Expert System for Diesel Electric Locomotive Repair

Harold E. Johnson, Jr. *Piero P. Bonissone*

Flexible Automation Systems Program and Knowledge Based Systems Program
Corporate Research and Development
General Electric Company
Schenectady, N. Y. 12345

Abstract

In recent years, expert systems have become the most visible and the fastest growing branch of Artificial Intelligence. General Electric Company's Corporate Research and Development has applied expert system technology to the problem of troubleshooting and the repair of diesel electric locomotives in railroad "running repair shops." The expert system uses production rules and an inference engine that can diagnose multiple problems with the locomotive and can suggest repair procedures to maintenance personnel. A prototype system has been implemented in FORTH, running on a Digital Equipment PDP 11/23 under RSX-11M. This sytsem contains approximately 530 rules (roughly 330 rules for the Troubleshooting System, and 200 rules for the Help System), partially representing the knowledge of a Senior Field Service Engineer. The inference engine uses a mixed-mode configuration, capable of running in either the forward or backward mode. The Help System can provide the operator with assistance by displaying textual information, CAD diagrams or repair sequences from a video disk. The rules are written in a representation language consisting of nine predicate functions, eight verbs, and five utility functions. The first field prototype expert system, designated CATS-1 (Computer-Aided Troubleshooting System - Version 1), was delivered in July 1983 and is currently under field evaluation.

Introduction

In the last few years, expert systems [7-9] have become the most visible and the fastest growing branch of Artificial Intelligence [1, 5, 12, 14]. The objective of these systems is to capture the knowledge of an expert in a particular problem domain, represent it in a modular, expandable structure, and transfer it to other users in the same problem domain. To accomplish this goal, it is necessary to address issues of knowledge acquisition, knowledge representation, inference mechanisms, control strategies, user interface, and dealing with uncertainty.

There are various approaches to the representation of the expert's knowledge, spanning from logic [11], to semantic network [3], frames [10] and production rules [6, 13]. Each representation has its own advantages and disadvantages, and this paper will limit itself to the description of an expert system implemented using production rules.

Rule-based expert systems consist of a body of knowledge (knowledge base) and a mechanism (inference engine) for interpreting this knowledge. The body of knowledge is divided into *facts* about the problem, and heuristics or *rules* that control the use of knowledge to solve problems in a particular domain.

The facts represent atomic pieces of evidence describing the problem to solve. They can be generated at the beginning of the session, by asking the user a fixed sequence of questions that establish the current problem at hand. Facts are also generated throughout the session as a direct result of system inference, and as additional questions are asked of the user.

The rules are conditional statements expressed in a subset of English, thus easy to understand. Each rule consists of a situation recognition part (premise) and an action part (conclusion). The situation part expresses some condition on the state of the data base, and at any given point it is either satisfied or not. The action part specifies changes to be made to the data base whenever a rule is satisfied.

*Received September 1983

The inference engine is an interpreter of the facts and the rules. Its task is to monitor the facts in the data base and execute the action part of those rules that have their situation part satisfied. The inference engine can operate forward (event-driven) or backward (goal-driven). In the forward mode, it tries to arrive at a goal, starting from the available facts. In the backward mode, it selects a goal and then verifies whether or not the supporting facts are present or can be inferred.

Problem and Proposed Solution

The General Electric Company's Corporate Research and Development has applied expert system technology to demonstrate the system's feasibility in the area of troubleshooting. To test these techniques, the problem selected was the repairing of diesel electric locomotives in "running repair shops" : railroad maintenance personnel must detect and repair a large variety of faults that have partially disabled a diesel electric locomotive. The *a priori* information available to them is the list of "symptoms" reported by the engine crew. More information can be gathered in the shop, by taking measurements and performing tests that may consume excessive "shop time" if performed by inexperienced personnel.

The result of this development effort is a rule-based expert system, DELTA (Diesel Electric Locomotive Troubleshooting Aid) [2], which guides the troubleshooter in his task, enforcing some disciplined troubleshooting procedures that will minimize the cost and time of the corrective maintenance.

Originally, a feasibility prototype system was developed in LISP [4, 15]. Subsequently a field prototype system has been implemented in FORTH, running on a Digital Equipment PDP 11/23 under RSX-11M. (The system also runs on a PDP 11/70 under RSX-11M-PLUS and in emulation model on a VAX 11/780 under VMS.) The choice of Forth was dictated by strong implementation issues, such as compactness of code, efficiency and portability to other microprocessor-based systems. This system contains approximately 530 rules, partially representing the knowledge of a Senior Field Service Engineer. Roughly 330 rules are devoted to the fault diagnosis and repair procedures, i.e., the Troubleshooting System, while about 200 rules form the Help System. The Troubleshooting System uses a mixed-configuration inference engine based on a backward chainer and a forward chainer, as illustrated in Figure 1. The Help System, uses the forward chainer of the same inference engine to respond to requests for information from the expert system. When the user hits the "HELP" key, the system provides additional information, such as the location and identification of locomotive components, replacement part classification, and description of repair procedures. To accomplish this task, the system uses CAD files stored in TEKTRONIX line graphics format and VIDEO pictures stored on a laser video disk.

A pictorial description of a session with this expert system is illustrated in Figure 2. A fixed sequence of questions is used to gather the initial facts about the locomotive problem, such as unit number, model year, reported symptoms, etc. An associative information table provides additional facts, such as unit standard features, unit history of failures, model failure propensity, etc. All these facts constitute the starting point for the troubleshooting process.

The set of rules (heuristics) that embeds the empirical knowledge about the diesel electric engine is functionally partitioned into knowledge spaces such as mechanical system, electrical system, etc. Within each knowledge space, the rules are subdivided according to hypotheses (fault areas), such as Operator Error, Engine Unable to Make Power, etc.

A set of meta-rules (a smart index of the knowledge partitions) retrieves from the various knowledge spaces the subsets of rules associated with all the hypotheses that could be relevant to the initial symptoms. This collection of hypotheses constitutes a preliminary diagnosis. Working in a backward mode, the interpreter tries to prove or disprove each hypothesis, based on both initial facts and additional facts inferred by the system or asked of the user.

The result of this process is a final diagnosis that indicates the successful hypotheses (faults) and their corresponding corrective actions (repairs).

Inference Engine

This expert system is based on a mixture of control strategies, since its inference mechanism can work in either forward or backward mode (Figure 1).

When the initial facts are input by the user, the META-RULES load a set of HYPOTHESES and a set of IFF-RULES, IF-RULES and WHEN-RULES (see discussion of rules below).

The BACKWARD INTERPRETER then tries to evaluate each hypothesis with the given set of rules and current facts. The evaluation of a hypothesis (goal) is a three-step process. First, the system

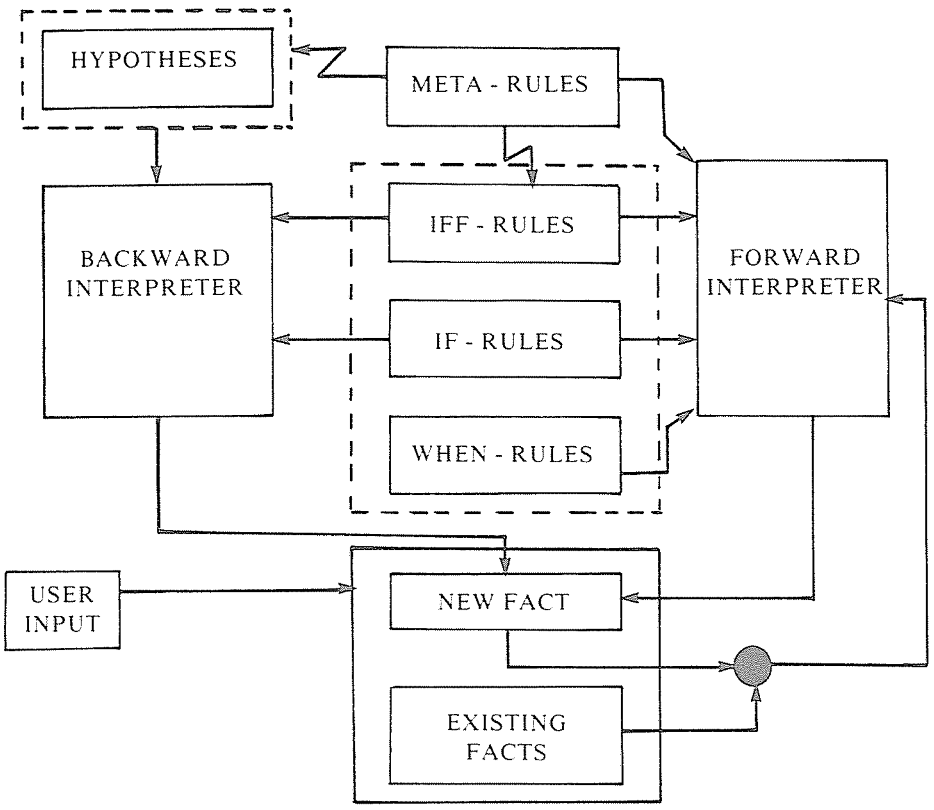

Figure 1 Inference Mechanism Configuration

scans the list of facts to verify whether the hypothesis is already known to be true or false. If this is the case, then evaluation terminates. Otherwise, the system scans the conclusion of each rule to determine whether the hypothesis could be proved by at least one rule. In such a case, the system recursively evaluates each clause (sub-goal) in the premise of that rule. Finally, when no hypothesis (or argument) can be directly inferred by a rule, the system requests information from an external source (either the user or a sensor).

During this deductive process, new evidence (NEW FACT) needed to prove a hypothesis could be inferred by the BACKWARD INTERPRETER or input by the USER/SENSOR. When NEW FACT is written in the list of facts, the FORWARD INTERPRETER is activated. This interpreter scans the META-RULES, IFF-RULES, IF-RULES and WHEN-RULES, trying to execute any rules containing NEW FACT in their premise.

The META-RULES verify whether or not some new knowledge (new set of hypotheses and corresponding IFF-RULES, IF-RULES or WHEN-RULES) is required and whether or not the existing knowledge should be reorganized, by reordering the set of current hypotheses, as a result of the presence of NEW FACT.

The IFF-RULES and the IF-RULES try to find some new evidence that can be inferred directly, based on the presence of NEW FACT. The new evidence could later provide a shorter path in the deduction process. These rules can be accessed by both the backward and forward chainers. IFF-RULES are "if-and-only-if" type rules (if A then B, and if not-A then not-B). IF-RULES are "if-then" type rules (if A then B).

The WHEN-RULES attach properties and activate procedures associated with NEW FACT. These rules can only be accessed by the forward chainer, thus preventing the backward chainer from using them to establish some goal. WHEN-RULES are "when-then" type rules (when A then B).

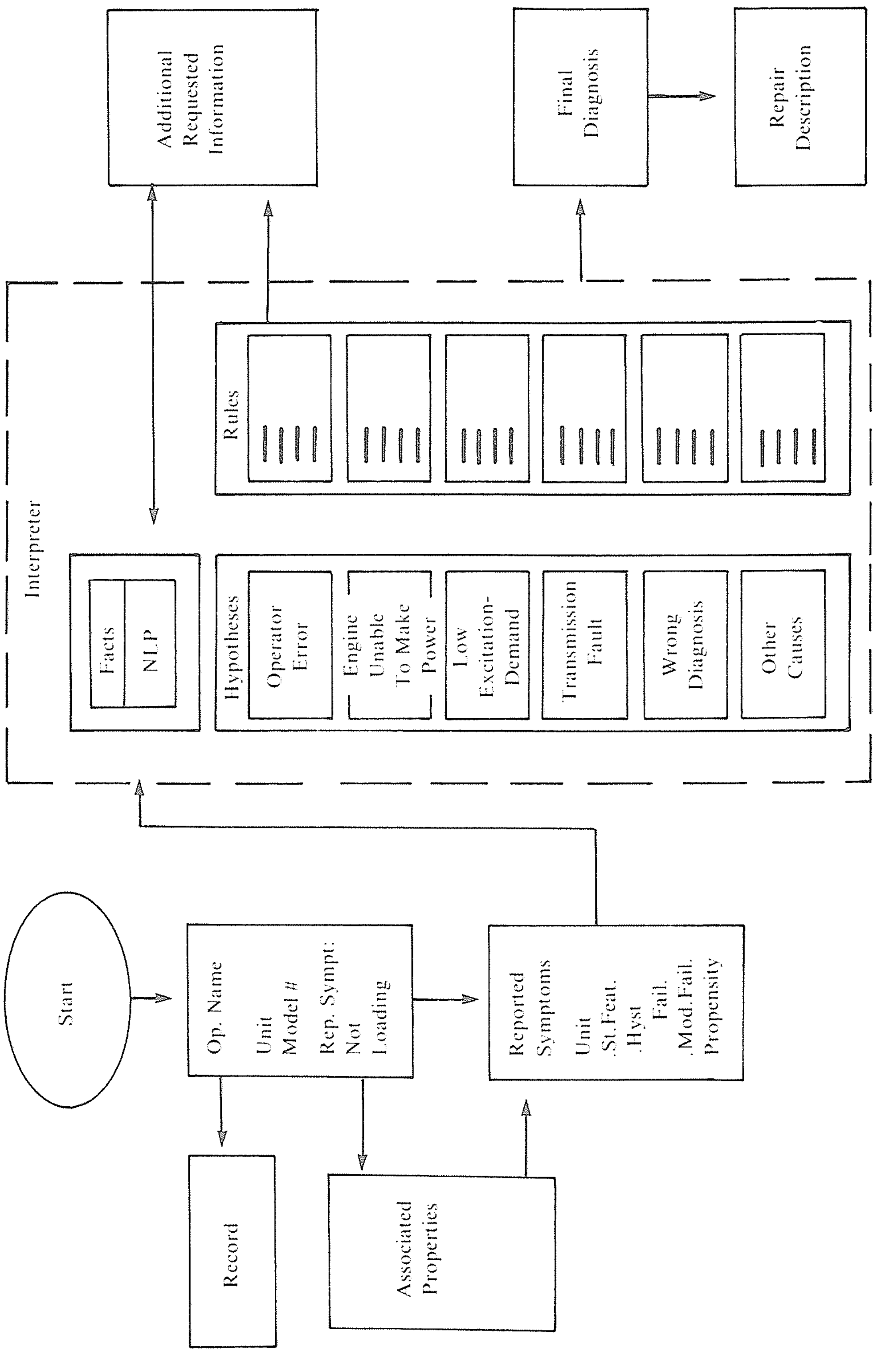

Figure 2 Expert Trouble-Shooting System Architecture

If any of the above rules has a fully satisfied premise (since no explicit rule-chaining or user-prompting is allowed during the evaluation of rules in the forward mode), then the FORWARD INTERPRETER executes that rule, writes another NEW FACT, and iterates again. This forward-chaining process stops when no rule can be executed by the FORWARD INTERPRETER, and control is returned to the BACKWARD INTERPRETER.

The BACKWARD INTERPRETER will continue its deductive process, until a hypothesis is proved or the entire set HYPOTHESES has been exhaustively evaluated.

Representation Language

The rules that form the knowledge base of the Troubleshooting System and the Help System are written in a special representation language. The user-extensible language currently contains:

- nine predicate functions to describe the conditions of the premise of each rule,
- eight verbs to describe the actions and inferences in the conclusions of each rule,
- five utility functions to interact with the user and display alphanumeric, graphic or pictorial information.

Each rule is a conditional statement describing the logical implication:

$$\text{(premise)} \xrightarrow{cf} \text{(conclusion)}$$

The weight "cf" is the certainty factor, a number between -1 and 1, which indicates the strength of such implication. This number is used to control the propagation of uncertainty in rule-chaining, to control the combination of different pieces of evidence supporting the same conclusion, and to evaluate the overall degree to which a premise is satisfied.

Each premise is an intersection of clauses. Therefore, a premise is satisfied if *all* its clauses are also satisfied. In this case, the intersection of clauses corresponds to a boolean AND. However, this operation could be extended to a fuzzy intersection, e.g., MIN, if the truth-value of the clauses can take values within the interval [-1, 1].

Each clause is defined by a predicate function and an argument composed of a 3-tuple <object attribute value>. Each clause is satisfied if its predicate function returns a true-value when applied to its argument.

Each conclusion is a disjunction of actions that are executed once the premise of the rule has been satisfied. Each action is defined by a verb and an argument.

Moreover, there are five utility functions that can be present in the premise or in the conclusion of the rule. These functions are transparent to the rule interpreter, in the sense that they do not affect the truth-value of the premise and do not modify the list of facts. The purpose of these functions is to help the user with text, graphics or video-images. These functions form the basis of a rule-driven help system.

A listing of the predicate functions, verbs and help-functions is provided in Appendix I. Appendix II illustrates three rules of the Expert System, describing a fault in the locomotive fuel system, and two rules of the Help System describing related available information.

Appendix III shows the FORTH source code used to compile Rule 5210 in Appendix II. All of the rules in Appendix II were listed by a pretty-printer which adds text identifying the end of a rule and removes the parenthesis in FORTH comments. The 3-tuples are offset by the executable word, [and closed by a]. As an example, in Rule 5210 EQ is a predicate function with an object of FUEL-REGULATING-VALVE, an attribute of VIDEO and a value of HELP . Sometimes expressions will have multiple values. The 3-tuple associated with VDSHOW gives the starting and ending frames, and specifies still mode for the video disk. Rules needn't be explicitly closed, as the source executes and builds data structures via the FORTH text interpreter.

Conclusion

The first field prototype of this expert system has already been implemented in a rugged unit, packaged by COMARK, containing a PDP 11/23 (running RSX-11M and an enhanced version of fig-FORTH), a 10 meta-byte Winchester disk, a VT100 terminal and a SELANAR graphics board. A SONY laser-video-disk player and an additional color monitor complete the configuration of this field prototype system. The system has already shown promising results since its recent delivery to the General Electric Company's Locomotive Operation in July 1983. The mixed-mode configuration of its inference engine performs very well. The FORTH implementation proved to be easily transportable to small microprocessor-based systems while maintaining fast execution speed. The man-machine interface is very user-friendly and allows the user to interact with the system via menu selections or simple (single keystroke) answers such as: Y, N, ?, W, H, (Yes, No, ?Unknown, Why?, Help).

During the next six months, the locomotive troubleshooting system will be tested in the field to verify the accuracy of its knowledge base and the reliability of the hardware configuration. In the following phase of this project, the knowledge base will be expanded to approximately 1200 rules, to cover, with increased depth, a larger portion of the problem space.

References

[1] A. Barr, E. A. Feigenbaum, *The Handbook of Artificial Intelligence* vol. 1 and 2, William Kaufmann, Inc., Los Altos, CA 1982.

[2] P. P. Bonissone, "Outline of the Design and Implementation of a Diesel Electric Engine Troubleshooting Aid", *Proceedings of Expert Systems 82*, pp. 68-72, Brunel University, Egham, England, September 14-16, 1982.

[3] R. J. Brachman, "On the Epistemological Status of Semantic Network", in *Associative Networks, Representation and Use of Knowledge by Computers*, pp. 3-50, Academic Press, 1979.

[4] E. Charniak, C. K. Riesbeck, D. V. McDermott, *Artificial Intelligence Programming*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1980.

[5] P. R. Cohen, E. A. Feigenbaum, *The Handbook of Artificial Intelligence* vol. 3, William Kaufmann Inc., Los Altos, CA, 1982.

[6] R. Davis, B. Buchanan, "Production Rules as a Representation for a Knowledge-Based Consultation Program", *Artificial Intelligence*, vol. 8, pp. 15-45, 1977.

[7] R. Davis, D. Lenat, *Knowledge-Based Systems in Artificial Intelligence*, McGraw-Hill, 1982.

[8] F. Hayes-Roth, D. Waterman, D. Lenat, *Building Expert Systems*, Addison Wesley, Reading, MA, 1983.

[9] D. Michie, *Expert Systems in the Micro Electronic Age*, Edinburgh University Press, UK, 1979.

[10] M. Minsky, "A Framework for Representing Knowledge" in *The Psychology of Computer Vision*, pp. 211-277, McGraw-Hill, 1975.

[11] N. J. Nilsson, *Principles of Artificial Intelligence*, Tioga Press, Palo Alto, CA, 1980.

[12] Y. Pao, G. W. Ernst, "Context-Directed Pattern Recognition and Machine Intelligence Techniques for Information Processing", *IEEE Computer Society Press*, 1982.

[13] D. Waterman, F. Hayes-Roth, *Pattern-Directed Inference Systems*, Academic Press, NY, 1978.

[14] P. H. Winston, *Artificial Intelligence*, Addison Wesley, 1977.

[15] P. H. Winston, B. K. Horne, *LISP*, Addison Wesley, 1981.

Mr. Johnson received a BSE in computer science and engineering in 1976 and an MSE in systems engineering in 1977, both from the University of Pennsylvania. He has worked on hardware and software for a spectrometer, real-time image processing and a multi-user business operating system. Currently he is using threaded interpretive languages to implement knowledge-based "expert" systems and an interactive robotic controller.

Dr. Bonissone received a BS in mechanical and electrical engineering from the University of Mexico City in 1975, an MS in electrical engineering and computer science in 1978, an MS in mechanical engineering in 1979 and a PhD in electrical engineering and computer science in 1979 from the University of California at Berkley. He is an adjunct professor at Rensselaer Polytechnic Institute and has interests in expert systems, fuzzy sets and areas of natural language processing. Currently he leads the DELTA development team at the General Electric Research and Development Center, which developed the expert system described here.

Appendix I

Description of the nine types of predicate functions used in the FORTH implementation:

(1) EQ: evaluates the argument, and returns a true-value if the argument was proven true (Equal).
(2) EVAL-ALL: forces an exhaustive evaluation of the argument and returns a true value if the argument was proven true at least once during its evaluation.
(3) NE: evaluates the argument and returns a true-value if the argument was proven false (Not Equal).
(4) NC: evaluates the argument and returns a true value if the argument was either proven false or unknown (Not Confirmed).
(5) ND: evaluates the argument and returns a true value if the argument was either proven true or unknown (Not Disconfirmed).
(6) ASK-Y: prompts the user with a particular question (the comment associated with the clause), writes the argument as a new fact (with an attached certainty-factor indicating the user's response) and returns a true-value if an affirmative response was input.
(7) ASK-N: like ASK-Y, but returns a true-value if a negative response was input.
(8) UDO: requires the user to perform a given action (the comment associated with the clause), waits for a confirmation from the user, writes the argument as a new fact and returns a true-value if the action was confirmed.
(9) MENU: displays a menu of choices, prompts the user for a specific menu entry selection, writes the selection as a new fact and returns a true value if a legal entry was selected. This function has three components:
    MENU-T: displays the title of the menu
    MENU-E: displays a menu entry
    MENU-S: prompts the user for an entry selection

Description of the eight types of verbs used in the FORTH implementation:

(1) WRITE: writes the argument as a fact in the list of facts.
(2) CLR: deletes from the list of facts any existing fact which matches the argument.
(3) EVAL: activates the backward chaining interpreter, trying to verify the argument.
(4) EVAL-ALL: activates the backward chaining interpreter, performing an exhaustive verification of the argument.
(5) ASK: prompts the user with a particular question (the comment) and writes the argument (with an attached certainty factor indicating the user's response) as a fact in the list of facts.
(6) UDO: requires the user to perform a given action (the comment), waits for a confirmation and writes the effect of the action (the argument) as a fact in the list of facts.
(7) STOP: displays a termination message (the comment) to the user and terminates the session, disregarding any pending tasks.
(8) MENU: displays a menu of choices, prompts the user for a specific menu entry selection and writes the selection as a new fact. This verb has three components:
    MENU-T: displays the title of the menu
    MENU-E: displays a menu entry
    MENU-S: prompts the user for an entry selection

Description of the five utility functions used in the FORTH implementation:

(1) DISPLAY: displays a message to the user.
(2) PAUSE: displays a message and waits for an acknowledgement from the user.
(3) SHOW: displays a CAD file (graphic picture) or an alphanumeric file on the user's terminal.
(4) SCREEN: clears the graphic plane of the user's terminal.
(5) VDSHOW: displays a video-image (still frame or film sequence) on the auxiliary monitor.

Appendix II

Sample of three rules in the Expert System related to a fault in the fuel system.

```
Rule 760
    there is a fault in the fuel system at idling speed and readings were taken from the locomotive fuel
    pressure gage
IF:
    EQ  [ ENGINE SET IDLE ]
        Is the engine at idle?
    EQ  [ FUEL PRESSURE BELOW NORMAL ]
        Is the fuel pressure below normal? (Less than 38 psi?)
    EQ  [ FUEL-PRESSURE-GAGE USED IN TEST ]
        Did you use the locomotive gage?
    EQ  [ FUEL-PRESSURE-GAGE STATUS OK ]
        Is locomotive gage known to be accurate?
THEN:
    WRITE  [ FUEL SYSTEM FAULTY ]  1.00
        Establishes that there is a fuel system fault.
End of rule 760

Rule 1270
    the locomotive fuel-pressure gage is OK
IF:
    UDO  [ FUEL-PRESSURE-TEST-GAGE STATUS ATTACHED ]
        Attach a known good pressure gage.
    ASK-Y  [ FUEL-PRESSURE-TEST-GAGE READING SAME-AS FUEL-GAGE ]
        Is test-gage reading the same as locomotive-gage reading?
THEN:
    DISPLAY  [ FUEL-PRESSURE-GAGE STATUS OK ]
        The locomotive-pressure-gage is OK.
    WRITE  [ FUEL-PRESSURE-GAGE STATUS OK ]  1.00
        Establishes that the locomotive-pressure-gage is OK.
    WRITE  [ FUEL-PRESSURE-GAGE STATUS ALREADY TESTED ]  1.00
        Establishes that the locomotive-gage has been tested.
End of rule 1270.

Rule 1460
    there is at least one faulty fuel system component
WHEN:
    EQ  [ FUEL SYSTEM FAULTY ]
        The fuel-system is faulty
THEN:
    DISPLAY  [ FUEL SYSTEM FAULTY ]

        There is a fuel system fault.

    WRITE  [ FUEL PROBLEM SOLVED ]  -1.00
        Establishes that the fuel problem is not solved.
    EVAL-ALL  [ FUEL SYSTEM-COMPONENT FAULTY ]
        Tests for a faulty fuel system component.
End of rule 1460
```

Sample of two rules in the Help System describing available information relevant to the subgoal of verifying the accuracy of the fuel pressure gage.

```
Rule 5190
    you want to see the Fuel Pressure Test Gage Menu
WHEN:
    EQ  [ FUEL-PRESSURE-TEST-GAGE MENU HELP ]
          Requests FUEL PRESSURE TEST GAGE Menu
THEN:
    CLR  [ FUEL-PRESSURE-TEST-GAGE MENU HELP ]
         Forgets Request
    MENU-T  [ FUEL-PRESSURE-TEST-GAGE SELECTION INVALID ]  1.00
         This Menu contains information of the
         FUEL PRESSURE TEST GAGE

         These are your choices:
    MENU-E  [ FUEL-TEST MENU HELP ]  1.00
         I want to go back to FUEL TEST Menu
    MENU-E  [ FUEL-PRESSURE-TEST-GAGE PICTURE HELP ]  1.00
         CAD Picture of pipe plug where test gage should be attached
    MENU-E  [ FUEL-REGULATING-VALVE VIDEO HELP ]  1.00
         VIDEO Picture of regulating valve where pipe plug is located
    MENU-E  [ GOAL: BACK TO EXPERT OR STOP ]  1.00
         End of help. Back to our problem
    MENU-S  [ FUEL-PRESSURE-TEST-GAGE SELECTION FINISHED ]
         Please enter your selection by number:
End of rule 5190

Rule 5210
    you want a VIDEO picture of fuel regulating valve
WHEN:
    EQ  [ FUEL-REGULATING-VALVE VIDEO HELP ]
          Request Picture of regulating valve where pipe plug is located
THEN:
    CLR  [ FUEL-REGULATING-VALVE VIDEO HELP ]
          Forgets request
    VDSHOW  [ 16120 16120 0 ]

    WRITE  [ FUEL-PRESSURE-TEST-GAGE MENU HELP ]  1.00
          We want to use Fuel Pressure Test Gage Menu
End of rule 5210
```

Appendix III

```
RULE 5210
    ( you want a VIDEO picture of fuel regulating valve)
WHEN:
    EQ  [ FUEL-REGULATING-VALVE VIDEO HELP ]
      ( Requests Picture of regulating valve where pipe plug is located)
THEN:
    CLR  [ FUEL-REGULATING-VALVE VIDEO HELP ]
      ( Forgets request )
    VDSHOW  [ 16120 16120 0 ]
      (     )
    WRITE  [ FUEL-PRESSURE-TEST-GAGE  MENU  HELP ]
      ( We want to use Fuel Pressure Test Gage Menu)
```

DELTA system is tested on a GE locomotive by Frank Lynch (seated), Manager of the R&D Center's Knowledge Based Systems Program, and Dave Smith, GE locomotive expert.