAL	IF THE STATUS OF WIRE.B IS NO-SIGNAL THEN THE STATUS OF MOTOR IS NO-SIGNAL
Motor		IF THE STATUS OF MOTOR IS NO-SIGNAL THEN THE STATUS OF MOTOR IS FAILED
Wire C	IF THE STATUS OF WIRE.C IS FAILED THEN THE STATUS OF MOTOR IS FAILED AND THE STATUS OF BATTERY IS FAILED	

Table 3-1: Rules for the simple circuit example

The process of analyzing each component failure mode continues for every component in the system. The end result is a group of rules as illustrated in Table 3-1. This group of rules, along with the KEE data units entered into the knowledge base, complete the construction of the model. This new model can now be used to generate Fault Tree Analyses and Failure Mode and Effects Analyses, as is described in the next section.

3.3. Fail Safe Analyses Generation

Now that the objects have been captured from the schematic, the connections between objects specified, and the behavior for each individual object defined, it is finally time to see how a Fault Tree Analysis (FTA) and Failure Mode and Effects Analysis (FMEA) can be generated from the model. Recall, that an FTA is a graphical representation of all of the reasons a particular component can fail, while a FMEA is an examination of all the consequences of a component failure.

3.3.1. Failure Mode and Effects Analysis

The Failure Mode and Effects Analysis is generated through a technique called *forward chaining*. Forward chaining adds a fact to a database, such as THE STATUS OF WIRE.1 IS FAILED, and analyzes all the consequences of that fact. This is achieved by matching the fact to the premises of the rules in the knowledge base. If a premise of a rule is matched, then the conclusion of the rule is added to the knowledge base, and the system starts the search all over again. The result can be represented as a graphical tree representing all of the facts that have been added to the knowledge base during the forward chain. However, this tree can also be interpreted in a different way. Since the rules actually represent the behavior of the system in response to different failure modes, the tree also represents a fail safe analysis. The tree is not only a map of how facts are added to the knowledge base, but is also representation of how the system would behave if the original condition was met. In other words the system has generated a FMEA.

To better understand this concept, refer once again to the simple circuit that has been modeled. If a sample fact such as THE STATUS OF BATTERY IS FAILED, then the system forward chains to examine all of the consequences. In this case, the graphical tree produced is represented in Figure 3-6. With the initial fact added to the database, MAS examines all of the rule premises which are now true. In this, case the first rule to be matched is:

IF THE STATUS OF BATTERY IS FAILED
THEN THE STATUS OF WIRE.A IS NO-SIGNAL

Therefore, the fact THE STATUS OF WIRE.A IS NO-SIGNAL is added. This fact in turn matches the rule,

IF THE STATUS OF WIRE.A IS NO-SIGNAL
THEN THE STATUS OF SWITCH IS NO-SIGNAL.

Which adds another fact to the database. This process continues, until the final event, which is motor failure. Thus, the system has generated a FMEA using the behavior modeled in the rules.

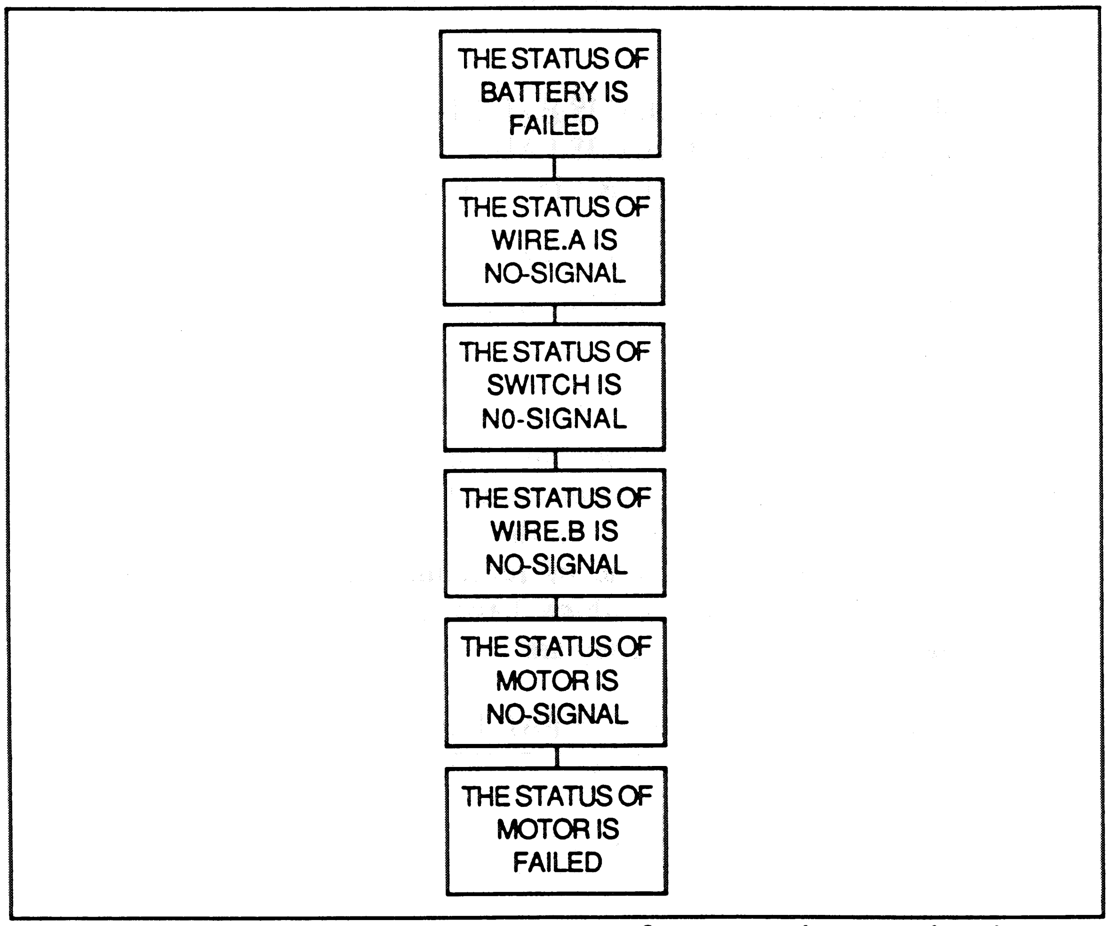

THE STATUS OF BATTERY IS FAILED
THE STATUS OF WIRE.A IS NO-SIGNAL
THE STATUS OF SWITCH IS N0-SIGNAL
THE STATUS OF WIRE.B IS NO-SIGNAL
THE STATUS OF MOTOR IS NO-SIGNAL
THE STATUS OF MOTOR IS FAILED

Figure 3-6: FMEA produced for the simple circuit

3.3.2. Fault Tree Analysis

The fault tree analysis is generated in almost exactly the same way as the FMEA. However, to generate the FTA, MAS uses *backward chaining*. For example, the system takes a fact, and searches for all the rules whose conclusion matches the fact. For each of those rules, the premises are then added to the knowledge base. Then the system backward chains off of the new facts. Again, the output can be represented as a graphical tree, and again the tree is not just a map of how facts were added to the knowledge base. Rather, the tree shows all of the facts that might lead to the truth of the original fact. In other words, the system has generated a Fault Tree Analysis.

Again, by referring to the example circuit, an actual FTA is illustrated in Figure 3-7. This time, the fact THE STATUS OF MOTOR IS FAILED was added to the database. This particular fact matches the conclusion of two rules:

```
IF     THE STATUS OF WIRE.C IS FAILED
THEN   THE STATUS OF MOTOR IS FAILED
       THE STATUS OF BATTERY IS FAILED

IF     THE STATUS OF MOTOR IS NO-SIGNAL
THEN   THE STATUS OF MOTOR IS FAILED
```

Notice, either of the two conditions could have caused the fact to be added to the knowledge base. This is represented in the FTA by the use of an OR logical gate. From this point, MAS backward chains from the facts that have just been added to the database. Since there is no rule conclusion which matches the fact THE STATUS OF WIRE C IS FAILED, the end of this line of reasoning has been reached. However, there are two rules which have the conclusion, THE STATUS OF MOTOR IS NO-SIGNAL. They are:

```
IF     THE STATUS OF WIRE.B IS FAILED
THEN   THE STATUS OF MOTOR IS NO-SIGNAL

IF     THE STATUS OF WIRE.B IS NO-SIGNAL
THEN   THE STATUS OF MOTOR IS NO-SIGNAL.
```

Hence, the system continues backward chaining. This process continues until all lines of reasoning are exhausted. Therefore, the Fault Tree Analysis is created in much the same way as the Failure Mode and Effects Analysis.

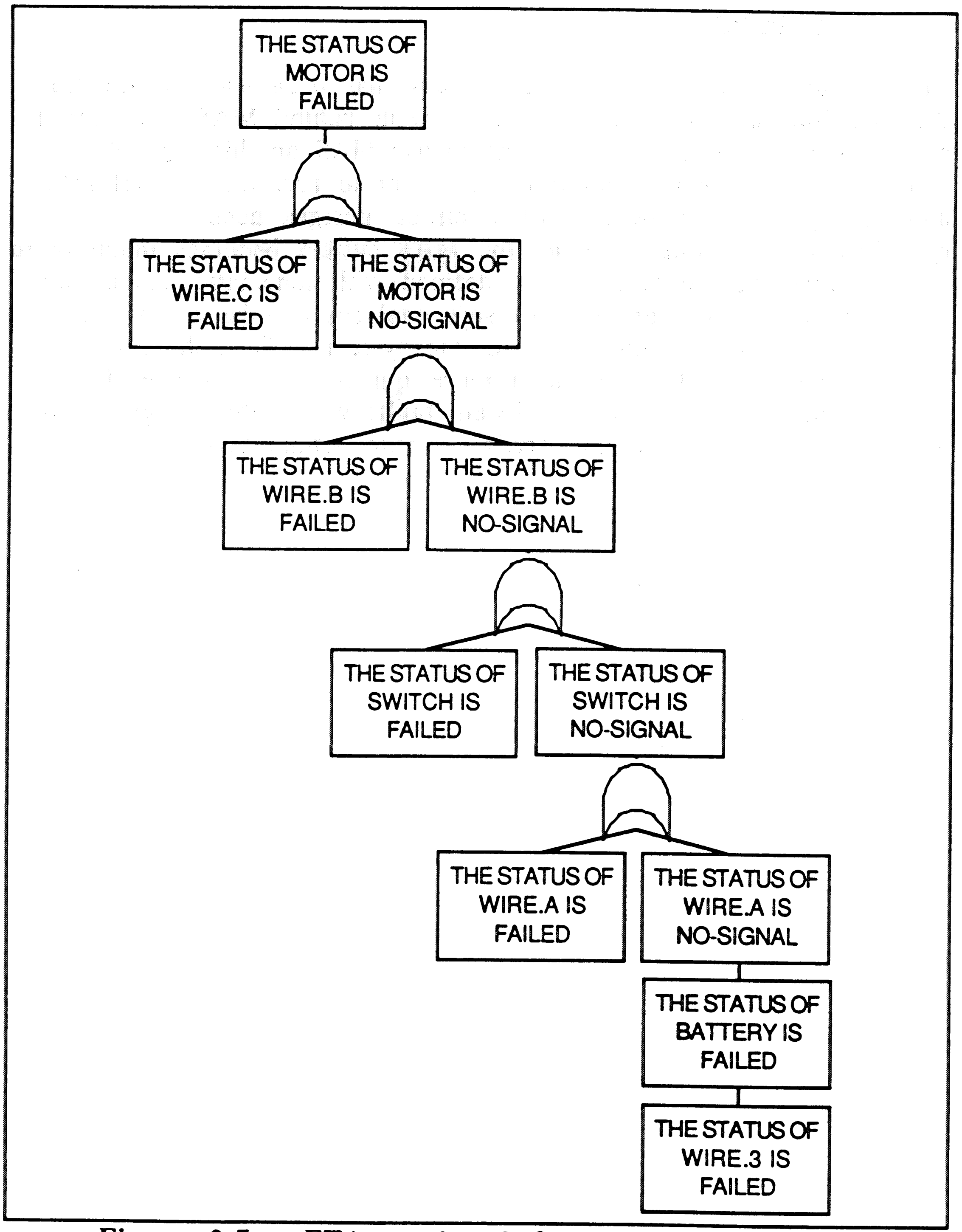

THE STATUS OF MOTOR IS FAILED
THE STATUS OF WIRE.C IS FAILED
THE STATUS OF MOTOR IS NO-SIGNAL
THE STATUS OF WIRE.B IS FAILED
THE STATUS OF WIRE.B IS NO-SIGNAL
THE STATUS OF SWITCH IS FAILED
THE STATUS OF SWITCH IS NO-SIGNAL
THE STATUS OF WIRE.A IS FAILED
THE STATUS OF WIRE.A IS NO-SIGNAL
THE STATUS OF BATTERY IS FAILED
THE STATUS OF WIRE.3 IS FAILED

Figure 3-7: FTA produced for the simple circuit

3.4. Conclusion

Throughout this chapter, a simple circuit has been used to illustrate the modeling and fail safe analysis concepts behind MAS. In fact, it almost seems to be a waste of time to use MAS on this type of example, since it would certainly be easier to perform the fail safe analysis by hand. However, when circuit designs become complicated, the value of using the MAS system becomes much more clear. Once the model has been defined with schematic capture and behavior definition, then the fail safe analyses can be performed on any component and failure mode in the system. Thus, the fail safe analysis can be performed much more quickly and consistently, allowing the safety engineer to locate faults within the design. This goal, above all others is the driving reason behind this prototype.

Chapter 4:

Library Objects and Their Role in MAS

Now that the basic operation of the system has been laid out, it is time to point out one important feature that has been ignored up to this point. As the system has just been described, rules have to be defined for every system component. However, writing detailed rules for every component is usually not necessary. Certain common objects can be given default behaviors which will greatly simplify the modeling process. For example, the behavior of a wire usually does not change, even when it is connected to non-standard components. Therefore, when defining one of the commonly used components, or *library objects*, the user only has to supply the component type and connections, and the behavior rules will be automatically written. Non-standard objects, called *generic objects*, can be defined, but their behavior rules must be specifically written.

As the library objects are defined now, there are a number of assumptions made about their behavior. These assumptions have been made in order to both simplify the modeling process and tailor it towards fail safe analysis. Currently, MAS supports five library objects; wires, batteries, terminal strips relays and grounds. These objects have been chosen to satisfy the requirements of typical real world circuit, which is described in Chapter 5. The rest of this chapter is dedicated to describing the modeled behavior of the five library objects that exist in MAS.

4.1. Directionality

Perhaps the most important behavior idea that has been introduced by defining the library functions is one of *directionality*. In order to model the behavior of the library objects, the concept of input and output connections has been introduced. By defining each object's connection as either an input or output, the model captures a sense of electrical directionality.

Directionality is particularly useful in fail safe analysis, because for any component failure, the failure will affect the outputs of the object, rather than the inputs. While this is not obviously true all of the time, this idea does make a good simplifying assumption. For example, if a relay fails open, the failure does not affect the input wires but does have an effect on the output wires. In this case, directionality accurately models real world behavior. However, if the relay were to short, it could have some effect not only on the input wires, but also on other aspects of the system. In MAS, the types of failures that affect both the input and output connections of the component were ignored, and a very simple set of failure modes that affect only the outputs of an object were utilized.

4.2. Failure Modes

As described before, each object has a number of different *failure modes*. Failure modes exist as a way of categorizing the ways in which a particular circuit component can fail, and the type of failure mode suffered by an objects has direct bearing on other objects. As a result, many types of failure modes were utilized throughout the definition of both library and generic objects. However, the failure modes that were chosen are a small subset of the number of possible modes. For example, generic objects have only two failure modes, *ok* and *no-signal*. Library objects have only a few more. Obviously, realistic modeling with such a limited number of failure modes is not possible, and the failure modes chosen have quite a broad definition. Nevertheless, the modes chosen for the modeling serve the proof-of-concept nature of this thesis.

4.3. The Library Objects

The following section describes in detail each of the library objects in MAS. In general, it is assumed that each object has been defined with both input and output connections. The failure of a library object then affects these connections in some way. The behaviors are defined, as always, with IF-THEN rules.

4.3.1. *Battery*

The battery's output connections have been defined as the wires connected to the positive terminal. If the battery were to fail, then these wires would be affected, since there would be no voltage, or signal on the wires. On the other hand, the input connections have been defined as the ground wires. Following the same line of reasoning, if the wires that are connected to the negative terminal fail, then the battery would fail. Thus, since ground connection failure affects the battery while a battery failure does not affect the ground wires' behavior, the ground wires become the inputs to the battery. In short, the default behavior of the battery has been defined as follows.

```
IF      THE STATUS OF negative-wire-1 IS NO-GROUND
        ...
        THE STATUS OF negative-wire-n IS NO-GR0UND
THEN    THE STATUS OF battery IS FAILED

IF      THE STATUS OF battery IS FAILED
THEN    THE STATUS OF positive-wire-1 IS NO-SIGNAL
        ...
        THE STATUS OF positive-wire-n IS NO-SIGNAL
```

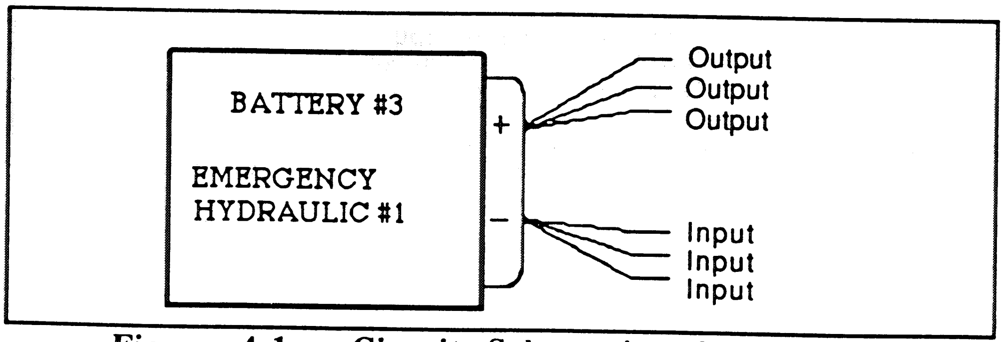


Figure 4-1: Circuit Schematic of a Battery

4.3.2. Terminal Strip

For a terminal strip, certain assumptions are made to simplify the model. It is assumed for a terminal strip that the inputs and outputs of the component are dependent on whether the terminal strip is either the input or the output device for the object connected to each wire. Thus, the terminal strip essentially translates faults from input wires to output wires. If the terminal strip itself is failed, the assumption is made that all the connections are invalid, therefore a fault is propagated to all output wires.

Thus, the default behavior of the terminal strip is as follows.

For each input wire, the following rules are defined:

IF THE STATUS OF *input-wire* IS NO-SIGNAL
THEN THE STATUS OF *output-wire* IS NO-SIGNAL

IF THE STATUS OF *input-wire* IS NO-GROUND
THEN THE STATUS OF *output-wire* IS NO-GROUND

Finally, for each output wire, the following rule is written:

IF THE STATUS OF *terminal-strip* IS FAILED
THEN THE STATUS OF *output-wire* IS NO-SIGNAL

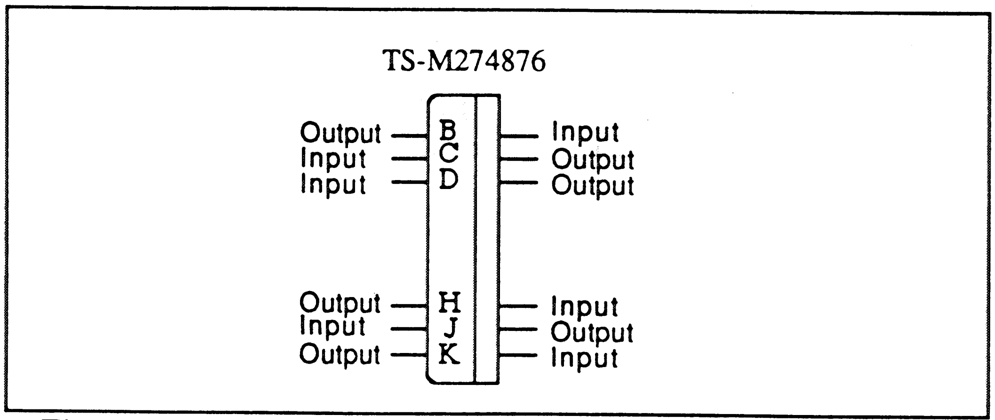


Figure 4-2: Circuit Schematic of a Terminal Strip

4.3.3. Wire

The wire is by far the most common type of component encountered in the schematic. Therefore, the behavior for the wire must be entirely generic, and depend in no way on other aspects of the objects connected to it. To enforce this type of behavior, the failure modes for the wire have been kept quite simple. Basically, there are three failure modes defined for a wire, *no-signal, no-ground* and *failed.* The no-signal failure mode is a general way of stating that there is no current through the wire. In the case of motor, the failure mode of no-signal on the input wires would be cause for the motor not to run. The second failure mode, no-ground, occurs when components must be properly grounded in order to run. For example, if a battery is not grounded properly, then the battery would fail, which would then cause a no-signal on the output wires. However, both these behaviors are dependent on the type of object connected to the wire. Since there is no way of telling to what object a wire will be connected, the behavior is defined for only one failure mode, failed. The default behavior of a wire is defined as

IF THE STATUS OF *wire* IS FAILED
THEN THE STATUS OF *wire* IS NO-SIGNAL

IF THE STATUS OF *wire* IS FAILED
THEN THE STATUS OF *wire* IS NO-GROUND

The default behavior assumes that if a wire undergoes a primary failure, then the wire will be incapable of either a proper grounding or of carrying a current.

4.3.4. Relay

In MAS, the behavior of a relay is assumed to be much the same as a terminal strip, in that input wires are matched to output wires. However, the relay has one major difference, notably that it acts as a switch. For each relay, there are two failure modes that are defined; *failed-open* and *failed-closed.* In the interest of simplicity, for each failure mode, the same default behavior is defined, namely that the *no-signal* fault will transfer to the output wires of the relay. Note, in this case the *no-signal* fault acts like a generic, propagating fault, rather than an actual lack of voltage on the wire.

The behavior for a normal relay must also be described, since faults are propagated from input wires to output wires in much the same way as the terminal strip. Finally, it is assumed that relays are normally closed. More specifically, if the control wire has *no-signal* then the relay is assumed to be *failed-closed.*

Thus, the default behavior can be captured with the following rules. As described above, the *failed-open* and *failed-closed* faults lead to *no-signal* faults on the output wires.

IF THE STATUS OF *relay* IS FAILED-OPEN
THEN THE STATUS OF *output-wire* IS NO-SIGNAL

IF THE STATUS OF *relay* IS FAILED-CLOSED
THEN THE STATUS OF *output-wire* IS NO-SIGNAL

The control wire behavior is defined with the following rule:

IF THE STATUS OF *control-wire* IS NO-SIGNAL
THEN THE STATUS OF *relay* IS FAILED-CLOSED

Since faults are assumed to transfer through the relay, the following rules are also added for each input/output wire pair.

IF THE STATUS OF *input-wire* IS NO-SIGNAL
THEN THE STATUS OF *output-wire* IS NO-SIGNAL

IF THE STATUS OF *input-wire* IS NO-GROUND
THEN THE STATUS OF *output-wire* IS NO-GROUND

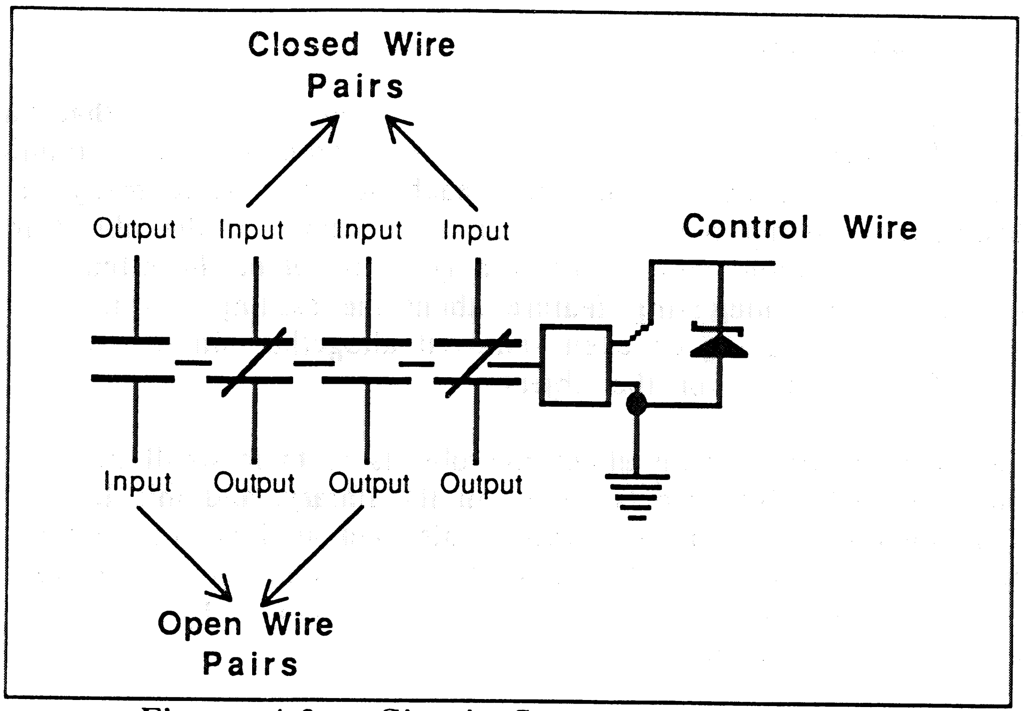


Figure 4-3: Circuit Schematic of a Relay

4.3.5. Ground

The ground is a component that has been modeled with a slightly different behavior than the rest of the library objects. In MAS it is assumed that if a ground fails then all of the wires connected to the ground are affected. For MAS modelling purposes, the ground is modeled with only one failure mode, *failed*. If a ground fails, then the fault *no-ground* is transferred to all of the connecting wires. Note, since the specification of input and output wires is dependent on which connections the failure of the object affects, the ground has only outputs as its connections.

The previous behavior description is captured in the following rule:

IF THE STATUS OF *ground* IS FAILED
THEN THE STATUS OF *output-wire* IS NO-GROUND

4.4. Conclusion

It is important to note several features about the objects that have been described in this chapter. First and foremost, many simplifying assumptions have been made about each object. While many of the assumptions do not precisely reflect the behavior of the object in a real world situation, they do offer a good model of the circuit behavior. The interesting feature about the assumptions is that they can often be relaxed, or even removed altogether simply by defining more failure modes for the object.

The next feature to note about the objects is their small number. There are only five objects stored in the library, and this limits the modeling process. However, this is also not an inherent limitation of the system, but rather a function of the implementation. More library objects could easily be implemented simply by following the same style as illustrated here.

Finally, the effectiveness of the library objects should be recognized. Even though a relatively limited number of objects are represented, they do cover a surprisingly large number of objects. In terms of modeling time, the fewer the *generic* objects that must be defined in an ad hoc fashion, the shorter the modeling time. When dealing with large circuits, the time savings becomes increasingly substantial. Thus, this is perhaps the most compelling reason to utilize and develop library objects as an inherent part of the model authoring process.

Chapter 5:

A Fault Tree Analysis on an Emergency Hydraulic System

In this chapter a simplified version of the Emergency Hydraulic System (EHS) for a Navy F-18 fighter is modeled using MAS. This application of MAS was selected to further explore the capabilities and limitations of MAS on a simple and real life-critical circuit design. The example differs from the motor control example explained earlier in two respects. First, the EHS is made significantly more complex than the first example by introducing redundancy. Second, the EHS example makes extensive use of the library objects discussed in Chapter 4. These factors combine to make this system an interesting example.

5.1. The Circuit Schematic

Figure 5-1 shows the schematic of the circuit that is modeled. Since the circuit is part of the emergency system for an F-18, the emphasis is on reliability. For example, the pump contractor is powered by three alternate power sources, battery 3, battery 5 and battery 6. Throughout the example, numerous wires provide a number of different paths, providing many layers of redundancy.

While the circuit itself is fairly self explanatory, there are a few features that should be noted. In general, there are three main power sources which can supply power to the the hydraulic pump, battery 3, battery 5 and battery 6. The power from these batteries is controlled by a relay, which in turn is controlled by a switch in the cockpit. Once this switch is thrown and the pump is activated, a signal is sent to the instrument bay which activates the appropriate lights.

First, note that almost all of the components consist of library objects described in Chapter 4. While this does not cut down the time needed for schematic capture, it does markedly reduce the time needed for behavior definition. Of the 47 components, only four components need specific rules written for them.

Second, the main feature of the circuit is the emergency hydraulic pump. Specifically, this pump failure will be the target for a Fault Tree Analysis. Finally, the circuit is still relatively simple. While the example does come from a real world application, it is only a simplified version. The full emergency hydraulic system is roughly four times more complicated.

5.2. The Fault Tree Analysis

The actual Fault Tree Analysis appears in the figures at the end of this chapter. This analysis was achieved using MAS as described in earlier chapters. First the schematic was entered into the system and schematic capture was performed to represent each circuit component. Next, the connections were defined for each object, thereby specifying input and output ports. Finally, the behavior was defined for the four components that were not contained in the library. Once the model was defined, the Fault Tree Analysis was calculated from the fact, THE STATUS OF PMP-1 IS FAILED.

In FTA-0, the first diagram of the FTA, note that there are two conditions which could cause the pump to fail. First, the pump would fail if both ground connections are failed. The use of a logical AND gate signifies that *both* conditions must be met for the failure to occur. From the diagram, it is evident that a ground failure could occur either if the ground itself failed, or the wire connecting the pump and ground failed.

The second cause of pump failure is shown in FTA-1. Simply, the pump would not work if there was no current supplied to the pump. Again, in order for the pump to have no current, both wires leading to the pump must have no current. By referring to the circuit, there are three conditions that will lead to no current on a wire: the wire is failed, the pump relay is failed or the pump relay has no current present.

FTA-2 details the conditions necessary for the pump relay to have no signal. Three conditions could cause this failure: the control wire could be failed, the control wire could have no signal present, or the power lines do not have any current. The conditions that could cause the control wire to have no signal are threefold. If the terminal strip is failed, then the control wire will have no current. If the connection wire pair to the control wire is failed or has no current, then the control wire will also have no current. Finally, there are a few reasons why the connection wire will have no current. The no-signal failure mode will occur if the connection wire is failed, if the instrumentation component is failed, or the instrumentation component is not producing a signal.

One of the reasons the pump relay might have no signal is if all seven of the power lines have no signal. The first three FTA's are similar, and appear in FTA-3, FTA-4 and FTA-5. In order for one of these three power lines to have no signal, one of three conditions must be met. Namely, if the terminal strip is failed, the connecting power line has no signal or if the power line itself is failed. The connecting power line follows the same fault tree pattern, and would have no signal is the terminal strip is failed, the wire is failed or the wire connection the battery to the terminal strip has no signal. Finally, the wire connecting the battery to the terminal strip would have no signal for one of two reasons, either the wire is failed, or the battery is failed.

The Fault Tree Analysis for battery 3 failure is shown in FTA-12. The battery would fail if the ground wire is not properly grounded. The ground wire would not be properly grounded if either the wire was failed, or the connecting wire was improperly grounded. That wire would be improperly grounded if it was failed or its connecting wire was ungrounded. That wire would be improperly grounded if the wire was failed, or if the ground was failed.

Returning back to to FTA-2, there are still four reasons that would lead the pump relay to have no signal. Again, these four FTA's are similar and appear in FTA-6, FTA-7, FTA-8, and FTA-9. These wires would fail either if the wire failed or if the battery failed. The FTA's for battery 5 and battery 6 failure are outlined in FTA-10 and FTA-11. These batteries would fail if the ground wire was improperly grounded, which would occur if the wire is failed or if the ground was failed.

5.3. Conclusion

From this example, it is evident that even a simple circuit can produce a relatively complicated fault tree. Dealing with this complexity is the main advantage in using MAS. A safety engineer might miss a crucial step in the fault tree analysis and unknowingly invalidate the entire tree.

However, there is one more important conclusion that can be drawn from this example. MAS is useful only as a tool, and like many tools, it is only as useful as the skill of the engineer. The results of a safety analysis must be reviewed with two thoughts in mind. First, the validity of the analysis is dependent on the modeling process. Creating a poor circuit model will certainly not yield accurate analyses. Second, MAS is not a circuit verification tool. The actual verification must be done by the engineer. MAS only produces useful information towards the verification process and in no way does actual verification. Despite these constraints, in the hands of a safety engineer, MAS can contribute significantly to the verification process.

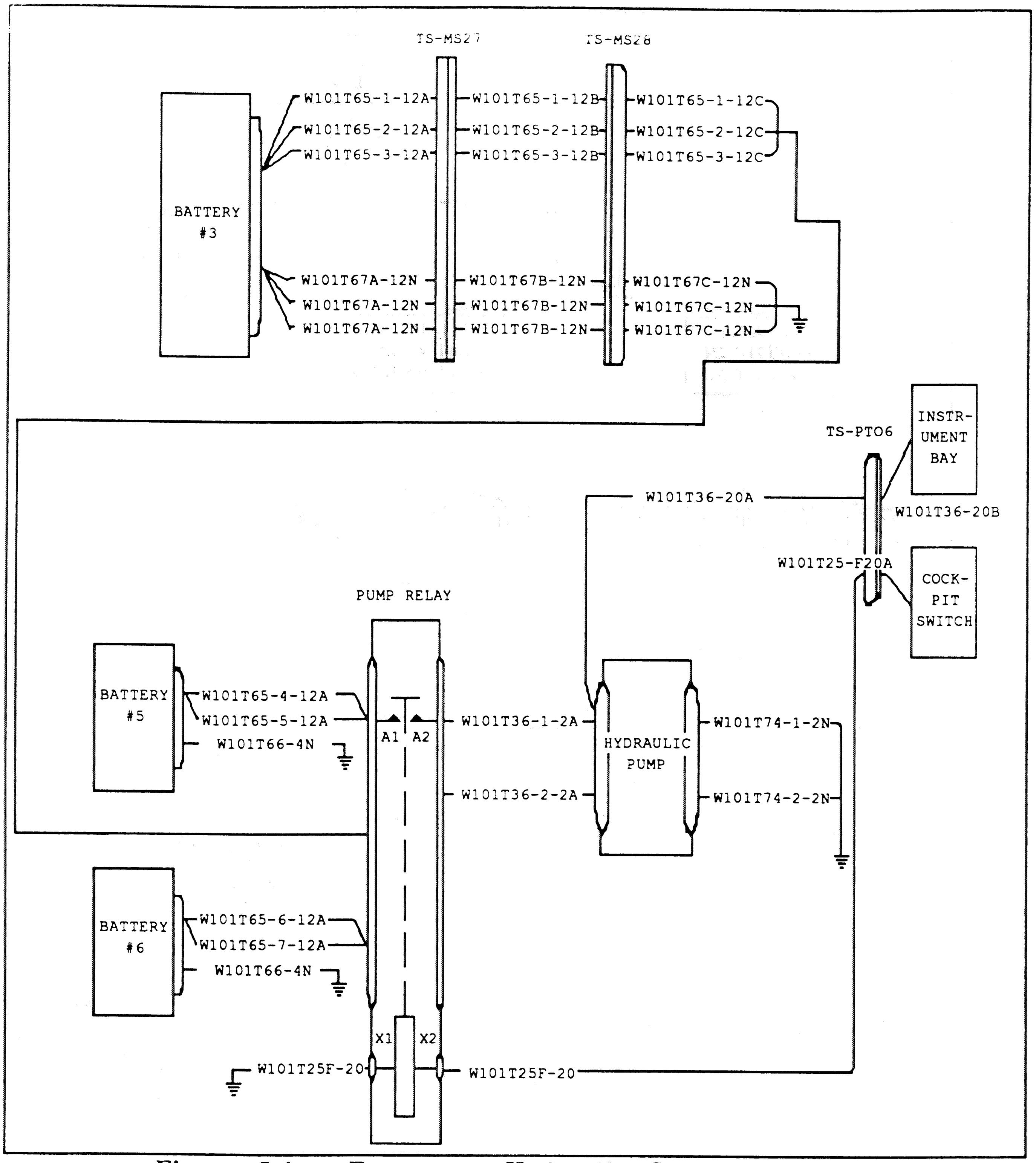
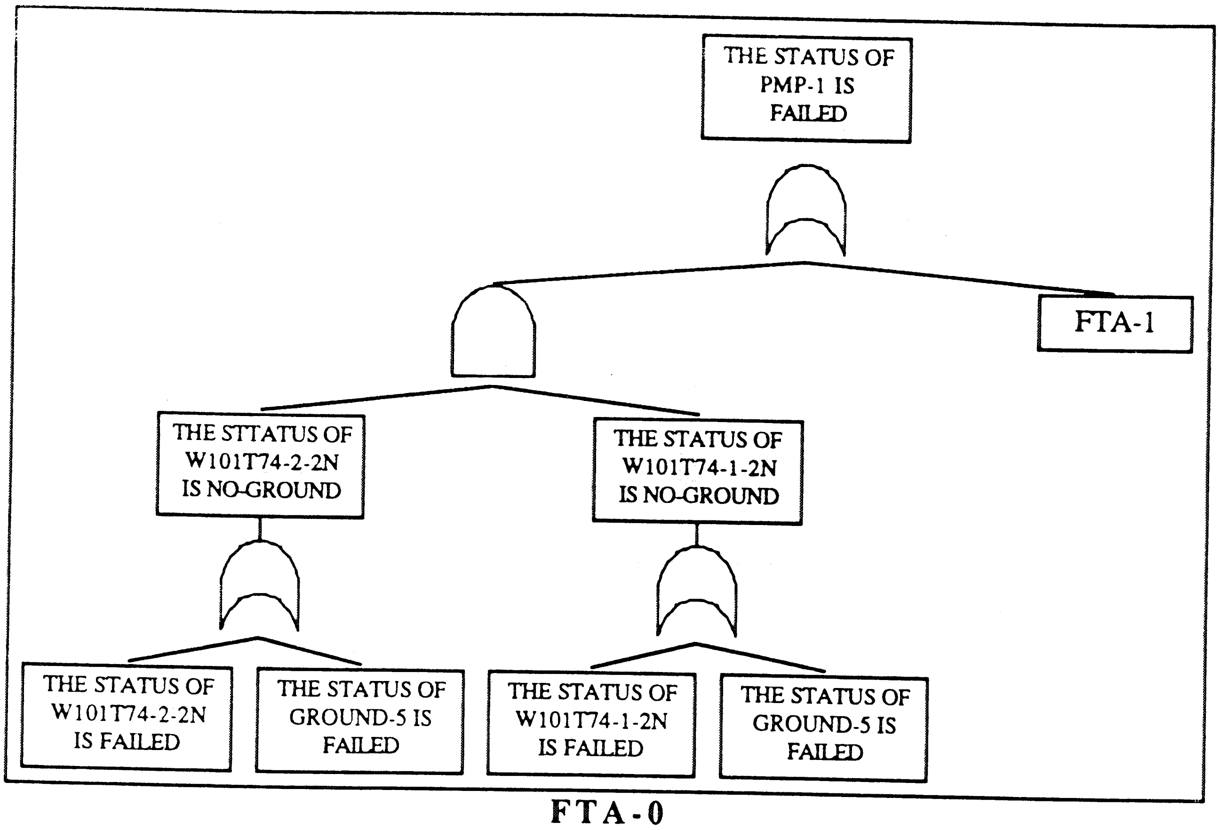
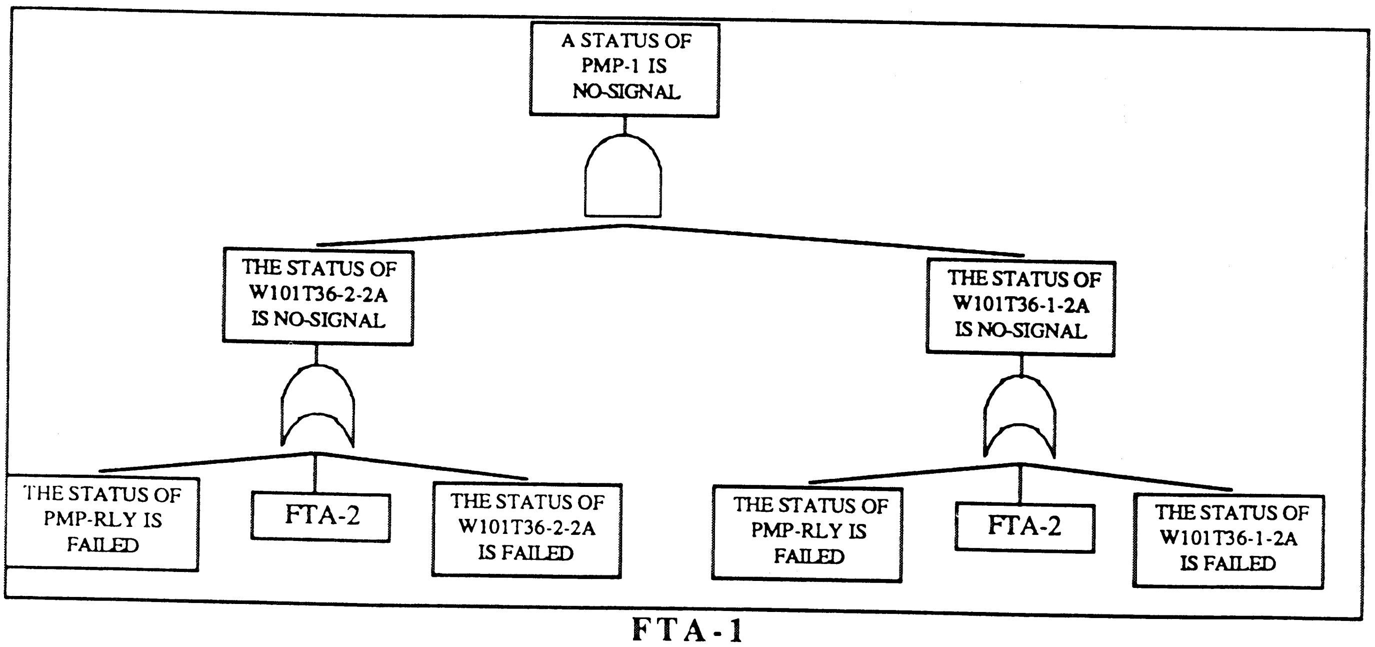


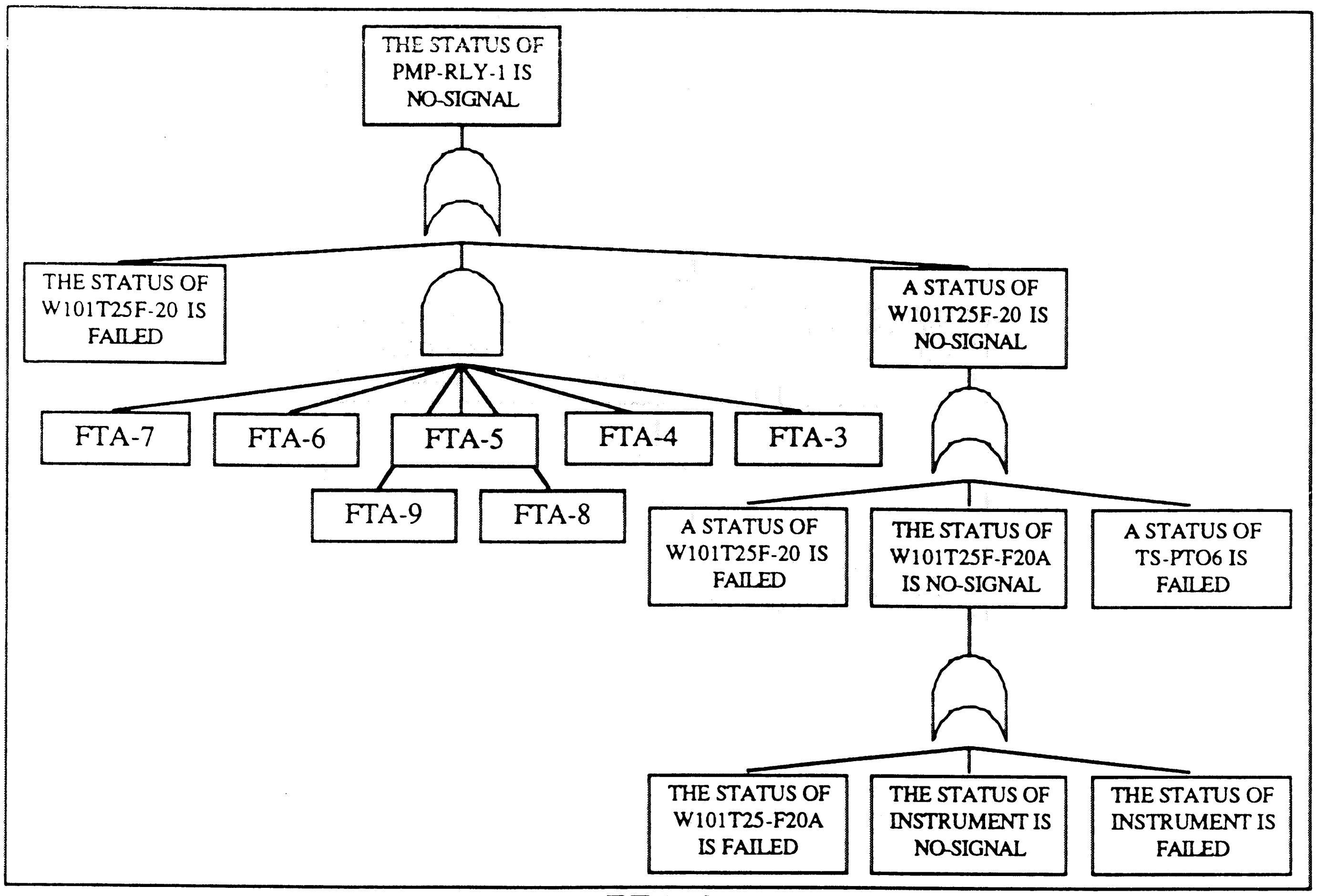
Figure 5-1: Emergency Hydraulic System Schematic


THE STATUS OF PMP-1 IS FAILED
FTA-1
THE STTATUS OF W101T74-2-2N IS NO-GROUND
THE STATUS OF W101T74-1-2N IS NO-GROUND
THE STATUS OF W101T74-2-2N IS FAILED
THE STATUS OF GROUND-5 IS FAILED
THE STATUS OF W101T74-1-2N IS FAILED
THE STATUS OF GROUND-5 IS FAILED

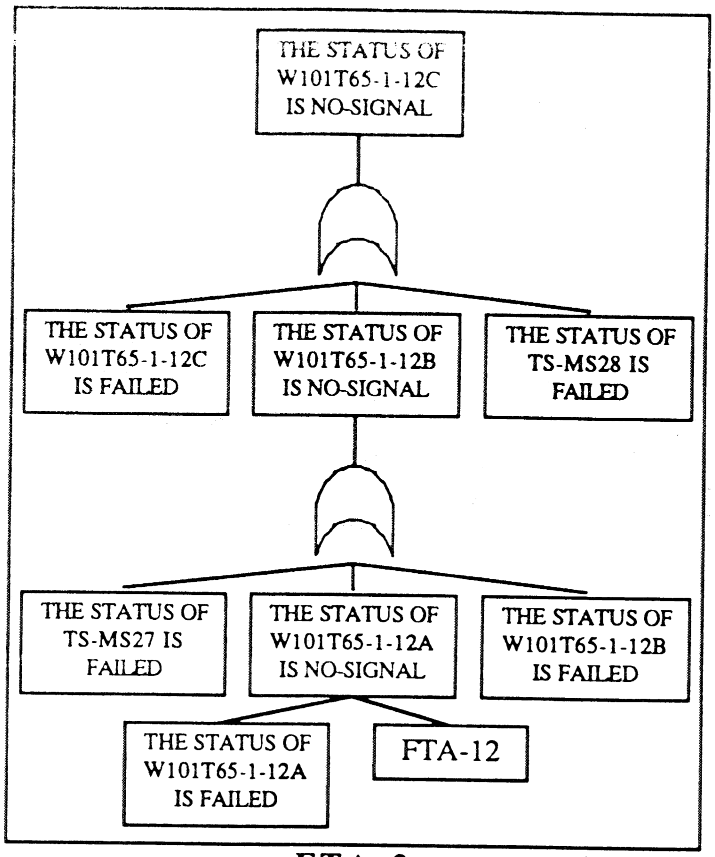
FTA-0


A STATUS OF PMP-1 IS NO-SIGNAL
THE STATUS OF W101T36-2-2A IS NO-SIGNAL
THE STATUS OF W101T36-1-2A IS NO-SIGNAL
THE STATUS OF PMP-RLY IS FAILED
FTA-2
THE STATUS OF W101T36-2-2A IS FAILED
THE STATUS OF PMP-RLY IS FAILED
FTA-2
THE STATUS OF W101T36-1-2A IS FAILED

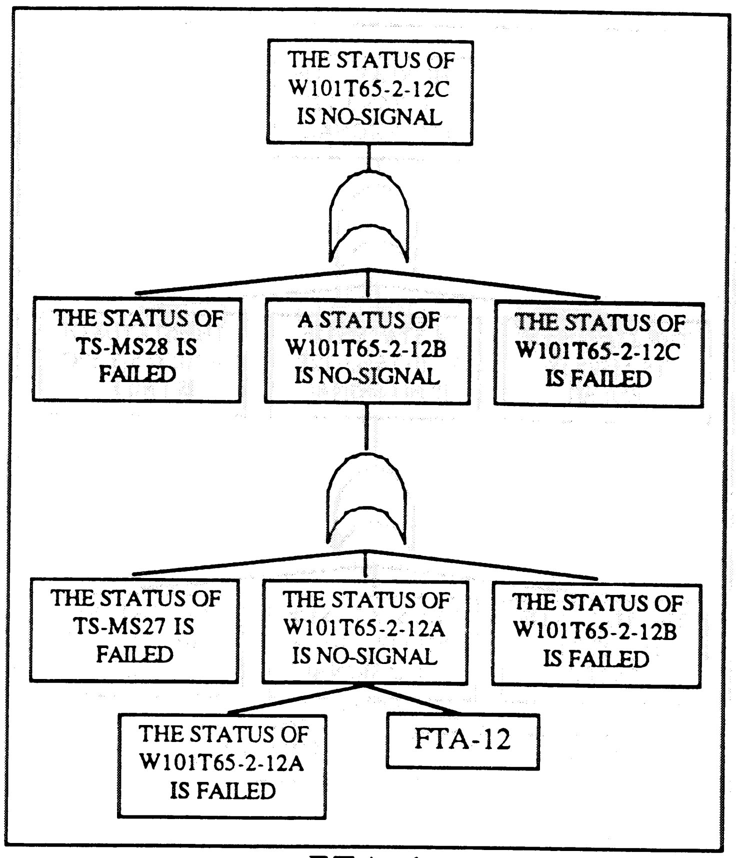
FTA-1


THE STATUS OF PMP-RLY-1 IS NO-SIGNAL
THE STATUS OF W101T25F-20 IS FAILED
A STATUS OF W101T25F-20 IS NO-SIGNAL
FTA-7
FTA-6
FTA-5
FTA-4
FTA-3
FTA-9
FTA-8
A STATUS OF W101T25F-20 IS FAILED
THE STATUS OF W101T25F-F20A IS NO-SIGNAL
A STATUS OF TS-PTO6 IS FAILED
THE STATUS OF W101T25-F20A IS FAILED
THE STATUS OF INSTRUMENT IS NO-SIGNAL
THE STATUS OF INSTRUMENT IS FAILED

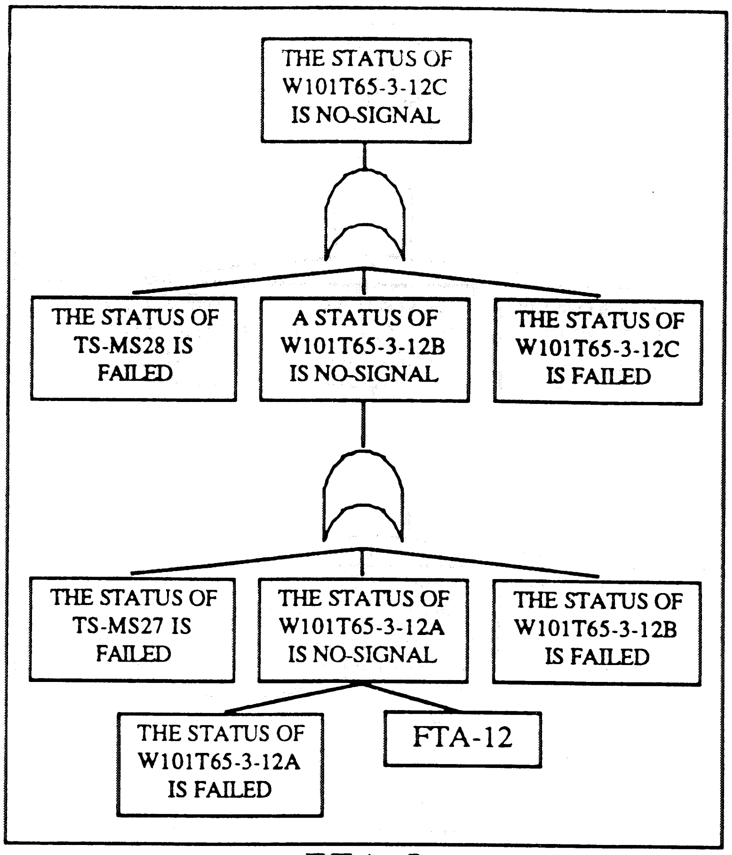
FTA-2


THE STATUS OF W101T65-1-12C IS NO-SIGNAL
THE STATUS OF W101T65-1-12C IS FAILED
THE STATUS OF W101T65-1-12B IS NO-SIGNAL
THE STATUS OF TS-MS28 IS FAILED
THE STATUS OF TS-MS27 IS FAILED
THE STATUS OF W101T65-1-12A IS NO-SIGNAL
THE STATUS OF W101T65-1-12B IS FAILED
THE STATUS OF W101T65-1-12A IS FAILED
FTA-12

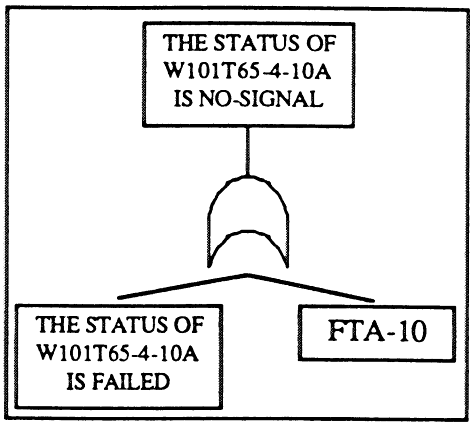
FTA-3


THE STATUS OF W101T65-2-12C IS NO-SIGNAL
THE STATUS OF TS-MS28 IS FAILED
A STATUS OF W101T65-2-12B IS NO-SIGNAL
THE STATUS OF W101T65-2-12C IS FAILED
THE STATUS OF TS-MS27 IS FAILED
THE STATUS OF W101T65-2-12A IS NO-SIGNAL
THE STATUS OF W101T65-2-12B IS FAILED
THE STATUS OF W101T65-2-12A IS FAILED
FTA-12

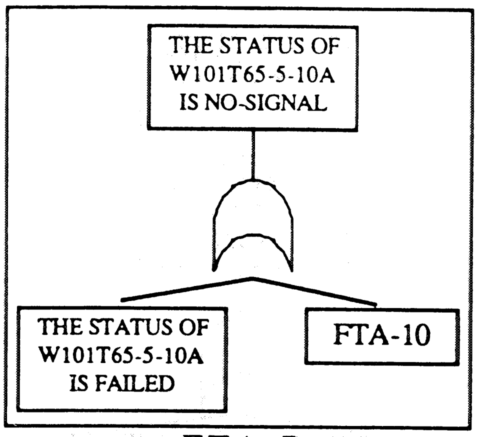
FTA-4


THE STATUS OF W101T65-3-12C IS NO-SIGNAL
THE STATUS OF TS-MS28 IS FAILED
A STATUS OF W101T65-3-12B IS NO-SIGNAL
THE STATUS OF W101T65-3-12C IS FAILED
THE STATUS OF TS-MS27 IS FAILED
THE STATUS OF W101T65-3-12A IS NO-SIGNAL
THE STATUS OF W101T65-3-12B IS FAILED
THE STATUS OF W101T65-3-12A IS FAILED
FTA-12

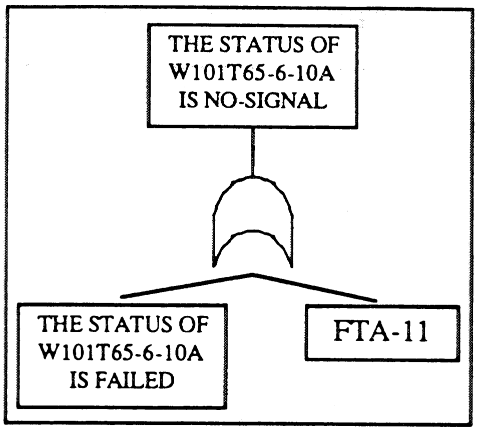
FTA-5


THE STATUS OF W101T65-4-10A IS NO-SIGNAL
THE STATUS OF W101T65-4-10A IS FAILED
FTA-10

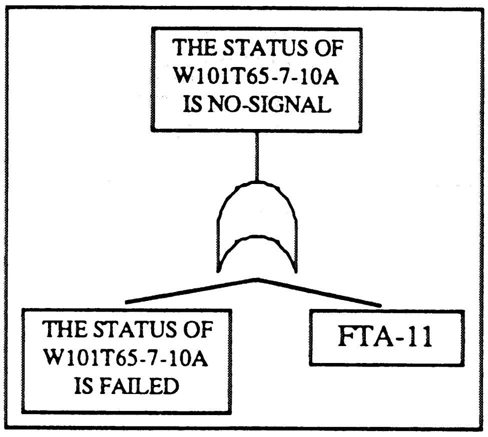
FTA-6


THE STATUS OF W101T65-5-10A IS NO-SIGNAL
THE STATUS OF W101T65-5-10A IS FAILED
FTA-10

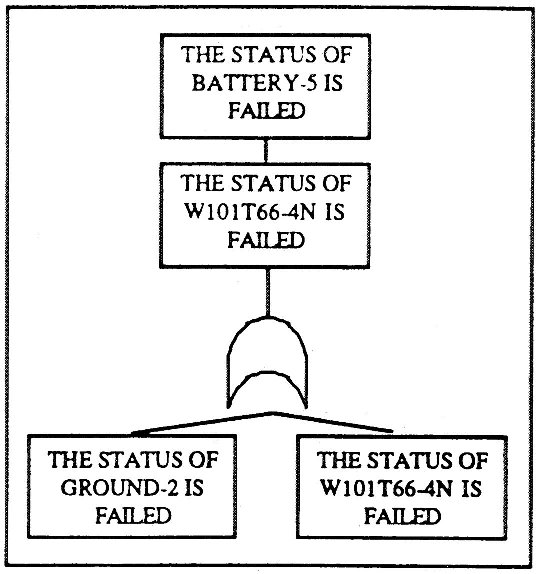
FTA-7


THE STATUS OF W101T65-6-10A IS NO-SIGNAL
THE STATUS OF W101T65-6-10A IS FAILED
FTA-11

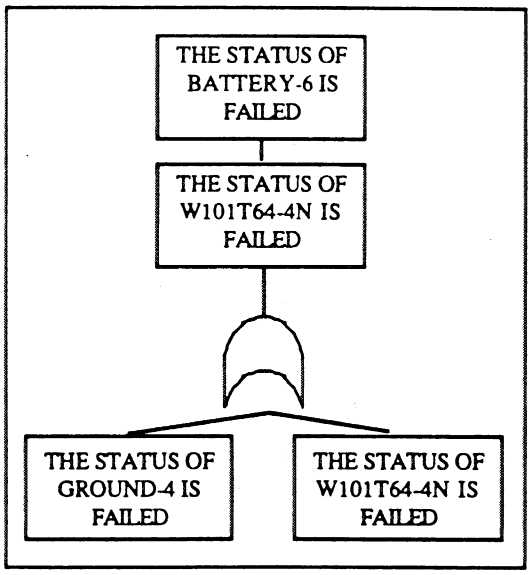
FTA-8


THE STATUS OF W101T65-7-10A IS NO-SIGNAL
THE STATUS OF W101T65-7-10A IS FAILED
FTA-11

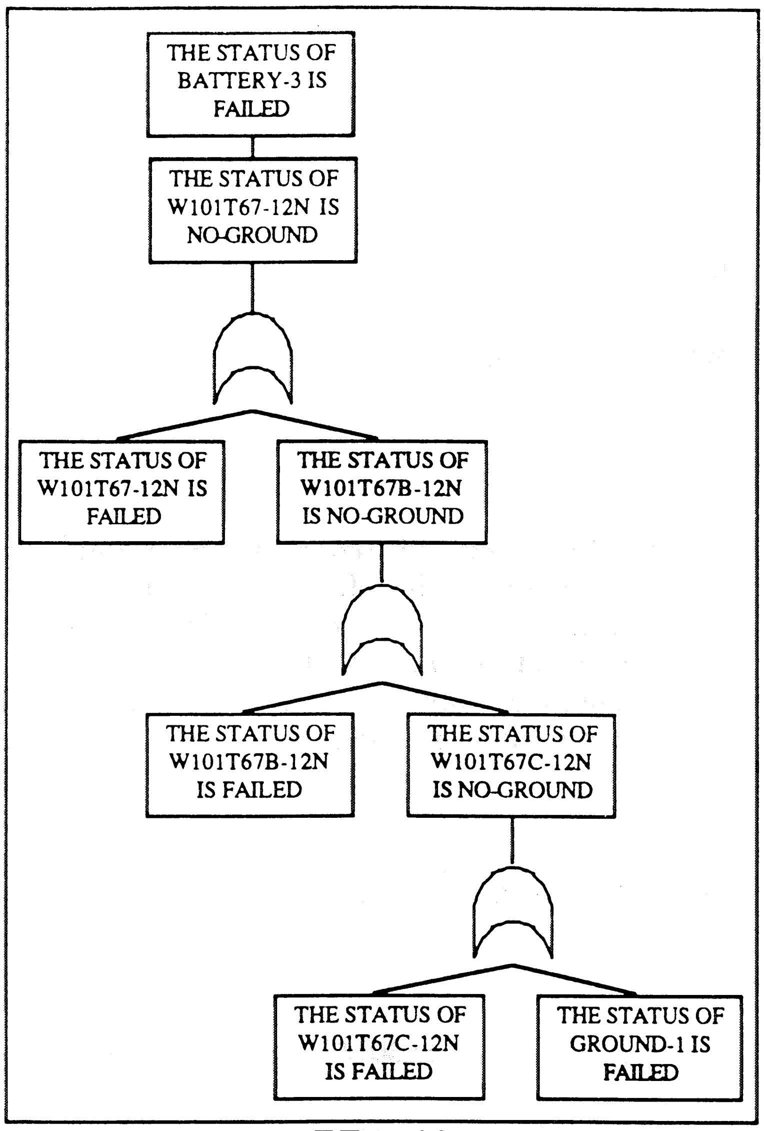
FTA-9


THE STATUS OF BATTERY-5 IS FAILED
THE STATUS OF W101T66-4N IS FAILED
THE STATUS OF GROUND-2 IS FAILED
THE STATUS OF W101T66-4N IS FAILED

FTA-10


THE STATUS OF BATTERY-6 IS FAILED
THE STATUS OF W101T64-4N IS FAILED
THE STATUS OF GROUND-4 IS FAILED
THE STATUS OF W101T64-4N IS FAILED

FTA-11


THE STATUS OF BATTERY-3 IS FAILED
THE STATUS OF W101T67-12N IS NO-GROUND
THE STATUS OF W101T67-12N IS FAILED
THE STATUS OF W101T67B-12N IS NO-GROUND
THE STATUS OF W101T67B-12N IS FAILED
THE STATUS OF W101T67C-12N IS NO-GROUND
THE STATUS OF W101T67C-12N IS FAILED
THE STATUS OF GROUND-1 IS FAILED

FTA-12

Chapter 6:

Results and Conclusions

Now that MAS has been detailed and a simple example has been worked, it is time to evaluate the effectiveness of the system. This chapter has been divided into three parts. First, initial system requirements are reviewed in light of final results. Second, other disadvantages and possible improvements are discussed. MAS is still a prototype, and many more features need to be implemented before a reasonable application can be written. In the final section, the inherent strengths of MAS will be reviewed. MAS has served as a proof-of-concept, and it is important to evaluate the things the system inherently does well. Finally, the last section concludes with a look to the future in automating reliability engineering.

6.1. The Basic System Requirements, Revisited

The best place to start with an overall system critique is a discussion of the initial requirements, and the success experienced in matching these requirements. Throughout this section, many possible improvements are outlined.

6.1.1. Ease of Use

- *MAS must be easy enough to use so that it can be utilized on many design iterations with many types of designs.*

The first requirement was that MAS must be easy enough to use so that many design iterations and design types could be analyzed. This requirement was satisfied to a degree, but not quite as completely as possible. First, MAS can only be used on one type of design, circuit schematics. While the technique is certainly generalizable to any type of design, it has only been implemented with circuits. Since the behavior of most physical components can be modeled by writing the appropriate rules, many different physical systems could be modeled. Thus, one future expansion might be to expand MAS to other system types, such as hydraulic systems.

The second part of the first requirement was ease of use. When evaluating MAS, it is safe to say the system has satisfied that goal. The user interface is not perfect, but does rely on an intuitive mouse and menu graphical technique, which communicates ideas readily to the user. Certain details and implementations could be improved, but the overall user interface concept is quite strong.

6.1.2. Automation

- *MAS must automate the laborious accounting involved in tracing the causes and effects of component failures, while allowing the safety engineer to effectively employ his own knowledge and experience.*

The second goal required MAS to automate the laborious accounting involved in tracing the causes and effects of component failures. This requirement has certainly been met by MAS. The rule based paradigm effectively accounts for the causes and effects of component failure by keeping track of every component in the system. Thus MAS effectively relieves this bookkeeping requirement, while allowing the engineer to concentrate on higher level tasks.

MAS also allows an engineer to employ his own knowledge and experience, thus satisfying the rest of the second goal. However, this feature can be regarded both as an advantage and as a disadvantage. MAS allows the engineer to tailor the model by using the abstraction features to obtain the proper level of complexity, which can be potentially very useful. However, MAS allows almost too much freedom in the modeling process. It is currently very easy to define a model in a way that will lead to erroneous or incomplete results. More constraints need to be implemented in the modeling procedure so that the engineer needs to know very little about the expert system mechanics that make MAS work. An interesting problem would be to explore just how transparent the underlying expert system could become, while still generating useful analyses.

6.1.3. Bottom Up Approach

- *Because the FTA and FMEA techniques are used on existing designs to analyze modifications and improvements, MAS should utilize a bottom-up approach to knowledge capture.*

The third requirement for MAS was that the system provide for a bottom-up approach. This requirement was explicitly built into the system design, so that the requirement is satisfied by the very nature of the tool. MAS is a tool for reliability analysis on existing systems, it is assumed that these systems are already designed. If MAS used a top down approach to design reliability, the system would become more of a design tool, rather than a verification tool. Thus, this last requirement exists as a defining part of the system.

6.1.4. Output

- *MAS must output the type of graphical representations and worksheets familiar to safety engineers.*

The fourth requirement is that MAS should output the type of graphical representations and worksheets familiar to safety engineers. The figures throughout this document show the graphical output of the system, and it is obvious that there is room for improvement. For Fault Tree Analysis, the system outputs only three symbols, the *and-gate*, *or-gate* and *box*. In a true FTA there are many types of objects that are used to designate primary failures, secondary failures, outside conditions etc. A logical extension to MAS should be the exploration of using the full range of FTA symbols to convey the maximum possible information.

For the Failure Mode and effects Analysis, the graphical output differs from the traditional worksheet style. While all the information is present, it is presented in a graphical tree-like form. In order to make the FMEA similar to familiar styles, the tree would have to parsed into a worksheet form, as described in the Chapter 2. While this improvement would merely be cosmetic and no change the overall system design, it would have a great deal to do with the usability of the system.

6.1.5. Documentation

- *Because the information entered and created in an analysis is often very useful as reference data, MAS must organize and retain this information for documentation purposes.*

The last requirement MAS must fulfill involves storing information. The information entered and created in an analysis can be a very valuable reference tool. As it exists, MAS saves the hierarchical and behavioral definition of the models. While the information is designed to provide reliability analysis, the *structure* of the information could be a very valuable documentation tool. The end result of the bottom up design is a top down hierarchy of system components. This hierarchy, once created, could be used as a supplement for, or with certain extensions, a replacement for existing documentation. MAS could be expanded to provide documentation "hooks" into the abstraction modules, which would provide a very valuable reference tool. Thus, MAS would not only be a tool for reliability analysis, but also a documentation tool. MAS currently provides the graphical and informational hierarchy, but expanding the system into a documentation tool would require additional development.

6.2. Other Disadvantages and Possible Improvements

In almost any scientific or technical endeavor, attempting to solve one problem leads to the discovery of two more. Therefore, this section will be larger than anticipated, only because there is so much work that could be done.

6.2.1. Scaling

In addition to the ways in which MAS fulfills the basic requirements, there are some more issues that must be discussed. First, there is a question about how well the system will scale up to real world models. Experience with expert systems seems to be that large groups of unclassified rules make the rule system unstable. The problem lies in the fact that there is no inherent way to order the assertions that rules add. Therefore, a rule that has added a fact may invalidate another fact. When there are large groups of rules, the inter-relations between the rules become very complicated. The result could lead to an unstable system that may provide erroneous output.

Thus, how does MAS scale? The question is left unanswered due to time and resource constraints. However, some guesses might be proposed. First, the scaling problem with large groups of rules seems only to be a problem when rules have cause interactions between a large group of objects. In MAS, behavior rules are very well defined, and affect only a small group of connected objects. Therefore, the interconnection problem and invalidation problem is reduced. This local rule behavior could avoid the large rule number scaling problem.

6.2.2. Accuracy

The second issue is one of accuracy. Notably, the system is forced to make numerous approximations and simplifications in order to model the design. The question then becomes, are these assumptions enough to invalidate the model? With MAS as it exists now, the design is simplified enough to make the analysis barely accurate. Thus, the question is actually, is the accuracy problem an inherent part of the system design, or can it be improved? By examining the system, it seems there is nothing inherently wrong with the design. Improving the accuracy of the model can be achieved simply by modeling the behavior of the system more realistically. For example, the failure modes for objects are currently very simplistic. One extension that would improve the accuracy of the model would be to expand upon the number of failure modes for objects. More failure modes would model the system more realistically, and thus make the results much more meaningful.

However, the level of detail in the modeling process is only one aspect of the accuracy problem. As mentioned before, real world FTA's require much more information than simply *or* and *and* gates. However, improving the detail of the modeling will also help improve the detail of these FTA's. By defining the behavior for a wide variety of faults, the FTA's will reflect the increase in information. Thus, the accuracy problem seems to be only in the implementation of the modeling, not the system design. In order for MAS to be transferred into a working application, the modeling process must be greatly expanded. However, as explained here, the problem seems to have a workable solution. But exactly how the modeling process and consequent analysis is to be expanded is one of the major open problems in MAS. More work is required to discover if MAS can gracefully transfer into a second stage and handle the increased complexity of a detailed failure mode implementation.

6.2.3. Failure Likelihood

Whenever reliability of designs is discussed, the issue of probability always arises. Calculating the failure likelihood of components is an integral part of reliability analysis, and no reliability tool would be complete without this capability. Unfortunately, MAS is not complete and the capability does not exist. However, the issue of failure likelihood calculations has been part of the basic design and its addition to the system would be a logical extension. By assigning each object with a failure likelihood, the system should be able to calculate the critical path from either a Fault Tree Analysis or a Failure Mode and Effects Analysis. Failure likelihood was just one of the many features that should be implemented, but for which time and resources do not allow.

6.2.4. Spatial Interactions

There is one more feature that should be placed on the wish list of improvements. MAS as it exists now only deals with the effects of objects physically connected to one another. However many system faults have nothing to do with objects that are connected to each other, but with objects that are located physically next to each other. For example, a capacitor might explode, affecting not just its connection objects but objects that are located near the capacitor. If many redundant paths are in close physical proximity, a particular fault may affect them all, and have disastrous effects on the system. If this type of connectivity could also be included in the reliability of the system, the overall accuracy of the analysis would be greatly improved. Note, this capability could be implemented by expanding the behavior of the rules, so that a failure mode might be able to affect not just connection objects, but other objects as well. The resulting system would be considerably more complicated, but also extremely more illuminating. Just how this could be implemented would be an interesting and fruitful area of research.

Thus, this concludes an outline of the major improvements and disadvantages of MAS. While this is certainly not an exhaustive list, it does serve as a good guide to the major areas of improvement and interest that the MAS implementation has encountered.

6.3. The Inherent Strengths of MAS

It is useful to summarize the results of MAS. First, the system really does save the reliability engineer an enormous amount of time. Although the initial modeling costs are quite high, a major advantage of the system is the fact that this initial definition only has to be performed once. Once the system is modeled, Fault Tree Analyses and Failure Mode and Effects Analyses can be generated literally at the click of a button. Thus, an entire system can be analyzed at a level that was never before possible. It should also be noted that the initial modeling costs can be greatly reduced by the use of library objects. This feature can save an engineer a great deal of otherwise redundant work.

Another strength of MAS lies in the graphical interface. When thinking about abstract ideas such as design, it is useful to present these ideas in the form of pictures. MAS allows the engineer to operate in these graphical terms, which results in increasing the effectiveness of the program. In fact, MAS is so completely graphically interfaced that it is hard to think about defining a system in non-graphical terms. The interface is as much an important part of the system as the rule-based paradigm, and the overall viability of the system improves as a result.

One of the interesting features of MAS is the use of local behavior to get results on a global scale. Objects are only defined in a purely local sense, that is, how their failure will affect their immediate connections. It is exactly this local behavior definition that supports the use of library objects, which are very important features of MAS.

Finally, it is important to think about the next phase of MAS, and about the role of automating reliability engineering in the future. In the second phase of MAS, the system should exist in two levels. First, the system should operate at a level completely transparent to the design engineer. MAS could operate directly from a Computer Aided Design (CAD) system which contains the system design. Each of the CAD objects and their connection information could be automatically transformed into the MAS system. MAS would then model each of the objects from an extensive list of library objects. Thus, the MAS would be able to essentially model itself, with the goal of requiring no human intervention. Thus, once the model is built by MAS, the engineer could request at any point in the design either a Fault Tree Analysis or a Failure Mode and Effects Analysis. In the second level, once the base system is modified, the engineer should be able to organize groups of objects into abstract design blocks. MAS should be intelligent enough to distill the behavior of the group objects from the underlying behavior, and leave just the system breakdown to the engineer. Thus, the designer could produce much more lucid high level analysis, and get a much more comprehensive view of the reliability of the system. This information would then, in turn, affect the system design, which MAS would then model again. This recursive design process would produce final designs that are tailored exactly to the application's reliability needs.

This scenario might seem an impossible dream, but the beginning foundation has been laid in this work. While there is still much more work to be done, MAS has proven the viability of the basic solution. However, the step from prototype to application is always a rather large one, and it will be interesting to see the state of the art in the next decade.

REFERENCES

Davis Randall, and Hamscher, Walter Model-based Reasoning: Troubleshooting. *Exploring Artificial Intelligence*, Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1988

Gevarter, William B., The Nature and Evaluation of Expert System Building Tools, pg 24-41 Computer, May 1987

Hamscher, Walter C. Model Based Trouble Shooting of Digital Systems. Technical Report 1074, MIT Artificial Intelligence Laboratory, August, 1988

Lewis, E. E., Introduction to Reliability Engineering, John Wiley and Sons, New York, 1987.

Allen, James G., A Knowledge-Based System Design/Information Tool for Flight Control Systems AIAA Computers in Aerospace VII, Paper #89-2978, October, 1989

NASA Safety Handbook, NHB 1700.1(V7), April 10, 1985

Williams, Brian C., Qualitative Analysis of MOS Circuits, Technical Report 767, MIT Artificial Intelligence Laboratory, July 1984

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No. NASA CR-4317	2. Government Accession No.	3. Recipient's Catalog No.

4. Title and Subtitle Model Authoring System for Fail Safe Analysis	5. Report Date August 1990
	6. Performing Organization Code
7. Author(s) Scott E. Sikora (Charles Stark Draper Laboratory, Inc.)	8. Performing Organization Report No. H-1620
	10. Work Unit No. RTOP 505-68-27
9. Performing Organization Name and Address Charles Stark Draper Laboratory, Inc. 555 Technology Square Cambridge, MA 02139	11. Contract or Grant No. NAS2-12451
	13. Type of Report and Period Covered Contractor Report
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546-3191	14. Sponsoring Agency Code

15. Supplementary Notes

NASA Technical Monitor: Dale A. Mackall, NASA Ames Research Center, Dryden Flight Research Facility, Edwards, CA 93523-0273.

16. Abstract

The Model Authoring System is a prototype software application for generating Fault Tree Analyses and Failure Mode and Effects Analyses for circuit designs. Utilizing established artificial intelligence and expert system techniques, the circuits are modeled as a frame-based knowledge base in an expert system shell, which allows the use of object oriented programming and an inference engine. The behavior of the circuit is then captured through IF-THEN rules, which then are searched to generate either a graphical Fault Tree Analysis or Failure Modes and Effects Analysis. Sophisticated authoring techniques allow the circuit to be easily modeled, permit its behavior to be quickly defined, and provide abstraction features to deal with complexity.

17. Key Words (Suggested by Author(s)) Artificial intelligence Flight control Knowledge base Structured analysis	18. Distribution Statement Unclassified-Unlimited Subject category 66

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

NASA FORM 1626 OCT 86 *For sale by the National Technical Information Service, Springfield, Virginia 22161-2171*

NASA-Langley, 1990