AUTOMATED PRODUCTION OF INFORMATION MODELS FOR USE IN MODEL-BASED DIAGNOSIS

John W. Sheppard
William R. Simpson

ARINC Research Corporation
2551 Riva Road
Annapolis, MD 21401

ABSTRACT

The most difficult part of system maintenance is diagnosing failures. System expertise is needed, but manuals cannot cover all the unique situations and in-house experts are difficult to retain because of today's budgets. The resulting "expertise gap" has led to the development of "intelligent" diagnostic aids to compensate for the missing expertise.

The first computer-assisted diagnostic aids attempted to capture and emulate the reasoning process of experts by encoding heuristics they used for diagnosis. Despite some early successes, capturing expertise is a high-risk process.

An alternative approach to system maintenance uses a model of the system as a foundation for its knowledge base. This model-based approach differs from the rule-based approach in the way it incorporates and processes knowledge about the system. A particular type of model, the information flow model, is very useful for diagnosing problems. This approach, however, suffers from a complex model building process that is labor-intensive.

Recently, the model building process has been automated by a tool that enables both top-down and bottom-up modeling. As discussed in this paper, this type of tool has the potential to enable widespread usage of the information modeling approach to system maintenance. We also discuss how the automated modeling approach can be used with machine learning techniques to enable the process to adapt with changing systems.

INTRODUCTION

Developing diagnostic systems constitutes a complex and labor-intensive task. In the past, analysts developed diagnostic systems (generally as technical manuals and fault trees) using simulation and failure-modes analysis. With the advent of the expert system, knowledge engineers began encoding heuristics used by expert diagnosticians in the hopes of developing computer-based systems that performed at the level of the expert. In the medical domain, Shortliffe showed that rules could successfully represent complex medical knowledge, and his MYCIN system was capable of diagnosing several infectious diseases.[1] Unfortunately, the expert system has failed to satisfy the maintenance requirements of modern systems. This arises largely because of the lack of system experts or because experts often cannot agree, experts cannot explain intuitive conclusions, and new systems have no experts.

In response to problems associated with manual diagnosis and the deficiencies of expert systems, analysts and engineers are exploring an alternative approach to intelligent diagnosis–model-based reasoning. Model-based reasoners use mathematical representations of the systems to be diagnosed and generally appear in one of two forms: behavioral or structural. The *behavioral* model incorporates knowledge from the theory of operation and establishes mathematical transfer functions that characterize the functions of the system. When a system fails, its behavior changes in a such a way that the discrepancy with the simulation model identifies the specific fault. The *structural* model begins with knowledge about the topology of the system (i.e., how the components are connected) and combines information obtained from functional testing. The resulting model defines a knowledge network (similar in function to a rule base but different in the type of knowledge used) that is used to choose a test and to draw inferences from the test outcome. Davis[2] developed a system that combined the behavioral model and the structural model for electronic circuit diagnosis. Unfortunately, his system was limited to simple circuits.

CH3158-3/92/0000-0972 $1.00 © 1992 IEEE

Today, several companies provide model-based reasoners as diagnostic aids. Most of these aids use a variation of the structural model as their knowledge base due to the apparent simplicity in constructing the models. In particular, Cantone, et al., Simpson and Sheppard, and Pattipati and Alexandridis have developed generic architectures for fault diagnosis using dependency-based models (i.e., information flow models).[3-6] We assume a pair of tools that function together that use an information flow model to assess system testability and diagnose faults—the System Testability and Maintenance Program (STAMP®) and the Portable Interactive Troubleshooter (POINTER™).

Tremendous research and commercial potential in computer-based diagnosis exist in the area of developing knowledge bases for the diagnostic system. In the past, analysts constructed knowledge bases through a cycle of interviewing experts, coding rules, generating prototypes, and testing the prototypes against data bases derived from the experts. The process continues until the resulting diagnostic system performs according to some set of user requirements. Unfortunately, the approach is generally ad hoc and incomplete.

For model-based systems, modelers study design documents and existing technical manuals to learn the "physics" of the system to be diagnosed. Resulting models become the knowledge base for diagnosis. Although more rigorous in their development, models constructed in this way are highly prone to error because of the inherent complexity of the systems modeled. And with advances in technology, complexity grows proportionately. Therefore, system modelers need to control the complexity of the diagnostic problem while still producing robust and efficient models for diagnosis.

Research in machine learning has provided a new avenue for developing models and knowledge bases for diagnosis. These new approaches evolve models based on actual examples of failure and diagnosis. Using these examples (and possibly an initial model of the system), diagnostic systems learn the relationships between the system being tested and the faults causing anomalous behavior.

In this paper, we explore an alternative learning approach to assist the modeling process. We describe a system in which we use a simulation model as a "teacher" to identify test attributes automatically for the system to be diagnosed. The automated modeling system begins with a simulation model and evaluates tests in a nominal situation to determine the limits and tolerances on these tests. Then the system sequentially fails the components in the system and reruns the simulation to determine which tests will fail. The results of these simulations define an *attribute map* for the system that becomes the basis for an information flow model to be processed by STAMP and POINTER.

THE INFORMATION FLOW MODEL

The model-based approach incorporates techniques from information fusion and artificial intelligence to guide analysis. Tests provide information, and diagnostic inference combines information from multiple tests using symbolic logic and pattern recognition.

The structure of the information flow model includes two primitive elements: tests and fault-isolation conclusions. Tests include any source of information that can be used to determine the health of a system. Fault-isolation conclusions can include failures of functionality, specific nonhardware failures (such as bus timing errors), specific multiple failures, and the absence of a failure indication. The information obtained may be a consequence of the system operations or a response to a test stimulus. Thus, we include observable symptoms of failure processes in the information flow model as tests. Doing this allows us to analyze situations that involve information sources other than formally defined tests (here, however, we discuss only defined tests). The purpose of our model, of course, is to combine these information sources (tests) to derive conclusions about the system being diagnosed.

The basic representation of the information flow model includes both a dependency representation (structural model) and a logical representation (logic model) of the system being analyzed. In addition, the information flow model includes the definition of groups of logically related tests and conclusions. In this representation, we define logical values for tests and fault-isolation conclusions. Specifically, if a test fails, it is true; if a test passes, it is false. An asserted conclusion (i.e., a failure) is true; a conclusion eliminated from consideration is false. (see Ref. 5 for a discussion of the information flow model.)

We simplify the scope of the model in the following way. First, we assume that the tests correspond to specifically defined observation points within a system. Our example is a simple analog circuit, and these points correspond to probes in the circuit. In addition, we limit fault-isolation conclusions to correspond to specific failure modes of the circuit components. For example, a transistor may fail in at least one of six ways:

- Open collector
- Open base
- Open emitter
- Collector-emitter short
- Base-emitter short
- Collector-base short

For a given transistor in a circuit, we include one fault-isolation conclusion for each of these failure modes in the model. Here, we consider developing only an attribute

map, so tests depend only on fault-isolation conclusions (later processing recovers test-to-test dependencies). A test depends on a fault isolation conclusion (i.e., failure mode) if and only if the test fails when the failure mode exists, and the failure mode does not exist when the test passes.

LEARNING IN FAULT DIAGNOSIS

A problem common to all computer-aided diagnostic systems is that the knowledge bases (whether rule bases or models) are difficult to develop. As a result, errors are common. This leads to inefficient and even incorrect diagnosis. Further, as the complexity of systems increase, the likelihood of erroneous models increases. Two questions naturally follow from this problem:

1. How does one develop models that minimize the chance of error?

2. In the event errors occur, how does one identify and correct the errors?

Work in the area of machine learning suggests potential solutions to both problems. Through a process of simulation or fault insertion, examples can be generated that capture the failed behavior and determine test-to-fault relationships. Also, discrepancies in repair recommendations and actions taken to repair the system help identify errors in the knowledge base. Finally, actual use of a diagnostic system provides information related to improving diagnostic performance. Before addressing the first question, we review learning techniques.

Simpson and Dowling describe a system that learns optimization data through actual diagnosis.[7] As systems are tested, the diagnostic aid times tests and records failures. Then the aid modifies the test time and failure rate weights to reflect the recorded events. Future diagnoses use the modified weights during optimization.

Another approach becoming popular involves connectionist systems (also called neural networks). Perhaps the most frequently used approach is back-propagation.[8] Diagnostic neural networks map test results to diagnoses through a process of training. When training a neural net, the network processes several example test-fault combinations and learns by minimizing the error in its output through a hill-climbing algorithm. In addition, neural networks have been trained to interpret signals generated from running a test to determine if a test has passed or failed.[9] We have also developed a neural network to interpret the results of the inference process (under uncertainty) with the information flow model to determine if additional testing is necessary.[10]

The learning method applied for neural networks frequently falls in the category of similarity-based learning (SBL). DeJong describes SBL as "discovering a combination of features that best classifies the regularity in a set of examples. The resulting generalization over the examples is the new concept."[11] In SBL, a teacher presents several examples to the learning system until the system recognizes the regularity discussed by DeJong.

In the event domain knowledge is available, this knowledge can be used to reduce the number of training instances in learning. Explanation-based learning (EBL) incorporates a detailed domain theory and a detailed functional specification of the concepts to be learned. Using the domain knowledge, the system learns concepts with single training instances.[11] In fault diagnosis, an example of a misdiagnosis together with a model of the physics for the technology employed in the system can be used to derive an explanation of an appropriate diagnosis. From this explanation, the learning system modifies the diagnostic model to include the knowledge of the correct diagnosis.

The automated modeling approach we describe fits loosely within explanation-based learning. The simulation model forms the domain theory to be used to diagnose. In this case, there will be no misdiagnoses. Rather, the teacher inserts faults in the simulation model and runs another simulation. The learning system examines the behavior of the simulation and compares that behavior to what it expects given a functional system. It identifies discrepancies at the probe points in the simulation and explains the discrepancy with the knowledge that the system has failed. From this explanation, the learning system identifies a set of attributes to be used in the attribute map for the circuit. The complete attribute map then forms the basis for the information flow model.

SYSTEM SIMULATION

The first step in the automated modeling process is to develop a baseline simulation model of the system to be diagnosed. For our discussion, we use the example circuit in Figure 1. This circuit has 5 transistors, 1 capacitor, and 10 resistors. In addition, a 12-volt dc power supply is available, and 10 volts dc are applied as input. An analyst begins by writing a simulation model of the circuit. This simulation should reflect nominal behavior of the circuit because it will be used to determine test limits. We chose to use the PSpice™ circuit simulation package to implement our prototype modeling tool. (PSpice is a registered trademark of MicroSim Corporation.) The baseline PSpice simulation model for this circuit is shown in Figure 2.

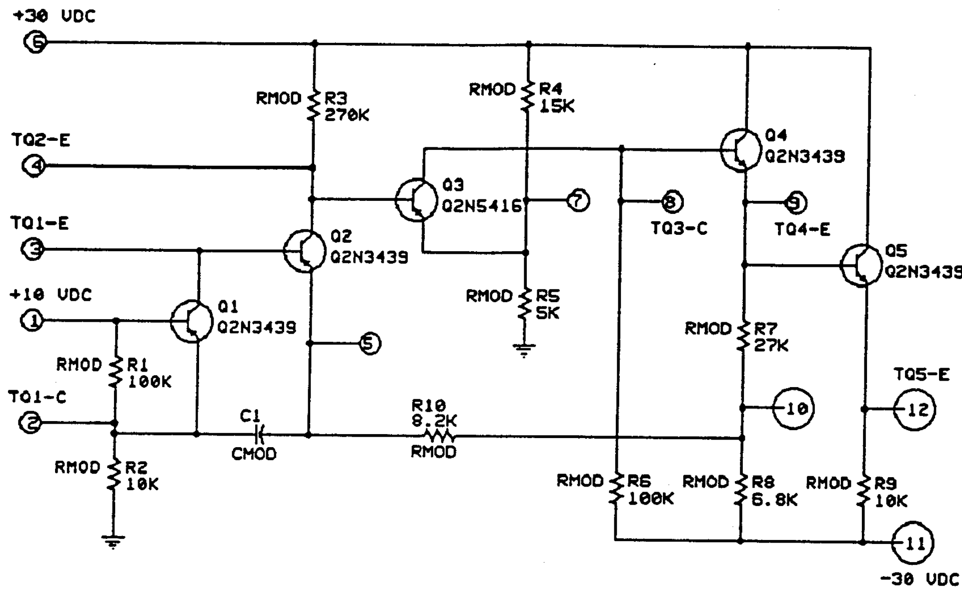

Figure 1. Example Circuit

```
Q1  2  1  3     Q2N3439
Q2  4  3  5     Q2N3439
Q3  8  4  7     Q2N5416
Q4  6  8  9     Q2N3439
Q5  6  9  12    Q2N3439
*
C1  2  5  CMOD  500PF
*
R1  1  2  RMOD  100K
R2  2  0  RMOD  10K
R3  6  4  RMOD  270K
R4  6  7  RMOD  15K
R5  7  0  RMOD  5K
R6  8  11 RMOD  100K
R7  9  10 RMOD  27K
R8  10 11 RMOD  6.8K
R9  12 11 RMOD  10K
R10 10 5  RMOD  8.2K
*
.LIB
.MODEL RMOD RES(R=1 DEV=10%)
.MODEL CMOD CAP(C=1 DEV=105)
*
VPS     6  0  30
VNPS    11 0  -30
VINPUT  1  0  10
*
.END
```

Figure 2. PSpice Baseline Simulation File for the Example Circuit

In addition to the circuit file, PSpice requires additional information to be included with each simulation. PSpice runs simulations for determining test limits (the worst case analysis) and then runs several simulations for each possible failure mode for each possible component in the circuit. Currently, we have developed a library of 23 failure modes and 3 types of tests. The 23 failure modes address the resistor, the capacitor, the transistor, the diode, a voltage source, and various integrated circuits; and 3 types of tests include dc, noise, and transient.

For each simulation, several special files need to be written.

- *Test Description File.* The test description file specifies the location of probe points in the circuit and the parameters necessary to evaluate the probe. Usually, an entry in the file includes a test label, the test type, the test location, and time parameters for making the measurement.

- *Test State File.* The test state file specifies the tests that belong to each state in the circuit.

- *Special Component File.* At times, the basic component in the PSpice library is insufficient to describe a particular component in the circuit. The special component file allows the engineer to specify the reference designator and the component type so that the correct failure modes can be associated.

- *Common Setup File.* The common setup file allows the engineer to group simulation setup commands in a separate file rather than in the actual circuit file.

This permits easy modification of operating conditions without changing the actual circuit file.

- *Statement Addition File.* The statement addition file associates PSpice commands with specific operating states so that only relevant commands are included in the simulation. When the automated modeling program invokes PSpice, it first inserts the relevant commands from this file into the circuit file.

- *Statement Deletion File.* Similar to the statement addition file, the statement deletion file specifies commands in the PSpice circuit file that do not apply to particular operating states. When the automated modeling program invokes PSpice, it begins by deleting the relevant commands from the circuit file.

- *Special Test File.* The special test file permits tests to be associated with specific failure modes in the circuit. This prevents the automated modeling system from applying the special test to components not relevant for analysis.

These data files define the complete input data package for the automated modeling system. Once the files have been defined, the automated modeler proceeds to run several simulations to compute test limits and identify attributes of system failures. The following two sections describe these processes.

Determining Test Limits

After the engineer constructs the baseline circuit file, he or she runs PSpice to determine test limits. In addition to the circuit file, the engineer constructs a test file describing probes in the system. The automated modeling program inserts the probes in the circuit file and runs the circuit using a worst-case analysis. PSpice runs each test individually in a default state, although all the tests associated with a given state could be run simultaneously. The worst-case analysis varies the input through a range of nominal values to determine the limits at the probe points in the circuit. These limits define the range of values for the corresponding test to pass. If the probe detects a value beyond the limits, the test fails.

For the sample circuit, we have defined five tests: two are dc tests, two are transient analyses, and one is a noise test. Because of the nature of transient analysis, the two transient tests expand to three tests corresponding to "voltage-peak-positive," "voltage-peak-negative," and "voltage-peak-to-peak." The five tests examine signals from the Q4 and Q5 transistors, and their test limits are given in Table 1.

Table 1. Test Limits for Sample Circuit

Test	Label	Lower Limit	Upper Limit
DC, Q4	TQ4-E	-15.12	-11.11
Voltage Peak-to-Peak, Q4	TQ4-ETR-VPP	14.05	17.13
Voltage Peak Pos., Q4	TQ4-ETR-VP	1.01	4.29
Voltage, Peak Neg., Q4	TQ4-ETR-VN	-13.04	-12.84
DC, Q5	TQ5-E	-15.64	-11.62
Noise, Q5	TQ5-ENO	0.00	1.91×10^{8}
Voltage Peak-to-Peak, Q5	TQ5-ETR-VPP	13.92	17.21
Voltage Peak Pos., Q5	TQ5-ETR-VP	0.48	3.75
Voltage Peak Neg., Q5	TQ5-ETR-VN	-13.46	-13.44

Fault Insertion

Once the test limits have been determined, the automated modeler inserts faults into the circuit according to the failure mode library. Only one component is considered at a time, and only one operating state is considered at a time. All tests appropriate to an operating state are considered simultaneously. To insert a fault, the automated modeler modifies the circuit file by eliminating the appropriate component line and inserting appropriate commands from the failure mode library. For example, if the capacitor in the sample circuit shorts, the following two lines will be inserted in the circuit file:

```
RADD1 2 5 RES  0.0001
C1    2 5 CMOD 500PF
```

As noted, the automated modeler runs simulations for each fault insertion and examines the PSpice output report to determine which (if any) tests detect the failure. The attributes of the inserted failure mode make up the failed test. For the capacitor that has shorted, the automated modeler finds the following attributes:

```
TQ4-ETR-VPP, TQ4-ETR-VP, TQ4-ETR-VN,
TQ5-ETR-VPP, TQ5-ETR-VP, TQ5-ETR-VN
```

As a result, the automated modeler has determined that each transient analysis test depends on the failure of capacitor C1 (i.e., if C1 shorts, each transient test will detect the failure).

When the automated modeler generates the complete attribute map, it provides a corresponding dependency list to STAMP for processing. In order for STAMP to use the attribute map, it first identifies any additional dependency relationships between the tests in the model. STAMP uses an inference procedure called *logical closure* to compute these dependencies. (The algorithms for logical closure are provided in Ref. 5.) Following closure, STAMP analyzes the model to assess system testability. In particular, it evaluates test resource utilization, identifies ambiguity, considers false alarms, and assesses multiple failure problems. STAMP also provides several fault trees for manual diagnosis.

As an alternative, the model processed by STAMP can be used by POINTER for interactive diagnosis. POINTER does not use a precomputed fault tree but incorporates the optimization and inference algorithms of STAMP to diagnose a system adaptively. Each step in the diagnostic process is determined based on the current context. In addition, information relating to test confidence, grouping, and alternate inference can be incorporated in the diagnostic process. Finally, POINTER includes additional learning algorithms to improve performance and identify errors in the model (as described above).

FUTURE DIRECTIONS

Whether using fault trees, rule bases, simulation models, or dependency models, developing knowledge bases for complex system diagnosis is a time-consuming and error-prone task. To improve the knowledge acquisition process (i.e., reduce the cost and increase the accuracy), we have developed a system for manipulating simulation models to construct structural diagnostic models. To automate the process further, a CAD system that produces PSpice models (e.g., SCHEMA™ registered by Omation, Inc.) can be used to develop circuits and feed the automated modeler. In addition to the basic circuit file generated in SCHEMA, the engineer simply provides information on test points in the system. The automated modeler then provides a first-level model of the circuit. It is unlikely that this model will be complete. We anticipate that the engineer will need to adjust the model to include additional tests and operating conditions not included in the simulation, but this first model provides a significant savings in the total modeling and analysis process.

In the future, we anticipate adding more tests, increasing the size of the failure mode library, and incorporating information for other simulators. For example, we plan to include frequency tests and digital tests that will expand the range of components and failure modes that can be processed. We also anticipate developing interfaces with VHDL simulators and various digital simulators such as HITS. With the current emphasis on concurrent engineering, we believe the automated modeling approach will help designers effectively include design for testability in the design process for complex systems.

REFERENCES

1. Shortliffe, Edward, *Computer Based Medical Consultations: MYCIN*, New York, Elsevier, 1976.

2. Davis, Randall, "Diagnostic Reasoning Based on Structure and Behavior," *Artificial Intelligence*, Vol. 24, 1984.

3. Cantone, Richard R.; Frank J. Pipitone; W. Brent Lander; and Michael P. Marrone; "Model-Based Probabilistic Reasoning for Electronics Trouble-shooting," *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, 1983.

4. Simpson, William R., and John W. Sheppard, "System Complexity and Integrated Diagnostics," *IEEE Design and Test of Computers*, Vol. 8, No. 3, September 1991.

5. Sheppard, John W., and William R. Simpson, "A Mathematical Model for Integrated Diagnostics," *IEEE Design and Test of Computers*, Vol. 8, No. 4, December 1991.

6. Pattipati, Krishna R., and Mark G. Alexandridis, "Application of Heuristic Search and Information Theory to Sequential Fault Diagnosis," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 20, No. 4, July/August 1990.

7. Simpson, W. R., and C. S. Dowling, "WRAPLE: The Weighted Repair Assistance Program Learning Extension," *IEEE Design and Test of Computers*, Vol. 3, No. 2, April 1986.

8. Rumelhart, D. E.; G. E. Hinton; and R. J. Williams; "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing*, The MIT Press: Cambridge, Massachusetts, 1986.

9. Katz, W. T., and M. B. Merickel, "Translation-Invariant Aorta Segmentation from Magnetic Resonance Images," *Proceedings of the International Joint Conference on Neural Networks '89*, Washington, DC, June 1989.

10. Sheppard, John W., and William R. Simpson, "A Neural Network for Evaluating Diagnostic Evidence," *Proceedings of the National Aerospace & Electronics Conference*, Dayton, Ohio, May 1991.

11. Gerald DeJong, "An Introduction to Explanation-Based Learning," in *Exploring Artificial Intelligence*, Morgan-Kaufmann Publishers: Palo Alto, California, 1988.