Intelligent Diagnostic Tutoring Using Qualitative Symptom Information

Douglas M. Towne

University of Southern California
Behavioral Technology Laboratories

1120 Pope St., Suite 201C
St. Helena, CA 94574
(707) 963-3060
dtowne@usc.edu

Abstract

An intelligent diagnostic tutor, DIAG[1], selects problems adaptively and generates all tutoring dialogs from its analysis of a model of the target system. The learner performs tests on a graphical representation of the target system and calls on DIAG for assistance when needed. During exercises, the tutoring functions can advise the learner about the implications of particular test outcomes, the rationality of suspecting particular replaceable units, and advisability of performing various diagnostic actions. After exercises, DIAG can critique the learner's testing strategy, and it can generate and explain an expert diagnostic strategy for the previous fault. A prototype application of a very complex system demonstrates the range of tutoring capabilities achieved by the system.

Overview

Fault diagnosis is one of the most ubiquitous and difficult tasks performed in everyday life, and it is an unavoidable component of a wide range of medical, commercial, industrial, and military operations. From the broadest viewpoint, the diagnostic process is surprisingly equivalent across domains. While the particular costs, risks, and payoffs of testing actions vary greatly over different applications, the diagnostic process can be characterized in a manner that takes these variables into account. Consequently, excellent diagnostic strategies can be artificially generated in highly specific domains, based upon a deep generalized conception of the diagnostic reasoning task.

It would seem, therefore, that this class of human performance would be a natural and highly amenable subject area for intelligent tutoring approaches. In fact, a number of intelligent diagnostic systems have been produced, and several have demonstrated excellent instructional power. Those approaches based upon expert systems (Lesgold, Eggan, Katz, and Rao 1992) are capable of generating discussions of impressive verbal and technical content. Unfortunately, this instructional depth and facility has come at a very high expense in capturing and structuring the expert knowledge.

At least two other methodologies have been explored to combat this discouraging cost-to-power relationship: 1) a *structural model* approach (Johnson, Norton, Duncan, and Hunt 1988), in which the target system is specified in terms of its normal input-output structure, and 2) a *functional model* approach (Towne, Munro, Pizzini, Surmon, and Wogulis 1990), in which the target system is modeled in a manner that it can be executed in various normal and abnormal conditions.

The structural model approach specifies the target system as a network of system elements, connected via directed paths, and it then reasons about symptoms by assuming that failures will propagate along the directed paths that connect the system elements. Unfortunately, these assumptions are commonly violated by faults in real systems, and the developer is then burdened with devising a representation of the target system that will somehow yield valid fault effects under the many possible aberrations that faults produce. This approach appears to function well in instructing normal operation, however.

The functional model approach specifies the target system as a set of system elements each characterized in terms of its normal behavior and some of its abnormal behaviors. When a specific fault is introduced into the system model, its effects can be correctly determined under a wide range of configurations and modes. The weakness of this approach is a consequence of two facts, one of which is obvious and one of which is devious:

[1] This work was funded by the Office of Naval Research under Contract No. F33615-90-C-0001.

1. the effort required to specify the abnormal behaviors of the system elements increases with the number of abnormalities to be embraced by the model (obvious); and
2. achieving intelligent and effective tutoring demands that the consequences of a very large number of faults be available for consideration (devious).

We know from experience and from analytical studies that a very modest number of faults can support highly effective instruction in fault diagnosis. If this set of failures— the *training set*—is carefully constituted, then a relatively wide range of fault conditions can be covered with a surprisingly modest number of fault cases. However, during the instruction of any of the faults in the training set, the tutoring system must be able to reason about the fault possibilities, i.e., what failures could produce the symptoms seen. If the scope of this reasoning is limited to the faults in the training set, the training system is blind to the true complexity of the target system, and this will substantially distort the diagnostic task from that which the learner will confront in the field.

DIAG

This paper describes DIAG (Towne 1996), a system conceived with the objective of generating all tutoring interactions automatically, based solely upon a functional model of the target system and symptom information that a knowledgeable technician could readily supply. DIAG (Diagnostic Instruction and Guidance) resembles its direct ancestor, IMTS, (Towne, Munro, Pizzini, Surmon, and Wogulis, *ibid;* Towne and Munro 1988) in that it employs a functional model as the instructional vehicle, i.e., there is an underlying representation of the target system that models its functionality and maintains a graphical model of the system as the learner operates upon it.

Whereas IMTS was a single system combining model building tools along with diagnostic reasoning functions, DIAG is a separate diagnostic tutoring system that operates upon functional models developed with the RIDES[2] (Munro, Johnson, Surmon, and Wogulis 1993; Towne 1994) system. While producing an interactive functional model of a complex system is a specialized skill, the great bulk of the effort is devoted to producing a normally functioning model. Once this is completed, each of the faults to be presented in exercises is then modeled by enhancing the specification of the affected system element, to include the impacts of that fault on the behavior of the host element.

To overcome the limitations of the functional model approach, however, DIAG required some means for capturing and retaining much more extensive symptom knowledge than could be drawn from the functional model, with its limited set of faults. Furthermore, this data needed to be in a highly compressed and even imprecise form, to be manageable. The methodology employs qualitative descriptors that express the strengths of functional relationships between particular system elements and particular test outcomes at system indicators[3]. This methodology was adapted from that previously termed *fuzzy logic* and now termed *Zadehan logic* (Zadeh and Kacprzyk 1992).

During development, the applicator may command DIAG to automatically generate all the symptom information for the faults that have been simulated in the training set. This information may then be expanded and enhanced by informing DIAG about other fault conditions of which it is not aware via the system model.

During instruction, DIAG has control over the graphical device model. It can introduce failures into that model, remove failures when the learner makes the correct replacement, and maintain the depicted state of the entire representation. In addition, it can set the device model into modes or configurations that are instructive, and point out significant indications.

By combining a working, interactive device model with a data base of qualitative symptom possibilities, DIAG has the abilities to:

- select problems, based upon the characters of the faults available in the training set;
- reason about the implications of the symptoms that the learner's tests produce;
- generate verbal advice in response to requests for guidance;
- produce a step-by-step critique of the individual's performance;
- demonstrate and explain an expert approach to a completed problem.

The tutoring functions rely heavily upon 1) complete and precise fault effect information for the relatively small number of faults in the training set, and 2) incomplete and qualitative symptom information about all other possible faults. The latter information is manageable because it does not explicitly identify what faults could produce what abnormalities.

---

[2] RIDES was developed under Air Force funding, Contract No. F33615-90-C-0001.

[3] The term *indicator* will be used generically throughout this paper, signifying any system element that reflects system status, e.g., lights, meters, test points, sounds, smells, and even vibrations.

DIAG has been used to produce one very large prototype application, and is currently in use to produce a section of a basic electronics course. The prototype application, a shipboard radar transmitter, involved 17 separate screens modeling the system, 86 replaceable (failable) units, 49 active indicators, 17 test points, and 31 active controls. The most significant outcomes of this prototype application process were these:

1. The hierarchical model of the target system proceeded in a top-down fashion, working from existing technical documentation. This permitted a simplified version of the system model to be operational in a very brief time, and to be expanded until the desired level of detail was produced.
2. The technician[4] providing the qualitative symptom information began work after less than an hour of explanation of the required form and content. He accomplished this task independently and without error in less than two calendar weeks.

The top-down development process is an essential feature. The more common bottom-up approach requires that the most detailed level of the application be completed before any tutoring functions become operational. Under the top-down approach, one can rapidly produce a tutoring system for diagnosing major units of a system, and later extend the level of detail wherever desired. It is hoped that this will encourage and allow instructional developers to introduce fault diagnosis early in instruction of system operation, and to continue with such instruction throughout training.

Application Development

A DIAG application is produced in five steps, all of which are typically revisited as the application is refined:

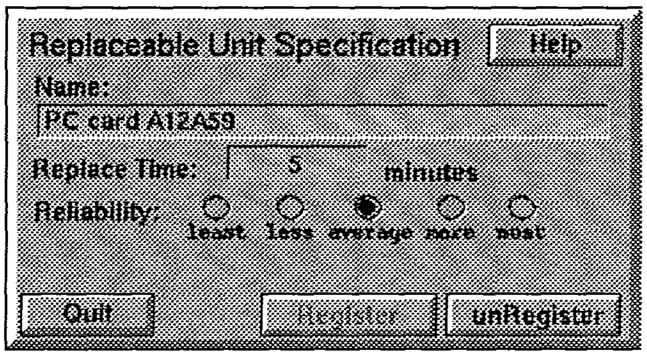
1. produce a working model of the target system;
2. specify the modes of operation in which instruction will be provided;
3. define the fault set;
4. provide broad, system-wide, symptom information;
5. specify parameters governing exercise presentation.

Produce a Working model

Using the tools provided within RIDES, the developer creates graphical objects representing the physical elements of the target system. Each object is identified as one of these DIAG object classes:

- *system block*, representing an equipment, a module, a circuit board or any other physical unit of the target system. A system block may be decomposed without restriction into more detailed blocks.
- *control*, which the learner manipulates to operate the modeled system.
- *indicator* or *test point*, which provides information about the modeled system;
- *test equipment*, which displays values at test points; and
- *replaceable unit*, which the learner may replace with a known good spare.

As each object is created, a dialog box appears that accepts the object name and other parameters relative to that object type. The following dialog box is presented when a replaceable unit is identified.


Replaceable Unit Specification

Name: PC card A12A59

Replace Time: 5 minutes

Reliability: least less [✓ average] more most

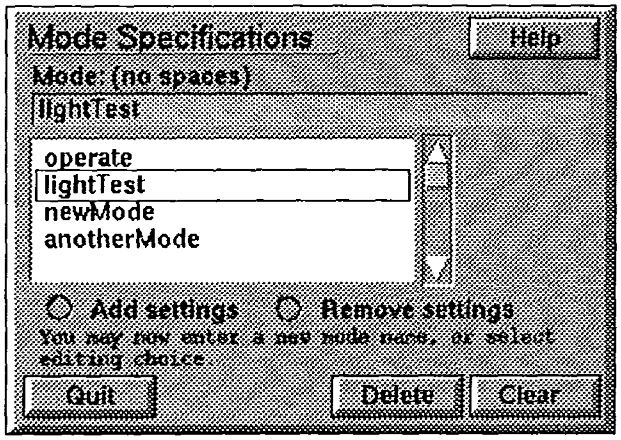
At the conclusion of this major phase, a user can navigate through the hierarchical system model using DIAG's built-in navigation buttons and can operate and test the system model (all symptoms will be normal at this phase since there are no faults defined).

The developer can also construct a functional hierarchy of the target system and link the physical views to the most equivalent functional views. This equivalency permits the learner to practice operating upon the physical model while considering the internal functional architecture of the target system.

Specify the modes of operation

The developer specifies each mode in which tutoring can be conducted by setting the controls for that mode, via the graphical representation of the system, and then entering a name for the mode to the following dialog box.

[4] Petty Officer James Armstrong, AEGIS Training Center, Dahlgren, VA.



Mode: (no spaces)
lightTest

operate
lightTest
newMode
anotherMode

○ Add settings ○ Remove settings
You may now enter a new mode name, or select editing choice

DIAG can then recognize what mode the learner has established at any time, and it can set the system model into particular modes at the initiation of exercises and during tutoring.

Establish the Faults in the Training Set

For each fault in the training set the technical expert provides an *executable specification* and a *verbal explanation.* The executable specification provides the abnormal effects of the fault, expressed in a form that the fault can be simulated when it is the subject of an exercise. The verbal explanation of a fault is presented to the learner at the conclusion of an exercise involving that fault. This recap may summarize the most significant symptoms, and it may explain how the fault effects propagated through the system and affected other operational functions. The intent of this explanation is to allow the learner to fully understand the effects of the fault just experienced.

Provide Broad Symptom Knowledge

The previous step provides complete and precise symptom knowledge about the faults to be presented in exercises, thereby supporting simulation of those faults and the generation of discussions about their impacts upon the target system. To support intelligent fault isolation reasoning DIAG must also have access to a broad bank of symptom knowledge which expresses the effects of other faults, ideally all other faults. In essence, DIAG must be able to answer two types of questions during its reasoning:

1. Can a fault in <some replaceable unit> produce <some abnormal symptom>?

and

2. Could <some normal symptom> be obtained if <some replaceable unit> were faulty?

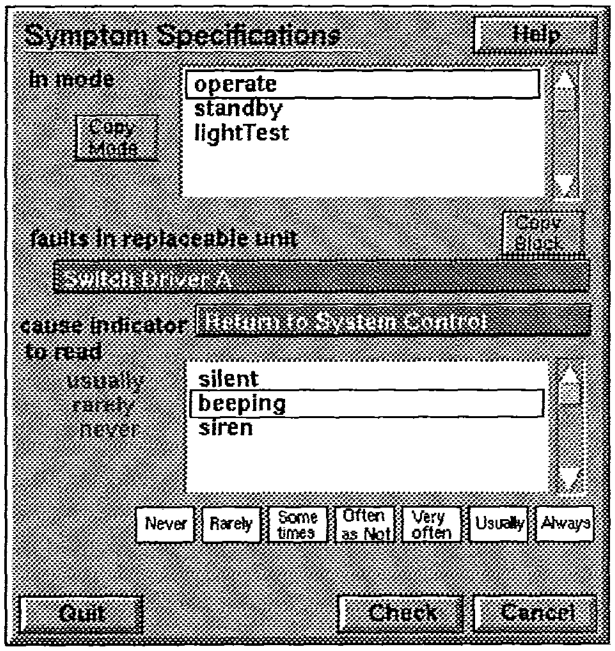
These two issues are central to all of diagnostic reasoning, from the drawing of inferences of observed symptoms to the decision concerning the next test to perform. The knowledge base required to answer these questions need not be at all precise, and fortunately it also need not identify which faults within a unit might produce which abnormalities.

DIAG provides two alternative or complementary ways to supply this knowledge:

1. Automatic generation. The developer clicks a **Generate** button which causes DIAG to insert each defined fault from the training set into the device model, in each system mode, recording the abnormal symptoms produced at each indicator. The result is a data set that expresses the frequency with which various symptoms result from faults in each replaceable unit.

and/or

2. Manual input of qualitative symptom descriptors. Under this approach the technical expert employs the following dialog box to provide qualitative judgments about the frequency with which each replaceable unit could produce various symptoms, when faulty.



in mode
operate
standby
lightTest

faults in replaceable unit
Switch Driver A

cause indicator to read
Return to System Control

usually silent
rarely beeping
never siren

The technical expert first selects the mode, the faulty unit, and the indicator to be specified, then selects one of the seven likelihood descriptors (Never, Rarely, ..., Always) for each symptom. The dialog box shown above is reflecting the following assertion (about an indicator that happens to be audio):

In **Operate** mode, faults in the **Switch Driver A** unit:
usually cause the Return to System Control indicator to be silent;
rarely cause it to be **beeping**; and
never cause it to be **siren**.

The automatic generation approach may produce sufficient broad symptom information about a target system *if the training set is large compared to the number of possible faults.* This might be true for a relatively simple system, or one which is being instructed to a relatively superficial degree. For most applications, however, the developer will need to supply additional symptom information, via the manual approach.

The labor involved to provide symptom information manually is reduced by using the automated approach first, then refining the symptom possibilities manually via the dialog box. Effort is further reduced by the fact that *only abnormal relationships need be given.* If one were developing an application for a personal computer, for example, there would be no need to specify relationships between the floppy disk drive, a replaceable unit according to most maintenance policies, and the mouse cursor (an indicator), since faults in the former could not affect the latter, at least according to this writer's understanding.

Specify course parameters

The developer shapes a course by providing the following in a simple text file:

- the problem selection scheme (fixed order, random order, or adaptively according to problem difficulty and individual proficiency);
- the maximum time allotted to the course; and
- a set of candidate problems, each expressed as a fault and a problem statement..

Because a DIAG *problem*, or exercise, consists of a particular fault coupled with a particular problem statement, any fault can be used in multiple exercises under different problem statements. This gives the developer some control over the difficulty of the problems presented. For example, a class of highly experienced students might be given a fault with the problem statement

```
There seems to be something wrong with the Transmitter.
```

while a class of new students could be given the same fault with the more focused problem statement

```
There is a rectification failure somewhere in the High Voltage
power supply.
```

Such a problem statement allows a novice to engage in fault diagnosis much earlier in instruction, since it restricts the search area to an one that has been addressed in the course. Alternatively, the same fault could be presented with the statement:

```
We're getting a zero reading at the Power meter.
```

Finally, the problem statement could be purposely vague, incomplete, or even incorrect, an extremely common condition in the field that is rarely addressed in the school environment.

At the conclusion of this development process, we have a working system model into which faults can be automatically inserted; tests can be conducted, either by observing indicators directly or employing simulated test equipment; and system units can be replaced. Unseen to the user, DIAG records the tests performed and the symptoms displayed. From this it generates instructional advice during and following each exercise, as requested by the learner. The next section outlines these instructional and consultation functions.

Instruction and Guidance

DIAG performs a number of general instructional management functions as well as highly domain-specific advisement functions. This section deals first with management functions.

Adaptive problem selection

When a learner starts a session, DIAG determines the next problem to present to that individual, depending upon previous progress, if any, and the course plan provided. For adaptive problem selection the first problem is selected at random, then for all ensuing problems DIAG searches through the problem set for that one that presents the most appropriate level of difficulty, considering the individual's success and progress. Higher success leads to increased difficulty, lower success leads to reduced difficulty.

The difficulty presented by a particular fault is computed as the extent to which that fault corresponds to its broad, or archetypal, symptom complex, as expressed via the qualitative descriptors. For example, suppose the content expert has indicated that faults in a floppy disk drive can affect three indicators, as follows:

Indicator	Symptom	Likelihood
1	ON	Rarely
	OFF	Usually
2	0	Often as not
	1-10	Sometimes
	11-20	Sometimes
	>20	Rarely
3	Low	Often as not
	Medium	Often as not
	High	Never

Suppose further that the particular fault being simulated is in the floppy disk drive, and that it produces the following symptoms:

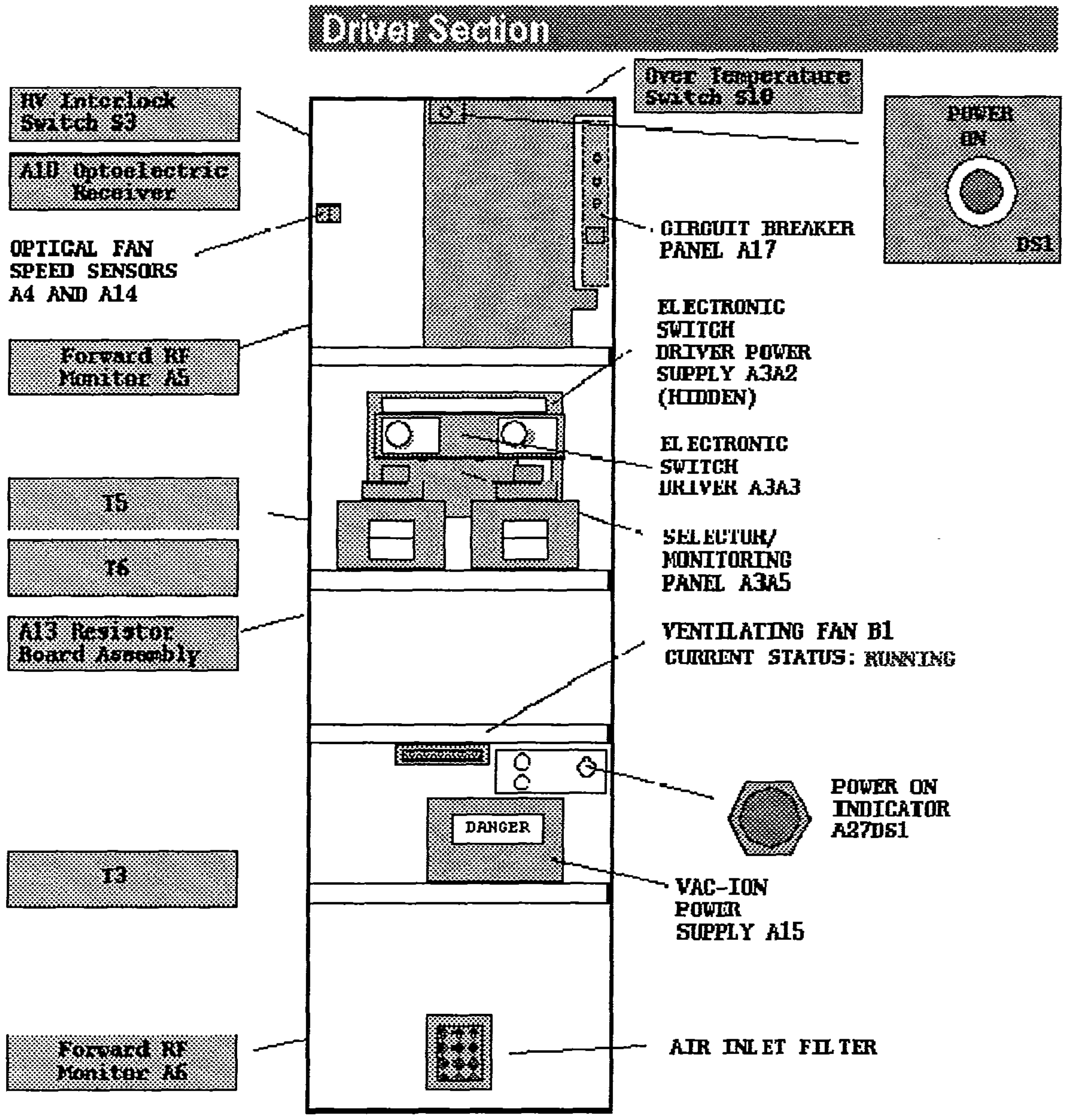
Indicator	Symptom
1	ON
2	25
3	Low

This fault will yield a very high difficulty measure since two of the symptoms it produces—indicator 1 is ON and indicator 2 reads above 20—are not typical of faults in the floppy drive. Alternatively, the following floppy drive fault would be much easier to identify:

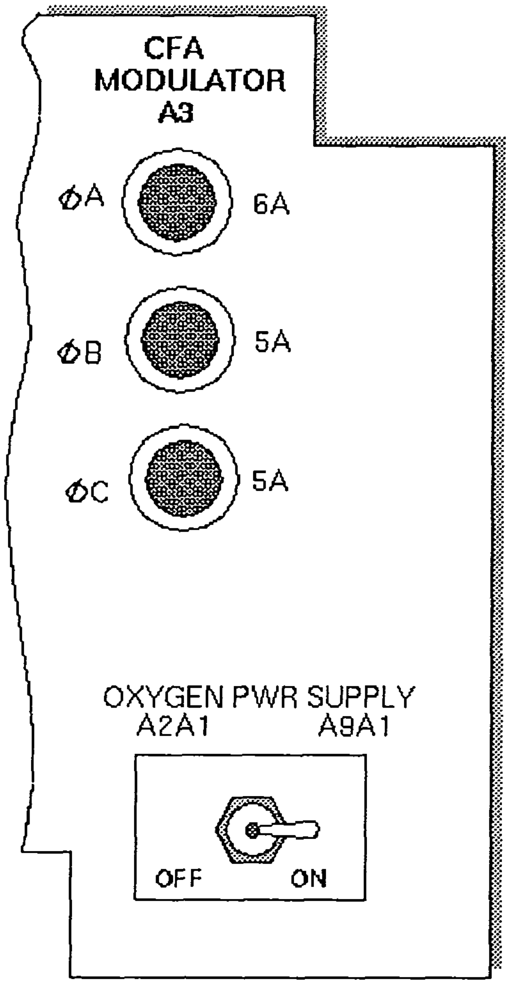
Indicator	Symptom
1	OFF
2	10
3	Low

Performance Tracking

After the problem report is displayed at the start of an exercise, the learner sees the top level of the system representation. For the prototype application the top view is a simple diagram of major equipment units, one of which is a Driver unit. A selection of the Driver unit produces the following display.


Driver Section
HV Interlock Switch S3
A10 Optoelectric Receiver
OPTICAL FAN SPEED SENSORS A4 AND A14
Forward RF Monitor A5
T5
T6
A13 Resistor Board Assembly
T3
Forward RF Monitor A6
Over Temperature Switch S10
POWER ON
DS1
CIRCUIT BREAKER PANEL A17
ELECTRONIC SWITCH DRIVER POWER SUPPLY A3A2 (HIDDEN)
ELECTRONIC SWITCH DRIVER A3A3
SELECTOR/ MONITORING PANEL A3A5
VENTILATING FAN B1
CURRENT STATUS: RUNNING
POWER ON INDICATOR A27DS1
DANGER
VAC-ION POWER SUPPLY A15
AIR INLET FILTER

This view includes some observable indicators, some replaceable units, and two modules for which there is more detail. Selecting the Circuit Breaker Panel, A17, in the upper right corner, produces this more detailed view:



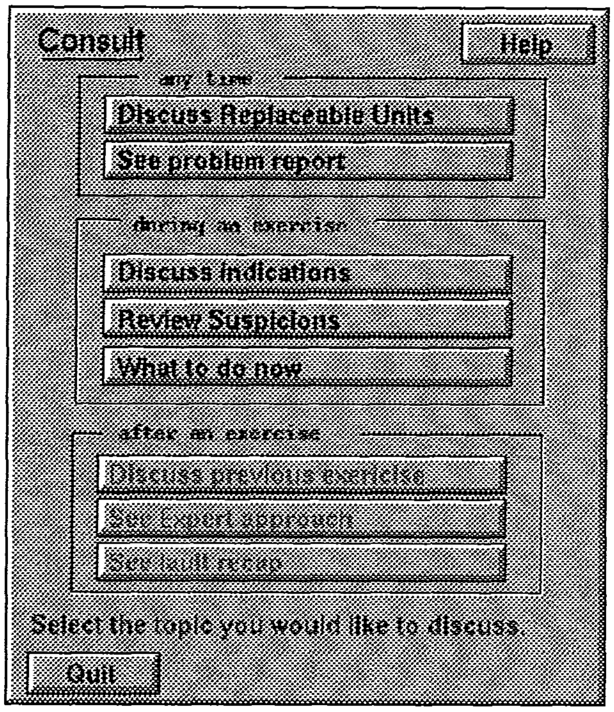
This is the most detailed view of this branch of the hierarchy, a view that happens to provide a control that the learner can operate. From here the learner may move upward and explore other sections of the system. As learners navigate through the system representation they see the indications, both normal and abnormal, corresponding to the current fault condition. The learner may change the controls at any time and may select test equipment to probe any of the test points. As the learner works, DIAG maintains a record of the symptoms that were displayed and the modes under which those observations were made. At any time, the learner may:

- explore the functional view of the system;
- replace any of the system units;
- call on DIAG for guidance;
- claim that the modeled system is now operational, or
- . give up on the current exercise;

The next section will describe the diagnostic guidance DIAG provides when requested.

Diagnostic Guidance

When the learner clicks on the **Consult** button, the following dialog box appears:



Select the topic you would like to discuss.

The **Help** button provides guidance in using DIAG's consultation features. The remaining buttons provide the following types of consultations:

Discuss Replaceable Units. The learner is prompted to select a unit of interest, then DIAG generates an analysis of the symptoms that have been observed, relative to the unit. The following is an example (the learner has selected PC card A12A4):

```
PC card A12A4 is one of the stronger suspects,
  however some indications you have seen contradict that
theory.
Here are some of them:
Vent Fan B2 sound was NOT_RUNNING in Radiate mode.
  This is an abnormal symptom (normal is RUNNING)
  which never results when this unit fails.
```

Here, the the selected unit was still high in the suspicion rankings, but one symptom conflicted with the theory that PC card A12A4 was faulty. If more disproving indications had been seen, DIAG would list several of them, and it would indicate that the unit is not likely failed. If the symptoms were consistent with a failure in the unit under discussion, DIAG lists the indications that most strongly point to that unit.

See Problem Report. DIAG displays the problem report again, for review.

Discuss Indication. The learner is prompted to select an indicator of interest, then DIAG discusses the implications of that indicator's *current* status relative to the units that *should be* most suspected considering the symptoms that have been seen. Here the learner has selected the PC Card Interlock Indicator.

```
The PC Card Interlock Indicator is off which is
  normal in this Radiate mode.
Ventilating Fan B2 has no affect on this test.
Optical Fan Sensors A3 and A14 has no affect on this test.
T3 has no affect on this test.
PC card A12A47 has no affect on this test.
PC card A12A36 sometimes
  allows this normal indication even when it fails.
```

The first sentence simply identifies the state of the indicator and whether that is a normal state in the current mode. Then, the current indication is discussed in relation to the units that should be most suspected at this stage of the problem. In this example, the indication was normal, which happens not to provide significant new information. If the learner thought that this normal indication confirms that T3 is functioning correctly, this consultation would correct that misconception.

At times, this discussion may reveal more to the learner than we might wish. In a future version of DIAG, the learner may also be asked to indicate the units of interest, so that true suspicion levels are not revealed by this consultation.

Review Suspicions. When prompted, the learner selects those units that he or she most suspects. DIAG attempts to comment on those suspicions without revealing too much, as shown here.

```
The symptoms you have seen so far do not justify your
suspicions of 3 of the units you indicated.
These symptoms implicate 6 other units that you should also
suspect.
```

While the discussion purposely avoids mentioning particular replaceable units, it is based upon DIAG's ongoing maintenance of suspicion levels for the individual replaceable units. This process initially ranks the replaceable units according to their relative reliabilities (least reliable units are ranked highest) then moves units down in the ranking when symptom information is shown that conflicts with the hypothesis that the unit has failed. The amount by which a unit is moved down in the ranking depends upon the degree of the conflict, as expressed in the qualitative symptom descriptors. Thus, if failures in a particular unit could never produce the symptom seen, its suspicion is reduced greatly.

What to Do Now. If a learner reaches an impasse, or simply wonders what an expert would do at the current point in an exercise, he or she may call for this consultation type. First, DIAG reviews the learner's suspicions, as outlined above. If the learner's suspicions are seriously flawed, then the remediation provided by the suspicion review may resolve the impasse. If not, DIAG will suggest a good test to make next. The analysis yields that indicator and mode most likely to provide new useful information, essentially the measure used in IMTS (Towne and Johnson 1987) but now working with qualitative data.

```
A good action now is to check Vent Fan B2 sound in Radiate
mode.
```

When this consultation is presented, DIAG also displays the scene containing the indicator to be checked and displays a graphical pointer to it. Currently DIAG does not explain why the recommended test is a good one, although it has the capacity to do so (by outlining the inferences that could be drawn from each possible outcome).

Discuss Previous Exercise. At the completion of an exercise, the learner may request a review of the previous problem. DIAG steps through each *significant* indication that was displayed, stating whether that indication was normal or abnormal, and listing the most significant inferences that could be drawn from the indication. The following is the analysis of one indication.

```
Channel 2 Arc indicator was on in Radiate mode,
  which was abnormal (normal is off).
The most suspected units are now:
T6
Traveling Wave Tube V2
Vac-ion Power Supply A15
PC card A12A48
PC card A12A39
PC card A12A37
PC card A12A4
```

As each indication is discussed DIAG displays that indication graphically and points to it.

Expert Approach. Following an exercise the learner may ask to see how an expert would have performed the same problem. DIAG inserts the fault into the system model again, then it performs and explains a very good diagnostic sequence. At each step DIAG points to the indicator or test point that it is checking, and it produces a verbal explanation of the significance of the indication seen. The following is the explanation of one test.

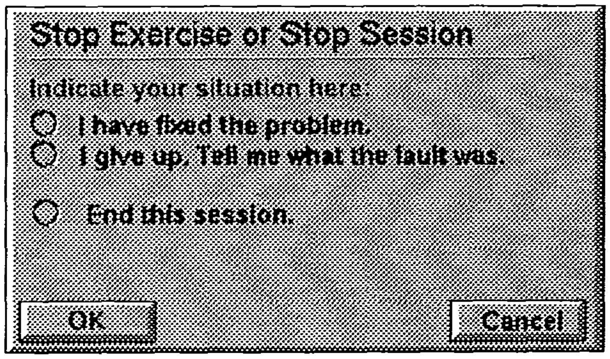
```
We check A1 Standby Button in Radiate mode.
  It is green which is abnormal.
(Normal is off).
The most suspected units are now:
PC card A12A48
PC card A12A41
PC card A12A39
PC card A12A37
T6
PC card A12A32
```

It is very interesting to observe the suspicion rankings as the expert demonstration proceeds. In easy problems, the true fault moves to the head of the list after just a few tests. In contrast, a very difficult fault might never reach the top of the suspicion ranking, since its symptoms do not identify it clearly. Thus, even the expert within DIAG may make incorrect replacements on some problems, just as human experts sometimes do. *This is not a flaw in the instructional system or in the expert diagnostic reasoning.* It is a fact of life that faults in the real world can present symptoms that disguise their true identity.

See Fault Recap. This function simply redisplays the fault recap for the just completed exercise.

Problem Termination

The learner declares an exercise completed by clicking on a **Stop** button and indicating via the following dialog box what result has been achieved .



Stop Exercise or Stop Session

Indicate your situation here:

○ I have fixed the problem.

○ I give up. Tell me what the fault was.

○ End this session.

OK Cancel

If the learner incorrectly claims to have fixed the problem DIAG so states, and the problem continues. Such erroneous claims are recorded in the individual's performance records and are considered serious errors. Ultimately, exercises are only terminated with correct system restoration or by giving up. After terminating a problem, the learner may end the DIAG session.

Records of Individual Performance

The records that DIAG maintains about each individual reflect the exercises successfully completed, the times taken to completion, serious errors committed, and the extent to which the individual relied upon DIAG consultations. These records are sufficient to select problems adaptively, and they provide measures of progress and proficiency that can be used by the instructor to address areas of difficulty or to modify the course plan.

Conclusions

Briefly, the prototype application of DIAG leads to the following conclusions:

1. Combining an interactive graphical device model with a bank of qualitative symptom data yields a very robust tutoring system, in which all context-dependent tutoring presentations can be generated automatically.

2. The process of developing the interactive graphical device model requires considerable training and computer literacy.

3. The process of providing domain-specific qualitative symptom information is one that can be done successfully by a knowledgeable technician, regardless of his or her computer skills.

4. DIAG's tutoring power is relatively wide ranging and supportive. It returns very high tutoring value for the development effort invested, but it cannot generate the same technical verbiage that can be loaded into an expert system.

5. Under some tutoring conditions, DIAG may reveal more about an ongoing exercise than is desirable.

References

Johnson, W. B.; Norton, J. E.; Duncan, P. E.; and Hunt, R. M. 1988. Development and Demonstration of an Intelligent Tutoring System for Technical Training (MITT), Technical Report AFHRL-TP-88-8, Brooks AFB, TX: The Air Force Human Resources Laboratory.

Lesgold, A.; Eggan, G.; Katz, S.; and Rao, G. 1992. Possibilities for Assessment Using Computer-based Apprenticeship Environments. In J. Regian & V. Shute (Eds.*), Cognitive Approaches to Automated Instruction* (pp 49-80). Hillsdale, NJ: Erlbaum.

Munro, A.; Johnson, M. C.; Surmon, D. S.; and Wogulis, J. L. 1993. Attribute-centered Simulation Authoring for Instruction. In Proceedings of AI-ED '93 World Conference on Artificial Intelligence in Education.

Towne, D. M. and Johnson, M. C. 1987. Research on Computer-aided Design for Maintainability, Technical Report, 109, Behavioral Technology Laboratories, University of Southern California.

Towne, D. M.; Munro, A., Pizzini, Q. A.; Surmon, D. S.; and Wogulis, J. L. 1990. Intelligent Maintenance Training Technology, Technical Report, 110, Behavioral Technology Laboratories, University of Southern California.

Towne, D. M. and Munro, A. The Intelligent Maintenance Training System. In Psotka, J.; Massey, L. D.; and Mutter, S. A. eds. 1988. *Intelligent Tutoring Systems: Lessons Learned* (479-530). Hillsdale, NJ: Erlbaum.

Towne, D. M. Model-based simulations for instruction and learning. Proceedings, Delta '94 Conference: Telematics for education and training. Dusseldorf, Germany, Nov. 1994.

Towne, D. M. DIAG: *Diagnostic Instruction and Guidance, Application Guide*. Los Angeles: Behavioral Technology Laboratories, University of Southern California, 1996

Zadeh, L. & Kacprzyk, J. *Fuzzy logic for the management of uncertainty* (Eds.). New York: John Wiley & Sons, Inc. 1992.