TEST: A MODEL-DRIVEN APPLICATION SHELL

Gary S. Kahn, Al Kepner, and Jeff Pepper
Carnegie Group Inc.
Pittsburgh, Pa. 15219

Abstract

TEST (Troubleshooting Expert System Tool) is an application shell that provides a domain-independent diagnostic problem solver together with a library of schematic prototypes. **TEST** fills a design niche halfway between rule-based and causal-model approaches.

This approach has resulted in a design that meets several functional requirements for an effective troubleshooting shell. Most critically, **TEST** can represent both the impact of failure-modes on a machine or system of interest, as well as the heuristic problem-solving behavior which can lead to rapid conclusions.

This paper provides an overview of **TEST**'s approach to diagnosis. As a special purpose application shell, **TEST** provides considerably more leverage to developers than can be gained through the use of general purpose heuristic classification systems.

1. Introduction

TEST[1] (Troubleshooting Expert System Tool) is an application shell that provides a domain-independent diagnostic problem solver together with a library of schematic prototypes. These prototypes constitute the object types and the structure required by each domain-specific **TEST** knowledge base. **TEST** applications for factory floor machines, vehicles, and computers, are currently in development.

TEST fills a design niche halfway between rule-based and causal-model approaches. On one hand, **TEST** uses a weak causal model to describe causal links between failure-modes; and on the other, **TEST** uses rules to constrain and direct diagnostic reasoning. **TEST** is, in some respects, similar to several other recent attempts to develop problem-solving architectures suitable to the troubleshooting task [Bylander *et al.* 83, Hofmann *et al.* 86]. **TEST** differs, however, in offering a more differentiated knowledge base and a more powerful set of control and inference options.

[1]**TEST** is an internal name used at Carnegie Group Inc. **TEST** is implemented in Knowledge Craft™.

TEST's approach has resulted in a design that meets several functional requirements for an effective troubleshooting shell. Most critically, **TEST** can represent both the impact of failure-modes on a machine or system of interest, as well as the heuristic problem-solving behavior which can lead to rapid conclusions. The underlying representation and the problem-solving method are easily understood by both design engineers and diagnostic technicians. This has had a positive impact on knowledge acquisition. Systems built in **TEST** have been found to be more easily maintainable than those built using Emycin-like [VanMelle *et al.* 81] belief rules.

TEST's approach to diagnosis is explained in the following sections. The first provides a context of previous work in the field, identifying limitations which motivated the development of **TEST**. The following two sections present overviews of the **TEST** knowledge base and diagnostic problem solver. The subsequent section describes **TEST**'s unique use of rules.

Many features of the **TEST** system cannot be covered within the scope of this paper. A comprehensive account of **TEST**'s functionality can be found in [Pepper and Mullins 86]; **TEST**'s approach to repair is described in [Pepper and Kahn 87]; and finally, development of a specialized knowledge acquisition workbench is reported in [Kahn 87].

1.1. Limits of Current Approaches

Many diagnostic expert systems have been built over the last several years. Typically, these systems use either evidential or causal reasoning. Most evidential reasoning systems, such as Mycin [Shortliffe 76] and Mud [Kahn and McDermott 86], are rule-based. Each rule represents a belief association between evidential considerations and a conclusion warranted by the evidence. There may be many rules bearing on the same conclusion. A numeric algorithm is used to compose evidence provided by each applicable rule. A simple decision rule is used to identify the best or most warranted conclusion. Causal systems, such as Abel [Patil *et al.* 81], Caduceus [Pople 82], and more recently the qualitative reasoning models of Dekleer and Brown [DeKleer and Brown 84], among others, operate on the basis of an underlying model of entities and the explicit representation of causal, behavioral, and/or structural relations. These models are used either to support differential diagnosis, or in the case of qualitative reasoning, a simulative approach to diagnosis. Casnet [Weiss *et al.* 78] used a probabilistic model of causal relations to drive an essentially Bayesian analysis.

While the above approaches have proved quite effective within certain domains, none have proved ideal for troubleshooting tasks, such as machine fault diagnosis. The Emycin approach provided by many expert system shells typically proves inadequate for three reasons. First, troubleshooting expertise is less a matter of generating beliefs on the basis of observed symptoms, than of using observations as they become known to effectively guide one to conclusive tests. Thus, the simple representational semantics of evidential belief rules typically does not correspond to the way experts think about diagnosing failure-modes. Secondly, the backward-chaining control strategy provided within this paradigm does not easily allow systems to modify the order in which candidate failures are considered as new evidence bearing on expected likelihoods is accumulated. Finally, pure rule-based approaches tend to require exhaustive search. This is impractical for many troubleshooting tasks where there are a large number of problems which could explain any particular failure-mode.

Problem solvers which rely on causal models to support differential diagnosis or simulation typically run into three problems when used to reason about machine faults. The first occurs during development, as the task of constructing large models becomes bogged down in complexity and issues of behavioral validation. The second occurs at run-time as these techniques typically result in intensive search and, as a consequence, inadequate performance. Finally, domain-specific heuristic solutions to the performance problem are often difficult to achieve within the strong representational and control constraints of these problem-solving architectures.

In troubleshooting tasks, a diagnostic conclusion is reached by performing a test or series of tests that combine to isolate an underlying failure-mode. Expertise is less a matter of evaluating the evidence in aggregate than of effectively searching for a conclusive test. While this search can often be explained in terms of differential diagnosis, much of the underlying reasoning is "precompiled" on the part of diagnostic technicians. **TEST** takes advantage of this to greatly reduce diagnostic search. At the same time **TEST** preserves a "weak" causal model with several benefits. In particular it supports knowledge base maintenance, explanation, as well as several effective problem-solving strategies.

2. The Knowledge Base

Unlike rule-based diagnostic systems, **TEST** uses a semantic network of schematic objects, called frames or schemata, to represent its key concepts [Pepper and Kahn 86]. These concepts are typically familiar to the troubleshooting technicians and design engineers who provide the expertise **TEST** is designed to model. Most critical is the **failure-mode**. A failure-mode represents a deviation of the unit under test from its standard of correct performance. Failure-modes are organized in a causal hierarchy. At the top of the hierarchy are observable problems, e.g., raster display problems, as shown in figure 2-1.[2] Each node is linked to a disjunctive set of its possible causes via a *due-to* link. At the bottom of the hierarchy, as shown, are failure-modes of individual components, e.g. the particular power supplies (R502, X501, R505), or the picture tube. Intermediate failure-modes typically represent functional failures which are causal consequences of component failures, e.g. "Hum in LV power supply", or classes of failures, e.g., "LV Power Supply Problem". Many levels of intermediate failure-modes are common. Typical **TEST** networks have 4 to 10 levels of concerns, though much deeper networks occur on occasion. Networks can be used to represent both taxonomic hierarchies, as in CSRL [Bylander *et al.* 83], and causal paths, as in CASTER [Thompson and Clancey 86].

Other conceptual objects within **TEST** include **questions, tests, test-procedures, repair-procedures, rules, decision-nodes**, and **parts**. Each of these concepts has an obvious mapping into the troubleshooting domain. Questions are simply queries to users which result in factual responses. Tests represent manual or sensor-based tests. Test-procedures describe sequences of tests, each of which must be carried out before the diagnostic significance of the overall procedure can be evaluated. Repair-procedures describe corrective actions to pursue upon determining the occurrence of a failure-mode. Rules represent a variety of contingent actions rather than evidence/belief propositions alone, as is typical in Emycin-like diagnostic systems. **TEST**'s use of rules is described in section 4. Parts provide descriptors of parts that are associated with component failures or with repairs.

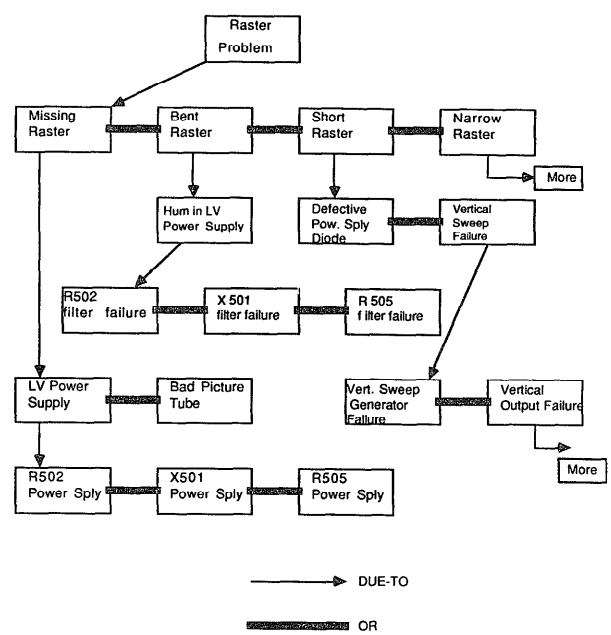

Figure 2-1: A failure-mode hierarchy

[2]The example used here is based on television troubleshooting as described by Tinnell [Tinnell 71]. Actual **TEST** knowledge bases are proprietary to Carnegie Group clients.

Decision-nodes provide a mechanism for integrating conventional diagnostic decision logic into the otherwise failure-mode oriented knowledge base. Although **TEST** can generate its own decision logic from the failure-mode knowledge base, domain experts often prefer to provide the decision logic directly. This may be done by building a decision-node network. Each decision-node represents a test together with branches to other tests contingent on the result of the first. Decision-node networks typically terminate with the failure-modes that could cause the problem associated with the network's entry point.

Knowledge base maintenance is facilitated by clustering information around failure-modes (see figure 2-2). Since the failure-mode is the key concept in most troubleshooting tasks, such aggregates provide an easily understood and readily accessible structure. Inspection of a failure-mode provides direct access to associated tests, repairs and documentation, as well as to forward and backward causal links to other failure-modes in the network.

3. The Diagnostic Problem Solver

Domain-specific knowledge bases represent pre-compiled search spaces and serve as input to the problem solver. Given the failure-mode hierarchy and other auxiliary information, the problem solver searches for a diagnostic conclusion, interactively prompting a technician, or sampling sensors and databases as necessary to obtain evidence to proceed with the diagnostic session. The search space can be dynamically altered by rules (see below) sensitive to information acquired during a diagnostic session. Knowledge engineers can also choose the appropriate level of granularity for the representation of causal chains, thus constraining the depth of search required prior to hypothesizing a particular failure. Moreover, the preferred order in which to consider candidate causes may be easily specified.

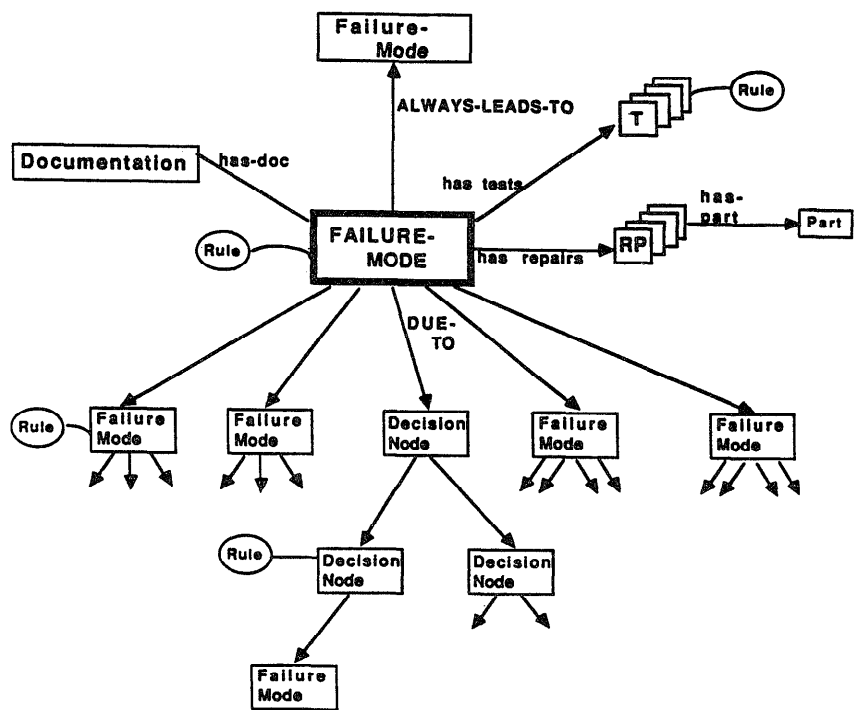

Figure 2-2: A failure-mode aggregate

In general terms, the problem solver pursues a depth-first recursive strategy starting with an observed or determined failure. It seeks the cause of an occurring failure-mode considering candidate causes (other failure-modes) referenced in the *due-to* slot of this failure-mode. Candidate causes can have three states: confirmed, disconfirmed, and unknown. Failure-modes are confirmed when the problem solver determines that they have occurred.

If a candidate cause is disconfirmed, the problem solver moves on to consider another possibility. If a candidate cause is confirmed, the problem solver will consequently seek to determine its causes. This procedure continues until a terminal failure-mode is identified. Terminal failures, those without instantiated due-to slots, are typically repairable faults.

Following the example in figure 2-1, let's assume that a short raster was observed. In this case, the problem solver would first consider "defective power supply diode" as the cause of "short raster". If this were ruled out, it would proceed to consider a "vertical sweep failure". If a vertical sweep failure were to be confirmed, or was unknown, the problem solver would proceed to consider its causes -- "vertical sweep generator failure" and "vertical output failure."

As new failure-modes come up for consideration, the problem solver chooses a method of confirmation provided by the knowledge base developer. It may be a direct test, a rule-based inference procedure, or the disconfirmatory recognition (*modus tollens*) that a necessary consequent of the failure-mode had not occurred. If the failure-mode cannot be confirmed or disconfirmed, the problem solver will nevertheless proceed to examine potential causes. If a failure-mode can have multiple causes, the diagnostic analysis will not terminate until all potential candidate causes are evaluated.

3.1. Additional Features

Apart from the failure-mode hierarchy, the problem solver can also be driven by decision-nodes, and data-gathering activities. The former are used to represent conventional diagnostic decision logic. Decision-nodes represent steps in a conditional sequence of tests which terminate in a decision to rule-out, confirm, or focus on a failure-mode. **TEST**'s ability to integrate test- and failure-mode-driven diagnosis has been critical to knowledge acquisition as both approaches are typically prevalent in the procedures used by technicians and referenced by manuals.

Data-gathering activities are used when tests should be run as a matter of convenience rather than for immediate diagnostic purposes. For instance, if dismantling is required for a particular test, it may be desirable to run other tests that require similar dismantling before reassembly, even though the latter tests are not of immediate relevance.

Additionally, **TEST** allows users to volunteer unsolicited information, as well as to dynamically change the course of the diagnosis. Since troubleshooting systems tend to be highly interactive, it is desirable to take advantage as much as possible of user input, particularly the human ability to notice diagnostically critical information, even though the system may not be asking for it. Moreover, the hunches of experienced technicians can often prove valuable in reducing diagnostic search. Supporting voluntary user-input for both hunches and observations, **TEST** not only makes use of its human partners, but is perceived as being more user friendly and less frustrating to use.

Finally, the problem solver supports a belief maintenance system that is used to provide explanation and an undo facility. The latter provides the ability to selectively modify any prior input. The impact of a modification is propagated through the belief system, possibly resulting in a change of diagnostic focus.

4. Using Rules

The troubleshooting task, like any other, can be characterized in terms of standard procedures and default knowledge which must be altered when special considerations hold. Rules provide the means to dynamically change a knowledge base under specified circumstances. Rules are conditional expressions of the form "IF (condition) THEN (action)." The condition is a boolean combination of (schema, slot, value) triples, each of which which represent a piece of information in the knowledge base. The action specifies a value or change in value for a schema/slot location. Rules may be characterized as *immediate* or *on-focus*. Immediate rules act as demons, firing as soon as their conditions are satisfied. On-focus, or goal-driven, rules are evoked only when the rule is relevant to the current focus of the problem solver.

TEST allows knowledge-base developers to characterize four types of dynamic alterations to a default knowledge bases. **Derivational rules** are used to recognize when failure-modes may be confirmed or ruled-out, as well as to infer factual data. In particular, as new information is collected, the problem solver updates data-driven rules that monitor for co-occurring conditions that would lead to the immediate recognition of a failure-mode.

Causal-modeling rules are used to alter the failure-mode hierarchy. For instance, when the LV power supply is under consideration, and it is known that the television can emit sound, X501 and R505 can be removed from the due-to slot of possible causes, as these power sources disable the audio on failure (See figure 2-1). Conjunctive causes are similarly modelled with causal-modeling rules. That is, a failure-mode could be added to a due-to list only under the condition that another failure-mode has been determined to occur.

Procedural rules are used to modify the several kinds of procedural and methodological knowledge that may be represented in a **TEST** knowledge base. Most critical are the rules used to modify the failure-node and decision-node networks.

Background information acquired during execution may suggest altering the order in which failure-modes are considered. This is typically done to more closely reflect evidential impact on the likelihoods for each failure-mode. A rule would be used, for instance, to indicate that "a bad picture tube" should be considered prior to "LV power supply problem" when the raster is missing and the picture tube is of a series known to be defective. In the context of machine diagnosis, such rules provide a mechanism for ensuring that failures due to part wear are investigated first for older machines, but only after manufacturing parts problems in the case of newer machines.

Procedural rules also facilitate the process of modeling conventional decision logic. Rules overlaid on decision-nodes may alter the transition path to a subsequent decision, or respecify which failure-modes are confirmed or disconfirmed as new information is acquired at each decision-node. Thus, **TEST** permits developers to focus on the default decision logic without worrying about working atypical alternatives into the network. These are easily added as special case rules.

Finally, **relevance rules** may be used to filter the knowledge base by deactivating objects. Rules attached to failure-modes, for instance, can remove the failure-mode from consideration during a diagnosis. This feature is used, for example, in multiple model knowledge bases when the component part associated with a failure-mode is not actually used in the model (or manufacturing run) represented by the unit presenting the fault.

5. Conclusion

TEST provides an effective approach to modeling troubleshooting knowledge. Domain-dependent knowledge bases can be built using concepts familiar to diagnostic technicians and design engineers. Default diagnostic strategies as well as special case heuristics can be easily represented in the knowledge base, with the causal relations that underlie diagnostic reasoning. By structuring the knowledge base around the failure-mode concept, significant modularity and maintainability is achieved. **TEST** offers a unique mixture of schematic and rule-based reasoning.

Unlike most of its predecessors, **TEST** provides mechanisms for readily expressing search behavior, as well as for adapting search to newly acquired information. Because search behavior is determined by heuristic rules, **TEST**'s performance is better than systems which must compute alternative hypotheses on the basis of a causal model.

Several features of **TEST** may carry over well to the design of other application shells. These include the distinction between model and problem solver, as opposed to knowledge base and inference engine. The problem solver knows much more about the task domain and the model assumes much more about the problem solver than the knowledge base/inference engine distinction implies. The problem solver is driven by the model, and as such, preference for various control strategies can be expressed in the model. Secondly, **TEST**'s success as a knowledge engineering tool has depended on the use of domain-familiar concepts. This has enabled knowledge engineers to easily map information from expert sources into the knowledge base; and to explain to their experts the import of the representations used. Finally, a model within which heuristic search constraints may be expressed appears critical to the performance of model-driven systems.

References

[Bylander et al. 83] Bylander, T., Mittal, S., and Chandrasekaran, B. CSRL: A Language for Expert Systems for Diagnosis. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*. 1983.

[DeKleer and Brown 84] DeKleer, J. and Brown, J.S. A Qualitative Physics Based on Confluences. *Artificial Intelligence* , 1984.

[Hofmann et al. 86] Hofmann, M., Caviedes, J., Bourne, J., Beale, G., and Broderson, A. Building Expert Systems for Repair Domains. *Expert Systems* 3(1) , January, 1986.

[Kahn 87] Kahn, G.S. From Application Shell to Knowledge Acquisition System. In *Proceedings of International Joint Conference on Artificial Intelligence*. 1987.

[Kahn and McDermott 86] Kahn, G.S., and McDermott, J. The MUD System. *IEEE Expert* 1(1) , Spring, 1986.

[Patil et al. 81] Patil, R.P., Szolovits, P., and Schwartz, W. Causal Understanding of Patient Illness in Medical Diagnosis. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*. 1981.

[Pepper and Kahn 86] Pepper, J. and Kahn, G. Knowledge Craft: An Environment for Rapid Prototyping of Expert Systems. In *Proceedings of Artificial Intelligence for the Automotive Industry*. SME, 1986.

[Pepper and Kahn 87] Pepper, J. and Kahn, G.S. Repair Strategies in a Diagnostic Expert System. In *Proceedings of International Joint Conference on Artificial Intelligence*. 1987.

[Pepper and Mullins 86] Pepper, J. and Mullins, D. Artificial Intelligence Applied to Audio Systems Diagnosis. *In Proceedings of the International Conference on Transportation Electronics*. 1986.

[Pople 82] Pople, H. Heuristic Methods for Imposing Structure on Ill- structured Porblems. In Szolovits, P. (editor), *Artificial Intelligence in Medicine*, pages 119-190. Westview Press, 1982.

[Shortliffe 76] Shortliffe, E. *Computer-Based Medical Consultation: Mycin*. Elsevier, 1976.

[Thompson and Clancey 86] Thompson, T. and Clancey, W. J. A Qualitative Modelling Shell for Process Diagnosis. *IEEE Software* 3(2) , March, 1986.

[Tinnell 71] Tinnell, R.W. *Television Symtom Diagnosis*. Howard W. Sams & Co., 1971.

[VanMelle et al. 81] Van Melle, W., Scott, A.C., Bennett, J.C., and Peairs, M. A. *The Emycin Manual*. Technical Report, Stanford, 1981.

[Weiss et al. 78] Weiss, S., Kerm, K.B., Kulikowski, C.A., and Amarel, S. A Model-Based Method for Computer-Aided Medical Decision-Making. *Artificial Intelligence* , 1978.