FOREST - AN EXPERT SYSTEM FOR AUTOMATIC TEST EQUIPMENT

Tim Finin
Computer and Information Science
University of Pennsylvania
Philadelphia PA

John McAdams
Advanced Technology Lab
RCA Corporation
Camden NJ

Pamela Kleinosky
Automated Systems
RCA Corporation
Burlington MA

1. ABSTRACT

This paper describes **FOREST**, an expert system for fault isolation and diagnosis in the Automatic Test Equipment (ATE) domain. Current ATE systems can correctly handle 90 to 95 percent of faults, but the residue accounts for a considerable cost in terms of equipment downtime and the human expert's time. FOREST is an attempt to handle residue of hard problems with current expert systems techniques. In particular, the incorporation of an AI approach allows us to handle two serious problems which the existing decision tree techniques can not: problems involving multiple faults and problems caused by components or systems which gradually drift out of calibration. FOREST is implemented in PROLOG and has an architecture that combines an object-oriented representation language (FIR), a general inferencing mechanism (PROLOG), an inferencing engine for reasoning with certainty factors (PINE) and a explanation generation system (ELM).

2. INTRODUCTION

This paper describes **FOREST**, an expert system for fault isolation and diagnosis in the Automatic Test Equipment (ATE) domain. Current ATE diagnostic software can correctly handle 90 to 95 percent of faults, but the residue accounts for a considerable cost in terms of downtime and the human expert's time to do backup diagnosis. FOREST is an attempt to handle this residue of hard problems with current expert systems techniques. In particular, the incorporation of an AI approach allows us to handle two serious problems which the existing decision tree techniques do not: problems involving *multiple faults* and problems caused by components or systems which *gradually drift out of calibration* (see [8] for additional details).

We feel that the architecture we have chosen to support FOREST has several aspects which are of general interest. We have designed an object-oriented representation language similar to KL-ONE and implemented it in Prolog. The use of Prolog provides the object-oriented language with a general inferencing mechanism that extends its expressiveness. This object-oriented language has been used in turn to build a language for defining rule-based system with both forward and backward chaining rules. In addition, these rules include mechanisms for reasoning with *certainty factors*.

In the remainder of this paper we will introduce the ATE domain, discuss some of the fault diagnosis principles we have encountered and describe the architecture of FOREST in more detail. Given the exploratory stage of our project, our conclusion is more a prescription for future research than it is a summary of results.

3. THE AUTOMATIC TEST EQUIPMENT DOMAIN

Automatic test equipment (ATE) is widely used in both industry and the military to troubleshoot electronic equipment. The ATE is a programable device which is hooked up to the faulty equipment, usually referred to as the UUT for ***Unit Under Test***. The ATE then runs a special program developed for the UUT which takes a number of measurements from it and attempts to diagnose the problem and identify the faulty component. Before taking the measurements, current sophisticated ATE systems perform a series of *self-tests* to determine that the ATE system is itself functional and calibrated.

The ATE self-test is particularly important since when the ATE system is unavailable, faulty equipment is simply shut down until a human expert can be called in to diagnosis and fix the ATE system. The ATE system we are working with performs the self-test as a two stage procedure involving a survey run followed by diagnostics. The survey run performs approximately 100 predetermined tests, records the resulting values (voltage, time measurements, etc.) and determines their status as either *pass*, *fail* or *nmc* for "no measure complete" (indicating that the amplitude of a pulse is too low to be read). If the status of any test is *fail* or *nmc* then diagnostic software is called to isolate the fault. This diagnosis is based on the first test to fail in the survey run and the results of any additional tests called for in the diagnostic program. Often the fault can only be localized to a group of circuit cards, requiring the entire group to be replaced.

The diagnostic software currently available for ATE can isolate and diagnose approximately 90 to 95 percent of the faults which occur in UUT's as well as in the ATE equipment itself. Unfortunately, the remaining 5 to 10 percent of undiagnosed faults are often those which are most difficult and costly to solve. These faults are currently diagnosed by human experts.

4. ATE DIAGNOSTIC KNOWLEDGE

The FOREST system was designed to extend rather than replace the testing strategies currently used in ATE systems. Similar work in the ATE doamin by Cantone, Pipitone and Lander [5] has focused on using expert system technology to automatically produce programs for current ATE systems. Our approach can be characterized as an attempt to emulate an experienced field engineer whose specialty was diagnosing faults undetectable by existing ATE software. This section describes our knowledge aquisition methodology and the structure and kinds of knowledge we found.

4.1. Knowledge Aquisition Methodology

Knowledge acquisition involved a three-phase interviewing process between the domain expert and a knowledge engineer. In the initial stage, the problems to be attacked were identified. The goal of the second stage was to elicit a set of stategies used by the expert to solve each of the problems. In the third stage we worked with the domain experts to find appropriate representations for the objects and relations involved in each problem and to articulate the rules more formally.

The first phase consisted of identifying faults which are beyond the abilities of conventional ATE software. This resulted not only in a well-bounded domain but it insured that the system accomplished a purpose that the expert considered significant.

The second phase involved obtaining the strategies used by the experts to diagnose these problems. We found that our domain experts had difficulties in articulating general troubleshooting strategies, so we used a technique of initially discussing *case histories*. As the expert detailed specific actions used to solve specific problems a framework of partial knowledge was developed. This initial framework was then used to generate direct questions to elicit the information needed to "fill in the holes".

In the third phase, we developed successively more formal expressions to represent the domain objects and relations and to represent rules to embody the diagnosis stategies discoved in the second stage. These wew presented to the expert for approval and any necessary revision. The process continued iteratively until both the expert and the knowledge engineer were satisfied. Sets of field data obtained from failure reports were used for preliminary testing and to refine the rule base.

4.2. The Experts' Knowledge

The knowledge which we collected from our experts can be generally divided into three categories. The first is procedural, consisting of strategies to obtain the information necessary to differentiate one type of failure from another. These are often "quick and dirty" looks which check for consistency among test results or check groups of tests without looking at a specific test result. These strategies generally take the form of experiential "rules-of thumb" and appear to be best represented as if-then rules with weights attached. In the ATE domain a rule of this type might be:

> *"If all pulse period measurements fail then look for a pulse amplitude problem."*

The second class of knowledge involves the use of circuit diagrams. In situations arising from inconsistent data or unusual failures, the expert cannot rely solely on his past experience and uses information obtained from the circuit diagram. Here he uses his knowledge of the relationships between specific test points and circuit topology to isolate possible areas of failure. These generally are straight-forward physical relationships and are so specific that they constitute a different level of reasoning. Our experts only turn to this type of problem solving when the heurisitic line of reasoning hits a dead-end and use it only as long as necessary. A relationship of this type might be expressed as:

> *"The generation of the main pulse triggers the system clock reset."*

Finally there are general electronic troubleshooting principles. These were not volunteered by the expert but were developed during the interviewing process. These principles underly the entire troubleshooting process whether the expert is using heuristic shortcuts or working through circuit diagrams. This is an important form of meta-knowledge and is part of the explanation process as well as guiding the reasoning process. To finish the previous example, a general principle would be:

> *"Insufficient amplitude of a signal may result from excessive resistance along the signal path."*

The experts were observed to integrate all three types of knowledge in most troubleshooting scenarios performing forward, backward and what may be referrred to as "vertical" reasoning. First he reasons forward from a limited set of data applying any heuristics he has developed through experience to hypothesize likely conclusions. He then attempts to narrow these down by backward chaining to identify additional data which is used to eliminate some of the candidates. If none are verifiable the expert "drops down" and continues forward using the specific "lower

level" knowledge usually obtained from circuit diagrams. At the end of a worst case scenario the expert is physically probing the low level circuit components-- usually, however, the problem will be found long before we look at this level.

5. THE FOREST ARCHITECTURE

The domain independant part of FOREST is a rule-based system implemented in Prolog (currently UNH Prolog on a Vax 11/780) which includes a general object-oriented representation language, a mechanism for dealing with certainty factors and a simple explanation system. A more complete description of the architecture we have developed can be found in [9]. The four major components are discussed in the sections to follow. They are:

- **FIR**, for Frame Inference Representation, provides a frame-based representation language on which everything else is built.
- **PINE**, for Prolog Inference Network Engine, is a language for building forward chaining inference networks. It is built up out of FIR objects.
- **ELM**, for Explanation Lending Mechanism, is a system that provides descriptions of the reasoning that supports the conclusions reached in a PINE network.
- **OAK**, for Our Actual Knowledge, is a particular PINE inference network and a collection of associated prolog predicates.

5.1. The FIR Representation System

FIR is a simple frame representation language. It allows one to define objects with various kinds of attributes, and to organize them into a taxonomy. Recently developed languages for representing knowledge such as KL-ONE [15], Krypton [4] and Nikl [16], have argued that general frame based languages need to be augmented by a general inferential mechanism. Our embedding FIR in Prolog provides a such a mechanism.

The attributes of an object include *superTypes*, *roles*, *demons* and *methods*. An object's *superTypes* are its immediate generalizations and determine its place within the taxonomy. The inheritance of roles, facets of roles, demons and methods is done with respect to this taxonomy. An object can locally own or inherit *demons* which are Prolog clauses which are executed under specified conditions, such as when an object is specialized. Similarly, an object can own or inherit *methods* which support a simple variety of message-passing semantics [14].

A *role* of an object is itself a complex entity which is composed of subparts called *facets*. These facets include:

- ***name*** - an arbitrary term used to access the role.
- ***value*** - one or more terms stored as values filling the role.
- ***default*** - a term used as a default value if no actual values are present.
- ***type*** - a FIR object which must subsume any value.
- ***range*** - a description of the number of values the role can hold.
- ***demons*** - Prolog goals which are executed under appropriate circumstances.

The demon mechanism is similar to that provided by other representation languages (e.g. KL-ONE [3], HPRL [13]) and allows one to specify demons to be executed before or after a value is added to a role, before or after a value is removed from a role and whenever a value is sought from a role which has no locally stored values.

For example, a generic node in PINE is represented (partially) by the Prolog term in figure 5-1. This states that a *node* is a specialization of a *thing* and has the locally defined roles named *description*, *mb*, *md* and *cf*. Both *mb* and *md* have a range equal to one (a range of one indicates that a role can hold only a single value). If another value is assigned to the role, it replaces the existing one.], default values of 0.0 and a demon which will be triggered just after a new value is given to the role. The *cf* role has a demon which will be triggered if one asks what value the role **has** and will compute an appropriate value.

Inheritance of roles and role facets other than the value facet is done (logically) by climbing up the taxonomy. If an object has more than one superType, then there is an implicit ordering in which they are processed. Role value inheritance is done following what Winston has termed *Z-inheritance* [18]. A depth first search is done up the hierarchy. At each object we look for one of the following (in order): a locally owned value facet, default facet or a *has*-demon which can be successfully evaluated.

FIR provides basic functions for creating and querying objects:

- **specialize(O1,O2)** - make object O1 a specialization of O2.
- **isa(X,Y)** - true if X is a role that is subsumed by role Y or if X is an object that is subsumed by object Y.

```
a node is a thing with

   a description = 'indescribable';

   a mb having
     default = 0.0;
     range = 1;
     demon = gave(mb of O,V) if propogateFromNode(O);

   a md having
     default = 0.0;
     range = 1;
     demon = gave(md of O,V) if propogateFromNode(O);

   a cf having
     demon = has(cf of O,Cf) if has(mb of O,Mb) and
                                has(md of O,Md) and
                                Cf is Mb - Md.
```

Figure 5-1: A Pine Node

- **give(R,V)** - Give role R a value V. This triggers a check for the satisfaction of any range and type constraints and triggers any *giving* or *gave* demons associated with the role.
- **give(F of R,V)** - Give facet F of role R the value V. This can be used to locally fill a role's facet with a term.
- **take(R,V)** - Remove value V from role R. Any range constraints are checked and any associated *taking* and *took* demons fired.
- **has(R,V)** - Find a value for the role. This will invoke an inheritance search for the nearest local value, default or has-demon to generate a value. Backtracking can be used to generate any additional values the role has.

A role R can be further decomposed into a name and the object owning the role. We might say, for example:

```
?- give(default of age of person, 30)
?- give(age of john, 32).
?- findAll(X,
           (isa(X,person),
            has(age of X,Age),
            Age>21),
           Adults).
```

5.2. The PINE Inference Engine

The PINE inference engine is built up out of the FIR objects shown in the inheritance hierarchy shown in figure 5-2. The approach of using a frame representation language to build a rule base and inference engine is similar to that employed by Fox [10] and also the the expert system building tools HPRL [13], KEE [12], and LOOPS [2]. The ***nodes*** represent either data about the UUT or conclusions reached by the inference network. The ***rules*** specify the method by which conclusions are derived from data and other conclusions. Truth and belief are handled in a manner similar to that used in MYCIN [17] and PROSPECTOR [7]. Each node has a measure of belief (MB), and a measure of disbelief (MD) whose values range from 0.0 (none) to 1.0 (maximal). The overall support that the system has for the proposition which a node represents is called the *certainty factor* (CF) and is defined as (MB - MD) and can range from -1.0 to +1.0, representing total disbelief and total belief, respectively.

```
thing
    rule
        logicRule
        valueRule
    node
        value
        proposition
            data
                given
                question
            conclusion
                goal
                intermediate
            logical
                andNode
                orNode
                wandNode
                notNode
```

Figure 5-2: Pine Object Taxonomy

Nodes in PINE

Nodes in PINE come in two varieties: those representing domain *values* and those representing *propositions*. A *value* node is an arbitrary variable which represents some parameter in the domain. In the ATE domain, for example, we have *value* nodes which represent the the measured voltages of each of the survey tests. A *value* node can be automatically mapped into a propositional node (with associated degrees of belief and disbelief) by *valueRules*. A *proposition* node represents an arbitrary fact about the domain and has associated degrees of belief and disbelief. Propositions can be *data*, which represents incoming knowledge about the domain, *conclusions* which represent facts derived by the inference network and *logical nodes* which are system generated nodes representing logical relationships between other nodes. Data enters the system from ***data*** objects, which will always appear as leaves in the inference network. *Data* objects are either *Givens*, propositions whose degree of belief or disbelief is directly provided as initial input, or ***Questions***, proposition whose degree of belief or disbelief is determined by asking the user when the values are first needed.

A *conclusion* is a proposition whose certainty is determined by a rule. It can be an *intermediate* or a *goal* conclusion. The user is kept informed of the status

of *goal* nodes as the system operates. The final type of node is a *logical node* which includes the usual and, *or* and *not* relations as well as the weighted and relation described later.

Rules in PINE

Rules are the arcs which connect nodes and come in two varieties: *value rules* which map values into conclusions and *logic rules* which map a Boolean combination of propositions into a conclusion.

ValueRules are needed in order to map measures taken from the ATE survey (e.g. a voltage) into propositions (e.g. the belief that a subsystem is functioning properly). In general we want to define a function from the range of the measurement into a certainty measure. For example, a voltage measurement of below 5.0v or above 10.0v for some reading may definitely be regarded as failing. Readings "near" the ideal value of 7.5v volts are definitely regarded as passing (where "near" is defined by our domain expert). The function assigns a certainty factor that a reading is within specification or has "passed" by assigning -1.0 to readings below 5.0 volts and above 10.0 volts and +1.0 to the small region surrounding the ideal reading of 7.5 volts. This leaves two "gray areas" surrounding the "ideal zone." In these areas the certainty is incremented according to how far the reading is from the defined "definite pass" and "definite fail" limits. This reflects the notion that a reading of 6.2 volts may tell the technician almost nothing about whether the test passed or failed. This "gray area" pass/fail decision making is critical to systems which gradually go out of calibration.

This mechanism we have provided in PINE is similar to the *active values* found in LOOPS [2]. Changing the value in a *valueNode* triggers the execution of a body of Prolog code in any associated *valueRules*. Figure 5-3 shows an example of a *valueRule*.

---

if mpRefSupplyVolt=V ***and*** not(between(5.0,V,7.0))
then disbelieve(mpRefSupplyVoltOK).

if mpRefSupplyVolt=V ***and*** between(7,V,8)
then believe(mpRefSupplyVoltOK).

if mpRefSupplyVolt=V ***and*** P is abs(V-7.5)/2 -0.25
and between(0,P,1)
then believe(mpRefSupplyVoltOK,P).

Figure 5-3: a Rule Mapping a Value to a Conclusion

LogicRules map a Boolean combination of propositional nodes into a conclusion and have the following roles:

- **Evidence** - Nodes which serve as evidence for the rule.
- **Conclusion** - The node which is the conclusion of this rule.
- **Pw** - Positive weight of the rule.
- **Nw** - Negative weight of the rule
- **Support** - A measure of the evidence that the rule currently provides the conclusion.

The positive weight (PW) of a rule measures its sufficiency and the negative weight (NW) its neccessity. as in PROSPECTOR [7]. The value of the support role is defined as the product of CF_e (the certainty factor of the evidence for the rule) and PW, if CF_e is greater than 0, or NW if if CF_e is less than 0. The support given by a "logical and" rule is determined by the minimum of the CF's of the evidence nodes. The support given by a "logical or" rule is the maximum of the CF's of the evidence nodes. "Weighted and" rules' support is the weighted average of the CF's of the evidence nodes. Actual updating of the corresponding CF is performed using MYCIN's updating rule:

$$CF_{new} = CF_{old} + (1 - CF_{old}) * \text{ruleSupport}$$

Complex rules can be given as a Boolean combination of nodes. As the Boolean combination is parsed, new intermediate nodes are created to serve as logical nodes, one for each logical connective (and, or, not, wand [weighted and]). The bottommost logical node is connected, via the original rule, to the conclusion. New rules are created which serve to connect the evidence nodes to the logical nodes. In the case of the wand node the respective weights of each conjunct's new rules are determined by the user. For the other logical nodes, the new rules are given default positive and negative weights of 1. Figure 5-4 gives some examples of *logicalRules*. The system will give the first rule in figure 5-4 a name (e.g. rule17) then parse the body of the rule to set up the evidence and conclusion links. Since the evidence here involves an "and" the system creates a new logical node (e.g. *andNode43*) which becomes the evidence of *rule17*. New logical rules (*r18* and *r19*) are also created to link the evidence into the logical node. The second example in figure 5-4 shows a *weighted and* rule.

5.3. The ELM Explanation Mechanism

ELM (Explanation Lending Mechanism) provides a simple explanation facility similar to that found in MYCIN [17]. If asked for an explanation for a node, ELM will describe the nodes which effect the belief of this conclusion -- both those which increased the support of the conclusion and those which lowered the

if 'the DP rise-time check failed "no measure complete"'
and
'the DP fall-time check failed "no measure complete"'
then 'the amplitude of the DP into the counter is insufficient'
with pw = 0.9; nw = 0.5.

if 'the dp rise-time check succeeded' @ 1
+
'the dp rise-time check succeeded' @ 2
+
then 'the amplitude of the dp into the counter is sufficient'
with nw = 0.0.

Figure 5-4: Examples of Logical Rules

support. At this point some of the reasons to believe the conclusion are probably high level intermediate conclusions themselves. These intermediate conclusions can also be explained and the network can thus be traced all the way back to data from the survey tests. An example of the explanation facility is seen in figure 5-5. If instead the data had been such that the DP rise-time check did not fail "no measure complete", i.e. the counter was able to make DP rise-time measurements, the resulting explanation would change to figure 5-6.

In giving the above rule a non-zero negative weight we mean that if there is negative belief that the evidence conditions are present then firing of this rule results in decreasing the belief in the conclusion. The presence of a non-zero negative weight implies the presence of the "virtual" rule r1 in figure 5-7. This second rule can now be simplified by the use of DeMorgan's laws to rule r2. In the explanation of an "or rule" (such as this has now become), since only one disjunct must be present to give evidence to the conclusion (or rules choose the most believed disjunct) only the most strongly believed disjunct is displayed to the user as a reason to believe the conclusion.

5.4. The OAK Knowledge Base

OAK (Our Actual Knowledge) is the knowledge base for ATE troubleshooting. The rules are combined in a network fashion where belief in each intermediate or goal node can be affected by multiple pieces of evidence and in turn be evidence for one or more further conclusions. This results in a lattice form of knowledge (as opposed to a simple binary tree) allowing relaxation of the single fault assumption and providing some diagnostic capability even in the case of incomplete or conflicting data.

In its present state of development, OAK contains a set of approximately fifty nodes and and their

THE BELIEF THAT *the amplitude of the dp into the counter is insufficient* IS 0.81

REASONS TO BELIEVE:
0.90 N14: *the dp rise-time check failed "no measure complete"*
0.85 N15: *the dp fall-time check failed "no measure complete"*

THERE ARE NO RULES WHICH LOWER BELIEF IN THIS CONCLUSION.

WOULD YOU LIKE FURTHER EXPLANATION? >>

Figure 5-5: An Explanation of a Conclusion

THE BELIEF THAT *the amplitude of the dp into the counter is ok* IS 0.50

REASONS TO BELIEVE:
0.99 N14: *the counter was able to make a dp rise-time measurement*

THERE ARE NO RULES WHICH LOWER BELIEF IN THIS CONCLUSION.

WOULD YOU LIKE FURTHER EXPLANATION? >> ***N14***

THE BELIEF *that the counter was able to make a dp rise-time measurement* IS 0.99

REASONS TO BELIEVE:
1.00 n2: *result of test 14040 was a reading of 80.05 usec*
WAS GIVEN INFORMATION.

Figure 5-6: An Explanation

r1 : *if not*('DP rise-time check failed "nmc"' *and*
'DP fall-time check failed "nmc"')
then not('DP amplitude into counter is insufficient')

r2 : *if* 'counter able to make DP rise-time measurements' *or*
'counter able to make DP fall-time measurements'
then 'DP amplitude into the counter is sufficient'

Figure 5-7: Equivalent Rules

associated rules. This covers the pulse generation sussystem of the ATE and accounts for about one quarter of the ATE self-test circuitry.

6. CONCLUSIONS AND FUTURE DEVELOPMENTS

The use of artificial intelligence techniques, in particular a knowledge-based expert systems, has proven to be beneficial in supplimenting the fault detection and isolation capabilites of current ATE diagnostic software. The FOREST system can provide diagnositic capability

in the presence of incomplete or conflicting data and can deal with two problems that conventional ATE systems can not: multiple-fault failure and calibration errors.

Human experts appear to use three types of knowledge in troubleshooting electronic equipment: procedural heuristics based on experience, reasoning from circuit diagrams, and general troubleshooting meta-rules. FOREST currently contains rules of all three types but represents them at the same level. Our primary goal for the future is to seperate these kinds of knowledge and to represent them in different ways. This will allow us to encorporate more model-based and functional knowledge in the system, as have recent work by Genesereth [11] and Davis [6]. We feel that the architecture we have used for FOREST will allow us to experiment with different strategies for representing meta-rules.

Other development efforts center around three major areas. The first is aimed at providing the system with the ability to update its rule base by drawing conclusions from past failures on the unit. The system as currently designed will keep a history of failures and data values as well as any desired statistical manipulations of these numbers. A second area we are interested in is the possibility of using the system for prediction of failures. The third area involves the use of a FOREST like system to train human troubleshooting experts.

7. BIBLIOGRAPHY

[1] Bobrow, D. et. al. LOOPS Documentation. Xerox Palo Alto Research Center, 1983.

[2] Brachman, Ronald. A Structural Paradigm for Representing Knowledge. Technical Report 3605, Bolt Beranek and Newman Inc., May, 1978.

[3] Brachman, R.J., Fikes, R.E., Levesque, H.J. KRYPTON: Integrating Terminology and Assertion. Proceedings of the National Conference on Artificial Intelligence, AAAI, Washington, D.C., August, 1983, pp. 31-35.

[4] Cantone, R., F. Pipitone and W. B. Lander. Model-Based Propabalistic Reasoning for Electronics Troubleshooting. Proc. 8th Int'l. Joint Conf. on Art. Intelligence, IJCAI, August, 1983.

[5] Davis, Randall, et. al. Diagnosis Based on Description of Structure and Function. Proc. National Conf. on Artificial Intelligence, AAAI, CMU, Pittsburgh PA, August, 1982.

[6] Duda, R., Gaschnig, J., & Hart, P. Model Design in the PROSPECTOR Consultant System for Mineral Exploration. In *Expert Systems in the Micro-electronic Age*, D. Michie, Ed.,Edinburgh University Press, Edinburgh, 1979.

[7] Finin, T., P. Kleinosky and J. McAdams. Expert Systems for Automatic Test Equipment. 1984 Conference on Intelligent Systems, Oakland University, Rochester, Michigan, April, 1984.

[8] Finin, T., P. Kleinosky and J. McAdams. Forest - An Expert System for Automatic Test Equipment. Tech. Rept. MS-CIS-84-09, Computer and Information Science, Univ. of Pennsylvania, March, 1984.

[9] Fox, M., S. Lowenfeld and P. Kleinosky. Techniques for Sensor-based Diagnosis. Proc. 8th Int'l. Joint Conf. on Art. Intelligence, IJCAI, August, 1983.

[10] Genesereth, M. R. Diagnosis Using Hierachical Design Models. Proceedings of the National Conference on Artificial Intelligence, AAAI, 1982.

[11] Kehler, T.P. and Clemenson, G.D. "An Application Development System for Expert Systems." *Systems & Software* (January 1984).

[12] Lanam, D, R. Letsinger, S. Rosenberg, P. Huyun and M. Lemon. Guide to the Heuristic Programming and Representation Language Part 1 : Frames. Tech. Rept. AT-MEMO-83-3, Application and Technology Laboratory, Computer Research Center, Hewlett-Packard , January, 1984.

[13] Moon,D., R. Stallman and D. Weinreb. *Lisp Machine Manual*. MIT AI Laboratory, Cambridge, MA, 1983.

[14] Schmolze, J.G. & Brachman, R.J. Proceedings of the 1981 KL-One Workshop. Tech. Rept. 4842, Bolt Beranek and Newman Inc., Cambridge MA, 1982. Also FLAIR TR-4, Fairchild Lab for AI Research

[15] Schmolze, J.G., and Lipkis, T.A. Classification in the KL-ONE Knowledge Representation System. Proc. IJCAI-83, Karlsruhe, W. Germany, 1983.

[16] Shortliffe, E.. *Computer-based Medical Consultations: MYCIN*. Elsevier, New York, 1976.

[17] Winston, P.. *Artificial Intelligence*. Addison-Wesley, 1983.