GENERIC EXPERT SYSTEM SHELL FOR DIAGNOSTIC REASONING

Wei-Han Chu

Varian Research Center
611 Hansen Way
Palo Alto, CA 94303

Abstract

Rule based expert systems provide a modular and uniform approach to representing knowledge, however it has been recognized that rule-based systems become increasingly difficult to understand and maintain as the number of rules grow. Expert systems today are developed on general purpose inference shells that offer general purpose paradigms which do not take into considerations the type of problems being solved. It is up to the users to create the meta level control to prevent rule interference, and for the rules to function properly. This task tends to become increasingly difficult in direct proportion to the size of the accumulated knowledge.

The solution is in a new generation of Application Specific Expert System Tools that are designed with specific paradigms and knowledge representation methodology that meet the requirements of a specific domain. This concept is examplified in the work presented here that introduces a generic expert systems shell for diagnostic reasoning. Domain knowledge is represented as five different classes of objects. A paradigm for diagnostic reasoning is built into the inference algorithm to become part of the inference shell, replacing the usual general purpose forward or backward chaining algorithm.

1. INTRODUCTION

Trouble-shooting of electronic systems and electro-mechanical systems has long been recognized as a domain well suited for the application of expert system technology. A typical diagnostic expert system encodes the domain knowledge as a collection of rules that relate the cause and effect of the possible known faults. The benefits of separation of rules from data, and the representation of diagnostic knowledge as declarative rules have been publicized [Dav84]. It is fashionable for vendors of expert system development tools to claim that, with the aid of their respective tools, rule-based expert systems can be developed by domain experts without training in either knowledge representation or programming. The claim is to imply that knowledge acquisition to implement a fully functional expert systems is feasible without knowledge engineers.

To anyone who has been involved in the development of a rule-based expert system of a significant size, those claims appear to be dubious. While a reasonable tool can assist a domain expert in formulating cause-and-effect rules, a collection of such rules typically will not function as an expert system, except for the most simple cases. It is inadequate to perceive the domain expert's knowledge simply as a collection of cause-and-effect rules, but it must be recognized that the expert knowledge also include strategies in performing the diagnosis tasks. Expert system development tools typically allow the encoding of such strategies in meta-level rules, i.e. the rules that contain the object-level knowledge and strategic knowledge [Dav84a]. Meta-level rules are also necessary to prevent the interference of object-rules. The effective application of meta-level rules, however, requires extensive knowledge of the programming paradigm and the skillful use of the facilities offered by the tool.

The introduction of meta-level rules allows reasoning strategies to be encoded, but it also greatly increases the complexity of the system as a whole. The construction of meta-level rules requires thorough understanding of the programming paradigm, and it is no longer apparent that an average domain expert will be able to do so without the assistance of a knowledge engineer. As the number of rules grows, the number of meta-level rules grows at a disproportionate rate, and the resulting system becomes increasingly difficult to understand and maintain [Bru86].

Chandrasekaran [Cha86] recognized that after such a system is built, it becomes indistinguishable which portion of the knowledge represents domain expertise and which represents programming devices. The control strategies of the expert system is therefore "encrypted" in the massive collection of rules. The proclaimed simplicity and modularity of rule-based expert system is no longer evident. Understanding an existing system well enough to modify its behavior becomes an increasingly formidable task.

One of the contributing factors to the above situation is that today's expert systems tools are designed and offered as general purpose tools. While the tools may offer one or more paradigms, no consideration is given to the particular problems being solved. Thus the user of the tools is often fitting a problem to the available tool, rather than using a tool tailored to the problem. Meta-level facilities are not so much tools to solve a problem, as they are a means of adapting the problem to the tool.

In view of the limitations of the current generation of expert systems tools, research efforts are now directed to a new generation of Application Specific Expert Systems Tools. Of particular interest are applications in diagnostic and financial problems ([Fin85], [Ben88]).

2. EQUIPMENT SERVICING PARADIGM

The generic nature of different reasoning tasks has been suggested before [Cha86]. Chandrasekaran recognized that many knowledge base reasoning tasks can be categorized as generic tasks, and the organization and representation of knowledge in each generic task should support the control behavior of the task. A specific application may involve a number of such generic tasks at different stages.

Our research work is not to identify the generic reasoning tasks that constitutes diagnostic reasoning, but to identify a generic paradigm specific to the domain of equipment fault diagnosis, and to develop an expert system tool suited for building service expert systems across a wide range of product lines. Our motivation is to retain the advantages of modularity and clarity of declarative representation of the domain knowledge, yet relieve the burden of crafting meta-level control from the user.

The core process of diagnosis has been suggested as a classification process [Gom81]. While this is true at a conceptual level, at a practical level the processes of diagnosis differ in different domains. Generally, however, equipment fault diagnosis cannot be viewed as a pure classification process. Other practical considerations such as company standard test procedures, cost of testing a suspected fault, availability of replacement components, the operating environment of the equipment, etc., often enters into the picture.

Our work is based on the notion that there is a common generic paradigm for equipment fault diagnosis. Similar to the rule based expert systems, the separation of the domain knowledge, operating data, and the control algorithm is maintained. The control algorithm in our case has a broader perspective by including the service strategy, rather than just as an instrument for rule selection and chaining.

An obvious paradigm for equipment servicing is to consider a search down a fault tree organized in the component hierarchy. Fault location is through a top down search of this tree to lower and lower levels in the component hierarchy. Each arc leading from one node to its child represent the result of a test performed. This is the classic view of diagnosis as a classification process. Many theories have been formulated around this paradigm, in practice this does not appear to be the strategy of the experts in the equipment service domain. The service experts rarely traverse down the component hierarchy and query the possibility of a fault existing in each of the subcomponents at the next level. Test results are often not that clear cut always to be indicative of the subcomponents that are suspect at the next level. Experts cannot always relate a test result accurately to a subcomponent fault. Experts may also apply heuristic knowledge to quickly identify a few possible faults not necessarily in the same hierarchy.

The data that a service expert utilizes for diagnosing a system fault are the observed behavior, or symptoms of the system, and the data gathered from tests and measurements performed by the expert. Let each piece of such data be an observation O_i. The knowledge state of the expert on the system at any moment S_i is a collection of observations which we shall refer to as the *Working Set* of observations..

$$S_i = \{O_1, O_2, O_3, \ldots, O_n\}$$

Each new test or measurement generates new observations that adds to the understanding of the equipment being diagnosed. The additional data are added to the Working Set, taking it to a different knowledge state. Thus a test or measurement T generates additional observations $\{O_{n+1}, O_{n+2}, \ldots, O_m\}$ providing a path that traverses from a state S_i to a neighboring state S_{i+1} where

$$S_{i+1} = \{O_1, O_2, \ldots, O_n, O_{n+1}, \ldots, O_m\}$$

Each state S_i in the state space maps to one or more hypotheses of faults. If each state is represented as a node, then the entire state space is represented by a graph. The arc leading from node S_i to its neighbor S_{i+1} represent the test necessary to generate the additional observations $\{O_{n+1}, \ldots, O_m\}$.

The diagnostic process is therefore a process to traverse the state space network by performing tests and measurements at each node. Each state in the graph maps to a set of fault hypotheses. At each node the next test to be performed is the test to confirm the first fault hypothesis. If a test failed to confirm the fault hypothesis, it would have generated additional observations to take the Working Set to a different state. When new fault hypotheses are generated as a result of the new data, the new fault hypotheses are added to the hypotheses queue to be considered.

Successful confirmation of a hypothesis leads to a repair action, which is followed by the standard check out tests to verify the integrity of the repaired system. The standard check out tests, if successful, should generate a set of expected observations S_t (termination observations) that indicates the system is in normal operating conditions. Thus there exists a set of terminal nodes in the graph, each one of these nodes is entered by the same check out tests. The Working Set at each terminal node will contain the termination observations S_t as a subset. The reasoning process terminates when one of these terminal nodes is reached.

A few remarks may be added to this paradigm for equipment servicing.

1. The objective is to reach *a* terminal node. No attempt is made to find the theoretical optimal path. Unlike medical expert systems where much cost incurred in applying certainty factors to determine the best conclusion for the disease, and hence the treatment, there is no attempt to select the most probable hypotheses among those on the queue. The hypotheses are ranked according to cost, including testing cost, repair or replacement cost. This is consistent with the practice of the service expert. If a part can be easily reached, it will be tested before another part that requires half an hour of disassembly, even if the latter may be more probable to be the location of the fault. Similarly, for repair actions a component may be proven to be faulty, but if some adjustment may delay the need to replace an expensive component, that may be the preferred action even though it is clear the problem will reoccur some time later. The contention is that a service expert system should not only reach the same conclusion as the human expert, but should also follow the same strategy in determining the course of action. The strategy of the expert is as much a part of the domain knowledge as the cause and effect knowledge.

2. The path leading to a node is not unique, a node may be reached via different paths.

3. The mapping of the state to hypotheses is neither unique nor one-to-one mapping. It is possible that states exist that do not point to any fault hypotheses. In such cases the domain expert usually runs a number of standard tests to generate more data. The additional observations take the Working Set to a different state where corresponding fault hypotheses can be generated. In the case that more than one fault hypotheses is generated in a state, these are placed on a hypotheses queue to be processed in succession.

4. The hypotheses queue may not be empty when a terminal node is reached. The objective of a service expert is put the equipment back into operational condition. If one of the hypotheses on the queue has been confirmed and the equipment has been successfully repaired and operational, there is no further need or interest to explore the remaining hypotheses.

3. KNOWLEDGE REPRESENTATION

Instead of the usual knowledge representation as causal rules, objects are chosen to represent the domain knowledge. An object could be a physical entity, an operation to be performed, an observation, or an abstract concept. An object is represented in a slot-and-filler data structure, and associated with it is a set of operations that it would perform in response to certain messages that it understands. A message directed to an object specifies the operation to be performed, but not how it should be carried out. It is the receiving object that will determine the method by which the requested operation should be carried out. For instance, voltage across a bend magnet is an object in the class of voltage readings. All objects of this class respond to the read voltage message by returning a value of the voltage at an assigned test point. The voltage could be directly measured if sensor connections are available to do so, or a message is to be displayed on the screen to ask the human operator to take the measurement, and key in the value from the keyboard. To the sender of the message this is transparent, as it only expects the value of the voltage to be returned, and not in the manner it is acquired.

For the equipment servicing domain, we identified five classes of objects that collectively capture the domain knowledge. These are identified as: observation-object, fault-object, test-object, component-object, and repair-object.

Observation-Object

A observation-object is a description of an observable behavior of the system. That could be the visual observation of a human operator, measurements by a voltmeter, or a reading of a direct sensor. The set of accumulated observation-objects represents the domain expert's current understanding of the equipment condition.

Observation-objects are the working data in the reasoning process. They are continuously generated as various tests are performed by the service expert. The tests can be as simple as visually observing a blinking light, to a complex series of setup measurements. The current set of observation-objects is said to be in the Working Set of observation-objects. As each piece of observation is added to the Working Set, the service expert understands more about the condition of the equipment. From the Working Set, he would be able to hypothesize the faults that may contribute to the current behavior of the system, or performs additional tests if the current set of observations are insufficient to support any hypothesis. The observation-object contains the following:

1. a description of the observation,
2. the test that generates this observation,
3. possible faults, if any, that relates to this observation.

Fault-object

A fault-object represents a specific fault in the equipment. The fault-object contains the following:

1. a description of the fault,
2. the location of the fault,
3. tests and expected results that will confirm the fault,
4. repair action to be taken if the fault is confirmed.
5. frequency of occurrence of this fault.

The domain knowledge of the service expert includes a large set of known faults, in this case fault-objects. A fault-object is said to be a fault hypothesis when the Working Set of observation-objects suggest it as a possible cause of the problem. All fault hypotheses are placed on a hypotheses queue to be verified.

We associate a repair action with the fault-object, rather than with the component because there could be a number of different ways for a component to be

faulty, each requiring a different repair action. Thus it is more appropriate to associated the repair action with a fault.

Component-object

A component-object describes a field replaceable unit (FRU) in the equipment being serviced. The component-object include the following:

1. A description of the component,
2. the location of the component,
3. cost of replacement,

The complete set of component-objects describes all FRU for the equipment.

Test-Object

A test-object describes a test to be performed such as taking a voltage reading at a test point, or visually inspecting the connection of a piece of wire. In a highly automated system, a test could be the sending of a command through the bus to retrieve a sensor reading from a register. Alternatively, as most expert systems are today, it could be a message displayed to the user instructing him/her to take a measurement at a specific point and key in the reading. A test-object contains:

1. A description of the test,
2 instructions to be displayed to the user,
3 possible outcomes of the test,
4. cost of performing the test,
5. warning messages, if any.

Test-objects respond to the message of perform-test by invoking the associated method of performing the test, then returning the result of the test to the sender of the message.

As a result of performing the test, observation-objects will be generated and added to the working set. The cost of the test prioritizes the fault hypotheses on the hypotheses queue.

Repair-Object

A repair-object describes a repair action. In this sense a repair-object behaves similarly to a test-object. It contains the necessary information to instruct the user to perform a repair operation:

1. A description of the repair action,
2. instructions to be displayed to the user,
3. expected results of the repair action,
4. cost of performing the repair,
5. warning messages, if any.

Repair-objects respond to the message of perform-repair, and it returns to the message sender whether the repair action has been successfully carried out. As a result of the repair action additional observation-objects may be generated.

Special Objects

All objects referenced in this expert system shell falls within the five classes described above. However, there are a few instances of objects which deserves our special attention.

Initial Test Objects

To begin the diagnosis process, the service expert starts with an overall evaluation of the equipement performance. This action is reflected as the initial test object, which is an instance of the object class of test-object. This is the test that seeds the Working Set of observation-objects to initiate the reasoning process through the graph.

Standard Test Objects

When there is insufficient evidence from the observation-objects to suggest any fault hypothesis, the service expert will resort to a standard set of tests that exercises the different functioning subsystems of a equipement in order to gain a better understanding of the operating state of the equipment. In our terminology, this set of standard tests is represented by a set of test-objects collectively referred as stand test objects.

Checkout Test Objects

At the conclusion of a repair action, the integrity of the repaired equipement must be demonstrated by the successful execution of a set of checkout tests. Depending on the complexity of the equipment, there could be one set of checkout tests, or there could be serveral sets, each set designed to checkout one subsystem of the equipment. The ckeckout tests may be the same as the standard tests for certain equipments. The checkout tests are represented as instances of test-objects.

Checkout Test Observation Objects

The checkout test observation objects is a set of observation-objects that is expected if the system is accepted to be in a fully functional state. This set of objects corresponds to the termination observations S_t.

4. DIAGNOSTIC REASONING

With the domain knowledge encoded in objects, the diagnostic reasoning process of the generic expert system shell follows the equipment servicing paradigm described above very closely. Since the need for meta level control is eliminated, the process of knowledge acquisition is greatly simplified. It has become possible to develop a knowledge acquisition tool that assists the domain experts to encode their knowledge into objects [Lim88], this represents a significant improvement over rule based knowledge acquisition tools.

A service expert begins the diagnostic reasoning process by first observing the overall performance characteristics of the equipment. The expert system duplicates this action by sending the perform-test message to the initial test object. This test-object seeds the initial Working Set of observations-objects that begins the reasoning process. The fault-objects

corresponding to this working set are located and placed in the hypotheses queue. If at any time there are insufficient observation-objects to generate any hypothesis-object, or the fault-objects on the hypotheses queue have been depleted without any one being confirmed, the standard-tests are run by sending perform-test messages to the standard-test-objects in order to generate more fault-objects to be added to the hypotheses queue. To avoid repetition of tests, a test will not be performed more than once unless a repair action has taken place between the tests.

The fault objects on the hypotheses queue are examined one at a time, in the order of least cost to verify. Thus a simple operation such as tightening a wire connection to test for a loose connection is given priority over swapping a circuit board, even though it may seem more likely the circuit board is faulty.

When a fault-object is taken from the hypotheses queue to be verified, perform-test messages are send to the associated test-objects. These test-objects return a set of observation-objects. A fault is said to be verified if the observation-objects matches the expected set of observations in the fault-object. In such a case the perform-repair message will be sent to the repair-objects associated with the fault-object to initiate the repair actions. In the case that the fault hypothesis fails to be confirmed, the next fault-object is retrieved from the hypotheses queue for consideration.

In a rule based system, the standard explanation facility is a trace through the chain of rules in response to the "HOW" and "WHY" queries by the user. This scheme originated from the MYCIN project and has become the standard explanation facility for almost all subsequent expert system shells. In our design, the objects representation provides a easy means of keeping the user informed of all reasoning steps at all times. The current fault hypothesis is automatically displayed to the user. The user does not have to interrupt the reasoning process to query "HOW and "WHY" certain actions are necessary. This increases the user confidence in the capability of the expert system.

When a repair action is successful, i.e. the observation objects returned by the repair-objects matches the expected observations, the final check-out tests are performed. For each piece of equipment there is one or more standard set of tests that guarantees the integrity of the system operations. A repaired system must normally pass one set of these tests before it can be released to the user. This set of tests corresponds to the check out test objects, which will be send the message to proceed with the testing. The observation-objects generated from the checkout tests will be added to the working set.

Successful check-out tests will be recognized by termination observations in the Working Set. The presence of the termination observations indicates that a terminal node in the knowledge state graph has been reached.

5. SUMMARY CONCLUSIONS

Currently the Generic Service Expert System Shell (GENSES) is implemented on a PC-AT in Common-lisp. PC-ATs are widely available in most company divisions due to the rapidly falling price. Common-Lisp is emerging as the standard AI language available on a large number of PC's and workstations. Implementation of GENSES in Common-Lisp makes it portable to these hardwares. Objects are implemented in Portable Common Loops developed at Xerox PARC. This implementation of Common Loops is entirely in Common Lisp, and currently runs on a number of hardwares supporting Common Lisp.

This research has departed from the usual diagnostic expert system in three important areas:

1. Objects have chosen instead of rules for knowledge representation. A library of methods is associated with each object class. The domain expert need only select the preferred method without understanding the intricacy of programming the methods.

2. The search space comprises of a network of reasoning states, each represented by a set of observations, rather than the fault tree usually employed for diagnostic expert systems.

3. The strategy for diagnostic reasoning is built into the reasoning algorithm, rather than leaving it as the responsibility of the user to craft the meta-rules for effective control.

In designing this generic service expert systems shell, the following are the primary objectives:

1. It interacts with the domain experts in their familiar language and allows knowledge acquisition with the minimal amount of knowledge engineer intervention.

2. It is friendly and assumes a minimal amount of computer literacy from the user.

3. It supports rapid prototyping of service expert systems.

4. It is applicable to a wide range of diagnostic problems covering diversified product lines.

5. It may be fielded in a low cost environment.

Towards this end the objectives have been achieved. It may be argued that some flexibility have been sacrificed as the domain experts may not be able to customize a diagnosis strategy to a special situation. This may be a small price to pay compared to the monumental task of preventing rule interference in a large rule based expert system. What have been gained is predictability in the system behavior independent of the amount of knowledge added into the system. The trend in the evolution of Expert systems is towards Application Specific Expert Systems Tools (ASEST), much as the evolution of integrated circuits is towards Application Specific Integrated Circuits (ASIC). This work is an illustration of the advantage of ASEST.

As a topic for further investigation, however, we are studying the feasibility of allowing strategy selections without reintroducing the evils of meta-rules.

6. REFERENCES

[Ben88] Ben-Bassat, Moshe , et al, "AI-TEST A Real Life Expert System for Electronic Troubleshooting", *Proceedings of The Fourth Conference on Artificial Intelligence Applications*, March 1988.

[Dav84] Davis, Randall and Jonathan King, "The Origin of Rule-Based Systems in AI", in *Rule-Based Expert Systems*, Addison-Wesley, 1984.

[Dav84a] Davis, Randall and Bruce Buchanan, "Meta-Level Knowledge", in *Rule-Based Expert Systems*, Addison-Wesley, 1984.

[Bru86] van de Brug, Arnold; Judith Bachant, and John Mcdermott, "The Taming of R1", *IEEE AI Expert*, Fall 1986.

[Cha86] Chandrasekaran, B. "Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design", *IEEE AI Expert*, Fall 1986.

[Fin85] Fink, K. Pamela, John C. Lusth, and Joe W. Duran, "A General Expert System Design for Diagnostic Problem Solving", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 7, No. 5, September 1985.

[Gom81] Gomez, F. and B. Chandrasekaran, "Knowledge Organization and distribution for Medical Diagnosis", *IEEE Trans. Systems, Man and Cybernetics*, Vol.11, No. 1, Jan. 1981.

[Lim88] Limbek, Bela E., "Generic Diagnostic Knowledge Acquisition Tool", *The First International Conference on Industrial & Engineering Applications of Artificial Intelligence and Expert Systems*, June 1988.