A Model-Based Reasoning Architecture for System-Level Fault Diagnosis

A Dissertation presented to

The Academic Faculty

by

BHASKAR SAHA

In Partial Fulfillment

of the Requirements for the Degree of

Doctor of Philosophy in Electrical Engineering

School of Electrical and Computer Engineering

Georgia Institute of Technology

April, 2008

A Model-Based Reasoning Architecture for System-Level Fault Diagnosis

Approved by:

Dr. George J. Vachtsevanos, Advisor
School of Electrical and Computer Engineering

Dr. Patricio A. Vela
School of Electrical and Computer Engineering

Dr. Thomas E. Michaels
School of Electrical and Computer Engineering

Dr. Yorai Wardi
School of Electrical and Computer Engineering

Dr. Steven Y. Liang
School of Mechanical Engineering

Date Approved: November 16, 2007

To Schatzie and Sankalita,

whose patience and fortitude

will be a source of joy and inspiration forever

Acknowledgements

First of all, I would like to thank my advisor, Dr. George Vachtsevanos, who took me under his wing under difficult circumstances in 2003, and has continued to provide unstinting academic, professional and personal guidance throughout the Ph.D. program. He showed me different ways to approach a research problem and the need to be focused and persistent to accomplish any goal.

I would also like to thank my dissertation committee members, Dr. Thomas Michaels, Dr. Patricio Vela, Dr. Yorai Wardi and Dr. Steven Liang. The special attention from Dr. Michaels and Dr. Vela in helping me refine my thesis is greatly appreciated. My thanks also go to my proposal committee members, Dr. Magnus Egersted and Dr. Krishna Palem, who, unfortunately, could not attend my dissertation defense.

I would like to express my appreciation for the late Dr. John Uyemura, who introduced me into the Georgia Tech fold, and also to the faculty and personnel of the School of Electrical and Computer Engineering, especially Dr. David Hertling, Dr. Bonnie Heck Ferri, and Ms. Marilou Mycko, who enabled the initiation, persistence, and conclusion of all the procedures involved with attaining the doctoral degree.

A big thanks to all the members and friends of the Intelligent Control Systems Laboratory of Georgia Tech who have greatly aided me in this research, especially to Mr. Gary O'Neill, Dr. Biqing (Becky) Wu, Dr Liang Tang, Dr. Abhinav Saxena, Dr. Marcos Orchard, Dr. Graham Drozeski, Dr. Nathan Scott Clements, Dr. Irtaza Barlas, Dr. Ben

Ludington, Dr. Johan Reimann, Dr. Bin (Ben) Zhang, Dr. George Georgoulas, Dr. Otis Smart, Taimoor, Jackie, Matt, Lauren, Manzar, Sachin, Douglas, Phillip and Ms. Sharon Lawrence.

The support and collaboration of DARPA, Northrop Grumman Corp., Sikorsky Corp., Impact Technologies LLC, NAVAIR and NASA is acknowledged and appreciated, especially the collaborative efforts of the following people involved with these institutions: Mr. Steve Engel, Mr. Mark Davis, Mr. William Hardman, Mr. Greg Kacprzynski, Dr. Kai Goebel, Mr. Scott Poll, Dr. Ann Patterson-Hine and Dr. Serder Uckun. Special thanks goes to Dr. Kai Goebel for making my internship at the NASA Ames Research Center such a rewarding and fulfilling part of my Ph.D. experience.

My loving wife, Sankalita, who endured the rigors of the doctoral program along with me and turned them into an enjoyable experience, has been an invaluable asset since I have known her. My deepest gratitude also goes to my parents, Rabindra Kumar Saha and Suparna Saha, for giving me life, for educating me, and for their unconditional support and encouragement to pursue my interests, even when the interests went beyond boundaries of language, field and geography; and my sister Shelley Saha, for always believing in me. Last, but not least, I would like to thank my friends, Mithun, Divya, Mrinmoy, Bhaskar, Sunitha, Anand, Vivek, Vindhya, Rishi, Aruna, Umesh, Rahul, Charu, Atri, Bevin, Sanjeev, Banit, Jacob, Rajarshi, Rachana, Souvik, and all those that I fail to mention, who have shared my ups and downs and have encouraged me through my darkest moments.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

Summary

Critical components that are prone to failure are usually constituent elements of a larger subsystem or system which cannot be instrumented at the component level. The inception of such failures, thus, often goes undetected until the anomaly spreads from one component to another, in a sort of *domino effect*, until the overall system fails. It is crucial, therefore, to devise methodologies that are capable of determining, from monitoring external system behaviors like vibrations, temperature, etc., which component is defective and, additionally, how specific components' faults/failures may propagate from faulty to healthy components, thus causing a catastrophic failure. Model-based reasoning (MBR) belongs to this methodological category. MBR is a very generic but sufficiently potent technique, which is applicable to a wide variety of domains, from finding faults in digital circuits, to the diagnosis of automobiles, the monitoring of industrial machinery, and more recently, even to the debugging of large software.

This dissertation presents a model-based reasoning architecture with a two fold purpose: to detect and classify component faults from observable system behavior, and to generate fault propagation models so as to make a more accurate estimation of current operational risks. The first step is the compilation of the *database* of functional descriptions and associated fault-specific features for each of the system components. The system is then analyzed to extract structural information, which, in addition to the functional database, is used to create the *structural* and *functional models*. A *fault-symptom matrix* is constructed from the functional model and the features database. The fault threshold levels for these symptoms are founded on the nominal baseline data. Based on the fault-

symptom matrix and these thresholds, a *diagnostic decision tree* is formulated in order to intelligently query about the system health. For each faulty candidate, a *fault propagation tree* is generated from the structural model. Finally, the overall system health *status report* includes both the faulty components and the associated “at risk” components, as predicted by the fault propagation model.

The MBR diagnostic architecture incorporates a novel approach to system level diagnostics. It addresses the need to reason about low-level inaccessible components from observable high-level system behavior. In the field of complex system maintenance it can be invaluable as an aid to human operators. The contributions from this research are:

- A novel adaptation of MBR that implements a knowledge database aided diagnosis of inaccessible faulty components from observable system behavior.
- A fault propagation methodology based on the information abstracted by the model-based diagnostic reasoning step listed above.
- An integration of the diagnosis and fault propagation algorithms in an overall model-based reasoning architecture formulated in an industry preferred test-bench language like MATLAB®.
- Application of the above software to intelligent fault diagnosis of existing helicopter power-train modules and other electro-mechanical systems.
- Complexity analysis of the algorithms used by the reasoning architecture to ascertain performance guarantees for online implementation.
- Model verification technique for complex engineered systems in temporal logic.

1 Introduction

The *information age*, as we know it today, began with the invention of the telegraph in 1837. Since then people have been looking into the extraction and exploitation of information from all possible sources to further the quality of life. However, information by itself serves no purpose, unless it is assimilated in some intelligent manner. With the advent of computing machines, this quest to represent ever-increasing information in the form of readily applicable abstract knowledge, has given birth to the field of *artificial intelligence (AI)*. Over the years, the expanding research in AI gave rise to a variety of sub-fields like *knowledge engineering*, *expert systems*, and *automated reasoning*. Although the scopes of these topics often intertwine with each other, the ultimate goal remains to replicate human cognizance. The *model-based reasoning* approach to fault diagnosis, presented in this dissertation, is one such endeavor to facilitate the task of intelligent maintenance, which, till now, has primarily remained an area dominated by human personnel.

When it comes to the maintenance and fault diagnosis of life critical machinery, human expertise, gained from years of experience, has never been and probably should not be substituted. This, however, does not detract from the importance of automated diagnosis as an aid to human judgment. The research work presented here retains its significance in as much as it helps to reduce unscheduled downtime of critical machinery, facilitate faster fault diagnosis and guide the human operator in taking intelligent run-time decisions about overall system health and operational readiness.

1.1 Problem Statement

The primary purpose of this research is to formulate a model-based reasoning (MBR) architecture to detect and classify internal component faults from observable system behavior, and to generate component-level fault propagation models in order to make an intelligent estimate of imminent threats to the system.

1.2 Objectives

The research methodology described in the following chapters incorporates a novel approach to system level fault diagnosis. In order to effectively describe the approach, the work has been subdivided into smaller modular tasks like knowledge database construction, system analysis, model synthesis, logical representation, reasoning etc. Aggregating these tasks in logical groups, the major objectives of the research work can be stated as below:

- To implement an MBR based diagnosis of unobservable faulty components from observable system behavior, aided by a functional knowledge database of system components.
- To formulate a fault propagation methodology based on the component proximity information contained in the abstracted system models synthesized during diagnosis.
- To integrate the diagnosis and fault propagation steps in an overall model-based reasoning architecture coded in an industry preferred test-bench language like MATLAB®.
- To apply the above reasoning architecture to intelligent fault diagnosis of electro-mechanical systems like helicopter gearboxes and airplane fuel delivery systems.

- To perform computational complexity analysis of the algorithms used by the reasoning framework so as to ascertain performance guarantees for online implementation.
- To implement a model verification technique in temporal logic for functional models of complex systems.

1.3 Significance of Research

The importance of information is now being appreciated in the defense industry, more than ever before. There have been major changes in the ways that the premier armed forces of the world operate. With increasing instrumentation and the better modeling and simulation environments available today, the approach to fleet maintenance has gone from reactive to proactive. This has, in turn, led to increased war-fighting capability of existing platforms. Enhancing platform effectiveness, mission availability and intended life cycle while reducing in-theatre logistical demands and overall Whole Life Cost (WLC) have now become major priorities. Terms like Health and Usage Monitoring Systems (HUMS) and Condition-Based Management (CBM)/Prognostic Health Management (PHM) have become buzzwords in the maintenance paradigm in the aerospace and defense industries.

Figure 1 depicts the big picture for these CBM/PHM paradigms. However, despite the plethora of sensors, hundreds of small components that critically affect the safe operation of the system lie beyond the scope of observation, like the gears and bearings (components) inside a gearbox (system). Even so, their individual characteristics leave indelible marks upon the externally observable system behavior. Thus, to be able to read these marks from system-level sensor data and autonomously reason about them in order to diagnose component and overall system health, has great significance in a world where human beings are being increasingly bewildered by ever increasing data that need to be processed in very little time.

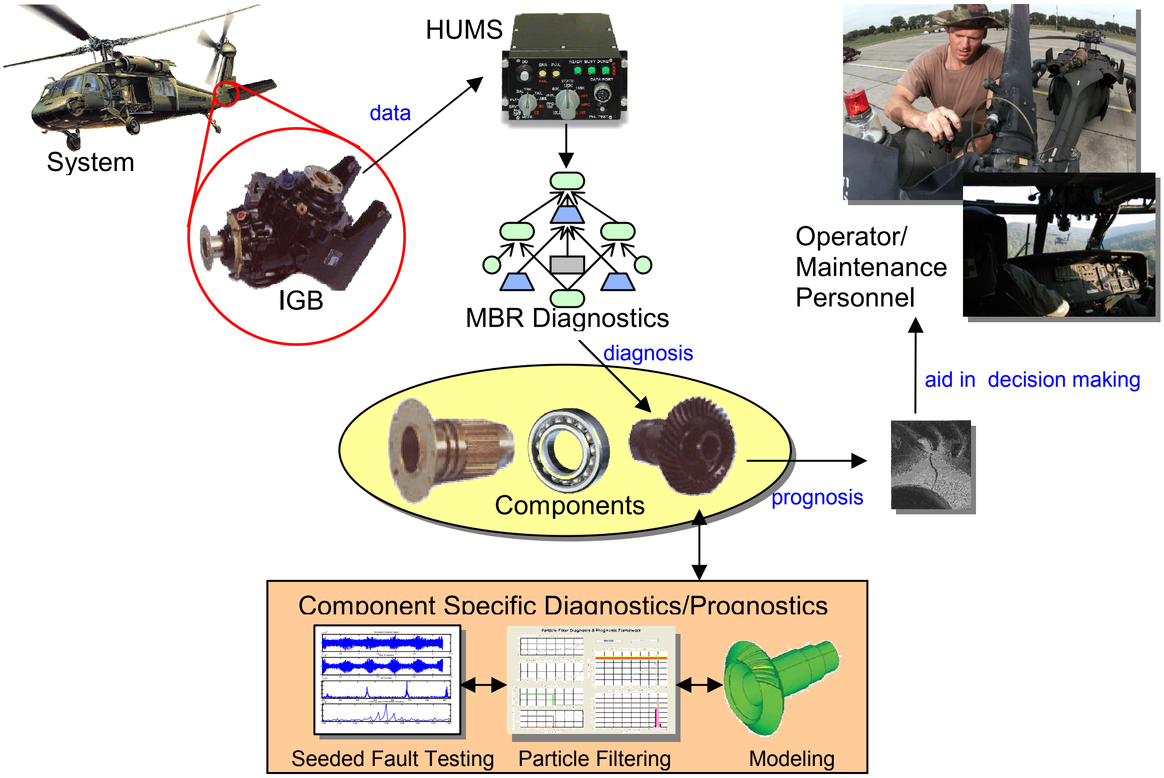


Figure 1. The overall CBM/PHM paradigm.

It is important, of course, to be able to integrate any proposed diagnostics/prognostics algorithm into the overall CBM/PHM scenario. MBR diagnostics fit nicely into this approach (shown in Figure 1), helping to identify faulty components from sensor data, and this is the focus of the research presented. In the context of helicopter power-train health monitoring, HUMS modules monitor data coming from sensors mounted on line replaceable units (LRUs) like the intermediate gearbox (IGB). In practice, most diagnostic sensors are accelerometers mounted externally on the casings of various power-train modules, and there is no way to monitor the internal components directly. To get the maximum advantage out of this setup the health data can be passed on to a model-based reasoning framework to come up with the likely fault candidates in the presence of anomalous behavior. These internal components can then be further investigated for fault progression by more computation intensive techniques, like particle filters (PF), utilizing high fidelity models. Once the identity, extent and progression rate of the fault have been appropriately determined then algorithms like particle filtering can again be used to estimate remaining useful life of the component (prognostics) in order to aid runtime decisions or to setup appropriate maintenance schedules.

2 Background Study

In this chapter we look at the motivation behind the application of MBR for diagnostic reasoning, as well as the salient points in the existing body of work in this research field. An inspection of the state-of-the-art gives us an insight into the potency of the current methods along with an understanding of the gaps that still remain to be addressed.

2.1 Motivation

The scope of MBR for fault diagnosis has been recognized by researchers in the field of AI since the late 1980's. As engineered devices became more and more complex, it gave rise to the need for some sort of automated diagnostics and testing. Although the basic principles of MBR are well understood, its application to specific domains, especially in mechanical systems, has been restricted due to insufficient model fidelity. However, the recent breakthroughs of model-based diagnosis in the field of automotive engineering have opened the door for MBR to conquer new vistas. The literature review presented in Section 2.2 describes the evolution of MBR as a diagnostic tool in greater detail.

Mechanical systems, like gearboxes, are difficult to model due to their highly nonlinear dynamics. Classical mathematical models, found in textbooks, tend to be too simplistic, whereas finite-element and other micro-modeling techniques become too tedious for real-time applications. An abstract model that captures the functional relationships within the mechanical system, thus, might hold the greatest promise of utility. Even though the information contained in such a model may be qualitative, the reasoning techniques developed in MBR are powerful enough to predict system behavior under different operational/fault modes. This motivates the application of MBR for fault diagnosis of systems that we can analyze offline but of which we cannot monitor the internal components once in operation.

2.2 Literature Review

The starting point of a model-based diagnostic reasoning methodology is the representation of a system's structure and behavior in the form of a suitable model that facilitates the diagnosis of a detected misbehavior [1]. There are two approaches to model-based diagnosis: the *consistency-based* approach and the *abductive* approach. In the *consistency-based* approach diagnosis is defined as a set of assumptions about the faulty behavior of system components, such that a detected anomaly in one component's behavior is consistent with the assumption that the other components are operating correctly [2]. This methodology only requires the expected nominal behaviors of the system components for diagnosis [3]. It follows the natural progression of reasoning of cause and effect. The simplicity of implementation makes this approach attractive; however, in complex interconnected systems the underlying assumption of one fault at a time may not hold. Although, the probability of multiple components of a system developing a fault at the same time may be small, the possibility of not detecting the single fault before it spreads to interconnected components cannot be ignored.

On the other hand, the *abductive* approach reasons from effects to causes [4]. Unlike the previous approach, abductive diagnosis looks at the models of the system's faulty operational modes. Diagnosis is defined as a set of abnormality assumptions that covers or implies the observations. Luca Console et al. analyzed the logical definitions of model-based diagnosis in the literature and proposed a unified framework that describes the relationship between consistency-based and abductive reasoning [5]. There are, of course, other diagnostic reasoning approaches, like causal reasoning and Bayesian networks, that have achieved great successes in recent times. K.W. Przytula et al. developed a procedure for the efficient creation of Bayesian networks for diagnostics with applications in diagnostic systems for diesel locomotives, satellite communication systems, and satellite testing equipment [6].

Among the great variety of domains that MBR has been applied to, the ones that have had the most successes are software debugging, configuration/reconfiguration of technical equipment, automobile diagnostics, and spacecraft systems. MBR techniques for diagnosing compromised software have successfully aided self diagnosis and failure recovery [7]. Successful applications in hardware debugging have also been reported [8]. Debugging of increasingly complex configuration knowledge databases can be achieved by consistency-based diagnosis [9]. The automotive industry has seen the introduction of a car prototype with onboard model-based diagnosis [10]. The diagnostics focus on electronic components and involve Failure Mode and Effect Analysis (FMEA) or the creation of decision trees. On the space front NASA's Deep Space One used a model-based diagnosis unit and a reactive planner for control and automatic error correction based on the model-based diagnosis system suggested by Brian Williams et al. [11]. Although, the MBR approaches presented above may have been customized for their specific domains, recent research has produced generic and widely applicable reasoning algorithms with highly scalable properties [12].

Airframe component diagnostics has been a topic of research almost from the advent of flight itself. Traditionally, such research has concentrated on specific components studied and tested in isolation [13]. The focus now has shifted to an overall system level approach. Mimnagh et al. proposed a Hotelling's T2 technique based helicopter diagnostics methodology that outperformed contemporary diagnostic methods in detecting mature and propagating drive system faults while reducing fault misclassification [14]. The Helicopter Integrated Diagnostic System (HIDS) [15] and the Vibration Management Enhancement Program (VMEP) [16] diagnostic regimes adopted by the US armed forces have given a much needed boost to vibration based condition monitoring of rotary wing aircraft. However, these are primarily data-driven techniques that ignore the underlying physics. This deficiency is acknowledged by Mimnagh et al. when they write "Further study into the mathematics of the calculated indicators should be conducted to reveal the exact nature of their relationships and the reason these relationships change in the presence of a fault" [14]. In logical progression, the state-of-the-art has seen the evolution of many model-based reasoning tools being applied to a variety of applications, some of them being commercially available off-the-shelf while others are customized for their applications. For example, the Automated Ground Engine Test Set (AGETS) MBR diagnostic tool [17] is designed to isolate failures in the Pratt & Whitney F100-PW-100/200 engines and related test equipment. However, the volume and complexity of AGETS measurements make it unsuitable for implementation onboard existing airframes.

In conclusion, therefore, the need for a system level model-based diagnostic technique for helicopter power-trains, that takes into consideration the physics behind failure mechanisms and can be run online on existing HUMS systems, is still unfulfilled.

3 Comparative Study of MBR Applications

The suitability of applying MBR to fault diagnosis of critical systems is borne out by the Joint Strike Fighter (JSF) example, which is one of the most important and complex engineering projects of the US armed forces. Although researchers on this project are not allowed to publish specific details of the MBR routines, the following extract taken from their published work on SH-60 helicopter prognostics [18] (the same application domain as ours) underscores the need for MBR:

"Without features based upon the mechanics of failures, or a method of discriminating benign from detrimental novelty, the simplistic detection of deviations from some baseline is likely to produce many false alarms and inaccurate remaining life projections. This is one area where the integration of prognostic analyses with advanced diagnostic methods, such as model-based reasoning (MBR), is most advantageous. MBR algorithms can differentiate normal operational changes from detrimental novelty. This assessment, as well as its capability to analyze multiple failure conditions, will make prognostic estimations more robust."

The following section presents a comparative study of different applications of MBR to fault diagnosis of various engineered systems. We describe each methodology and its application in brief and list their salient features as well as the gaps left in their approach that are addressed by our methodology. These gaps are referenced by cell coordinates in the feature comparison matrix shown in Table 1 in Section 3.7. The chosen literature represents a wide spectrum of perspectives, from the theoretical to the application specific as well as a variety of domains ranging from digital circuits to advanced avionics.

3.1 General Diagnostic Engine (GDE)

Johan de Kleer and Brian C. Williams published a seminal work in model-based diagnosis in 1987 [2], in which they implemented the GDE and tested it in the troubleshooting of various digital circuits. Their underlying theory of diagnostics is the inference of the behavior of the composite device from the knowledge of the structure and function of its individual component units. Figure 2 represents the reasoning methodology that attempts to interpret the differences between the system model and observed behavioral artifacts in terms of a diagnostic methodology.

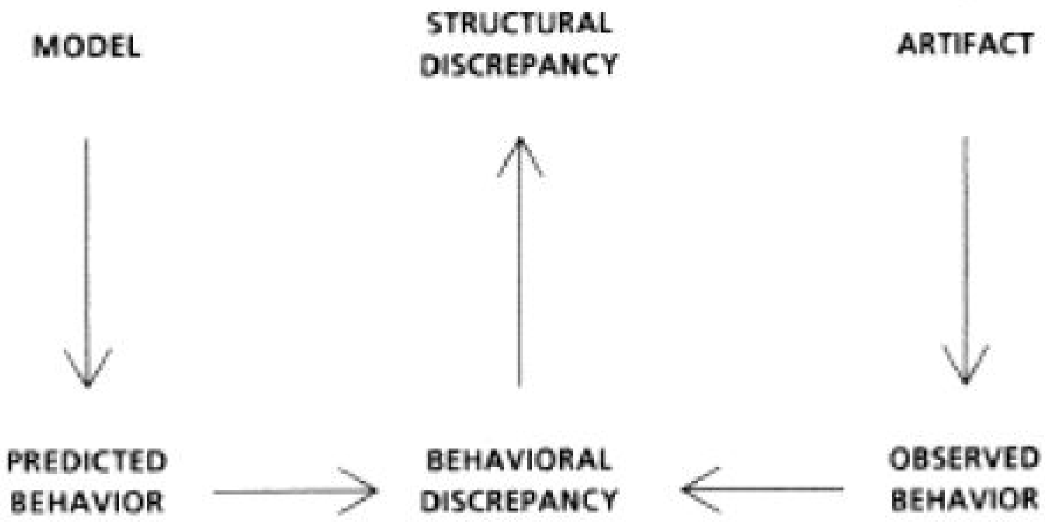


Figure 2. Model-artifact difference [2].

The novel contributions made by this research are:

- Multiple faults can be diagnosed.

- Failure candidates are represented and manipulated in terms of minimal sets of violated assumptions, resulting in efficient diagnosis.
- The diagnostic process is incremental with respect to iterations.
- Diagnosis is separated from behavior prediction, resulting in a domain independent methodology.
- GDE combines model-based prediction with diagnosis to suggest measurements to isolate the fault.

Although the theoretical groundwork for model-based diagnosis laid down by this body of work is extensive, there remain a few points upon which improvements may be made:

- This approach is *consistency-based*, meaning that a fault symptom is defined as any difference between a model inference and observation, leading to larger fault candidate sets (cell C-1 in Table 1).
- A priori knowledge for the chosen application consists of the circuit topology (structural model) along with the functional description of each of its components. Since the fault modes of the components are not modeled, this methodology cannot bypass the complications of constraint propagation by using a fault-symptom matrix. In the chosen application domain of electrical circuits, this constraint was simply Ohm's law, $v = iR$. Such simplicity is impossible in dealing with inherently more complicated systems like a gearbox (cell F-1 in Table 1).
- The incremental diagnosis process relies on the proposed measurements monotonically reducing the fault candidate search space, so as to guarantee the convergence and uniqueness of the minimal set solution. In complex, intractable

systems such measurements may not be possible requiring additional tests (cell G-1 in Table 1).

- This methodology uses a sequence of measurements to diagnose a fault scenario assuming there is no fault propagation (cell I-1 in Table 1).

3.2 AGETS MBR

Automated Ground Engine Test Set (AGETS) is a model-based diagnostic tool designed to isolate failures between Pratt & Whitney's F100-PW-100/200 engine and related test equipment using a system-level troubleshooting approach [17]. The backbone of the AGETS tool is the Qualitative Reasoning System (QRS) software that is comprised of two major components: a qualitative model developer and a qualitative reasoner. The salient steps of this methodology are listed below:

- QRS first uses constraint propagation on the model to detect the presence of a failure under given symptoms.
- QRS then uses hierarchical constraint propagation to determine the candidate failures that explain the observed symptoms.
- For this list of probable failures, QRS predicts the values for model parameters that have not been measured.
- Intelligent Test Selection is used to choose the measurement/test that has the greatest overall utility in terms of the probability of isolating the fault, the probabilities of various component failures and the cost of the test. This generates the *test set*.

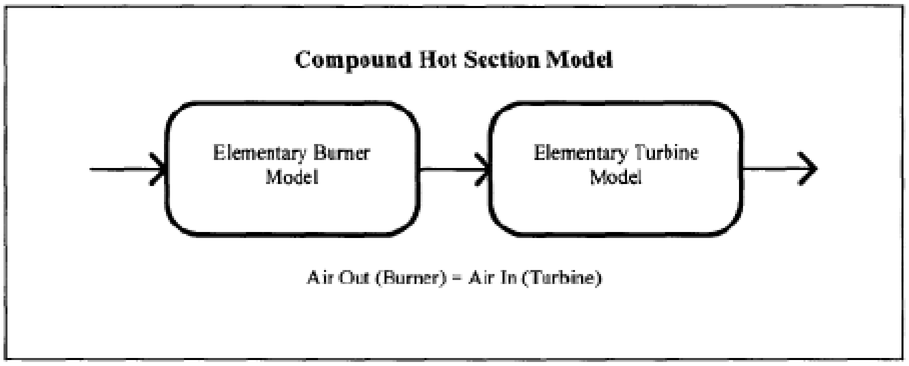

Compound Hot Section Model
Elementary Burner Model
Elementary Turbine Model
Air Out (Burner) = Air In (Turbine)

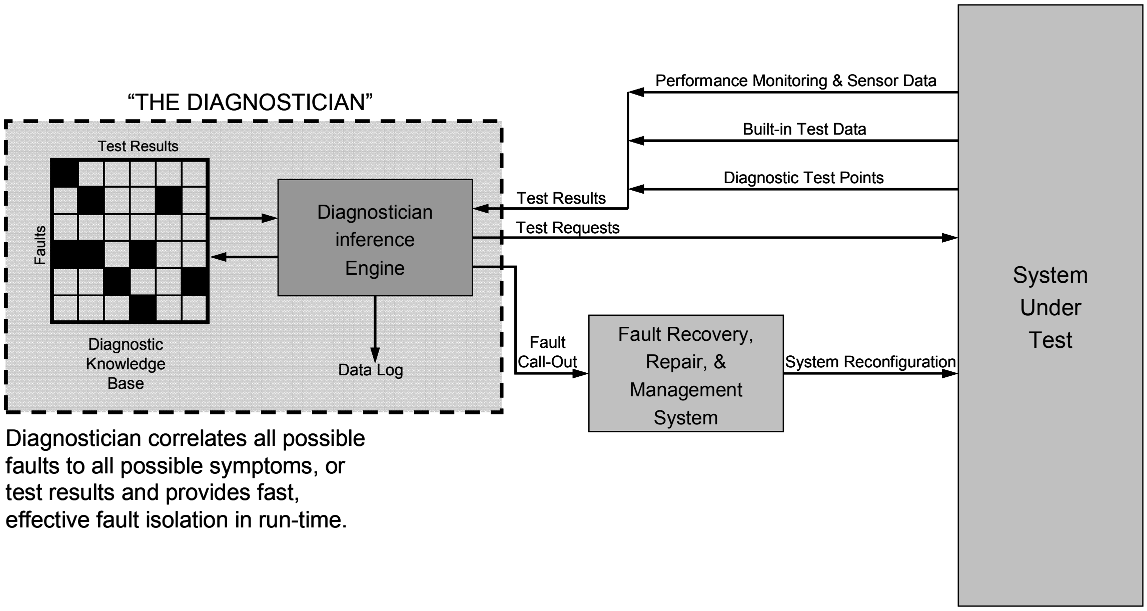
Figure 3. Compound model for turbofan engine [17].

The AGETS MBR, though an invaluable diagnostic aid to engine maintenance personnel, has its own share of cons, which in many cases led to difficulties in troubleshooting during F100 engine tests. The points of difference of the AGETS MBR scheme vis-à-vis our methodology are as follows:

- The physics underlying the failure mechanisms have been neglected while modeling the nominal modes (cell C-2 in Table 1).
- AGETS MBR is also a *consistency-based* approach, requiring a large number of complicated models to represent a highly sophisticated system like an aircraft engine. Across three AGETS configurations, 1362-1398 elementary and 229-242 compound models were used, with a high number of models being unique in structure. Figure 3 shows a compound model of a commercial turbofan engine. Such a model is a combination of both structural and functional information without looking at the component level (cell E-2 in Table 1).
- The sensor requirements for the test set produced may not be feasible in an onboard environment (cell G-2 in Table 1).
- The volume and complexity of the AGETS measurements may necessitate an offline analysis (cell H-2 in Table 1).

- The possibilities for fault propagation are not investigated (cell I-2 in Table 1).

3.3 Diagnostician-on-a-Chip (DOC)



Diagnostician correlates all possible faults to all possible symptoms, or test results and provides fast, effective fault isolation in run-time.

Figure 4. Schematic for Diagnostician-on-a-Chip (DOC) [19].

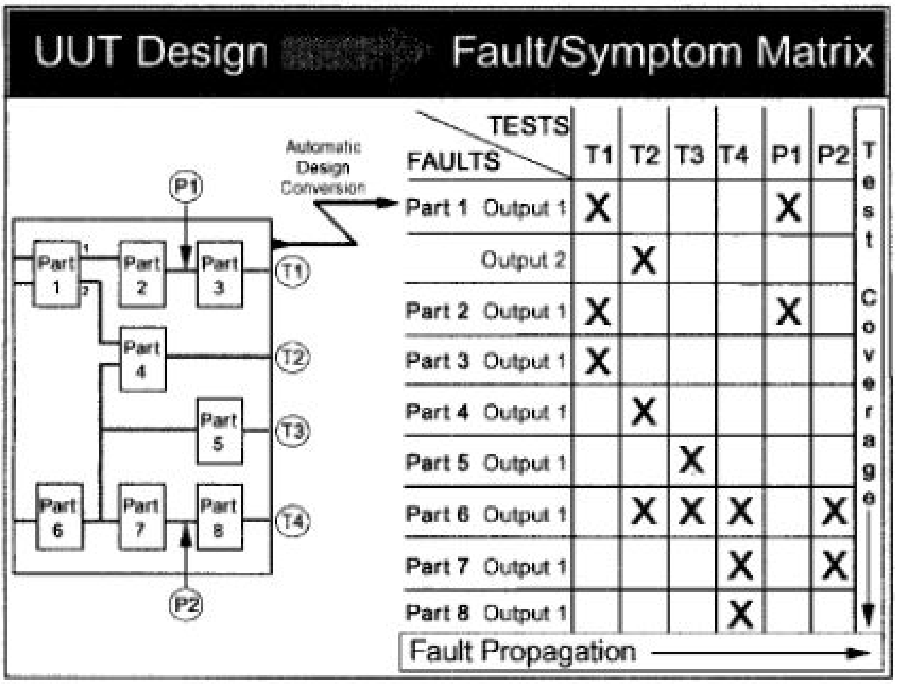
The DOC approach presented by Nolan and Giordano in 1997 [19] is based upon the use of microcontroller technology and an automated Concurrent Engineering Tool Set (CETS) comprising both a development environment (Diagnostic Profiler) and runtime software (Diagnostician). The Diagnostician reads system data, detects and isolates failure conditions and ultimately reconfigures the system in an optimal way. The key contribution was the capability to "adapt" the same diagnostic model to various reconfigured states. This methodology was applied to the design of a fault tolerant remote solid-state power controller. A schematic of the DOC concept is shown in Figure 4. The following technologies represent the salient points of this endeavor:

- Use of a design model to implement all diagnostic logic and the integration of the model with on-line, embedded performance monitoring and built-in test functions.

- Implementation of the model-based solution on a single-chip microcontroller for integration in an embedded environment.
- Full software support of operational and failure data supporting extensive operations monitoring and management from an off-system or remote location.
- The adaptability of the design-based model to new hardware configurations while maintaining the same functionality.
- The ability to dynamically reconfigure hardware resources in real-time to accommodate a failure event and maintain operations.
- The ability to detect, isolate and reconfigure around simultaneous multiple faults occurring in independent portions of the circuitry.
- Though not fully implemented, the model-base and software structure enables prognosis by monitoring “rate of change” of voltage levels to provide an indication of impending failure events.
- Implementation of above technologies in a structured, automated, generic systems engineering approach.

The above methodology uses the *abductive* approach to MBR implemented as a diagnostic model derived from design data. In the abductive paradigm, the fault modes of the system are modeled as opposed to the nominal behavior. Apart from the reconfigurability aspect (which is beyond the scope of our research), when compared to our methodology, some differences in the fault diagnosis approach stand out:

- The structural and functional representations are merged together into one model (Figure 5). Although this does not hamper reconfiguration of the targeted system, it inhibits the rapid prototyping of a similar system from the existing model database (cell A-3 in Table 1).



FAULTS \ TESTS		T1	T2	T3	T4	P1	P2
Part 1	Output 1	X				X	
	Output 2		X				
Part 2	Output 1	X				X	
Part 3	Output 1	X					
Part 4	Output 1		X				
Part 5	Output 1			X			
Part 6	Output 1		X	X	X		X
Part 7	Output 1				X		X
Part 8	Output 1				X		

Figure 5. Diagnostic model of system [19].

- The Diagnostic Profiler converts the CAD data into a Fault/Symptom matrix which represents the propagation of faults to sensors as rows, and the fault coverage of monitored locations as columns (Figure 5). The integration of fault propagation into the matrix necessitates that the sequence in which the sensors would indicate a spreading fault (if at all) would have to be the same for all fault modes. This is not necessary for all systems and hence a logical separation of these two steps may be required (cell D-3 in Table 1).

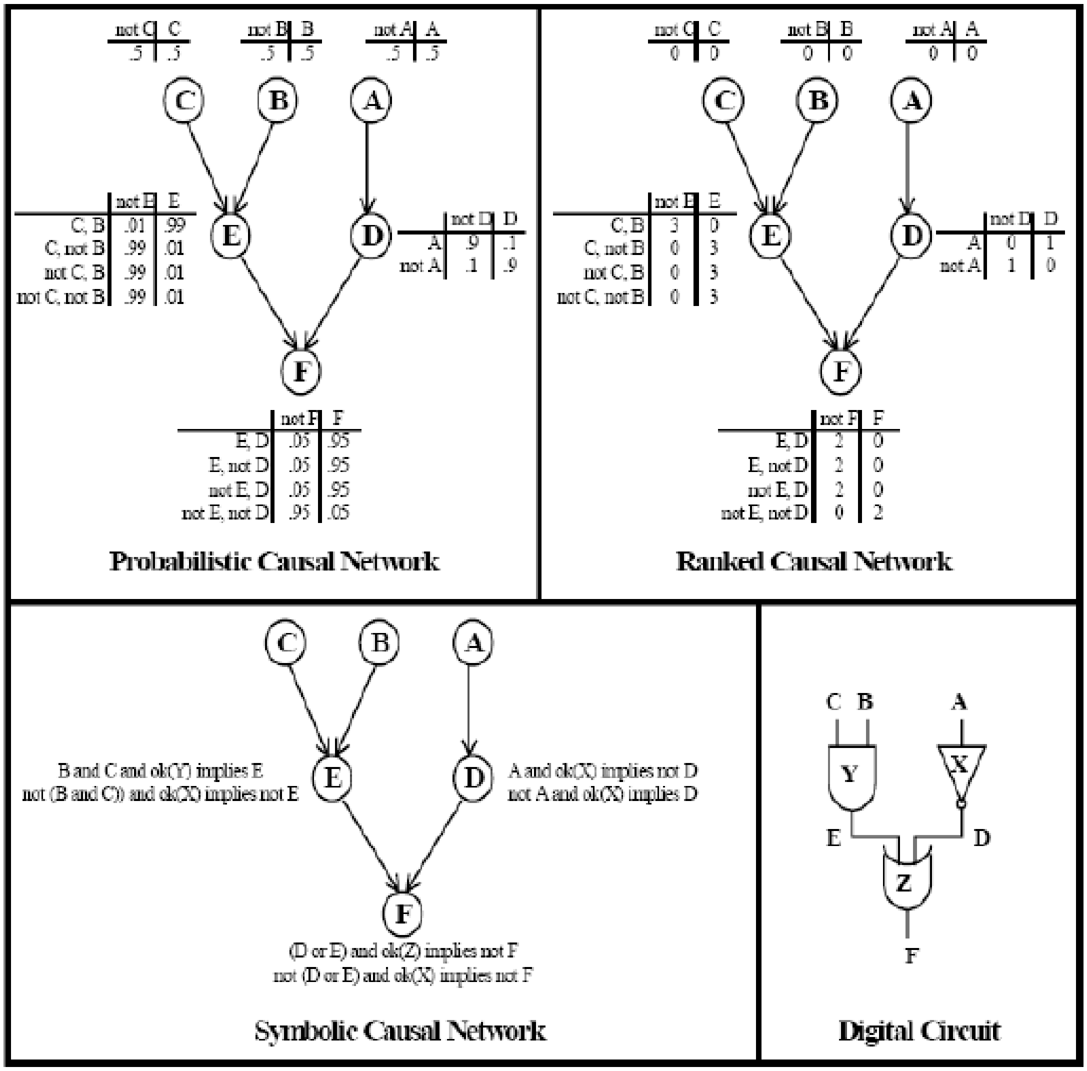
3.4 Causal Network Inference System (CNETS)

A causal network is a graph-based system representation that can be used to simulate both normal and abnormal behaviors, as well as to diagnose faults. In 1998 Misra et al proposed CNETS as a system for representing and reasoning with causal networks [12]. The workings of CNETS are described in brief below:

- First the user creates a model by specifying cause/effect relationships for the system.
- The user inputs the system measurements into the CNETS software.
- Finally the user makes queries about the system health with respect to the available evidence and the causal network representation of the system model.
- CNETS has a number of inference algorithms to deal with the queries, the main one being the clique-tree algorithm.
- CNETS has a compiler that allows the user to generate a runtime version of the system model.
- Order of complexity, in terms of model parameters, for the CNETS approach to diagnostic inference can be derived by analysis.

The CNETS approach differs from ours in a few key aspects:

- The system model combines the structural and functional aspects in a simple logical construct (Figure 6) which may be unsuitable for complex systems like a gearbox (cell B-4 in Table 1).



not C	C
.5	.5

not B	B
.5	.5

not A	A
.5	.5

	not E	E
C, B	.01	.99
C, not B	.99	.01
not C, B	.99	.01
not C, not B	.99	.01

	not D	D
A	.9	.1
not A	.1	.9

	not F	F
E, D	.05	.95
E, not D	.05	.95
not E, D	.05	.95
not E, not D	.95	.05

Probabilistic Causal Network

not C	C
0	0

not B	B
0	0

not A	A
0	0

	not E	E
C, B	3	0
C, not B	0	3
not C, B	0	3
not C, not B	0	3

	not D	D
A	0	1
not A	1	0

	not F	F
E, D	2	0
E, not D	2	0
not E, D	2	0
not E, not D	0	2

Ranked Causal Network

B and C and ok(Y) implies E
not (B and C)) and ok(X) implies not E

A and ok(X) implies not D
not A and ok(X) implies D

(D or E) and ok(Z) implies not F
not (D or E) and ok(X) implies not F

Symbolic Causal Network

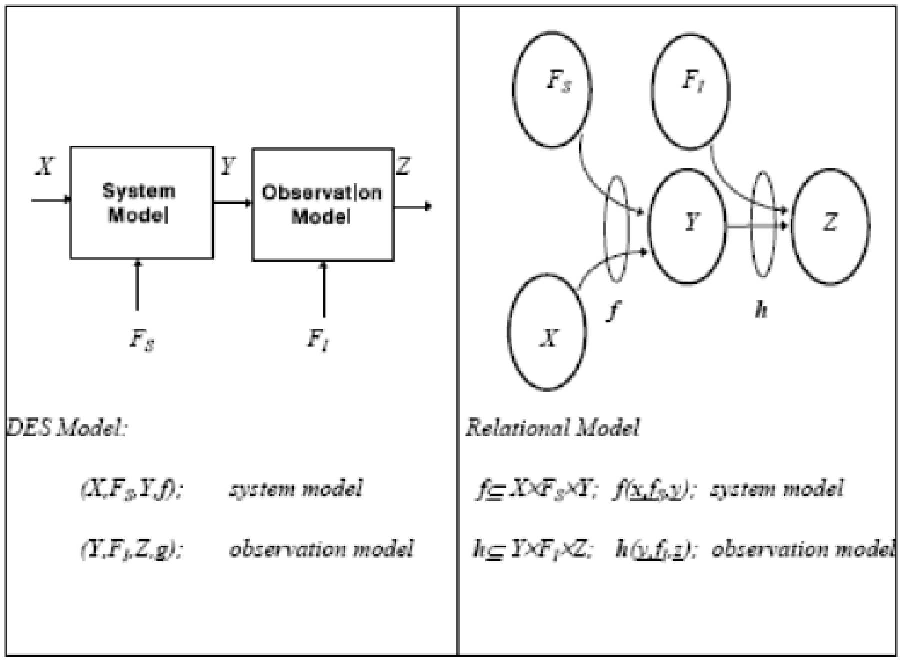
Digital Circuit

Figure 6. Quantification of causal networks [12].

- Fault modes are not modeled (cell C-4 in Table 1**Error! Reference source not found.**).
- In the CNETS paradigm the user has to make the system health queries as opposed to the automated system fault diagnostics, fault propagation and the intelligent status report generated by our approach (cell G-4 in Table 1).

3.5 *Ordered Binary Decision Diagrams (OBDD)*

Misra et al also proposed the concept of applying OBDDs for the diagnostic reasoning of Discrete Event Systems (DES) [12]. This is more of a unified approach in which they incorporate both the nominal inputs to the system as well as the fault modes as shown in Figure 7. OBDDs provide a symbolic representation for Boolean functions in the form of directed acyclic graphs. Diagnostic reasoning using OBDDs includes the following steps:



X
System Model
Y
Observation Model
Z
F_S
F_I
DES Model:
(X,F_S,Y,f): system model
(Y,F_I,Z,g): observation model
F_S
F_I
Y
Z
X
f
h
Relational Model
f⊆X×F_S×Y; f(x,f_s,y): system model
h⊆Y×F_I×Z; h(y,f_I,z): observation model

Figure 7. DES and relational models [12].

- Translating the DES or relational models into OBDDs can be done automatically. Domain specific models can be translated into OBDD format using model interpreters.
- Diagnosability and safety analyses are done symbolically using OBDDs. These criteria are expressed in terms of logical relationships on the discrete state trajectory by the models, and checked using the OBDD algorithms.

- Diagnosis occurs in two steps. First, a generic runtime support is created which includes a diagnostic engine implemented with OBDD algorithms. Second, the software synthesis component configures the runtime system using the computational model and synthesizes the OBDDs for the models.

As elegant as this method may be for a wide variety of systems like digital circuits, switching, communication networks etc., there are a few caveats:

- The a priori information (DES model) includes the structural and functional models as well as Fault mode information. This significantly reduces flexibility of the approach to prototype new but similar systems for diagnostic purposes (cell A-5 in Table 1).
- Not all application domains are suited to Boolean representation. For example, continuous systems cannot be intuitively modeled as discrete event systems. There will always be some approximation involved with DES models (cell B-5 in Table 1).

3.6 MBR Toolset for Power System Diagnostics

In 2003 Davidson et al proposed a software-based toolset to aid the application and development of model-based reasoning systems for validation and diagnosis in power systems [20]. System models are constructed with a component/structure based view of the system. During diagnosis, measurements are propagated through the models in order to predict the behavior of the system. This behavior is then checked for discrepancies with measured system observations. The salient points of their technique are:

- Provide a toolset that allows non-software engineers to construct MBR systems.

- Support various models and data types.
- Support various consistency-based and abductive techniques while incorporating temporal issues and modeling and measurement inaccuracies.
- Provide a flexible, reusable diagnostic engine (Figure 8) designed to allow the use of different component models, data types and MBR techniques within the same MBR framework.

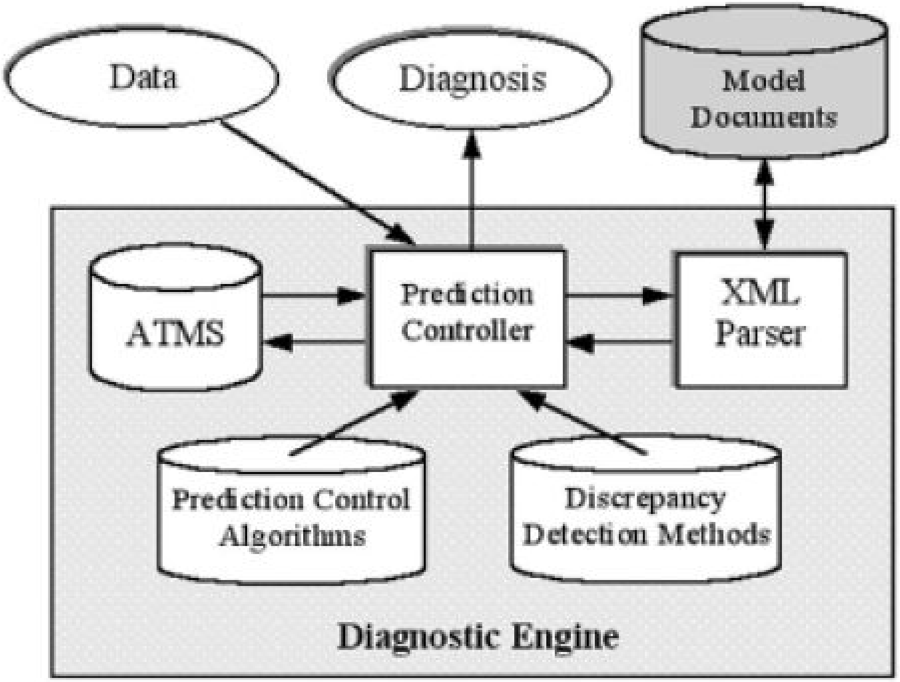


Figure 8. Diagnostic engine architecture [20].

The MBR toolset described above differs from the one presented in this dissertation in a few significant ways. The following list briefly touches upon each of these points:

- For every new data set, the evaluation of the diagnostic decision tree in our methodology needs to be carried out only once, whereas multiple simulations may need to be performed for multiple models stored in the model documents database in the MBR toolset described above (cell F-6 in Table 1).

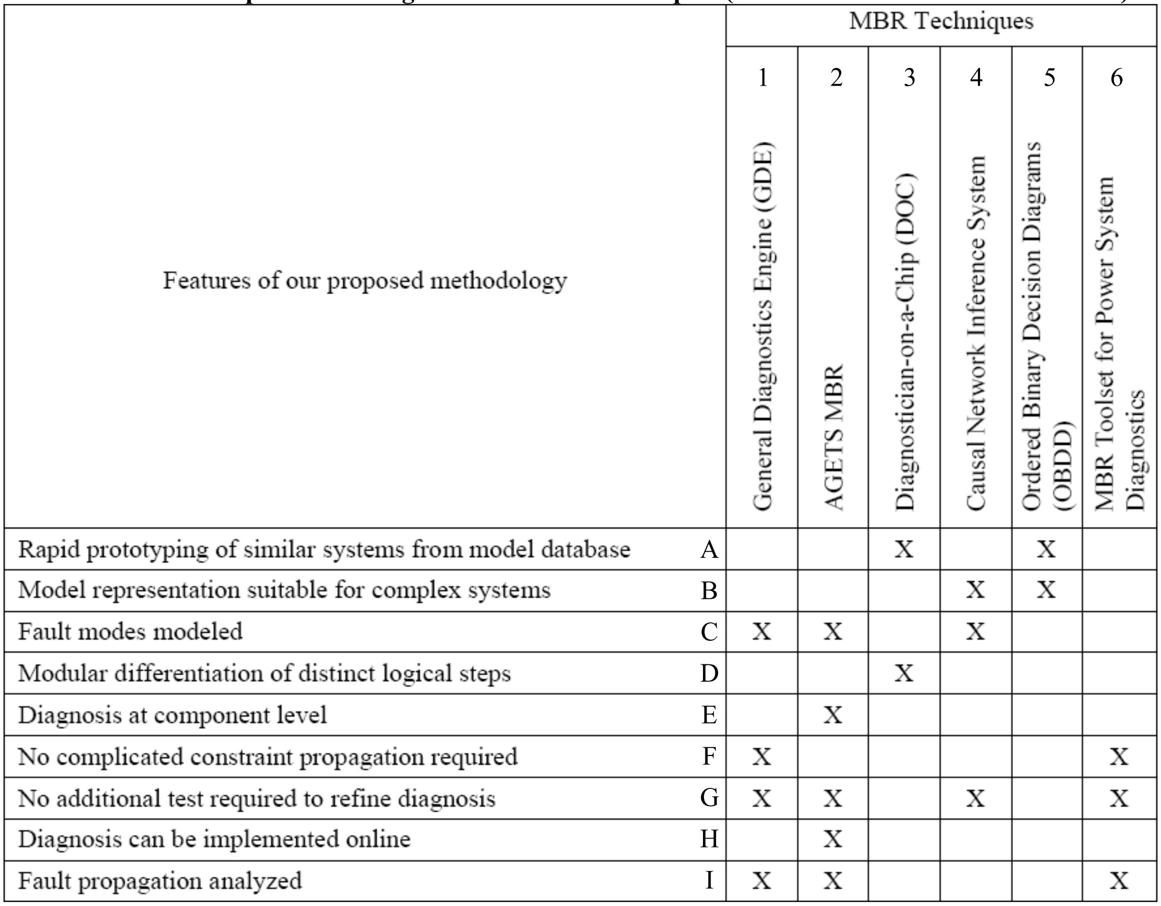
- As shown in Figure 8 the Prediction Controller works on system data. Consequently, it needs to simulate the system operation using the prediction control algorithms to generate the next system state. In the case of complex systems, such simulations are in general more resource-intensive than evaluating a decision tree based on features extracted from data (cell G-6 in Table 1).
- The issue of fault propagation is more easily and intuitively handled when addressed from the structural model perspective instead of the system simulation angle. In the latter case data manipulations tends to obscure the physics behind the process (cell I-6 in Table 1).

3.7 Comparison Summary

The MBR methodologies presented above have their individual viewpoints and their associated pros and cons. However, the need to have a modular MBR methodology for the rapid prototyping of diagnostic models of distinct but similar systems, and an automated scheme to bridge the gaps between modeling, diagnostic reasoning and fault propagation, in an intuitive way, has not been satisfied. These are the issues that our methodology attempts to address.

Table 1 summarizes the missing points of the above discussed techniques in terms of the features of our reasoning framework. Each cell with an X, representing a feature of our methodology (denoted by the row label) absent in the MBR technique (given by the column label), is referenced by the corresponding letter and number coordinates in the preceding sections.

Table 1. Feature comparison among different MBR techniques (X denotes features that are absent).



Features of our proposed methodology		MBR Techniques					
		1	2	3	4	5	6
		General Diagnostics Engine (GDE)	AGETS MBR	Diagnostician-on-a-Chip (DOC)	Causal Network Inference System	Ordered Binary Decision Diagrams (OBDD)	MBR Toolset for Power System Diagnostics
Rapid prototyping of similar systems from model database	A			X		X	
Model representation suitable for complex systems	B				X	X	
Fault modes modeled	C	X	X		X		
Modular differentiation of distinct logical steps	D			X			
Diagnosis at component level	E		X				
No complicated constraint propagation required	F	X					X
No additional test required to refine diagnosis	G	X	X		X		X
Diagnosis can be implemented online	H		X				
Fault propagation analyzed	I	X	X				X

4 MBR Theory

4.1 The Reasoning Paradigm

The model-based reasoning paradigm introduced in this dissertation goes beyond a simple analytic view of the problem at hand. Section 4.1.1 briefly describes the overall approach whereas the later sections explain each step of the methodology in greater detail, along with some simple examples. Following that, the application specific details for validating this approach on a helicopter intermediate gearbox are discussed. However, before we proceed to the technical aspects of our approach, it is necessary to introduce a few key concepts of MBR.

- **Model** – An executable, declarative structure representing an objective system (usually physical).
- **Model-Based Reasoning** – The use of explicit models to aid intelligent reasoning processes in achieving set goals within such domains as diagnosis, explanation, training, etc.
- **Structural Model** – A model representation that shows what components are present in the system and how they are connected together.
- **Functional Model** – An abstract model composed from representations of the fundamental physical processes manifested in the system. A functional model uses qualitative representation of real-world signals for reasoning about the interactions among these processes.

4.1.1 MBR Approach to Fault Diagnosis

Given a system to diagnose, we analyze it manually to extract information about the internal components. Literature survey of fault analysis of these components is conducted to build the *knowledge database*. Visual inspection of the component interconnectivity gives us the *structural model*. From there, the MATLAB® encoded MBR program goes through various intermediate steps to generate the *diagnostic tree*. In the presence of faulty data, this tree automatically triggers the *fault propagation tree*. The outputs of both these trees combine to provide the system-level fault diagnosis. The implementation steps of the model-based diagnostic reasoning approach outlined above are as follows:

- Build a database of functional models and condition-indicators for the basic building blocks of the specified application domain (manual process).
- Given a system, generate the structural model by analysis (manual process).
- Synthesize the functional model of the system from the functional description database (automated in MATLAB®).
- Generate a fault-symptom matrix from the functional model and the CI database (automated in MATLAB®).
- Create a diagnostic tree from the fault-symptom matrix (automated in MATLAB®).
- From the structural model, generate the fault propagation trees for the faults detected by the diagnostic tree (automated in MATLAB®).
- Update the health status of the system components based on the data evidence, the diagnosis and the fault propagation model (automated in MATLAB®).

A schematic for the above steps is shown in Figure 9. The model-based reasoning architecture depicted as well as the automated steps denoted by blue arrows are part of the contributions of this research. The following subsection provides a brief overview of the implementation steps of our methodology, following which Sections 4.2–4.5 elaborate further on the details of each step.

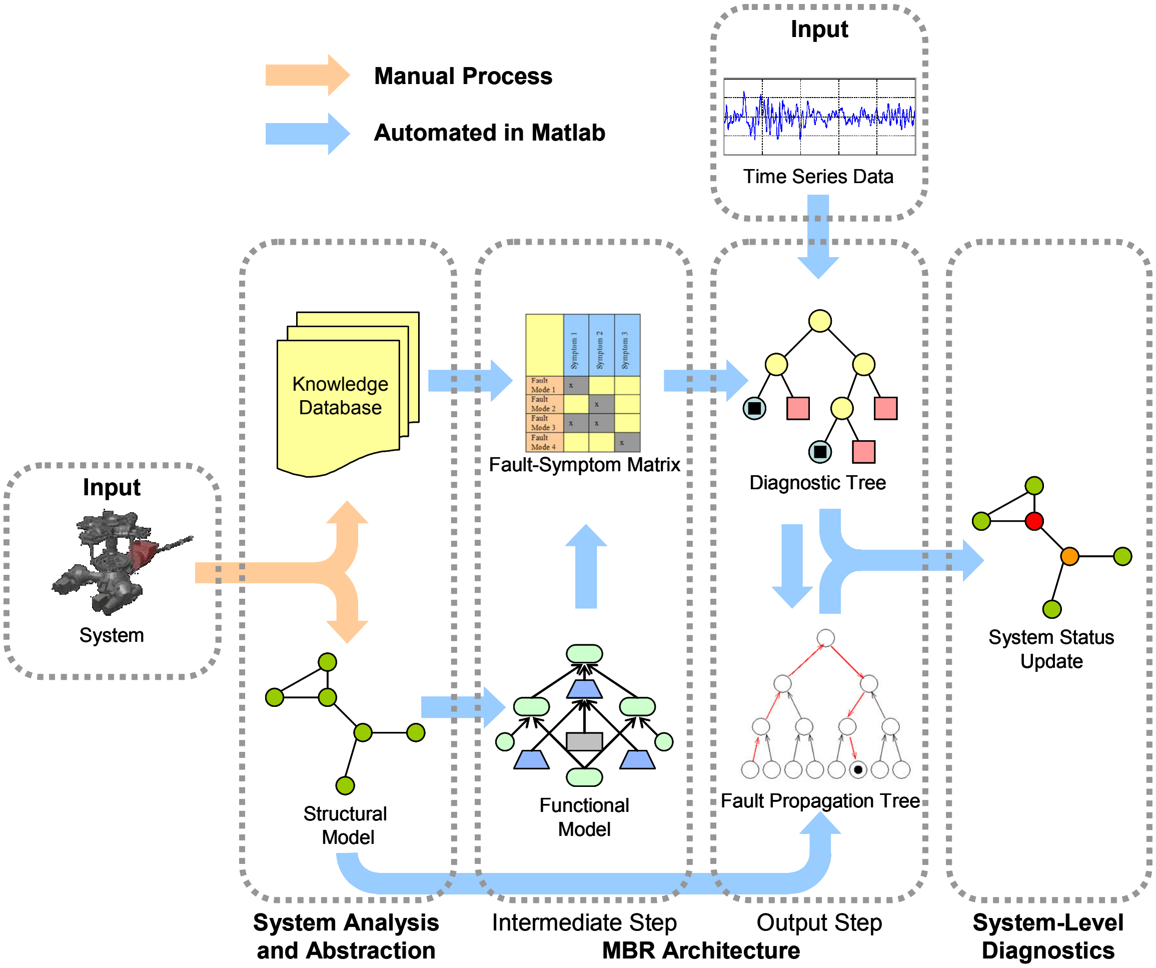


Figure 9. The model-based diagnostic reasoning architecture.

The MBR architecture presented in Figure 9 fits perfectly between the HUMS equipment and the component specific diagnostics/prognostics routine shown in the overall CBM/PHM paradigm in Figure 1. Time series data from the HUMS equipment can be passed on as input to our MBR architecture (as shown in Figure 9) to come up with the candidate list of faulty and "at risk" components (from fault propagation). Fault progression in these internal components can then be further diagnosed by more detailed techniques utilizing high fidelity models. For a given component, estimates of the extent of the fault and its rate of progression can be used as input to a prognostics routine that computes its remaining useful life. Such a prediction can be an invaluable aid to decision making for operators or maintenance personnel in critical situations. However, uncertainty management remains the key hurdle faced by such diagnostics and prognostics algorithms. A Bayesian treatment of this problem, perhaps in the form of a *particle filtering* framework, provides an elegant and theoretically sound approach to the modern CBM/PHM paradigm.

4.1.2 Brief Overview of Methodology

The system under observation is broken down into an interconnected set of component units, and subsequently, the expected overall system behavior is formulated as an interaction of the individual component behaviors. An *a priori* knowledge base of the functional characteristics of the underlying components of the specific application domain is created. In association with these functional descriptions the fault modes and their symptoms (called *features* or *condition indicators,* CI's*)* are stored for each component. Once the database is built, the application phase for a given system can begin.

Structural analysis of the target system is carried out to model it in terms of the components stored in the database and to extract the physical interconnections between them. This is denoted as the *structural model*. Proceeding further from the structural description of the system, we then focus our attention on the behavioral description, which forms the basis of model-based analysis. For each component in the structural model, using the corresponding functional descriptions from the database, a *functional model* of the system is synthesized. This is a higher level of abstraction of the system, one which allows us to reason about the overall system behavior. This analysis is exceptionally useful for diagnosing faults in systems where the internal components are not accessible to instrumented monitoring.

Based on the above described functional model and the features stored in the database a *fault-symptom* matrix is constructed. This matrix is a tabulation of the individual component fault modes and their respective symptoms that are observable in the overall system behavior. The thresholds for these symptoms are arbitrary and can be derived from theoretical constraints or from experimental baseline (non-faulty) data. The formulation steps described till now, including the structural and functional model and the fault-symptom matrix, sets the stage for the execution of the diagnostic reasoning algorithm.

The main step of our methodology is the creation of the *diagnostic tree* from the fault-symptom matrix. This tree lists the tests to be performed in order to determine the health status of the system components. The order of the tests, as laid out in the diagnostic tree, follows a descending order in the classification scope of the test. For example, a test that detects a fault in the broader class of gears would be performed before the tests that can isolate the faults of one gear from another. Subsequently, from the component interconnection information contained in the structural model a *fault propagation tree* is generated at run-time for every faulty component. Once the faulty components and those most likely to be affected next (as derived from the fault propagation model) are determined, their status is appropriately updated in the overall system health report.

The focus of this approach is to keep the whole procedure as little computationally intensive as possible so as to enhance the real-time online implementation potential. The scope of this methodology lies in assisting maintenance personnel as well as in preventing unscheduled down-time. It can help the operators of the system in making intelligent decisions about its reliability under operational conditions. Furthermore, it can provide useful input to prognostic algorithms about the initial conditions for fault growth models.

4.2 The Knowledge Database

The starting point of this diagnostic reasoning methodology is the creation of the *a priori* knowledge database. The target application domain is inspected to determine the primary building blocks along with their respective functional roles. In the decomposition of a system into its components, it may be possible to go down to extremely small micro-levels like finite-element models or even the molecular or atomic level and start to synthesize system level behavior from there. Although this process of micro-level characterization might give us a better understanding of the system, it is not always desirable since it can be extremely tedious and computationally intensive. Often satisfactory results can be achieved by looking only as deep as the macro level where we have commercially available, replaceable components whose behaviors have been studied in detail. It is to this level that the system analysis process of this methodology is restricted.

Once the component units have been identified, their behavior is studied and stored in the database in the form of functional descriptions. Later on, the functional description of the system will be described in terms of these behaviors. Associated with these functions the components also have specific fault modes. Unless externally affected, most system failures have their roots in these component unit fault modes. Depending on the resources at hand, e.g., the sensors and the data collection/analysis equipment available, the behavioral anomalies can be monitored in a variety of ways. These observations, represented in the form of features extracted from signals gathered from the system, are called *fault features* or *condition indicators* (CI's). The features are also stored in the database along with the functional description for each component. A schematic of the data structure of the database is shown in Figure 10. The basic unit of reasoning here is the fault mode *j* of component *i*, represented logically as:

$$F_{ij} = \bigwedge_{k} f_{ijk} \tag{1}$$

where, f denotes the feature threshold being exceeded, k is the symptom index and $\wedge$ denotes logical AND.

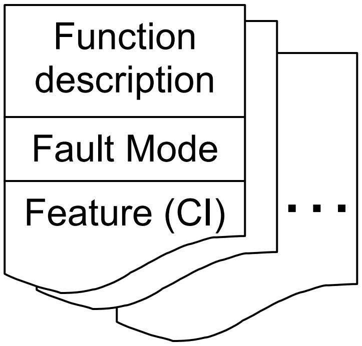


Figure 10. Data structure for representing system components in database.

Although outside the scope of this research, the effectiveness of the knowledge base can be further improved if the component information stored is derived from the Failure Mode, Effects, and Criticality Analysis (FMECA) [21] study of the concerned system. FMECA is a powerful design/analysis tool that is used to increase system reliability. It can be applied during the initial design phase or to existing equipment. To be most effective, the FMECA should be applied at the design stage itself. In either case, it considers overall design, operating, and service problems, while at the same time addressing process and safety problems. It is an enhancement of the FMEA methodology [21] in which a criticality analysis is performed. Criticality analysis involves assigning a frequency to each failure mode and a severity to each failure effect. Criticality is a function of the severity of the effect and the frequency with which it is expected to occur. The purpose of this analysis is to rank each potential failure mode identified in the FMEA study according to the combined influence of severity classification and its probability of occurrence. The knowledge database entries for the different failure modes of each component can be easily augmented with such information.

4.3 System Model Abstraction

Having created the database, we move on to the model abstraction part. Given a system, we start out by analyzing its structural links. The different component parts are identified and their specific structural organization is stored in the form of a *structural model*. This data structure stores information about which components from the database make up the given system, how many there are of each type and how they are spatially arranged and interconnected. Information about the location of each component and its relative proximity to other neighboring components is crucial in predicting how a local failure in one component may propagate though the entire system. Knowledge about the component interconnections is also important since the type of connectivity may sometimes restrict certain degrees of freedom of some component and hence affect its operational modes. For example, the vibration modes of a shaft mounted rigidly on bearings at both ends differ considerably from one which has an overhanging rotor on one end. The information stored in the structural model is, thus, indispensable in our diagnostic effort. In logical terms, this information is represented as:

$$\begin{aligned}&\textit{Set of components},\ \mathrm{V} = \{v_1, v_2, \ldots, v_n\}\\ &\textit{Interconnection},\ \mathrm{E} = \{<v_a,v_b>, <v_c,v_d>, \ldots, <v_i,v_j>, \ldots\}\\ &\textit{Structural Model},\ \mathrm{M_S} \equiv \mathrm{V}\cup\mathrm{E}.\end{aligned} \qquad (2)$$

where, $\cup$ denotes the union of two sets.

The *functional model* of the system is constructed by traversing the partially connected graph represented by the structural model and substituting the corresponding function for each component from the database. This model is also stored in the same data structure as the structural model, except that an additional function field is added to each component. This model allows us to explain the exhibited behavior of the overall system in terms of the functions performed by each individual component. Any anomaly in the system response is then reasoned about and expressed in terms of faulty operational mode(s) of one or more components. In fact, the nominal system behavior is described as the absence of all known fault modes. This concept is formalized as:

$$\textit{Functional Model},\ \mathrm{M_F} \equiv \bigwedge_i \bigwedge_j \neg F_{ij} \qquad (3)$$

where, F_{ij} denotes the fault mode j of component i and $\neg$ denotes logical NOT.

As a specific example we look at the oil cooler of the H-60 helicopter. Its primary function is to cool the helicopter transmission lubricant while transmitting power to the tail rotor drive shaft through the oil cooler shaft. Figure 11 shows the structural model of the oil cooler while Figure 12 depicts its functional model. The components of the oil cooler are centered on a splined shaft supported at the front by two shielded cartridge bearings and in the rear by a viscous damper bearing. The oil cooler fan assembly consists of a multi-bladed rotor housed inside a concentric stator. The physical arrangement of the above mentioned components as well as their interconnections comprise the structural model (Figure 11).

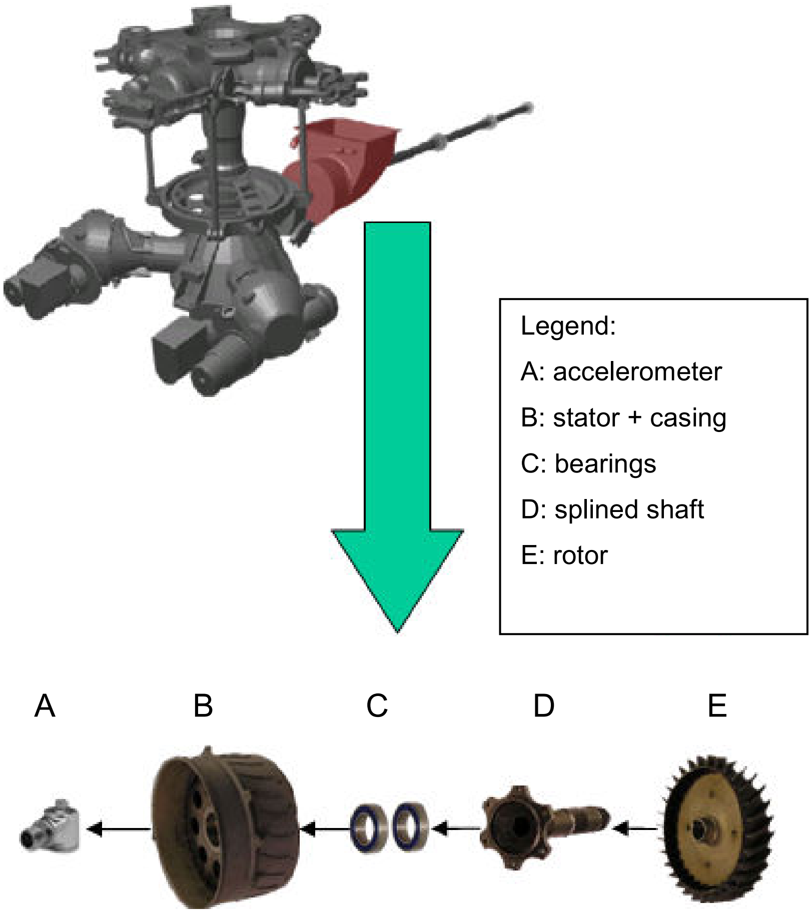


Figure 11. Structural model of the H-60 oil cooler.

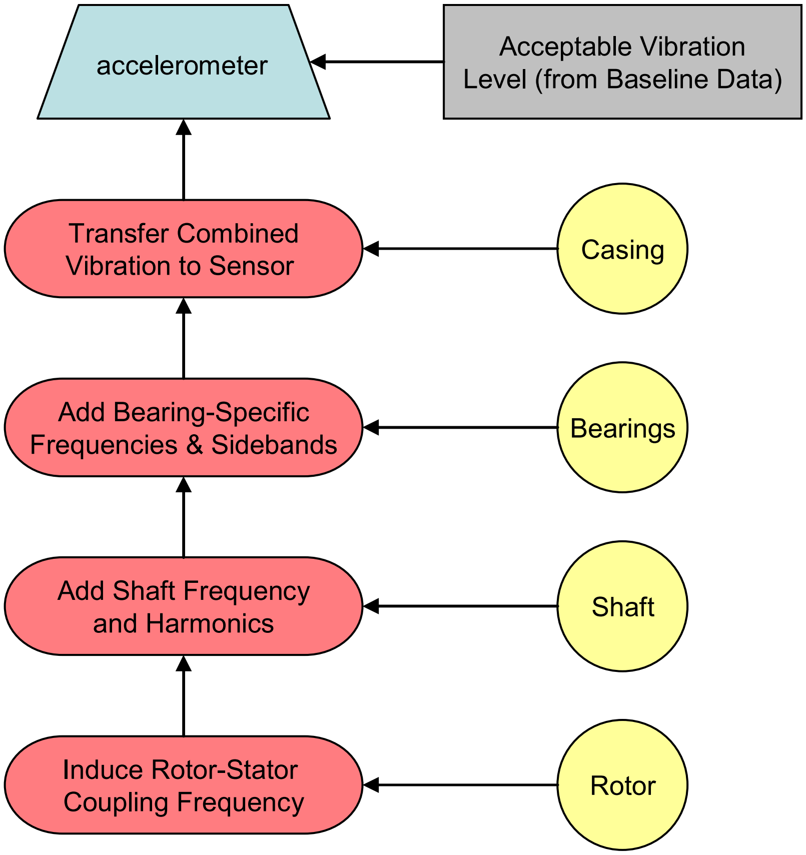

accelerometer
Acceptable Vibration Level (from Baseline Data)
Transfer Combined Vibration to Sensor
Casing
Add Bearing-Specific Frequencies & Sidebands
Bearings
Add Shaft Frequency and Harmonics
Shaft
Induce Rotor-Stator Coupling Frequency
Rotor

Figure 12. Functional model of the H-60 oil cooler.

We study the oil cooler system from a frequency domain perspective since the only sensors are accelerometers. The structural organization of its components (shown in Figure 11) can be abstracted in the form of a functional representation, where each component adds its own frequency signature to the overall vibration signal transmitted to the accelerometers by the casing. Figure 12 represents this in the form of the functional model. An accurate mathematical analysis of the relations mentioned above requires the introduction of complex concepts of rotordynamics and vibration analysis, as well as a good knowledge about the physical parameters of the units involved, e.g. the stiffness matrix of the shaft, the viscosity of the lubricants, etc.

However, without going into such detail, a simple qualitative analysis of the measurable quantities and their relations to each other offers a keen insight into the operation of the system. For example, since vibrations are additive in the frequency domain, the levels of specific frequency bands in spectrum analysis routines (like Fast Fourier Transform or Power Spectral Density), corresponding to the various components, are good indicators of fault modes. It is readily apparent that such a behavioral description is very convenient in formulating effective diagnostic queries as to the condition of the system.

4.4 The Fault-Symptom Matrix

The construction of the *fault-symptom matrix* is the main reasoning step regarding overall system behavior. Each unit in the functional model is associated with a number of fault modes, with each fault mode corresponding to one or more condition indicators. A system fault is defined as:

$$\begin{aligned} \mathrm{F} &\equiv \neg \mathrm{M_F} = \neg \bigwedge_i \bigwedge_j \neg F_{ij} \\ &= \bigvee_i \bigvee_j F_{ij} \end{aligned} \qquad (4)$$

i.e., at least one fault mode has been excited. Here $\vee$ denotes logical OR.

A matrix tabulating the various fault modes and their symptoms is generated by traversing all units of the functional model and extracting their features from the database. If sufficient data from seeded fault testing is available from a FMECA study, then the fault-symptom matrix can be enhanced with criticality metrics like severity and frequency of occurrence. Table 2 depicts a possible fault-symptom matrix for a generic system with 5 fault modes and 5 symptoms.

Table 2. A generic fault-symptom matrix (X denotes a valid fault-symptom relation).

	Symptom 1 (S1)	Symptom 2 (S2)	Symptom 3 (S3)	Symptom 4 (S4)	Symptom 5 (S5)
Fault Mode 1 (F1)	X		X		
Fault Mode 2 (F2)		X		X	
Fault Mode 3 (F3)	X		X	X	
Fault Mode 4 (F4)	X	X		X	X
Fault Mode 5 (F5)	X		X		X

This matrix gives us a handle on what anomalous behaviors we can expect to occur inside the system and what symptoms they would exhibit in terms of observable system behaviors. Using baseline data provided for the system under study, we calibrate this matrix for the acceptable levels of the fault mode symptoms. Here we **assume** that the model of the system is incapable of setting nominal thresholds on the expected system behavior. This is often the case when, constrained by the sensors available, only a narrow slice of the entire behavioral range of the system is observable e.g., gearboxes instrumented with accelerometers only allow for vibration signature analysis, which cannot be sufficiently modeled (so as to set operational limits) due to a variety of other operational and system parameters. Hence, we choose to bolster our model-driven approach here with some data-driven limits.

The choice of thresholds on these symptoms is arbitrary. In the real world, maintenance personnel pick these thresholds from operational experience. In the absence of such expert knowledge we **assume** that the data is *normally distributed* (equation 5), and we construct this distribution based on the mean μ and standard deviation σ of the given baseline data.

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/(2\sigma^2)} \tag{5}$$

It is an acceptable practice to base the control limits upon a multiple of the standard deviation. Usually this multiple is 3 and thus the limits are called 3-sigma limits. These limits give us a 99.7% confidence interval on our baseline data. If a data point falls outside the control limits, we assume that the system is probably demonstrating a fault mode and that an investigation into the cause is warranted.

4.5 Diagnostic Reasoning

Having determined our nominal thresholds, it is time to move on to the actual diagnostic step. A multi-branched *diagnostic tree* is constructed from the measurements of the overall system behavior. The internal nodes of the tree represent monitored system variables measurements of internal signals, whereas the leaves or terminal nodes denote the components that are fault candidates. To illustrate the above concepts let us look at a simple adder-multiplier example [22]. The simple adder-multiplier circuit shown in Figure 13 consists of three analog multipliers, labeled M1, M2*,* and M3, and two analog adders, labeled A1 and A2. For this simple circuit a ternary diagnostic tree is constructed as shown in Figure 14. In this diagram, the root and internal nodes (denoted by the letters F, G, X, Y and Z in circles) represent measurements, leaf nodes (denoted by the literals M1, M2, M3, A1 and A2 in circles) represent failed components, empty circles denote multiple simultaneous faults, while black squares denote either fault not detected or not isolatable. Fault diagnosis proceeds as follows: if terminal F is measured LOW (BAD) and terminal X is measured HIGH (BAD), then the ternary tree shown in Figure 14 correctly indicates a multiple fault (represented by an open circle).

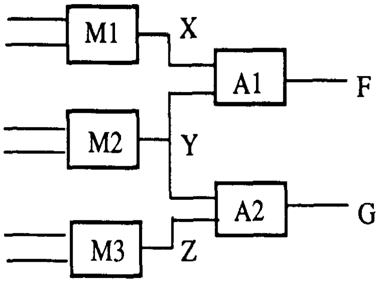


Figure 13. Analog adder-multiplier.

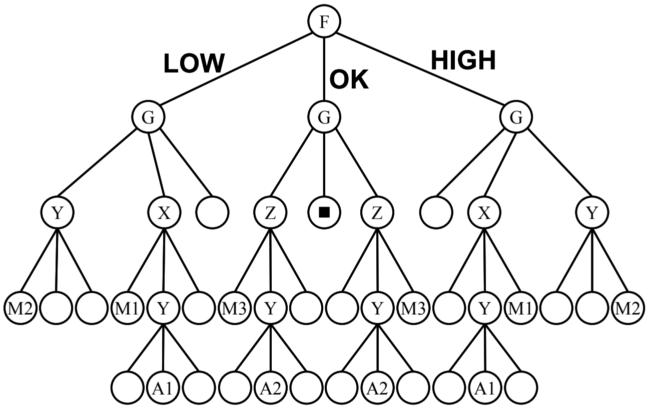


Figure 14. Diagnostic tree for analog adder-multiplier.

Before we delve into the nitty-gritty of the reasoning algorithm, we need to take another look at some of the basic concepts of MBR. As mentioned in the literature review (Section 2.2) traditional MBR algorithms have two different approaches:

Consistency-Based Approach: In this approach the nominal behavior of the target system is modeled. The diagnosis provided discounts the probability of multiple faults as well as the possibility of the fault signature of one component being influenced by its interaction with other components. Nevertheless, it can provide an insight as to whether nominal operation has been disrupted or not.

Abductive Approach: In this methodology the faulty operational modes of the system is modeled so as to provide more accurate fault detection and identification.

Our approach combines the two in the sense that some generic feature (e.g., energy) evaluated on the overall system behavior diagnoses whether some fault has occurred or not. This event in turn triggers the diagnostic decision tree evaluation that isolates and identifies the fault.

The mention of several performance metrics for MBR methods exists in literature like entropy, logical incompleteness etc. [2] However, before we discuss performance, we need to lay down some **assumptions**:

- The diagnostic decision tree is evaluated on the basis that some fault has occurred, i.e. the *no fault* condition does not exist.
- All symptoms appear in one or more fault modes but not in all of them.

For our reasoning paradigm we define some simple performance metrics. A measure of *speed* of the reasoning algorithm is the expected number of measurements needed to identify a fault. The *cost* of the diagnosis is measured in terms of the number of paths that diagnose a fault condition. If a component can be detected as being faulty, it should appear as a leaf or terminal node. Finally, the *resolving power* of the diagnostic tree is related to the number of multiple fault modes that can be discriminated. Our aim is to construct a decision tree maximizing speed and resolving power while reducing cost. For a given application, the construction of the fault-symptom matrix determines the resolving power of the algorithm. The speed and cost measures are thus the only metrics left to optimize while formulating the diagnostic tree.

Since faults are expressed as *minterms* (logically ANDed product of literals in which each variable appears exactly once) as shown in equation 1, we can apply the laws of logic minimization to reduce our diagnostic decision tree. These rules are presented below. Here, 1 signifies a TRUE value, which in the context of the fault-symptom matrix represents a symptom indicative of a fault, denoted by an X, while 0 or FALSE implies the absence of a symptom in a fault mode, + and $\bullet$ signify logical AND and OR respectively.

Operations with 0 and 1:

$$X + 0 = X \qquad\qquad X \bullet 1 = X$$

$$X + 1 = 1 \qquad\qquad X \bullet 0 = 0$$

Idempotent Law:

$$X + X = X \qquad\qquad X \bullet X = X$$

Involution Law:

$$\neg(\neg X) = X$$

Laws of Complementarity:

$$X + \neg X = 1 \qquad\qquad X \bullet \neg X = 0$$

Commutative Law:

$$X + Y = Y + X \qquad\qquad X \bullet Y = Y \bullet X$$

Associative Laws:

$$(X + Y) + Z = X + (Y + Z) \qquad\qquad (X \bullet Y) \bullet Z = X \bullet (Y \bullet Z)$$

$$= X + Y + Z \qquad\qquad = X \bullet Y \bullet Z$$

Distributive Laws:

$$X \bullet (Y + Z) = (X \bullet Y) + (X \bullet Z) \qquad\qquad X + (Y \bullet Z) = (X + Y) \bullet (X + Z)$$

Simplification Theorems:

$$X \bullet Y + X \bullet \neg Y = X \qquad\qquad (X + Y) \bullet (X + \neg Y) = X$$

$X + X \bullet Y = X$ $X \bullet (X + Y) = X$

$(X + \neg Y) \bullet Y = X \bullet Y$ $(X \bullet \neg Y) + Y = X + Y$

DeMorgan's Law:

$\neg (X + Y + ...) = \neg X \bullet \neg Y \bullet ...$ $\neg (X \bullet Y \bullet ...) = \neg X + \neg Y + ...$

Karnaugh maps or K-maps are a useful graphic technique to perform the minimization of a *canonical* (*minterm*) form. They utilize the above theorems in a mapping procedure which results in a simplified Boolean expression. There are five basic steps in the minimization procedure:

- Develop the first canonical expression from the associated truth table.
- Plot 1's in the K-map for each *minterm* in the expression.
- Loop adjacent groups of 1's (expressible as a power of 2) together.
- Write one *minterm* per loop, eliminating variables where possible.
- Logically OR the remaining *minterms* together to give the simplified *minterm* expression.

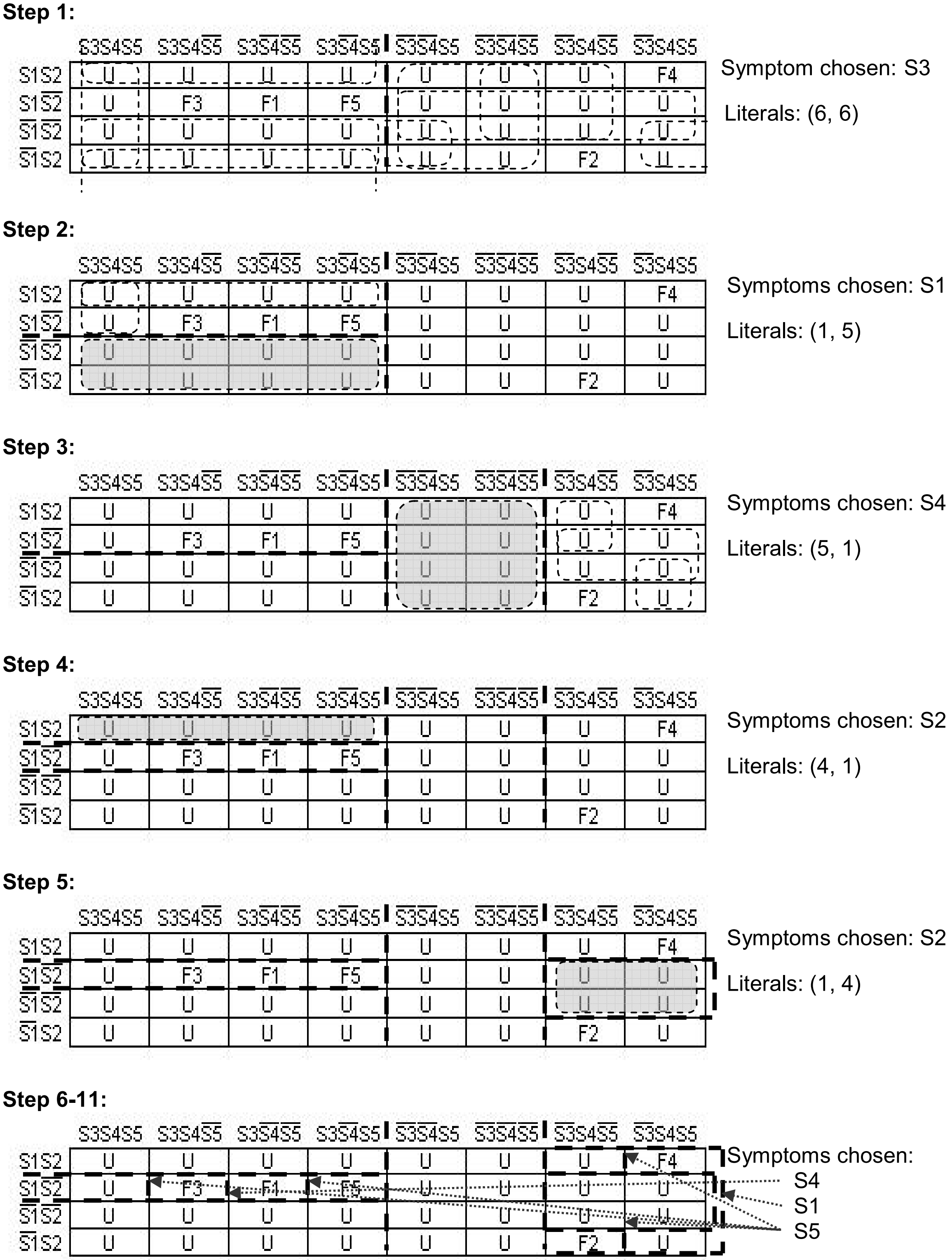
The specific rules for looping on a *k*-variable K-map are summarized here:

- Loops must contain 2^n cells set to 1, where *n* is an integer not larger than *k*.
- A single cell (loop of 2^0) cannot be simplified.
- A loop of 2 (2^1) is independent of 1 variable, a loop of 4 (2^2) is independent of 2 variables. In general a loop of 2^n cells is independent of *n* of the *k* variables involved.
- Using the largest loops possible will give the simplest functions.

- All cells in the K-map set to 1 must be included in at least one loop when developing the *minterm* or *maxterm* form.
- Loops may overlap if they contain at least one other unlooped cell in the K-map.
- Any loop that has all of its cells included in other loops is redundant.
- Loops must be square or rectangular. Diagonal or L-shaped loops are invalid. The edges of a K-map are considered to be adjacent. Therefore a loop can leave at the top of a K-map and re-enter at the bottom, and similarly for the two sides.

There may be different ways of looping a K-map since for any given truth table there may not be a unique minimal form.

In the case that we allow unknown faults (those not covered in the fault-symptom matrix), optimizing the decision tree for maximum efficiency as well as accuracy becomes difficult. The condition in every cell in any given row of the fault-symptom matrix must be satisfied before that fault is declared. For example, if the fault F4 (from Table 2) is to be declared then the thresholds for symptoms S1, S2, S4 and S5 must be exceeded while S3 feature value must be nominal. For any case not covered in the matrix an unknown fault is declared. Basically, for the 5 symptoms we have 25 = 32 fault modes (F1-F5 and 27 unknown modes). Since the number of decision steps for F1-F5 (the known faults) is 5 in each case, we can only optimize for the set of unknowns (denoted by U). The procedure is similar to K-map minimization. At each step we select the symptom to be tested in such a way so as to minimize the number of literals involving U in each half of the K-map differentiated by the symptom as shown in Figure 15 below.



Step 1:

	$S3S4S5$	$S3S4\overline{S5}$	$S3\overline{S4}\overline{S5}$	$S3\overline{S4}S5$	$\overline{S3}\overline{S4}S5$	$\overline{S3}\overline{S4}\overline{S5}$	$\overline{S3}S4\overline{S5}$	$\overline{S3}S4S5$
$S1S2$	U	U	U	U	U	U	U	F4
$S1\overline{S2}$	U	F3	F1	F5	U	U	U	U
$\overline{S1}\overline{S2}$	U	U	U	U	U	U	U	U
$\overline{S1}S2$	U	U	U	U	U	U	F2	U

Symptom chosen: S3

Literals: (6, 6)

Step 2:

	$S3S4S5$	$S3S4\overline{S5}$	$S3\overline{S4}\overline{S5}$	$S3\overline{S4}S5$	$\overline{S3}\overline{S4}S5$	$\overline{S3}\overline{S4}\overline{S5}$	$\overline{S3}S4\overline{S5}$	$\overline{S3}S4S5$
$S1S2$	U	U	U	U	U	U	U	F4
$S1\overline{S2}$	U	F3	F1	F5	U	U	U	U
$\overline{S1}\overline{S2}$	U	U	U	U	U	U	U	U
$\overline{S1}S2$	U	U	U	U	U	U	F2	U

Symptoms chosen: S1

Literals: (1, 5)

Step 3:

	$S3S4S5$	$S3S4\overline{S5}$	$S3\overline{S4}\overline{S5}$	$S3\overline{S4}S5$	$\overline{S3}\overline{S4}S5$	$\overline{S3}\overline{S4}\overline{S5}$	$\overline{S3}S4\overline{S5}$	$\overline{S3}S4S5$
$S1S2$	U	U	U	U	U	U	U	F4
$S1\overline{S2}$	U	F3	F1	F5	U	U	U	U
$\overline{S1}\overline{S2}$	U	U	U	U	U	U	U	U
$\overline{S1}S2$	U	U	U	U	U	U	F2	U

Symptoms chosen: S4

Literals: (5, 1)

Step 4:

	$S3S4S5$	$S3S4\overline{S5}$	$S3\overline{S4}\overline{S5}$	$S3\overline{S4}S5$	$\overline{S3}\overline{S4}S5$	$\overline{S3}\overline{S4}\overline{S5}$	$\overline{S3}S4\overline{S5}$	$\overline{S3}S4S5$
$S1S2$	U	U	U	U	U	U	U	F4
$S1\overline{S2}$	U	F3	F1	F5	U	U	U	U
$\overline{S1}\overline{S2}$	U	U	U	U	U	U	U	U
$\overline{S1}S2$	U	U	U	U	U	U	F2	U

Symptoms chosen: S2

Literals: (4, 1)

Step 5:

	$S3S4S5$	$S3S4\overline{S5}$	$S3\overline{S4}\overline{S5}$	$S3\overline{S4}S5$	$\overline{S3}\overline{S4}S5$	$\overline{S3}\overline{S4}\overline{S5}$	$\overline{S3}S4\overline{S5}$	$\overline{S3}S4S5$
$S1S2$	U	U	U	U	U	U	U	F4
$S1\overline{S2}$	U	F3	F1	F5	U	U	U	U
$\overline{S1}\overline{S2}$	U	U	U	U	U	U	U	U
$\overline{S1}S2$	U	U	U	U	U	U	F2	U

Symptoms chosen: S2

Literals: (1, 4)

Step 6-11:

	$S3S4S5$	$S3S4\overline{S5}$	$S3\overline{S4}\overline{S5}$	$S3\overline{S4}S5$	$\overline{S3}\overline{S4}S5$	$\overline{S3}\overline{S4}\overline{S5}$	$\overline{S3}S4\overline{S5}$	$\overline{S3}S4S5$
$S1S2$	U	U	U	U	U	U	U	F4
$S1\overline{S2}$	U	F3	F1	F5	U	U	U	U
$\overline{S1}\overline{S2}$	U	U	U	U	U	U	U	U
$\overline{S1}S2$	U	U	U	U	U	U	F2	U

Symptoms chosen:

Figure 15. Diagnostic tree formation based on K-map minimization. Si (i=1–5) denotes the threshold for Symptom i has been exceeded, while Si signifies nominal feature levels. The shaded boxes denote leaf nodes for unknown fault conditions.

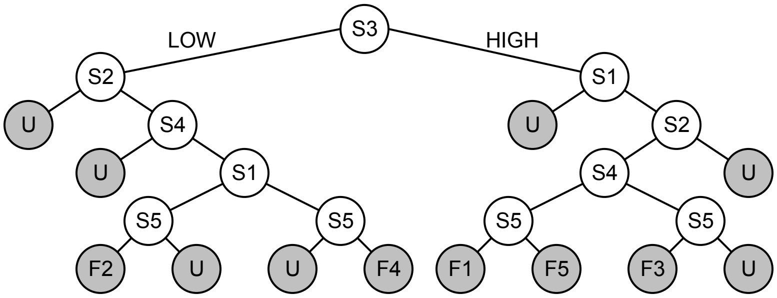


Figure 16. Diagnostic decision tree for the fault-symptom matrix in Table 2. Leaf nodes denoting faults are shown in shaded color.

The corresponding decision tree is shown in Figure 16. The performance metrics for the example discussed above are: speed = 50/12 = 4.17, cost = 12, resolving power = 5.

However, considering unknown faults may not always be practical from an engineering standpoint. The previous method, while optimizing the diagnostic process, is not suitable for fault propagation. Countless experiences have shown that single faults, left uncorrected, spread from one component to another, causing fault modes to be triggered elsewhere. In the diagnostic tree this would be represented as leaf nodes with multiple faults. Of course, the case of all possible faults happening in a chain before the system fails is also rare. Accordingly, the decision tree must be pruned to reflect these constraints. Before we discuss the tree formulation procedure in this case, it is imperative to restate our **assumptions** at this point.

- The fault-matrix table covers all possible fault modes (this is not an unreasonable assumption if the table is constructed from a FMECA study).
- Faults are independent of each other, i.e. the occurrence of one does not affect the probability of occurrence of another.

- The set of symptoms of a fault is not a subset of another fault, for example in Table 2 fault F1 was a subset of F3 and F5.
- The diagnostic tree is *not* computed under no-fault conditions.

This approach is more suited to the representation of faults as *minterms* (equation 1) since the absence of an unrelated symptom does not give us any more information about the concerned fault mode. The tree formulation steps are similar as before, where we construct a truth table from the fault-symptom matrix and then use K-map minimization to get the sequence of the symptoms to be tested. Although the theory behind the approach remains the same, the truth table formulation is slightly different. This is explained in the example below.

Table 3. Another generic fault-symptom matrix (X denotes a valid fault-symptom relation).

	Symptom 1 (S1)	Symptom 2 (S2)	Symptom 3 (S3)
Fault Mode 1 (F1)	**X**	**X**	
Fault Mode 2 (F2)	**X**		
Fault Mode 3 (F3)			**X**

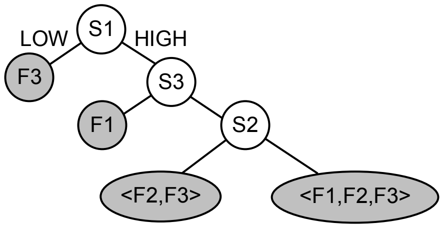
Table 3 gives the fault-symptom matrix of another generic case where we consider 3 symptoms spread over 3 faults. The fault modes can be written as:

$$\begin{aligned}
F1 &= S1 \wedge S2 = (S1 \wedge S2 \wedge S3) \vee (S1 \wedge S2 \wedge \neg S3) \\
F2 &= S1 = (S1 \wedge S2 \wedge S3) \vee (S1 \wedge S2 \wedge \neg S3) \vee (S1 \wedge \neg S2 \wedge \neg S3) \vee (S1 \wedge \neg S2 \wedge S3) \\
F3 &= S3 = (S1 \wedge S2 \wedge S3) \vee (S1 \wedge \neg S2 \wedge S3) \vee (\neg S1 \wedge \neg S2 \wedge S3) \vee (\neg S1 \wedge S2 \wedge S3) \qquad (6)
\end{aligned}$$

The truth table representing this set of literals is shown in Figure 17(a) while the corresponding diagnostic tree is shown in Figure 17(b). The asterisks in the truth table denote "don't care" conditions, whereas in the leaf nodes below <F_i, …,F_n> denotes that any possible combination of the included faults may occur. Thus, <F1,F2,F3> includes the fault modes F1, F2, F3, F1∧F2, F1∧F3, F2∧F3, and F1∧F2∧F3. The bold boxes show the partitions made after K-map minimization of literals at each step. The performance metrics are: speed = 33/12 = 2.75, cost = 4, resolving power = 4.

	$S1S2$	$S1\overline{S2}$	$\overline{S1}\,\overline{S2}$	$\overline{S1}S2$
$S3$	F1F2F3	F2F3	F3	F3
$\overline{S3}$	F1	*	*	*

(a)



(b)

Figure 17. (a) Truth table and (b) diagnostic decision tree for the fault-symptom matrix in Table 3.

As each fault mode is identified, the corresponding faulty component is determined from the database. Corresponding to each faulty component a *fault propagation tree* is generated based on the structural links stored in the structural model. In case of an unknown fault the fault propagation step is not performed. The theoretical foundation for logically reasoning about propagating a fault through a system represented by a partially connected graph can be specified using Z formalism [23]. Simply put, the fault propagation step takes the set of fault nodes as input and outputs the set of "at risk" nodes. Starting with the input set we search for the set of connected nodes (representing components) maintaining the acyclic property (we do not cycle back to the faulty nodes) such that all input nodes must be reachable from nodes in the output set with a jump size of one, i.e. the nodes (components) are physically adjacent. This is logically represented as:

$$\textit{Set of “at risk” nodes} = \{v_i|(v_i \in V)\wedge(\exists j,x{:}\neg M_F \Rightarrow F_{xj})\wedge(<v_i,v_x>\in E)\} \qquad (7)$$

Let us consider the process plant example studied in the Models and Techniques for Integrated System Safety Engineering (MATISSE) project [23]. The system consists of a computer-based controller, valve A, valve B, level sensor X and level sensor Y as shown in Figure 18. Let us assume that the aim is to maintain the liquid level in the tank between level sensors X and Y. Valve A is manually operated to drain the tank while the controller tries to maintain the level using valve B.

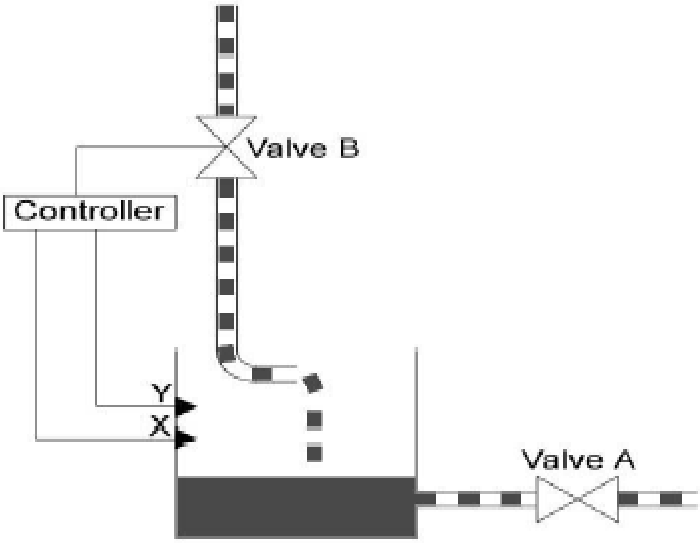


Figure 18. Process plant schematic.

Now, assume that the input ports of the Controller fail such that it can no longer read the X and Y levels. There can be two possible failure scenarios: a) if valve A was open and the tank was being replenished at the time of failure then the valve B would not be switched off even after the level reached Y and the tank would overflow, and b) if the level was between X and Y and valve B was closed then continual drainage through valve A would cause the tank to run dry. These possibilities are captured in the fault propagation tree depicted in Figure 19.

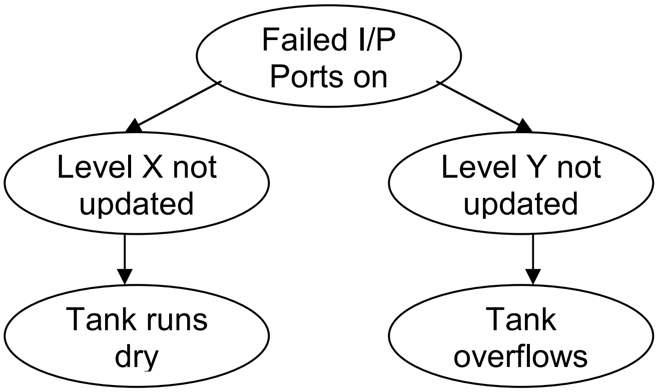


Figure 19. Process plant fault propagation tree for failed controller input ports.

In order to update the overall system health report, the full fault propagation tree is not required. Rather the components at the second level of the tree (i.e. adjacent components) are indicated as the components mostly likely to develop fault modes in the future.

5 Computational Complexity Analysis

As a branch of the theory of computation in computer science, computational complexity theory describes the scalability of algorithms, and the inherent difficulty in providing scalable algorithms for specific computational problems. The implications of the theory are important to any practical application. The speed and memory capacity of computers are always increasing, but then so are the dimensions of the problems that need to be analyzed. If algorithms fail to scale well, then even vast improvements in computing technology will result in only marginal improvement in solution time.

5.1 Analysis of Reasoning Steps

The following paragraphs describe the computational complexity analyses of the automated steps in the overall algorithm. The formulation of the model database and the structural model of the system are user inputs and hence are not part of this study. For the sake of simplicity, we assume that there are p components, each with q fault modes, e interconnect edges, and a total of r distinct symptoms.

5.1.1 Functional Model

The functional model of the system is constructed by traversing the partially connected graph represented by the structural model and substituting the corresponding function for each component from the database. Since this step involves the traversal of a p-node e-edge graph the computational complexity is $O(p+e)$.

5.1.2 Fault Symptom Matrix

The matrix tabulating the various fault modes and their symptoms is generated by traversing all units of the functional model and extracting their features from the database. For each component, we have *q* fault modes and *r* symptoms. Thus, to create this matrix we have a computational complexity of $O((p+e)\times q\times r)$.

5.1.3 Diagnostic Tree

The generation of the diagnostic tree is split into two subroutines: the formulation of the K-map representation of the fault symptom matrix, followed by the sequencing of the diagnostic queries depending upon the K-map minimization rules. The computational complexity of formulating the table is $O(2^r)$ as determined by the number of symptoms. In order to find the complexity of the diagnostic tree generation routine we carry out a worst case analysis. For *r* symptoms the largest diagnostic tree would be a balanced binary tree with 2*r* leaf nodes and *r* diagnostic decision levels. The number of nodes at any given level *i* of the tree (the root being the 0th level) is 2^i and the number of possible diagnostic choices at any node of that level is *r-i*. Also for a node at level *i* the number of cells in the K-map representation that need to be checked to see if they form a leaf node or not is $2^{(r-i-1)}$. Hence, the overall number of diagnostic tree possibilities along with the number of K-map cells to be checked at each level is given by

$$\prod_{i=0}^{r-1} (i+1).2^i.2^{(r-i-1)} .$$

The computational complexity is thus determined to be $O(r!\, 2^{r(r-1)})$.

5.1.4 Fault Propagation

The fault propagation step takes the set of fault nodes as input and outputs the set of “at risk” nodes, such that all input nodes are reachable from nodes in the output set and the nodes (components) are physically adjacent. In the worst possible case it amounts to the complete traversal of the structural model graph which is O(p+e).

5.1.5 System Status Update

This step simply involves displaying the list of the faulty and the “at risk” nodes from the diagnostic and fault propagation steps and hence the computational requirements are taken care of in the previous steps.

5.2 Analysis of Reasoning Steps

In computational complexity theory, NP ("Non-deterministic Polynomial time") is the set of decision problems solvable in polynomial time on a non-deterministic Turing machine. Equivalently, it is the set of problems whose solutions can be "verified" by a deterministic Turing machine in polynomial time. Thus, the challenge of NP problems is to efficiently find the answer, given an efficient (polynomial-time) way of verifying it once it is found.

NP-complete problems are the most difficult problems in NP ("non-deterministic polynomial time") in the sense that they are the smallest subclass of NP that could conceivably remain outside of P, the class of deterministic polynomial-time problems. The reason is that a deterministic, polynomial-time solution to any NP-complete problem would also be a solution to every other problem in NP [24]. A supposed answer is very easy to verify for correctness, but no one knows a significantly faster way to solve the problem than to try every single possible subset, which is very slow. Nobody has yet been able to prove whether NP-complete problems are in fact solvable in polynomial time, making this one of the great unsolved problems of mathematics [24].

In the celebrated Cook-Levin theorem (independently proved by Leonid Levin), Cook proved that the Boolean satisfiability problem is NP-complete [25], which makes the *Diagnostic Tree* generation step the most significant computational bottleneck in our methodology. Our approach, so far, has been to restrict the input size of our problem by focusing only on the most critical faults and a limited domain of features. Even so, it is preferable in practical applications that all steps upto *Diagnostic Tree* generation be done offline, since they need to be done only once for a given system, and only the *Diagnostic Tree* evaluation, *Fault Propagation* and *System Status Update* steps be done online in real-time.

5.3 Parallelizability

Parallel processing is the simultaneous execution of the same task (split up and specially adapted) on multiple processors in order to obtain results faster. The idea is based on the fact that the process of solving a problem usually can be divided into smaller tasks, which may be carried out simultaneously with some coordination. Currently, with the advent of multi-core cell processors the parallelized implementation of computation-intensive algorithms has gained greater importance.

The steps of our methodology can be broadly classified into two groups: those that involve look-up functions or updates on different nodes or components (e.g. generation of the functional model from the structural model, fault-symptom matrix formulation, conversion to the K-map minimization format, and system status update), and those that involve graph or tree traversals (like the diagnostic tree and fault propagation). The former class lends itself well to parallel implementations since operations on nodes are mostly independent of each other. The latter class would also benefit from parallelization; however, the benefits would not be significant if the node connectivity structure is not balanced.

6 The IGB Application Domain

6.1 The Helicopter Power-train Components

The application domain chosen for the model-based reasoning paradigm envisioned here is comprised of helicopter power-train modules. They are composed of three basic types of fundamental mechanical units – gears, shafts and bearings. Albeit the individual specifications of these units as components of different modules, like the main transmission or the intermediate gearbox, differ significantly, however, their underlying functions remain the same. Thus, a collection of condition-indicators (CI's) that are characteristic of anomalies in these functions provides a suitable platform for diagnostic reasoning.

The helicopter drive train assembly is shown in Figure 20. The tail rotor drive train consists of a drive shaft that transfers torque from the main transmission to the oil cooler drive shaft, four drive shaft interconnected sections that transfer torque from the oil cooler drive shaft to the intermediate gear box, and another drive shaft that transfers torque from the intermediate gear box to the tail gear box [26]. The intermediate gearbox (IGB) of a helicopter is an assembly consisting of a pair of meshed spiral pinion gears, whose shafts are supported at either end by bearings. It is located at the end of the tail boom of the helicopter and serves the purpose of changing the direction of the tail rotor drive shaft up towards the tail rotor gearbox. The tail rotor serves a very important function in the helicopter - not only as a directional (yaw) control but it prevents the machine from spinning out of control in the direction opposite to the main rotor rotation.

Hence, fault diagnosis of the intermediate gearbox, that powers the tail rotor, is of critical importance.

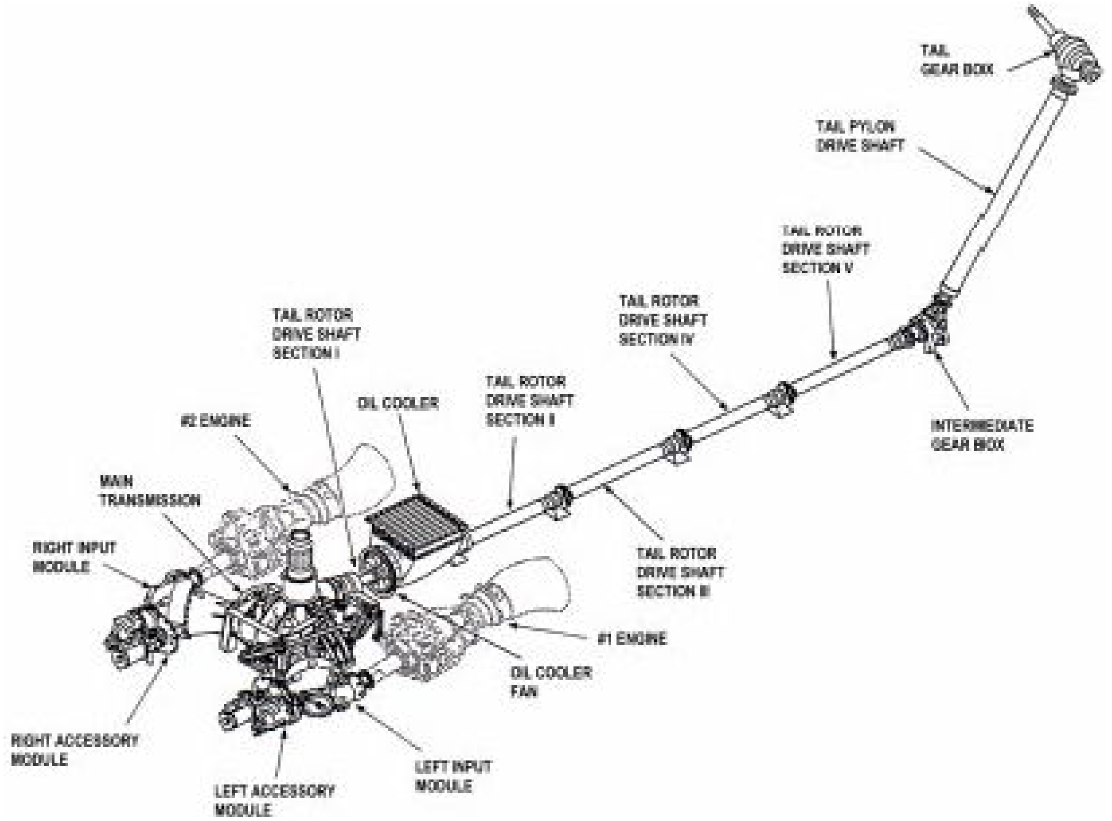


Figure 20. Helicopter Drive Train Assembly Drawing [26].

In our example, we shall discuss the power-train components of the H-60 (Black Hawk) helicopter. The H-60 helicopter plays a pivotal role in a variety of missions for the U.S. armed forces. Variations of the helicopter include Black Hawks, Pave Hawks, and Seahawks. The manufacturer, Sikorsky Aircraft Corp., says more than 2,500 H-60s are in service with the U.S. Army, Navy, Coast Guard, Air Force, and Marine Corps. A schematic view of the H-60 IGB is shown in Figure 21.

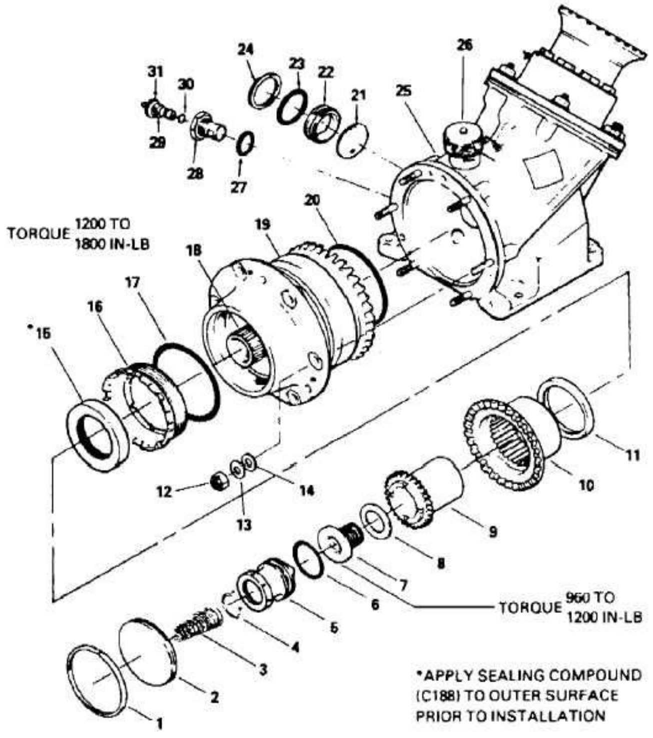

24
23
22
26
31
30
21
25
29
28
27
20
TORQUE 1200 TO 1800 IN-LB
19
18
17
16
*15
11
10
12
14
13
9
8
7
6
TORQUE 960 TO 1200 IN-LB
5
4
3
2
1
*APPLY SEALING COMPOUND (C188) TO OUTER SURFACE PRIOR TO INSTALLATION

Figure 21. Schematic of the H-60 intermediate gearbox.

6.2 IGB Diagnostics

We start with the database of functional descriptions and fault mode condition indicators for each of the power-train module components, namely gears, bearings and shafts. It is also possible to model the casing, but we shall ignore it for simplicity since casing fracture is a rare occurrence for the IGB. Also, by simply concentrating on the three types of components listed above, we retain a significant amount of commonly occurring fault modes. Since the effectiveness of the approach depends on the validity of the database, it is critical to have a thorough idea about the functions of each component and the fault modes resulting from these functions. Table 4 lists the three component types under study along with their relevant vibration related fault modes [27]. The database is stored in the form of an array of data structures with the component type, function, fault modes and associated feature calculation parameters as constituent fields.

Table 4. A priori knowledge of gearbox components.

Component	Function	Fault Modes	Features
Gear	Change rotational speed (RPM) by gear tooth ratio	• Crack in gear	• High vibration levels at the gear natural frequency (g.n.f.) • Sidebands around the g.n.f. spaced at the running speed of the bad gear
Bearing	Support rotating shaft with balls circulating in grooves	• Bearing race defect	• (*n*/2) x shaft speed, *n* is the number of balls
		• Excessive bearing clearance	• sub-synchronous whirl
Shaft	Transmit torque from one end to another	• Rotor imbalance	• 1 x shaft speed
		• Shaft misalignment	• 2 x shaft speed • high axial vibration
		• Mechanical looseness	• higher harmonics of shaft speed

Next, the H-60 intermediate gearbox is analyzed and the structural model is extracted. In this process the mechanical linkage between each component is noted and the model is again stored in the form of a linked graph. Each node of the graph is a data structure representing the component name (e.g. Input Gear, Output Shaft, etc.), its type (e.g. gear, shaft, etc.) and the pointers to its neighboring components. Figure 22 is a diagrammatic representation of the structural model of the H-60 IGB. As shown in the picture, the heart of the IGB is a pair of spiral bevel gears. Each gear is connected to the respective input and output sides of the gearbox by splined shafts. Each shaft is in turn supported by ball bearings at the exit ports of the IGB casing. The accelerometers are stud mounted on the input and output port flanges of the casing.

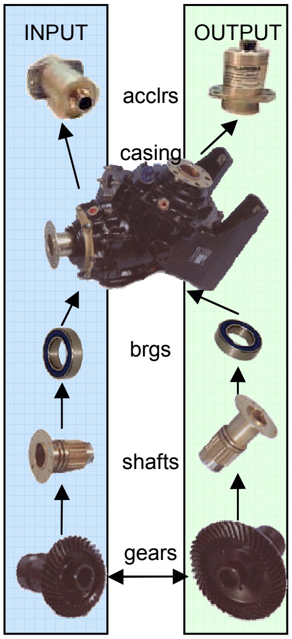


Figure 22. Structural diagram of the H-60 IGB.

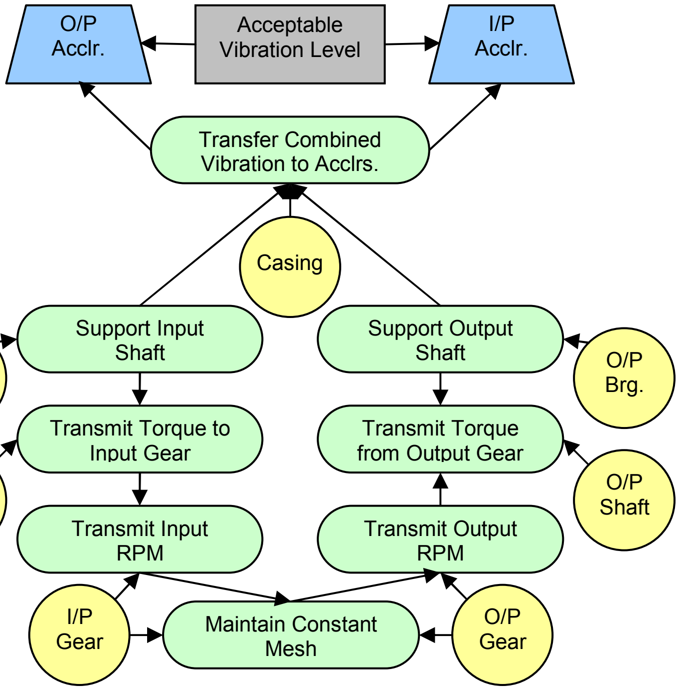


Figure 23. Functional diagram of the H-60 IGB.

The functional model of the system is constructed by traversing the structural model graph and substituting the respective functional descriptions for each node from the knowledge database. A block diagram of this functional model is shown in Figure 23. The function of the bevel gears is to maintain constant mesh between the input and the output sides and change the input rotational speed (RPM) into the output RPM through the 25:31 gear ratio. The input shaft transfers torque from the IGB input to the input bevel gear while the output shaft transfers the torque from the output gear to the IGB output. The functions of the input and the output bearings are to support their respective shafts and inhibit and motion other than axial rotation. The casing holds the entire assembly and transmits the combined vibrations from the internal components to the mounted accelerometers.

The response of a mechanical system to vibrating motion is simply modeled as a mass-spring-damper system [28]. However, in the chosen application, due to the system nonlinearities, precise first principle based modeling is impractical. Our approach takes advantage of the fact that during forced vibrations of mass-spring-damper systems the vibration frequency is the same as the driving frequency. As the vibration from one component passes to another, there is some attenuation, but the frequency remains unchanged. Hence, the signals picked up by the casing mounted accelerometers contain aggregated vibration signatures from each of the components, as shown in the functional description of the casing in Figure 23.

The IGB has multiple fault modes depending upon which components are subjected to damage. For example, a bearing may develop a defect in the inner/outer race, a shaft or a gear pinion might develop a crack, or even a gear tooth may be worn out or chipped. All of these fault modes give rise to characteristic vibration signatures. By looking at the corresponding frequency bands in the accelerometer readings, we can classify the fault modes present [21]. Some common IGB fault classification heuristics are given below. Although this list is not comprehensive, it is sufficient for our purpose, namely to demonstrate the application of our model-based diagnostic reasoning approach.

- Rotor Imbalance – 1 x shaft speed
- Shaft Misalignment – 2 x shaft speed, high axial vibration
- Mechanical Looseness – higher harmonics of shaft speed
- Excessive Bearing Clearance – sub-synchronous whirl instability
- Bearing Race Defect – (*n*/2) x shaft speed, *n* is the number of balls

- Gear Bevel Defect – gear natural freq., sidebands spaced at the running speed of the bad gear.

With the above concepts in view, the vibration specific fault-symptom matrix is constructed from the functional model and the database as shown in Table 5. The choice of thresholds on these symptoms is arbitrary. In this application, for all the symptoms, except for the side band spacing, the energy thresholds are set at the upper 3 sigma limit as determined from the baseline data. For the sideband spacing feature the symptom can take the value of either the input shaft speed or the output shaft speed. The system parameters required, as input to the MBR program for full IGB diagnostics, are given as 68.6 Hz input shaft speed, 25:31 gear ratio, and gear natural frequency around 822 Hz for both input and output gears.

Table 5. Fault-Symptom Matrix for the IGB (the symptoms are represented by associated frequency bands in Hz).

Symptoms Faults	Sub-synchronous	1x shaft speed	2x shaft speed	(*n*/2) x shaft speed	Higher harmonics	Gear Natural freq.	g.n.f. sideband spacing
Input Gear Crack						820-825	65-70
Output Gear Crack						820-825	82-87
Input Bearing Race Defect				200-350			
Output Bearing Race Defect				250-435			
Excessive Clearance I/P Brg	20-50						
Excessive Clearance O/P Brg	25-60						
Input Shaft Imbalance		50-100					
Output Shaft Imbalance		60-125					
Input Shaft Misalignment			100-150				
Output Shaft Misalignment			125-185				
Input Shaft Looseness					350-450		
Output Shaft Looseness					435-560		

Since the fault symptoms described above are primarily energy metrics, the energy content of each data sample received is used to determine whether the system is in nominal mode or not. Once a fault is diagnosed at the system level we evaluate the *diagnostic decision tree* to isolate and identify the fault. The **assumptions** involved are:

- All possible fault modes are covered in the fault-symptom matrix.
- There are no simultaneous multiple faults.
- Fault modes of each component are independent of each other. Hence multiple symptoms can be explained as the occurrence of multiple faults.

For the H-60, the fault-symptom matrix (Table 5) shows that other than the gear natural frequency the remaining symptoms are different for the input and the output sides. To exploit this feature two diagnostic decision trees are constructed for each of the input and output accelerometer data. Taking advantage of the fact that most faults have only a single symptom we devise a simpler linear decision tree. Although the decision tree is manually optimized in this case, automated generation by the MBR program also arrives at the same solution. Figure 24 shows the generic form of these decision trees for both the input and output side. The performance metrics are computed as *speed* = 4.1, *cost* = 8, resolving power = 7. This tree is traversed in the *depth-first* method with all left branches, starting from the root, being taken in each iteration. Each iteration corresponds to a new dataset recorded from the sensors. When any one of these nodes returns a fault, then the same traversal process is applied to the right sub-tree of that node. It is to be noted that the search space is not narrowed down after initial diagnosis since a fault initiation in one component does not rule out the possibility of other components going bad.

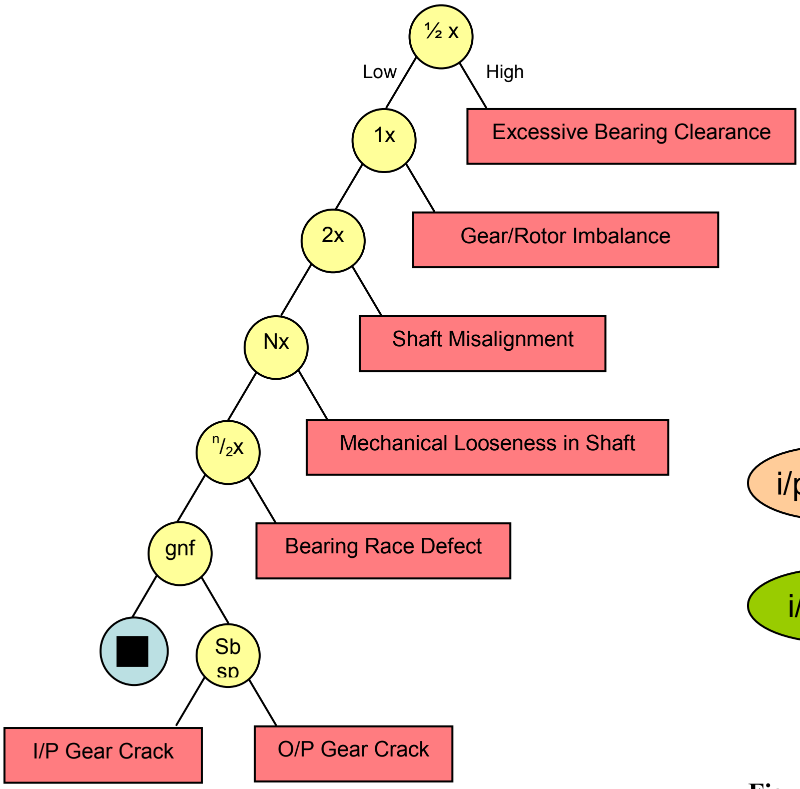


Figure 24. Generic diagnostic decision tree for IGB.

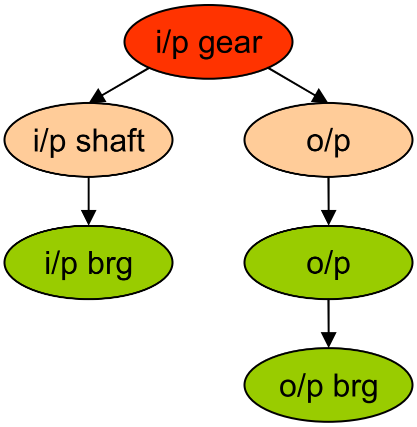


Figure 25. Fault propagation tree for input gear fault.

In any given iteration, after the detection of the fault mode(s), a fault propagation tree is generated corresponding to each faulty component based on the links stored in the structural model graph. In order to update the overall system health report, the full fault propagation tree is not required. Rather the components at the second level of the tree (just below the root) are indicated as the components mostly likely to develop fault modes in the future. Figure 25 shows an example of the fault propagation tree for an input gear fault. A schematic of this approach is presented in Figure 26. The double arrows going from the fault-symptom matrix to the diagnostic tree in the diagram denote that the latter was obtained from the former by both manual optimization as well as MATLAB® code.

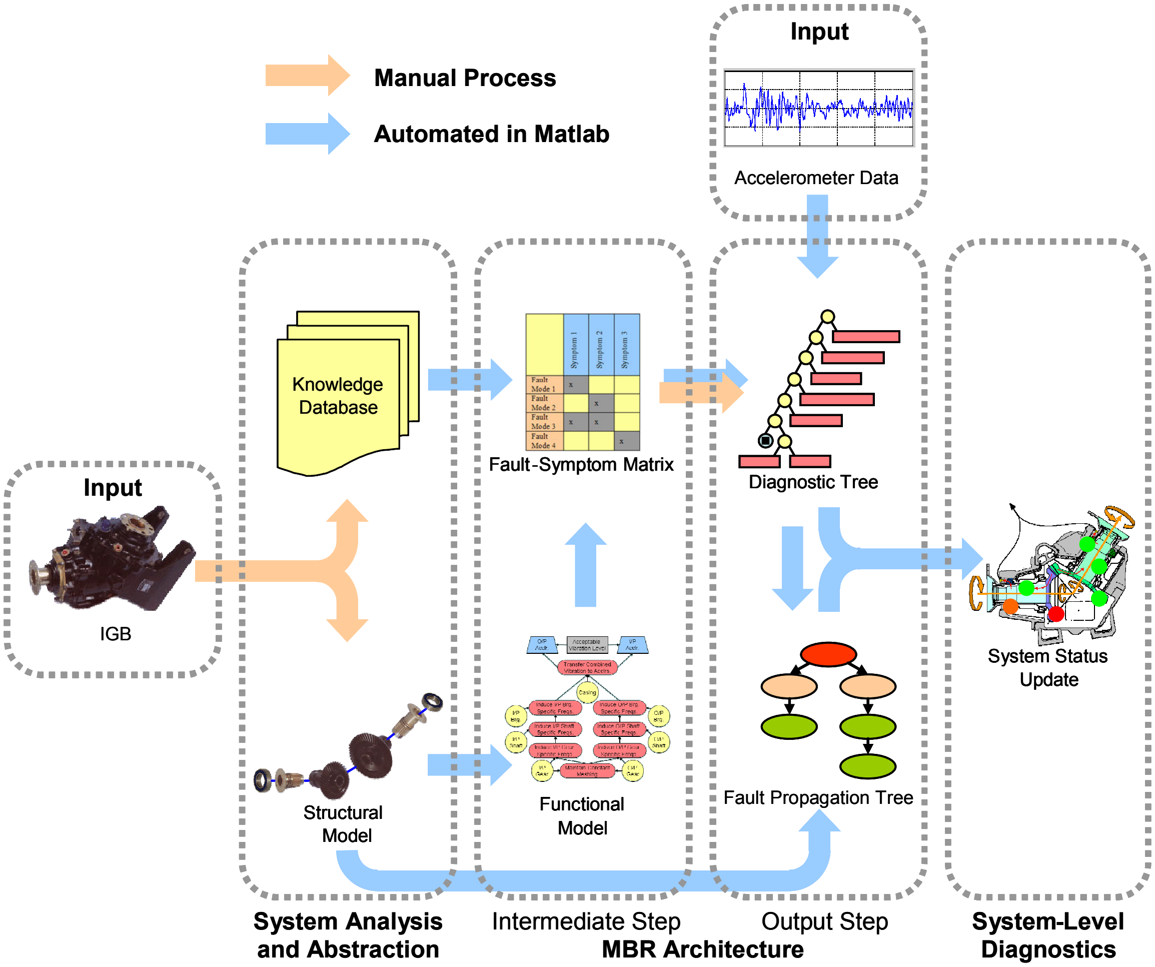


Figure 26. An MBR approach to fault detection and classification in the IGB.

6.3 Results

For fatigue crack analysis, an H-60 IGB pinion gear made of 9310 steel was used in a cyclic crack growth test. It was seeded with faults to initiate cracks. These faults consisted of notches made by an electric discharge machine (EDM), and were located at, and parallel to, the root of one of the gear teeth, as shown in Figure 27. The crack growth test consisted of rotation in a spin pit, at a constant high speed with a varying load cycle, to simulate flight conditions. Data collection was done using 2 stud mounted accelerometers at the input and the output of the IGB. The data consists of 36 sets of accelerometer readings from both the input and the output ends.

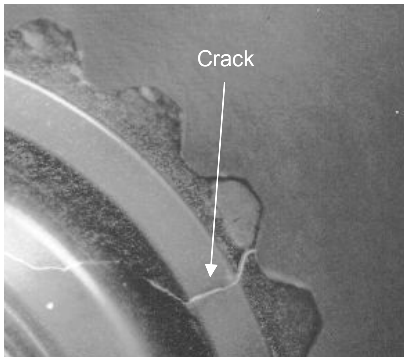


Figure 27. IGB crack.

A healthy gearbox exhibits certain characteristic frequencies. The primary component of the spectrum is the gear mesh frequency, which is 1714.5 Hz in our case. The harmonics of the mesh frequency are also present but their energy content varies with load conditions. When a crack initiates in the gear, the power spectrum starts to show a clear peak at the natural frequency of the gear. There are also characteristic sidebands spaced at the running speed of the bad gear. Thus, the major energy component of the signal shifts from the mesh frequency to the gear natural frequency and its sidebands.

For the frequency domain analysis of the given data, we primarily look at the power spectral density. First, we eliminate the mesh frequency and its prominent harmonics. Then we measure the features listed in Table 4. The normalized energy at the gear natural frequency (822 Hz) is observed to be the prime indicator of a growing fault condition. The spacing between the sidebands is then analyzed to see if it matches either the input or the output shaft speed. This provides a means of classifying which pinion gear is faulty. The simulation is done in MATLAB®. Figure 28 shows the plots of the feature values and their associated fault status flags.

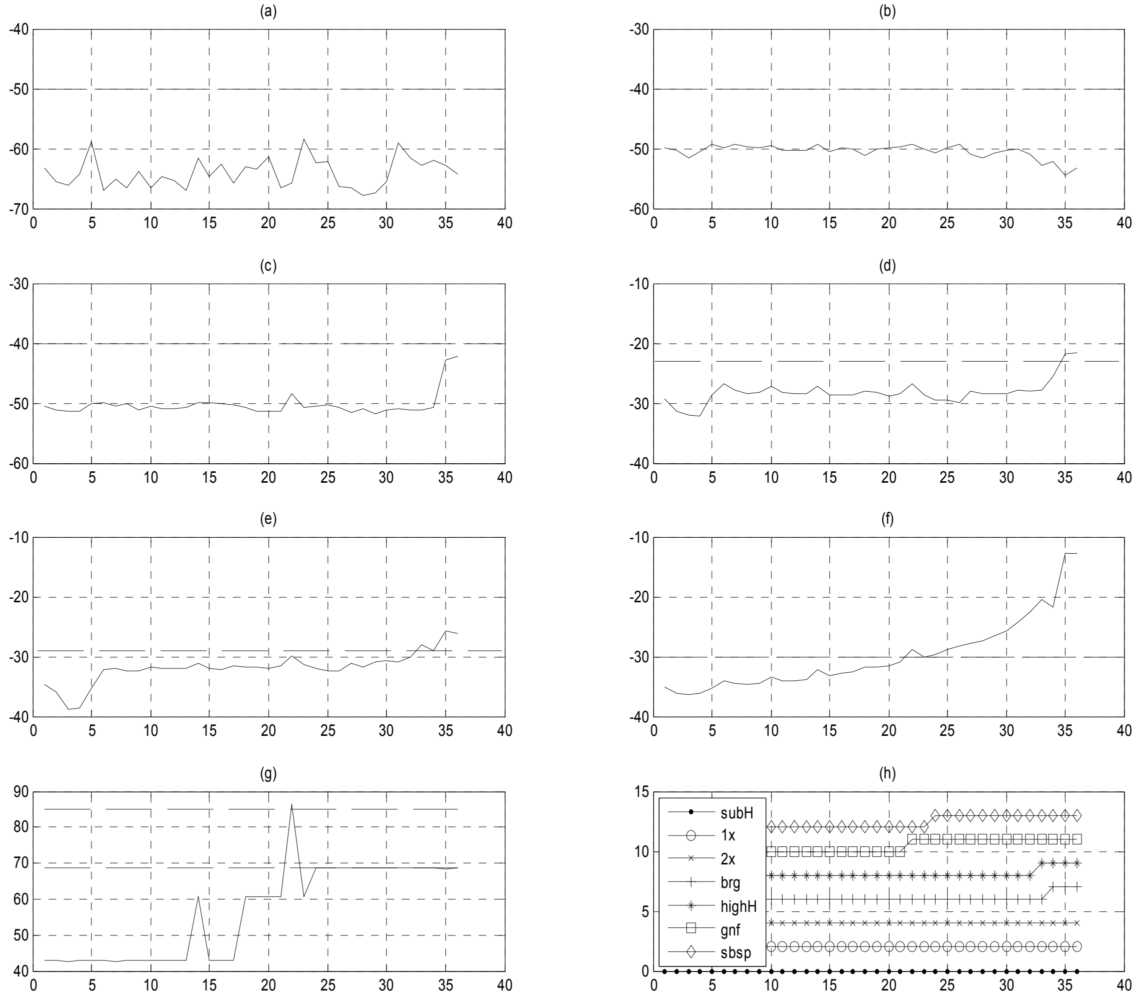


Figure 28. Plots showing various features and their flags vs. sample index. (a) Subharmonic energy (subH) in dB, (b) 1st harmonic or shaft speed energy (1x) in dB, (c) 2nd harmonic energy (2x) in dB, (d) bearing defect freq. energy (brg) in dB, (e) higher harmonic energy (highH) in dB, (f) gear natural freq. energy (gnf) in dB, (g) sideband spacing about gnf (sbsp) in Hz, and (h) the status flags of the 7 features listed above, displaced from each other.

In accordance with the plot shown in Figure 28(h) the text output received from the program is as follows:

```
time index 1 -> nominal
time index 22 -> gear fault
time index 24 -> i/p gear fault
time index 24 -> at risk: i/p shaft o/p gear
time index 32 -> i/p shaft mechanical looseness
time index 24 -> at risk: i/p brg o/p gear
time index 33 -> i/p brg race defect
time index 24 -> at risk: o/p gear
```

The photograph of the input pinion crack, shown in Figure 27, provides validation for our approach. The fault propagation tree shown in Figure 25 is also validated by the fault flags for the input bearing defect and the input shaft mechanical looseness being thrown. The physical explanation of the observed phenomena is as follows. As the gear crack is allowed to grow unchecked, the resulting vibrations are transmitted through the shaft to the input bearing. The shaft itself starts to deviate from pure axial rotation from the impulses provided by the opening and closing of the gear crack in each turn. This leads to the degradation of the input bearing and mechanical looseness of the input shaft.

7 The JSF Fuel Supply System Example

As an effort to show the applicability of the MBR diagnostic methodology to systems different from the H-60 IGB, we look at the fuel supply system of the F-35 Joint Strike Fighter (JSF) [29]. Figure 29 shows the structural model of the fuel supply system while Figure 30 depicts its functional model. As depicted, the system comprises of two independent fuel tanks with dedicated pumps feeding a common fuel line. The power circuitry to the pumps along with an extensive set of current and flow rate sensors make up the rest of the system. The physical arrangement of the above mentioned components as well as their interconnections comprise the ***structural model*** (Figure 29).

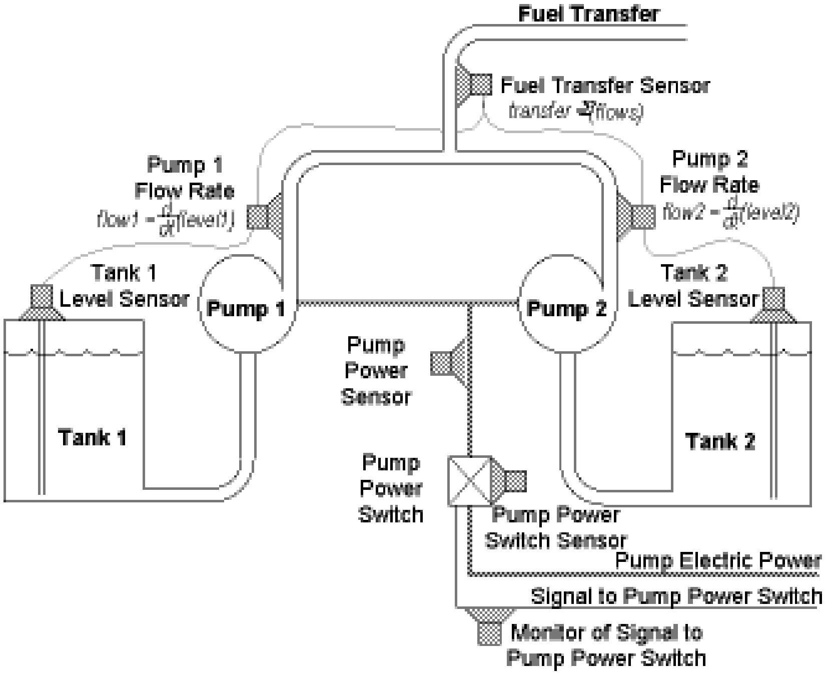


Figure 29. Structural model of JSF fuel delivery system [29].

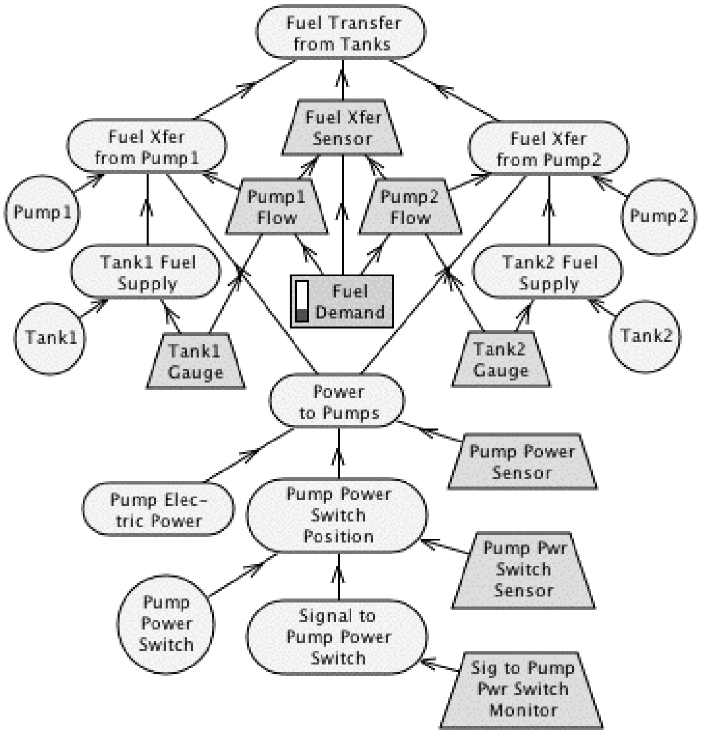


Figure 30. Functional model of JSF fuel delivery system [29].

7.1 System Abstraction

The functional specification of the system is to maintain the net required fuel flow rate from the two tanks. The fuel transfer sensor monitors the net flow rate, which is modeled as the simple sum of the individual pump flow rates, measured by their respective flow rate sensors. The individual flow rate of each tank is then expressed in terms of the change in fuel level in the tank. Assuming the pump to be the only source of fuel extraction from the tank, the decrease in level can be related to the current/power supplied to the pump through the power switch monitored by the pump power switch sensor. Finally, the power supplied through the switch is directly controlled by the on-time of the pump power switch signal. Thus, a functional flow diagram is built up to explain the overall system response in terms of its component units. Figure 30 represents this in the form of the ***functional model***. Accurate mathematical analysis of the relations mentioned above requires the introduction of complex concepts of fluid dynamics, as well as a good knowledge about the physical parameters of the units involved, e.g. the rotational speed of the pump, the cross-sectional profile of the fuel pipes, the viscosity of the fuel, etc. Again, without going into such detail, a simple behavioral analysis offers a good understanding of the condition of the system. For example, in the case of insufficient flow rate, if it is observed that the pump is powered but the tank level does not drop, then the tank must be faulty.

7.2 Reasoning

In order to draw up the fault symptom matrix we need to enlist the components of the system that can go wrong. Assuming that the fuel piping and the electrical wiring are OK and do not develop faults, we restrict ourselves to the following components along with the fault modes shown:

- Pump power switch
    - stuck ON
    - stuck OFF
- Tank 1
    - fuel leak
- Tank 2
    - fuel leak
- Pump 1
    - cracked seal
    - debris buildup

- Pump 2
    - cracked seal
    - debris buildup

Linking these fault modes with the sensors shown in Figure 29 we can build the ***fault symptom matrix*** as shown in Table 6.

Table 6. Fault-Symptom Matrix for the JSF Fuel Delivery System.

Symptoms Faults		Signal to Pump Power Switch	Pump Power Switch Status	Pump Power Consumption	Tank 1 Level	Tank 2 Level	Pump 1 Flow Rate	Pump 2 Flow Rate	Total Fuel Flow Rate
		S1	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**
Power Switch Stuck ON	**F1**	OFF	ON						
Power Switch Stuck OFF	**F2**	ON	OFF						
Tank 1 Fuel Leak	**F3**		OFF		DEC				
Tank 2 Fuel Leak	**F4**		OFF			DEC			
Pump 1 Cracked Seal	**F5**		ON	OK			LOW		LOW
Pump 2 Cracked Seal	**F6**		ON	OK				LOW	LOW
Pump 1 Debris Buildup	**F7**		ON	HIGH			LOW		LOW
Pump 2 Debris Buildup	**F8**		ON	HIGH				LOW	LOW

The next step is the construction of the ***diagnostic decision tree***. The tree is traversed in the *depth-first* method starting from the root in every iteration. Each iteration corresponds to a new dataset recorded from the sensors. Figure 31 shows the diagnostic decision tree as derived from Table 6.

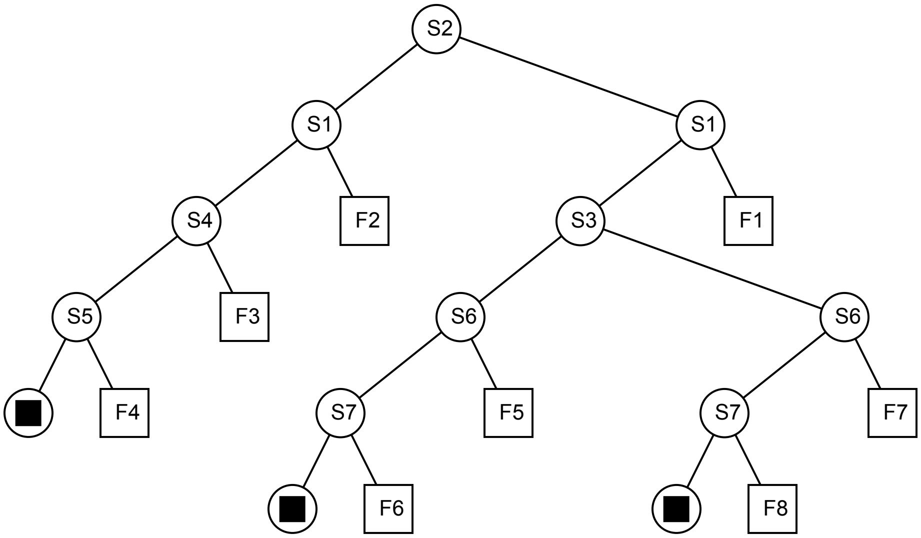


Figure 31. Diagnostic Decision Tree for the JSF Fuel Delivery System.

The ***figures of merit*** for the above binary decision tree are as follows:

- Speed = 29/8 = 3.625
- Cost = 8
- Resolving Power = 8.

A schematic of the MBR diagnostic approach in the JSF case is presented in . In this case, the structural and functional models of the fuel delivery system are given [29], and are, therefore, manually coded into the MATLAB® program. The system analysis and subsequent literature review to build the knowledge database about tanks, pumps, switches, etc. are also done manually. These steps are showed by the tan colored arrows in . The MBR program takes over from this point to generate the rest of the diagnostic reasoning framework (shown by blue arrows).

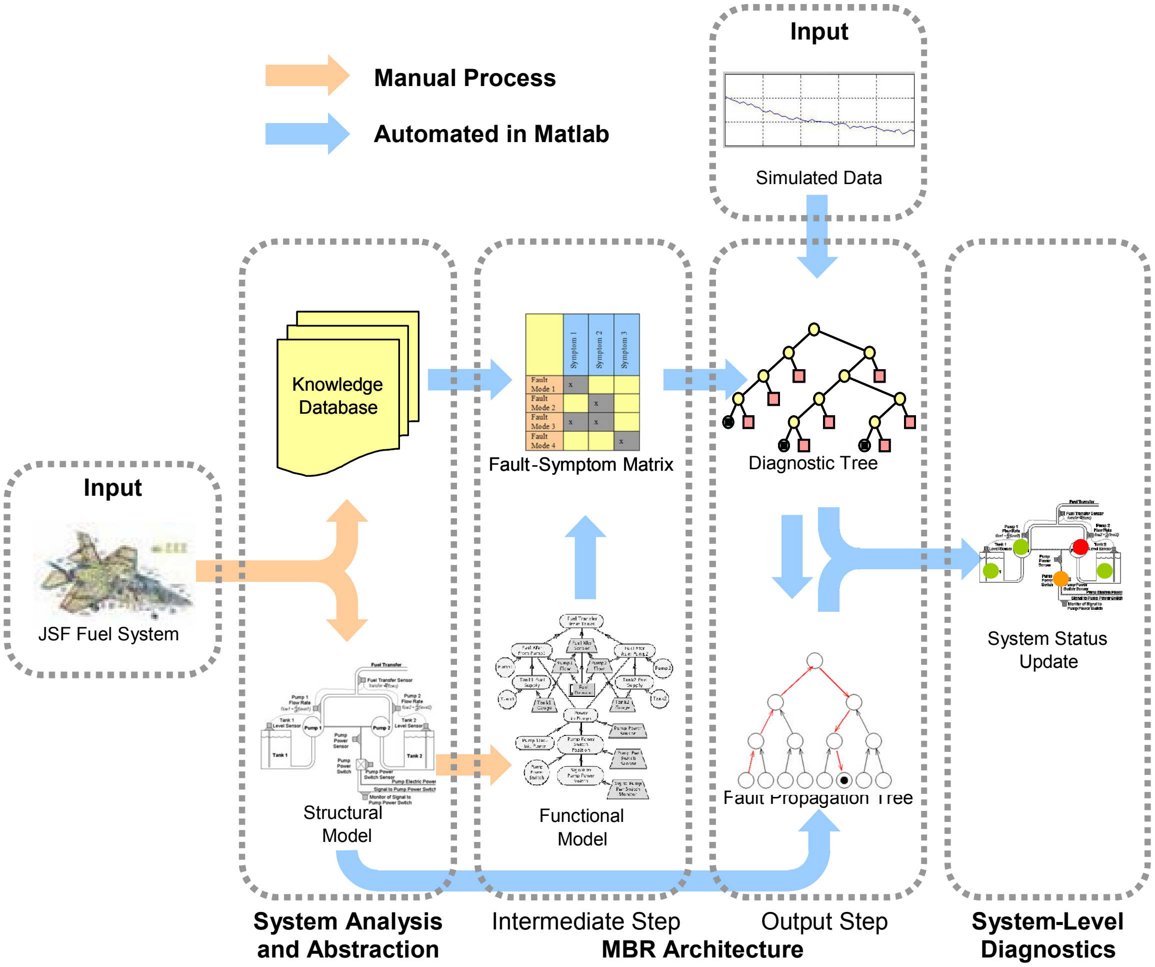


Figure 32. An MBR approach to fault detection for the JSF fuel delivery system.

7.3 Results

Since there exists no real data for this system we simulate sensor data by modeling the components as signal sources with superimposed Gaussian white noise. The tanks are modeled as containers with possible leakage flow. The pumps are modeled such that the flow rate is directly proportional to the input current and inversely proportional to debris buildup. The switches modeled with inline fuses that blow at preset current thresholds. Both tanks start at the arbitrary fuel level of 100. The flow rate of the pumps are 1 under nominal conditions. The flow rate and pump power consumption thresholds are set at 3-

sigma limits assuming normal distribution of data. During the simulation a fault is introduced at time unit 4 in the form of debris blockage in pump 2. Figure 33 shows the simulation plots of the fault symptoms (blue) along with the fault flags (red).

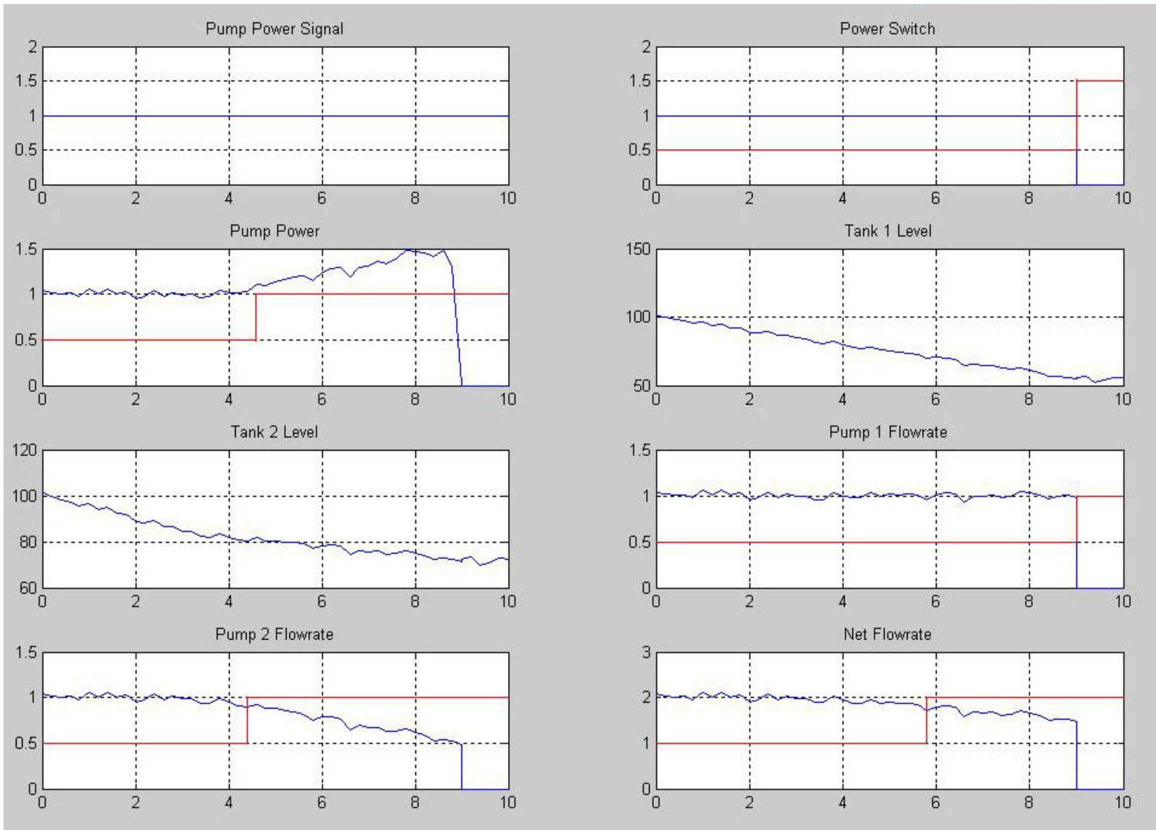


Figure 33. Plots of the JSF Fuel Delivery System fault symptoms (symptoms in blue; fault flags in red).

As shown by the plots, at time unit 4.4 the pump 2 low flow rate flag is thrown indicating some either a broken seal or debris in the flow path. The latter is confirmed by the pump power high flag being thrown at time unit 4.5. Rising consumption of power by pump 2 results in the power switch blowing a fuse and becoming stuck OFF. This is indicated by the power switch fault flag as well as the pump 1 low flow rate flag being thrown at time unit 9. This sequence demonstrates the ***fault propagation*** in the system.

8 Model Verification

One of the most important steps of the model-based diagnostic reasoning methodology described thus far is the formulation of the functional model. Consequently, it is vital to ensure that the generated model correctly captures the system behavior we are interested in. “Model verification is the process of determining that a model implementation accurately represents the developer’s conceptual description and specifications” [30]. Model checking is a technique for formally verifying finite-state concurrent systems. Specifications about the system are expressed as temporal logic formulas, and efficient symbolic algorithms are used to traverse the model defined by the system and check if the specification holds or not. Extremely large state-spaces can often be traversed in significantly less time than simulation methods.

There are a plethora of software tools for model checking applications. However, most are geared towards digital hardware design. Formal verification of complex real time systems requires support for specifying temporal properties built into the model checking software. Temporal logic of actions (TLA) is a *logic* for specifying and reasoning about concurrent systems [31]. Systems and their properties are represented in the same logic, so the assertion of the specifications that a system meets and the assertion of what a system implements are both expressed by logical implication.

TLA+ [31] is a specification language for concurrent and reactive systems that combines the temporal logic TLA with full first-order logic and ZF set theory [32]. TLC is a model checker for debugging a TLA+ specification by checking invariance properties of a finite-state model of the specification [31]. It accepts a subclass of TLA+ specifications that can be used to describe most real time systems. Major components of the overall system are modeled and verified individually. These temporal logic modules are then collated to form the description of the entire system. The verification plan for a system module can be summarized in the following steps:

- Express the system and its expected behavior as logical constructs in TLA+ using the system's functional model.
- Express the specifications for the system model under test in the form of TLA+ constructs.
- Run the TLC model checker to test the compliance of the system to stated specifications.

8.1 The IGB Example

In the case of the IGB, all that is required for the model verification process is to make sure that the input shaft speed of the IGB is multiplied by the gear ratio at the output. Looking at Figure 23 we can see that the gears are only components that modify the rotational speed, and hence the only components that need to be modeled in temporal logic. Figure 34 shows the TLA+ code that verifies the IGB model. The variables `IPGRrot` and `OPGRrot` represent input gear and output gear rotation angles in terms of the number of teeth respectively, while `ipspd` and `opspd` are the respective input and output shaft speeds. The constants `N1` and `N2` are the number of teeth on the input and output gears respectively. The initialization statement `IGBini` says that the initial rotation angles start from zero. The `IPGRnxt` and `OPGRnxt` statements describe how the rotation angles increment. The IGB statement expresses the system as a combination of the initial conditions and the state transitions, while the ASSRT specification states that as long as the gears are meshed, the output speed will be the product of the input speed and the gear ratio. The TLC model checker verifies this code, thus implying that the IGB functional model accurately captures the expected behavior.

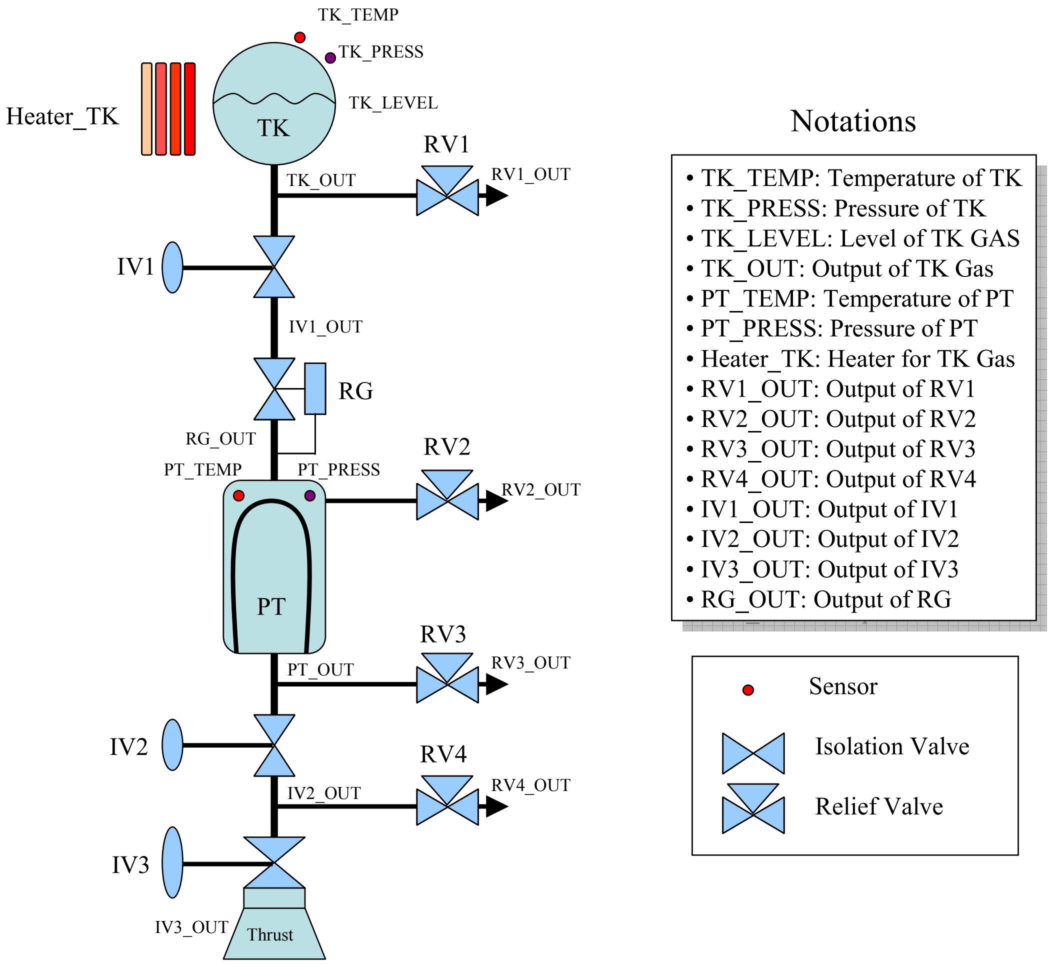
```
--------------------- MODULE IGB ------------------------
EXTENDS Naturals
VARIABLES IPGRrot, OPGRrot, ipspd, opspd
CONSTANTS N1, N2
---------------------------------------------------------

     IGBini  ==  /\ (IPGRrot = 0) /\ (OPGRrot = 0)
     IPGRnxt  ==  /\ ipspd’ = ipspd
                  /\ IPGRrot’ = IPGRrot + N1*ipspd
     OPGRnxt  ==  /\ opspd’ = opspd
                  /\ OPGRrot’ = OPGRrot + N2*opspd
     IGBnxt  ==  /\ IPGRnxt /\ OPGRnxt
     IGB  ==  /\ IGBini /\ [][HTRnxt]_<<IPGRrot, OPGRrot,
ipspd, opspd>>
     ASSRT  ==  (IPGRnxt = OPGRnxt) -> (opspd =
(N1/N2)*ipspd)
```

Figure 34. TLA+ Representation of the IGB.

Although this example has few logical expressions, this model verification methodology can be applied to more complex systems like the monopropellant propulsion system described in the next section.

8.2 *The Monopropellant Example*

The monopropellant propulsion system under study [33] uses hydrogen peroxide (H_2O_2) that passes over a catalyst and decomposes into oxygen, water, and heat, creating an expanding gas that produces the required thrust.



- TK_TEMP: Temperature of TK
- TK_PRESS: Pressure of TK
- TK_LEVEL: Level of TK GAS
- TK_OUT: Output of TK Gas
- PT_TEMP: Temperature of PT
- PT_PRESS: Pressure of PT
- Heater_TK: Heater for TK Gas
- RV1_OUT: Output of RV1
- RV2_OUT: Output of RV2
- RV3_OUT: Output of RV3
- RV4_OUT: Output of RV4
- IV1_OUT: Output of IV1
- IV2_OUT: Output of IV2
- IV3_OUT: Output of IV3
- RG_OUT: Output of RG

Figure 35. Schematic of the NASA monopropellant propulsion system.

The system, as shown in Figure 35, consists of a reservoir tank of inert gas that feeds through an isolation valve IV1 to a pressure regulator RG. The pressure regulator senses the pressure downstream and opens or closes a valve to maintain the pressure at a given set point. Separating the inert gas from the propellant is a bladder that collapses as the propellant is depleted. The propellant is forced through a feed line to the thruster isolation valve IV2 and then to the thrust chamber isolation Valve IV3. For the thruster to fire, the system must first be armed, by opening the IV1 and IV2. After the system is armed, a command opens the IV3 and allows H_2O_2 to enter the thrust chamber. As the propellant passes over the catalyst, it decomposes producing oxygen, water vapor and heat. The mixture of hot expanding gases is allowed to escape through the thruster nozzle, which in turn creates the thrust. The relief valves RV1-4 are available to dump inert gas/propellant overboard should an overpressure condition occur in any corresponding part of the system.

The corresponding functional model is depicted in Figure 36**.** The yellow circles denote the system components, the blue rounded rectangles represent the component functions, the grey boxes are external inputs, and the labeled arrows denote the system variables of interest. An autonomous contingency management (ACM) module is built around the monopropellant system so as to control the internal variables within acceptable limits in the presence of various fault scenarios. The main components that must be modeled include the Heater, Tank, Pressure Regulator, and Valves. As an example of the verification methodology we will look at a simplified behavior of the monopropellant and the ACM module together. We restrict ourselves to only a qualitative analysis of the interactions between the ACM module and the monopropellant. It is to be noted that the entire system may be modeled down to the level of partial differential equations involving the system variables as dictated by the equations of physics. However, it is our intention to only verify the logic of the ACM module as it applies to restricting the monopropellant from going in to unwanted states.

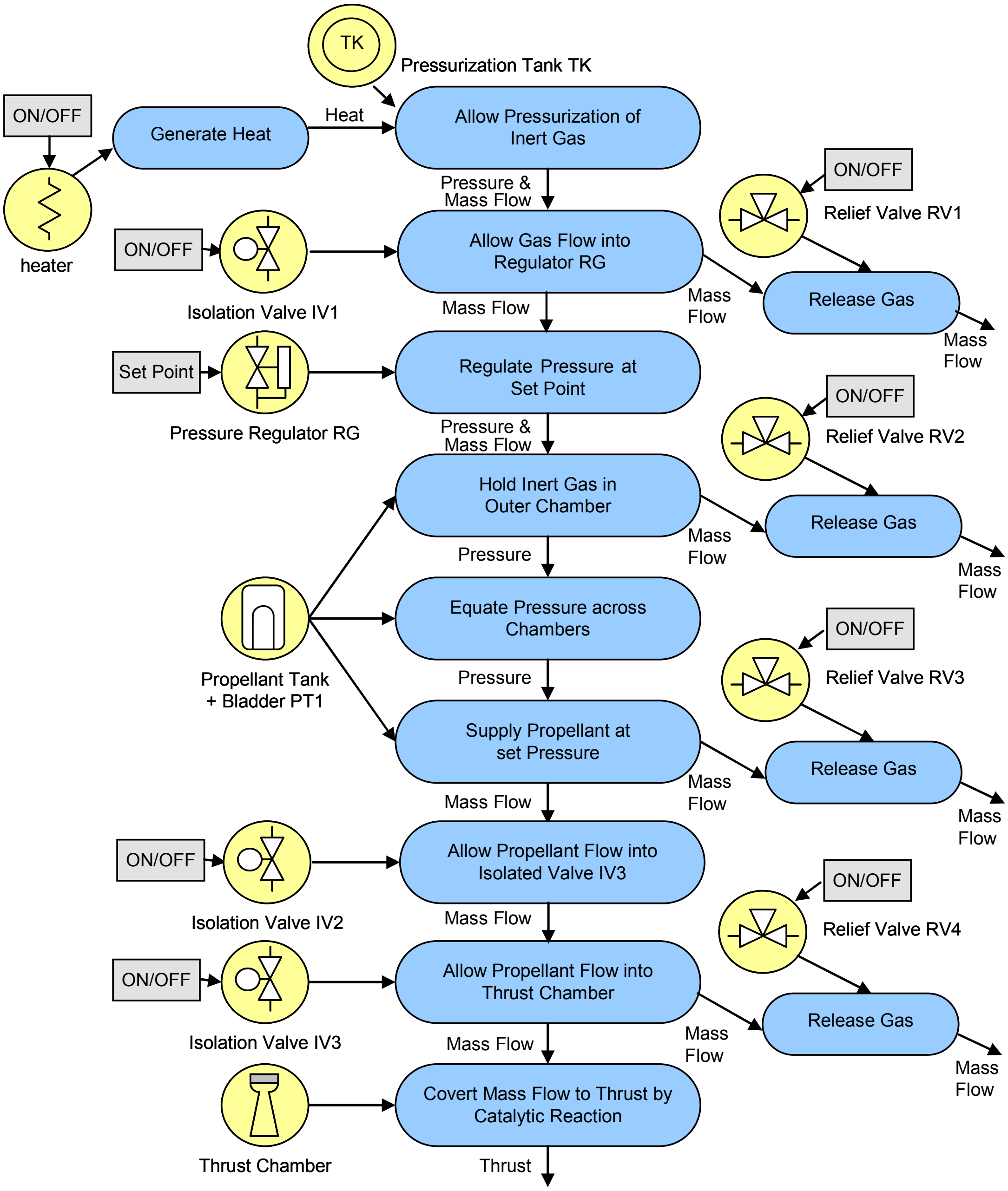


Figure 36. Functional model of the monopropellant propulsion system.

Figure 37 (the code section below) shows the TLA+ representation of the combined monopropellant-ACM system behaviors. The monopropellant system variables of interest are limited to the inert gas tank pressure (p_TK) and the propellant tank pressure (p_PT). These two pressures also respectively denote the input and output pressures of the pressure regulator. The ACM control variables are the heater ON/OFF switch (sw_HTR), release valve (RV1) flow control switch (flow_RV1), the regulator set point (sp_RG) and the system status (SYS_status). The HTR_cond, RV1_cond and the RG_cond variables simply reflect the statuses of the three fault flags of the system which are discussed later. All the values shown for the state variables are arbitrarily chosen so as to quantitatively reflect the qualitative changes during state transitions. The system specification is denoted by the SPEC statement that implies if the ACM module declares the monopropellant to be in NOMINAL state then the inert gas tank pressure must be within certain lower and upper thresholds (chosen to be 100 and 150) and if the propellant tank pressure dips below 80 then the heater must be switched on as a contingency management measure, or the ACM module must put the monopropellant in the FAILSAFE state.

```
--------------------------- MODULE Monopropellant-----------------------------------------
(****************************************************************************************)
(* This module specifies the behavior of the test system *******************************)
(****************************************************************************************)
EXTENDS        Naturals

VARIABLES      sw_HTR,
               p_TK,
               flow_RV1,
               sp_RG,
               p_PT,
               SYS_status,
               HTR_cond,
               RV1_cond,
               RG_cond

SPEC    ==     \/ (/\ p_TK >= 100
                   /\ p_TK <= 150
                   /\ (p_PT < 80 => sw_HTR = 1)
                   /\ SYS_status = "NOMINAL")
               \/ SYS_status = "FAILSAFE"
```

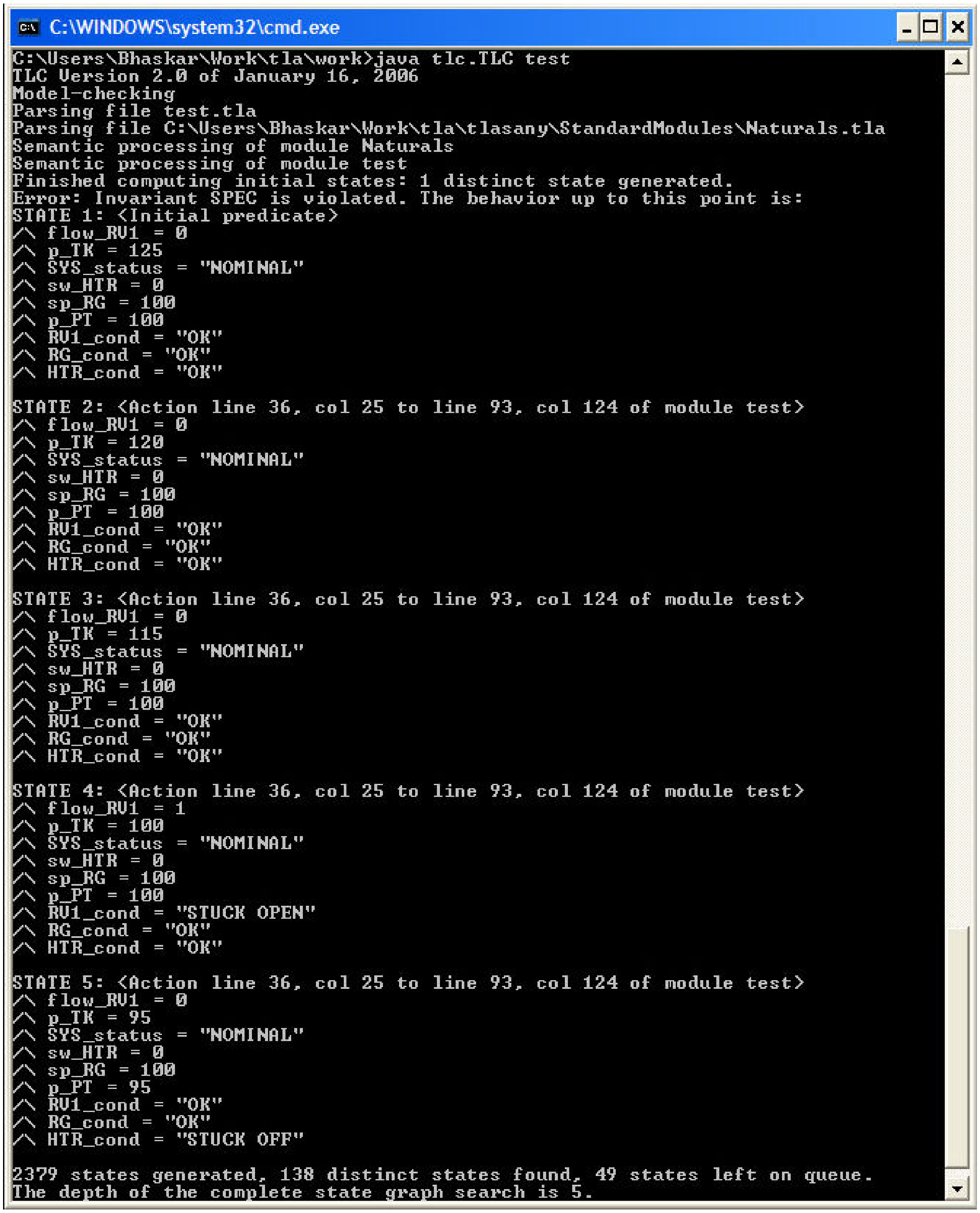
```
----------------------------------------------------------------------------------------

Init    ==      /\ SYS_status = "NOMINAL"
                /\ HTR_cond = "OK"
                /\ RV1_cond = "OK"
                /\ RG_cond = "OK"
                /\ sw_HTR = 0
                /\ p_TK = 125
                /\ flow_RV1 = 0
                /\ sp_RG = 100
                /\ p_PT = 100

SYS(HTR_fail, RV1_fail, RG_fail)
        ==      IF SYS_status = "NOMINAL"
                THEN
                /\ CASE
                        HTR_fail = 0 ->
                        /\ HTR_cond' = "OK"
                        /\ sw_HTR' = IF p_TK <= 100 THEN 1 ELSE IF p_TK >= 150 THEN 0 ELSE
sw_HTR
                        []
                        HTR_fail = 1 ->
                        /\ HTR_cond' = "STUCK OFF"
                        /\ sw_HTR' = IF p_TK >= 150 THEN 0 ELSE sw_HTR
                        []
                        HTR_fail = 2 ->
                        /\ HTR_cond' = "STUCK ON"
                        /\ sw_HTR' = IF p_TK <= 100 THEN 1 ELSE sw_HTR
                /\ CASE
                        RV1_fail = 0 ->
                        /\ RV1_cond' = "OK"
                        /\ flow_RV1' = IF p_TK > 150 \/ (p_TK = 150 /\ HTR_cond' = "STUCK
ON")
                                        THEN 1 ELSE 0
                        []
                        RV1_fail = 1 ->
                        /\ RV1_cond' = "STUCK CLOSED"
                        /\ flow_RV1' = 0
                        []
                        RV1_fail = 2 ->
                        /\ RV1_cond' = "STUCK OPEN"
                        /\ flow_RV1' = 1
                /\ p_TK' = p_TK + sw_HTR'*10 - flow_RV1'*10 - 5
                /\ CASE
                        RG_fail = 0 ->
                        /\ RG_cond' = "OK"
                        /\ sp_RG' = 100
                        /\ p_PT' = IF p_TK' > sp_RG' THEN sp_RG' ELSE p_TK'
                        []
                        RG_fail = 1 ->
                        /\ RG_cond' = "80%"
                        /\ sp_RG' = 120
                        /\ p_PT' = IF p_TK' > sp_RG' THEN sp_RG' - 20 ELSE p_TK' - 20
                        []
                        RG_fail = 2 ->
                        /\ RG_cond' = "STUCK @ 80%"
                        /\ sp_RG' = 100
                        /\ p_PT' = IF p_TK' > sp_RG' THEN sp_RG' - 20 ELSE p_TK' - 20
                /\ SYS_status' = IF     \/ (p_TK' > 150 /\ RV1_cond' = "STUCK CLOSED")
                                        \/ (p_TK' < 100 /\ RV1_cond' = "STUCK OPEN")
                                        \/ (RG_cond' = "80%" /\ HTR_cond' = "STUCK OFF")
                                        \/ RG_cond' = "STUCK @ 80%"
                                        THEN "FAILSAFE" ELSE SYS_status
                ELSE UNCHANGED <<
                        sw_HTR,
                        p_TK,
                        flow_RV1,
                        sp_RG,
                        p_PT,
                        SYS_status,
                        HTR_cond,
                        RV1_cond,
                        RG_cond>>

Next    ==      \E HTR_fail, RV1_fail, RG_fail \in {0, 1, 2} : SYS(HTR_fail, RV1_fail,
RG_fail)

SYSTEM  ==      /\ Init
                /\ [][Next]_<<
                        sw_HTR,
                        p_TK,
                        flow_RV1,
                        sp_RG,
                        p_PT,
                        SYS_status,
                        HTR_cond,
                        RV1_cond,
                        RG_cond>>

-----------------------------------------------------------------------------------------
THEOREM         SYSTEM => []SPEC
=========================================================================================
```

Figure 37. TLA+ specification of the combined monopropellant-ACM system.

The initial state of the system, as described by the Init statement, assumes the system to be in the NOMINAL state with no faults in any of the monopropellant subsystems, the heater being OFF, the release valve being CLOSED, the regulator set point being at its nominal value of 100 and the inert gas tank and propellant tank pressures being 125 and 100 respectively. The next statement SYS describes the possible behavior of the system. It takes 3 parameters HTR_fail, RV1_fail and RG_fail that respectively denote the fault conditions in the heater, release valve RV1 and the regulator. If the HTR_fail flag is 0 then the heater is in OK condition and according to the ACM strategy it is turned ON if p_TK ≤ 100, turned OFF if p_TK ≥ 150 and otherwise left as is. If HTR_fail is 1 then the heater is STUCK OFF, i.e. it can be turned OFF as before but cannot be turned ON. The conditioned is reversed for the STUCK ON case when HTR_fail is 2. The ACM strategy for the release valve RV1 in the OK state (RV1_fail = 0) is to keep it CLOSED unless p_TK > 150 or p_TK = 150 and the heater is STUCK ON. The fault scenarios are denoted by RV1_fail = 1 when RV1 is STUCK CLOSED (no flow) and RV1_fail = 2 when STUCK OPEN (constant continuous flow). At every state transition p_TK has a constant decreasing effect of 5 assuming constant thrust conditions and a decreasing effect of 10 if RV1 is open due to reducing mass and an increasing effect of 10 if the heater is ON due to temperature increase. These effects are simply a quantization of the behavior mandated by the ideal gas equation: $PV = nRT$, where P, V, n, R and T respectively denote the pressure, volume, mass (in moles), gas constant and temperature (in °K). The regulator, in the OK condition (RG_fail = 0), regulates output pressure (p_PT) at the set point sp_RG if the input pressure (p_TK) is higher than the set point or at input pressure if it is lower. The ACM module sets sp_RG at 100 in this case. Under faulty conditions p_PT is regulated at 20 below the value stated above. If the regulator is at 80% (RG_fail = 1) then the ACM strategy is to increase the set point to 120 to offset the output pressure reduction, but if the regulator is STUCK @ 80% then sp_RG cannot be changed and it remains at 100 with a correspondingly lower p_PT. The system status (SYS_status) is determined as follows: if p_TK < 100 and RV1 is STUCK OPEN, or p_TK > 150 and RV1 is STUCK CLOSED, or the regulator is at 80% and the heater is STUCK OFF, or if the regulator is STUCK @ 80% then change status to FAILSAFE, else maintain NOMINAL state. The state variables are only updated under NOMINAL conditions but not in the FAILSAFE mode. The Next statement declares the SYS expression as the rule for state change with the fail flag parameters taking any value from the set {0, 1, 2}.

The TLC model checker takes this system description and tries to prove the THEOREM, which states that the desired system specification SPEC can at all times (denoted by the symbol []) be derived from the system description SYSTEM (comprising of the Init and

Next statements). If there is some discrepancy between the system description and the specifications, the model checker will find it and output a counterexample to prove the case. For example if we run the model checker on the above test module (Figure 37), then we get a specification violation as shown in Figure 38.



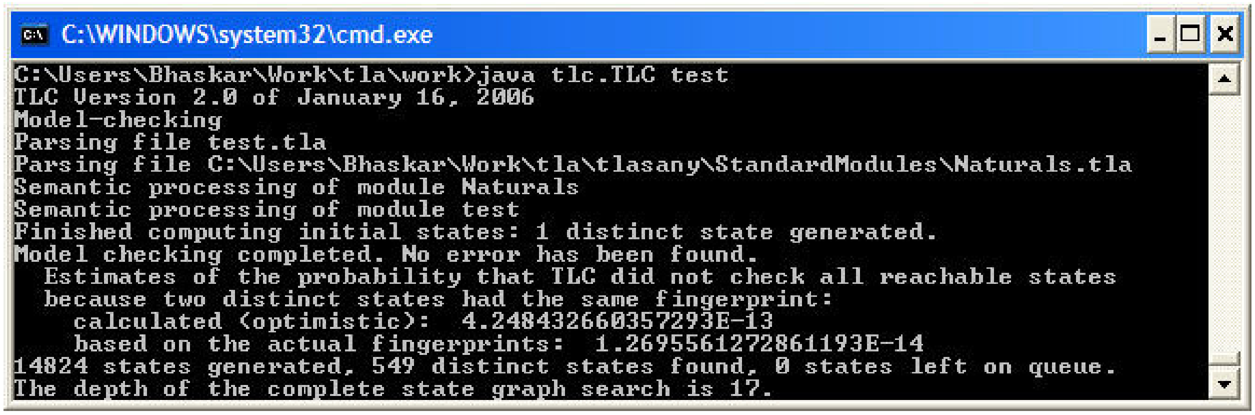
```
C:\Users\Bhaskar\Work\tla\work>java tlc.TLC test
TLC Version 2.0 of January 16, 2006
Model-checking
Parsing file test.tla
Parsing file C:\Users\Bhaskar\Work\tla\tlasany\StandardModules\Naturals.tla
Semantic processing of module Naturals
Semantic processing of module test
Finished computing initial states: 1 distinct state generated.
Error: Invariant SPEC is violated. The behavior up to this point is:
STATE 1: <Initial predicate>
/\ flow_RV1 = 0
/\ p_TK = 125
/\ SYS_status = "NOMINAL"
/\ sw_HTR = 0
/\ sp_RG = 100
/\ p_PT = 100
/\ RV1_cond = "OK"
/\ RG_cond = "OK"
/\ HTR_cond = "OK"

STATE 2: <Action line 36, col 25 to line 93, col 124 of module test>
/\ flow_RV1 = 0
/\ p_TK = 120
/\ SYS_status = "NOMINAL"
/\ sw_HTR = 0
/\ sp_RG = 100
/\ p_PT = 100
/\ RV1_cond = "OK"
/\ RG_cond = "OK"
/\ HTR_cond = "OK"

STATE 3: <Action line 36, col 25 to line 93, col 124 of module test>
/\ flow_RV1 = 0
/\ p_TK = 115
/\ SYS_status = "NOMINAL"
/\ sw_HTR = 0
/\ sp_RG = 100
/\ p_PT = 100
/\ RV1_cond = "OK"
/\ RG_cond = "OK"
/\ HTR_cond = "OK"

STATE 4: <Action line 36, col 25 to line 93, col 124 of module test>
/\ flow_RV1 = 1
/\ p_TK = 100
/\ SYS_status = "NOMINAL"
/\ sw_HTR = 0
/\ sp_RG = 100
/\ p_PT = 100
/\ RV1_cond = "STUCK OPEN"
/\ RG_cond = "OK"
/\ HTR_cond = "OK"

STATE 5: <Action line 36, col 25 to line 93, col 124 of module test>
/\ flow_RV1 = 0
/\ p_TK = 95
/\ SYS_status = "NOMINAL"
/\ sw_HTR = 0
/\ sp_RG = 100
/\ p_PT = 95
/\ RV1_cond = "OK"
/\ RG_cond = "OK"
/\ HTR_cond = "STUCK OFF"

2379 states generated, 138 distinct states found, 49 states left on queue.
The depth of the complete state graph search is 5.
```

Figure 38. TLC output for faulty system specification.

There are two options for rectifying this error. If the specification is correct then we must modify the system description till it matches with the specification. In our case, changing the SYS_status expression to include the case where the system status is changed to FAILSAFE when p_TK < 100 and the heater is STUCK OFF (as shown in Figure 39) enables a positive verification as shown in Figure 40. Alternatively, if the system description cannot be changed (as for a system past the implementation phase), then the model checker may be used to find out the limits of the system by altering the specifications until a positive verification is achieved.

```
…
/\ SYS_status' = IF
                      \/ (p_TK' > 150 /\ RV1_cond' = "STUCK CLOSED")
                      \/ (p_TK' < 100 /\ RV1_cond' = "STUCK OPEN")
                      \/ (p_TK' < 100 /\ HTR_cond' = "STUCK OFF")
                      \/ (RG_cond' = "80%" /\ HTR_cond' = "STUCK OFF")
                      \/ RG_cond' = "STUCK @ 80%"
                      THEN "FAILSAFE" ELSE SYS_status
…
```

Figure 39. Change in the TLA+ specification of the combined monopropellant-ACM system.



```
C:\Users\Bhaskar\Work\tla\work>java tlc.TLC test
TLC Version 2.0 of January 16, 2006
Model-checking
Parsing file test.tla
Parsing file C:\Users\Bhaskar\Work\tla\tlasany\StandardModules\Naturals.tla
Semantic processing of module Naturals
Semantic processing of module test
Finished computing initial states: 1 distinct state generated.
Model checking completed. No error has been found.
  Estimates of the probability that TLC did not check all reachable states
  because two distinct states had the same fingerprint:
    calculated (optimistic):  4.248432660357293E-13
    based on the actual fingerprints:  1.2695561272861193E-14
14824 states generated, 549 distinct states found, 0 states left on queue.
The depth of the complete state graph search is 17.
```

Figure 40. TLC output for corrected system specification.

9 Conclusions

On the conceptual front, the research work presented formulates a model-based reasoning architecture in order detect and classify fault modes of non-observable internal components from observable system behavior, and generates component-level fault propagation models so as to make an intelligent estimate of system health. This research attempts to bridge the gap between the overall system-level control and diagnostics applications of MBR and the computationally intensive quantitative analyses applied to component-level fault diagnosis.

On the software side, the end product is a MATLAB® program that takes a system structural model and a domain knowledge database as input and outputs a *reasoner*, comprising of the diagnostic decision tree and the fault propagation tree, which when applied to time-series data, gives an intelligent report of the system health.

9.1 Contributions

- A novel adaptation of MBR implementing a knowledge database aided diagnosis of inaccessible faulty components from observable system behavior.
- A fault propagation methodology based on the information abstracted by the model-based diagnostic reasoning step listed above.
- An integration of the diagnosis and fault propagation algorithms in an overall model-based reasoning architecture formulated in an industry preferred test-bench language like MATLAB®.

- Application of the above software to intelligent fault diagnosis of existing helicopter power-train modules and other electro-mechanical systems.
- Complexity analysis of the algorithms used by the reasoning architecture to ascertain performance guarantees for online implementation.
- Model verification technique for complex engineered systems in temporal logic.

9.2 Remaining Work

The data structures used in the MBR program presented, though suitable for the intended applications, would benefit from modifications that allow encoding of failure probability and criticality derived from FMECA studies. Also, MATLAB® representations of system models, though suitable for the chosen application domain, do not lend themselves easily to qualitative reasoning, while more intuitive representation in an AI programming language like TLA+ is not easily applicable to quantitative analysis. This underscores the need to combine the two approaches perhaps using tools like MATLAB® Stateflow.

Publications

1. Saha, B. and Vachtsevanos, G., "A Novel Model-Based Reasoning Approach to System-level Diagnostics of a Helicopter Intermediate Gearbox", *Proceedings of the 60th Meeting of the Society for Machinery Failure Prevention Technology*, pp. 281–290, Apr 2006.

2. Saha, B. and Vachtsevanos, G., "A Novel Model-Based Reasoning Approach to System Fault Diagnosis”, *10th WSEAS Int. Conf. On SYSTEMS*, July 2006.

3. Saha, B. and Vachtsevanos, G., “A Model Abstraction and Representation Approach to Helicopter Powertrain Fault Diagnosis”, *Integrated Systems Health Management Conference*, Aug 2006.

4. Saha, B. and Vachtsevanos, G., "A Novel Model-Based Reasoning Approach to System Fault Diagnosis”, *WSEAS Trans. on Systems*, Issue 8, Volume 5, pp. 1997–2004, Aug 2006.

5. Vachtsevanos, G., Zhang, B., Orchard, M., Saha, B., Saxena, A., Lee, Y., “Verification and Validation of Prognostics and Health Management Systems,” submitted to *Reliability Engineering and System Safety.*

References

[1] Davis, R. and Hamscher, W., “Model-Based Reasoning: Troubleshooting”, Exploring Artificial Intelligence, Shrobe, H. E., ed., Morgan Kaufmann, pp. 297–346, 1988.

[2] de Kleer, J. and Williams, B. C., “Diagnosing Multiple Faults”, Artificial Intelligence, vol. 32, no. 1, pp. 97–130, Apr. 1987.

[3] Reiter, R., “A Theory of Diagnosis from First Principles,” Artificial Intelligence, vol. 32, no.1, pp. 57–95, Apr. 1987.

[4] Poole, D., “Representing Diagnosis Knowledge,” Annals of Mathematics and Artificial Intelligence, vol. 11, nos. 1–4, pp. 33–50, 1994.

[5] Console, L., Dupré, D. T. and Torasso, P. “On the Relationship between Abduction and Deduction,” Journal of Logic and Computation, vol. 1, no. 5, pp. 661–690, Oct. 1991.

[6] Przytula, K. W. and Thompson, D., “Construction of Bayesian Networks for Diagnostics”, 2000 IEEE Aerospace Conference Proceedings, vol. 5, pp. 193 - 200, Mar. 2000.

[7] Shrobe, H. “Model-Based Diagnosis for Information Survivability,” Proc. 13th International Workshop Principles of Diagnosis (DX 02), 2002, www.dbai.tuwien.ac.at/user/dx2002.

[8] Friedrich, G., Stumptner, M. and Wotawa, F., “Model-Based Diagnosis of Hardware Designs,” Artificial Intelligence, vol. 111, nos. 1–2, pp. 3–39, July 1999.

[9] Felfernig, A., Friedrich, G. E., Jannach, D. and Stumptner, M., “Consistency Based Diagnosis of Configuration Knowledge Bases,” Proc. 14th European Conf. Artificial Intelligence (ECAI 2000), IOS Press, pp. 146–150, 2000.

[10] Williams, B. C. and Nayak, P. P., “Immobile Robots — AI in the New Millennium,” AI Magazine, vol. 17, no. 3, pp. 16–35, Fall 1996.

[11] Sachenbacher, M., Struss, P., and Carlén, C. M., “A Prototype for Model-Based On-Board Diagnosis of Automotive Systems,” AI Communications, vol. 13, no. 2, pp. 83–97, Nov. 2000.

[12] Misra, A., Provan, G., Karsai, G., Bloor, G., and Scarl, E., “A Generic and Symbolic Model-Based Diagnostic Reasoner with Highly Scalable Properties”,

1998 IEEE International Conference on Systems, Man, and Cybernetics, vol. 4, pp 3154 – 3160, Oct. 1998.

[13] Wu, B., Saxena, A., Khawaja, T. S., Patrick, R., Vachtsevanos, G., and Sparis, P., "An Approach to Fault Diagnosis of Helicopter Planetary Gears", IEEE AUTOTESTCON 2004 Proceedings, pp 475 – 481, Sep. 2004.

[14] Mimnagh, M. L., Hardman, W., and Sheaffer, J., "Helicopter Drive System Diagnostics Through Multivariate Statistical Process Control", 2000 IEEE Aerospace Conference Proceedings, vol. 6, pp 381 – 415, Mar. 2000.

[15] Hardman, W., Hess A. and Sheaffer, J., "A Helicopter Power Train Diagnostics and Prognostics Demonstration", 2000 IEEE Aerospace Conference Proceedings, vol. 6, pp. 355-366, Mar. 2000.

[16] Brotherton, T., Grabill, P., Friend, R., Sotomayer, B., and Berry, J., "A Testbed for Data Fusion for Helicopter Diagnostics and Prognostics", 2003 IEEE Aerospace Conference Proceedings, vol. 7, pp 3357 – 3370, Mar. 2003.

[17] Buchina, G., and Clark, R. T., "AGETS MBR: An Application of Model-Based Reasoning for Advanced Diagnostics", 1995 IEEE Aerospace Applications Conference Proceedings, vol. 2, issue 0, part 2, pp 169 – 180, Feb. 1995.

[18] Engel, S. J., Gilmartin, B. J., Bongort, K., and Hess, A., "Prognostics, The Real Issues Involved With Predicting Life Remaining", 2000 IEEE Aerospace Conference Proceedings, vol. 6, pp. 457–469, March 2000.

[19] Nolan, M. and Giordano, J. P., "Use of Adaptive Model-Based Reasoning for Embedded Diagnostics and Redundancy Management for Fault Tolerant Systems", 1997 IEEE Autotestcon Proc., 1997.

[20] Davidson, E. M., McArthur, S. D. J., and McDonald, J. R., "A Toolset for Applying Model-Based Reasoning Techniques to Diagnostics for Power Systems Protection", IEEE Trans. On Power Systems, vol 2, 2003.

[21] Kececioglu, D., "Reliability Engineering Handbook Volume 2", Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1991.

[22] Tong, D. W., Jolly, C. H., and Zalondek, K. C., "Multi-branched Diagnostic Trees" Conf. Proc., 1989 IEEE International Conference on Systems, Man and Cybernetics, vol 1, pp 92 – 98, Nov. 1989.

[23] Iwu, F. and Toyn, I., "Modelling and Analysing Fault Propagation in Safety-Related Systems", Proceedings of the 28th Annual IEEE/NASA Goddard Software Engineering Workshop (SEW'03), Dec. 2003.

[24] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C., “Introduction to Algorithms”, The MIT Press; 2nd edition, Sep., 2001.

[25] Cook, S., "The Complexity of Theorem Proving Procedures", Proceedings of the third annual ACM symposium on Theory of computing, pp 151–158, 1971.

[26] Technical Manual, “Aviation Unit and Intermediate Maintenance for Army Models UH60A, UH60L, and EH60A Helicopters”, US Army TM 1-1520-237-23-4, May 29, 1998.

[27] Vance, J. M., “Rotordynamics of Turbomachinery”, John Wiley & Sons, Inc., New York, USA, ISBN 0-471-80258-1, 1988.

[28] Broch, J. T., “Mechanical Vibrations and Shock Measurements”, Brüel & Kjær, Nærum, Denmark, ISBN 87-87355-36-1, 1984.

[29] Gandy, M., Line, K., “Joint Strike Fighter – Prognostics and Health Management (PHM)” Lockheed Martin Aeronautics Company, Public Release AER200307014.

[30] Hughes, W. P., “Military Modeling for Decisions”, Third Edition, Military Operations Research Society, 1997.

[31] Lamport, L., “Specifying Systems”, Addison-Wesley Professional, 1st edition, 2002.

[32] Cohen, P. J., “Set Theory and the Continuum Hypothesis”, W. A. Benjamin Inc., New York, Amsterdam, 1966.

[33] Vesely, W., Stamatelatos, M., Dugan, J., Fragola, J., Minarick III, J., Railsback J., “Fault tree handbook with Aerospace Applications”, version 1.1, NASA, August 2002.