A Development Environment for Field Diagnosis Tools

K.P. Lee, John Martin, Paul Rutter, and Richard Wexelblat
Philips Laboratories, North American Philips Corporation
Briarcliff Manor, New York 10510

Abstract

Field diagnosis of electro-mechanical equipment is a domain that abounds in empirical knowledge that skilled technicians have gained over years of experience. As such it is a fit area for application of expert systems methods. This paper describes a generic architecture for such applications in a system that combines empirical knowledge, a naive device model, an excellent user interface, and a set of tools to assist the knowledge engineer.

Subjects: diagnostic expert systems, field maintenance, knowledge engineering, device modeling

Introduction

The CATS expert system architecture is designed to support field diagnosis and repair of electro-mechanical equipment. The generic structure grew from an attempt to build a specific expert system for servicing equipment manufactured by Philips Medical Systems. "CATS" stands for Components, Actions, Tests, and Symptoms, the four classes of objects represented in the knowledge base. The first use of CATS is aimed at field diagnosis of the Philips Tomoscan 300 series of computed tomography scanners.

Diagnosis has always been important for manufacturers, users, and maintainers of equipment. Although recent designs provide for a significant amount of embedded self diagnosis, older equipment is less facile. In the Tomoscan, the control computer has built-in tests but these do not extend to the X-ray and mechanical subsystems. Further, this limited self test ability cannot be interfaced with an external diagnosis computer without unacceptably expensive reengineering in an old product line. Thus, the present CATS architecture is aimed at assisting the human field technician, using this technician as mediator between the expert system and the device tested. The intended user is a field engineer, experienced in servicing CT scanners, but not necessarily familiar with the Tomoscan. To this end, significant attention has been paid to the user interface - an aspect of performance that can in itself determine success or failure in the field.

The relationship between CATS and other diagnostic systems is discussed in the Conclusions section.

Project and System Overview

The current production version of CATS primarily implements a decision tree with certain extensions based on a device model of the equipment under repair. This model must be built anew for each device CATS is applied to, however the knowledge representation has primitives for such structural features as power distribution. The frame-based representation has, for example, a slot named power-depends.on.

CATS uses a commercial expert system shell, Intellicorp's KEE. The public tends to think of shells primarily as supporting rule-based expert systems. With this in mind, one could say that CATS is in but not of KEE. It uses the object-oriented programming environment of the host but not its rule-based inference scheme. Our first cut at diagnosis was a "traditional" rule-based system aimed at the Tomoscan 300. Early experience with the knowledge engineering led to an interesting discovery: Although the domain expert, an experienced field service engineer, was able to express his knowledge as rules, he did not think or reason in such terms. When the time came to extend the first rapid prototype, it became clear that the expert thought in terms of a naive domain model - indeed, a set of functional models relating to different aspects of the system under test: power distribution, mini-computer front panel, X-ray beam focus, etc. His internal models resembled a semantic net whose representation mapped very nicely with frame-based object-oriented programming. Thus, the second version of the system uses this paradigm.

A goal of the CATS interface design is "zero learning time." Given a potential user already familiar with a
mouse and with window-oriented displays, productive use of the system should begin immediately. As can be seen in the screen images, communication between user and system is through displays on a bitmapped multi-window screen and a mouse. More details on the user interface appear elsewhere.[4]

Knowledge Representation

As noted, the CATS knowledge base is composed of Components, Actions, Tests, and Symptoms. Figure 1 shows examples of each of these objects. An interpreter, described below, uses these objects to direct a diagnosis and repair session.

The knowledge representation is built on a device model that is a collection of components and their interrelationships. Each component is implemented as a frame with named attributes and values. They are arranged in a hierarchy and subsystems selectively inherit characteristics from multiple parents. Each component has a status attribute that is updated as diagnosis proceeds and additional attributes track details of the repair history. Components are additionally related by power.depends.on and functionally.depends.on attributes. During diagnosis, status of components can often be inferred from the status of related ones.

Malfunctions of the system under test are known as symptoms. A diagnosis session always begins with the user identifying an observable symptom. Other symptoms may be added to the list as diagnosis proceeds.

Associated with each symptom is a checklist of tests used to determine the causes of a malfunction. Tests may uncover additional symptoms or invoke actions that call for additional tests.

Actions, such as asking the user to repair or replace a component, are invoked as the result of running tests. Actions may invoke "post-action" tests that check the success of the action. Both tests and actions may update the status of the device model, providing an empirical non-monotonic truth maintenance function.

System Architecture and Interpreter

The interface architecture performs the main user interaction in a window at the upper right of the bitmapped screen. The left half-screen contains diagrams; the lower right contains information windows and, when appropriate, instructions for the user. At the lower right, frequently overlaid, is a window containing an event transcript.

At the beginning of an interaction, a user is presented with a list of high-level observable symptoms to choose from. Once the domain of the failure is identified, screens become more detailed. In Figure 1, the user has already noted inability to boot the internal minicomputer from hard or floppy disk and the system has asked him to run the microdiagnostics. The nested set of windows
on the left is a physical map of the backplane and power section of the minicomputer. Initially all components are UNTESTED. Later they may become OK, BAD, NEW.OK. If the user chooses the ALL or SOME result from the microdiagnostics test, several of the power section components will be updated to OK as the test could not have gotten that far without power. (Note the power.depends.on slot in the component frame in Figure 2.)

(component DISK.CONTROLLER.12 16
(status UNTESTED)
(power.depends.on EXPANSION.CHASSIS.POWER.SUPPLY))

(symptom NO.IPL.FROM.ANY.DEVICE
(checklist MICRO.DIAGNOSTICS IPL12 16)
(type TOP.LEVEL))

(test MICRO.DIAGNOSTICS
(question 'When you run micro diagnostics, how many switches work?')
(answers (ALL (SET.STATUS SWITCHES.AND.BULBS OK))
(Return NOT THE PROBLEM)
(SOME (SET.STATUS SWITCHES.AND.BULBS BAD)
(ACTION FIX SWITCHES.AND.BULBS))
(NONE (ADD SYMPTOM BAD CPU SUSPECTED)))

(components.tested (SWITCHES.AND.BULBS))

(how ('Turn key to test position, flip each switch, checking for corresponding light'
(bitmap P857.FRONT.PANEL.BITMAP))

(why ('To test CPU switches and bulbs')))

(action REPAIR VDUX.POWER.SUPPLY
(test 'Replace the defective VDUX power supply'
(components.affected VDUX.POWER.SUPPLY)
(post.action.tests (IPL FLOPPY))))

Figure 2: Knowledge Representations

Figure 3 is a simplified schematic view of the the CATS interpretation process. The top line of the figure is meant to indicate that an initial symptom gets the interaction started by identifying a checklist of tests to perform in sequential order. (For example, the symptom NO.IPL.FROM.ANY.DEVICE in Figure 2 identifies a checklist of two items, MICRO.DIAGNOSTICS and IPL12 16. Figure 1 shows the screen associated with the MICRO.DIAGNOSTICS test.)

The lower section of Figure 3 gives more detail of interpretation. Within each test, if all components have the status OK or NEW OK the test is omitted (i.e. the component has been previously tested and found correct); otherwise it is performed. Afterwards, the system status is updated and an appropriate result code returned. The test may lead to uncovering other symptoms. It may also specify actions or tests to perform. When other symptoms are found, they are processed immediately in a recursive fashion. In performing an action, it is necessary to avoid a loop such as

Symptom: fuse is blown
Action: replace fuse
Symptom: fuse is blown
Action: replace fuse

... so the user is warned when a previously repaired component is found faulty.

CATS INTERPRETER

![CATS Interpreter Diagram]

If all components under test are OK, skip this test.
Present question -- get user answer.
Update STATUS and process answer items:

```
{ Symptom  Action  Test  }
(Warn if component repaired previously)
Request user action.
Process post actions: Test  ... 
```

Figure 3: The CATS Interpreter

Explanation facilities in CATS are strictly oriented toward the empirical knowledge base. For example, when the testing process is involved or might be unfamiliar to the user, "how" information is automatically displayed as shown in Figure 4 (The action in this case is replacing an I/O control unit known as the DIOCU VDUX). It can be seen in this figure that several components in the model have an OK status. The DIOCU VDUX, however is still UNTESTED. It is not yet known whether this component is good or not. However the knowledge base suggests replacing it as part of an attempt to isolate the reason for the computer's refusing to load. At the end of the repair, the DIOCU VDUX will gain the status NEW OK and the
Figure 4: Instructions on Performing a Test

Figure 5: The Knowledge-base Browser
The process continues as long as there are actions or tests to perform. An empty symptom list defines the conclusion of a successful diagnosis/repair session. On the other hand, if, when the actions and tests are exhausted, symptoms remain, the session is unsuccessfully completed and the user is referred to a (human) specialist.

**Knowledge Engineering Interface**

Particular care has been paid in the design of CATS to two types of end user: the field engineer and the knowledge engineer. Transfer cannot succeed if the receiving organization cannot maintain, extend, and retarget the system. It is unrealistic to expect the field service organization to have AI-trained personnel—or to go out and hire them on the basis of not-yet-proven technology. Therefore, we are providing a basic toolkit that will allow knowledge engineering in domain-oriented terms. Space does not permit full description of the knowledge engineering interface. Here are a few highlights. The first knowledge engineers will have to be trained in knowledge engineering, in the knowledge representation, and in the workings of the CATS interpreter. They will not have to dig into the underlying code and frame structure of KEE. In fact the tools we have implemented display only domain-related aspects of the knowledge structures and their interrelationships.

The two basic tools we provide are domain browsers, one for objects and one for the interrelations of the knowledge base itself. Figure 5 shows the relationship between the test MICRO.DIAGNOSTICS and its neighbor objects. The user may invoke an object browser, Figure 6 to inspect or modify (via the system editor) the slots of the object. Items appearing in a box with a broad border are local to the current frame and can be edited directly. Those within a narrow border are inherited or otherwise computed and can be changed only by tracing them back to the source.

The basic testing and debugging mechanism is the ability to record and play a session back. At any point, a knowledgeable user can escape to debugging mode to enter the browser or to review the interaction. It is possible to back up a number of steps or to the beginning of the interaction and then to replay the session or branch to an alternative path. When replaying the interaction, the session can run by itself at various speeds or proceed one step at a time. The user can also follow all mouse motions or just go on a screen-by-screen basis. This mechanism has made it easy to do an exhaustive test of all branches of sections of the knowledge base.

![Figure 6: The Object Browser](image-url)
Conclusion

CATS has been in development for about a year. In its object-oriented structure and device model representation of knowledge, it is a significant step beyond the first generation rule-based diagnosis system. The effort fits well into current trends of AI-based diagnosis work. For example, the diagnosis methods resemble in some respects those used by the TEST project[2]. In Milne[3], four levels of diagnostic knowledge representation are described. Using his classification, the CATS architecture can be regarded as one that uses mainly compiled knowledge assisted by a simple functional model. The integrated diagnostic model of Fink[1] combines the knowledge of how a domain expert diagnoses the equipment with a structural view of the equipment. When a diagnostic session exhausts the experience of the expert, a reasoning component based on the connectivity of the components can attempt to explore further.

Avoiding an unstructured rule base was an explicit design decision in CATS. Power was traded for ease in understanding and knowledge engineering. The simple connectivity and functional model of CATS has made it easy to elicit diagnostic knowledge from the domain experts. On-screen display of the status of the device model has been helpful in visualizing the progress of a diagnostic session and in checking for omissions and errors in the knowledge base.

The CATS effort has been an experiment in expert system design and in knowledge engineering methods. Based on acceptance by the target organization, the experiment has been successful so far. The use of KEE frames and Lisp to implement an object-oriented programming environment specific to diagnosis and repair has proven to be a good choice. The interactive nature of the knowledge acquisition process will be crucial to building expert systems of a useful size in a reasonable amount of time.

Acknowledgments

The initial Tomoscan diagnosis prototype was implemented by David Schaffer with the assistance of John Martin. Robert Purdy of Philips Medical Systems, Inc. (Shelton CT) and Eric Timmer of Philips Medical Systems Division (Best, Netherlands) were the domain experts whose experience was mined for the knowledge base. The authors are grateful to reviewers of early drafts for their helpful criticisms.

References


