ACTES

SIXIEME CONFERENCE CANADIENNE SUR
L'INTELLIGENCE ARTIFICIELLE

PROCEEDINGS

SIXTH CANADIAN CONFERENCE ON
ARTIFICIAL INTELLIGENCE

Commanditée par:
La Société canadienne pour l'étude de l'intelligence par ordinateur

Sponsored by :
Canadian Society for Computational Studies of Intelligence

ÉCOLE POLYTECHNIQUE DE MONTRÉAL
MONTRÉAL QUÉBEC CANADA

21 - 23 mai / May 1986

ISBN 2-7605-0409-3

Dépôt légal — 2e trimestre 1986
Bibliothèque nationale du Québec
Bibliothèque nationale du Canada
Imprimé au Canada

DEVICE REPRESENTATION USING INSTANTIATION RULES AND STRUCTURAL TEMPLATES†

Mingruey R. Taie, Sargur N. Srihari, James Geller and Stuart C. Shapiro

Department of Computer Science
State University of New York at Buffalo
Buffalo, NY 14260, USA
taiemr%buffalo.csnet-relay

Abstract — A device representation scheme for automatic electronic device fault diagnosis is described. Structural and functional descriptions of devices (which are central to design-model-based fault diagnosis) are represented as instantiation rules and structural templates in a semantic network. Device structure is represented hierarchically to reflect the design model of most devices in the domain. Each object of the device hierarchy has the form of a module. Instead of representing all objects explicitly, an expandable component library is maintained, and objects are instantiated only when needed. The component library consists of descriptions of component *types* used to construct devices at all hierarchical levels. Each component *type* is represented as an instantiation rule and a structural template. The instantiation rule is used to instantiate an object of the component *type* as a module with I/O ports and associated functional descriptions. Functional description is represented as procedural attachments to the semantic network; this allows the simulation of the behavior of objects. Structural templates describe sub-parts and wire connections at the next lower hierarchical level of the component *type*. Advantages of the representation scheme are compactness and reasoning efficiency.

INTRODUCTION

First Generation diagnostic expert systems, such as MYCIN [10] for medical diagnosis and CRIB [5] for computer hardware fault diagnosis, are built on empirical rules that associate observed symptoms with possible fault (disease) hypotheses. While these systems are considered successful, experience has shown significant drawbacks in their design methodology: knowledge acquisition from domain experts is difficult; all possible faults (diseases) have to be enumerated explicitly, which results in limited diagnostic power; and they have almost no capability of system generalization.

Structural and functional descriptions, usually referred to as "design models" of a device, have been suggested as a solution to the difficulties of empirical rule-based diagnosis systems in knowledge acquisition, diagnosis capability, and system generalization [1, 2, 4]. Such systems are referred to as "design model-based" or "specification-based" as opposed to first generation systems which are "symptom-based"[6]. Diagnostic architectures for combining symptom based and specification-based reasoning have also been proposed [11].

† This work was supported in part by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and the Air Force Office of Scientific Research, Bolling AFB DC 20332 under contract No. F30602-85-C-0008.

The present work focuses on knowledge representation for design-model-based diagnosis. The knowledge needed for building such a system is well-structured and readily available at the time when a device is designed. There is no need to explicitly enumerate all possible faults since they are defined generically as violated expectations at the output ports. This approach makes adaptation of the system to a new device much easier, because all that is needed is to describe the device to the system.

Since a design model-based fault diagnosis system reasons directly on the structure and function of a device and usually uses a simple inference engine, the representation of the device is vital to system performance. We use a hierarchical representation of knowledge to provide abstraction levels of devices. This allows a fault diagnosis system to focus on either individual objects or on several objects at a time.

Compactness of device representation is desirable for memory economy and diagnostic reasoning efficiency. It is observed that many parts of an electronic device often have the same component type and thus show the same function. Therefore we find that representing every detail of a device creates unnecessary redundancy. Instead of representing all objects explicitly, an expandable component library is maintained, and objects are instantiated only as needed. An object, which may be the device itself or a sub-part of it at any hierarchical level, is represented as a module.

The component library consists of descriptions of all component *types* used to construct the devices at all abstraction levels. Each component *type* is in turn abstracted at two levels: at level-1, it is a module (a black box in the usual sense) with I/O ports and functional descriptors; at level-2, sub-parts and wire connections are envisioned. In a previous implementation, two instantiation rules were used for the representation [9], this was satisfactory for simple cases, but performance degraded when dealing with more complex devices. In this paper, we present a new device representation scheme that uses both instantiation rules and structural templates in a semantic network. Functional description is represented as a procedural attachment to the semantic network. This allows the simulation of the behavior of objects.

The representation scheme has been used to represent several devices, including several multiplier/adder boards and a six channel PCM (Pulse Code Modulation) board for telephone communication, in a Versatile Maintenance Expert System (VMES) [9]. The result shows that the representation scheme is effective, and that SNePS [7], the semantic network processing system used as an underlying representation tool and inference package, is suitable for this purpose.

In the following sections, details of the representation scheme are described, an example of using the representation scheme for electronic circuit board trouble-shooting is presented, and the method of "lazy instantiation" is investigated.

REPRESENTATION SCHEME

To build a design-model-based fault diagnosis system, it is necessary to extract structural and functional information from the design model of the device. This information has to be represented in an appropriate formalism. One way to represent the device is to describe every detail of the device directly to the system. This could lead to inefficiencies in memory usage and in system development. Instead of hand-coding every detail of the device, VMES keeps a component library which describes every "*type*" of component.

The representation scheme is implemented as a semantic network for several reasons. The semantic network representation has long been around as a knowledge representation technique for expert systems [3]. It is able to represent subset and element taxonomic information, and has the potential for a smooth interface with natural language subsystems [3]. Second, a printed circuit board can be viewed as a constrained network, and it is very natural to represent it as a semantic network. Third, SNePS provides mechanisms for representing both declarative and procedural knowledge; the former is good for representing device structure, and the latter for device function.

In the representation scheme, each component type is abstracted at two levels and represented by a SNePS rule and a SNePS assertion. The former is categorized as an "instantiation rule", and the latter a "structural template". The structural representation reflects the part hierarchy of a device. Sub-parts of a device are instantiated only when they are needed. This increases memory efficiency.

Level-1 Abstraction: Instantiation Rule for I/O Ports and Function

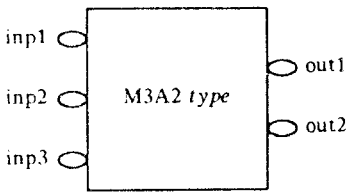
At level-1 abstraction, knowledge about a component *type* is represented as a SNePS rule. The rule is used later on to instantiate an object of the component *type* as a module with its own I/O ports and associated functional descriptor. The functional descriptor contains information about the functional description of the component *type*. The representation of the level-1 abstraction of component *type* "M3A2" is shown in Figure 1. (M3A2 is an artificial board which consists of three multipliers and two adders.) Its structure is shown in Figure 2.

Figure 1(a) shows the level-1 abstraction of the M3A2 *type*. The function of the component is abstracted as mathematical equations. This is good for digital circuits in general. Figure 1(b) and 1(c) contain our representation for the abstraction.

The first three lines of the instantiation rule shown on Figure 1(b) say that "if x is an M3A2 and is to be instantiated at level-1 abstraction (TBI-L1A), then do the following". The next five lines will instantiate the I/O ports of the object when this rule is fired. I/O ports of an object are the places where the input/output values of the object are stored. Measured (observed) I/O values depict the real behavior of the object, and calculated I/O values show its expected (normal) behavior. The last two "builds" create the functional descriptors of the object. The function of an object in the domain can be best abstracted as the relation between its inputs and outputs. The first one says "in order to simulate the value of the first output, use the function M3A2out1 which takes three parameters namely the inputs of the object x in order". Similar functional descriptors can be included for the input ports if the inference of input value from outputs and other inputs is desired (these are not shown in the figure).

The functional description should be usable to simulate the component behavior, i.e., to calculate the values of output ports if the values of the input ports are given. It should also be usable to infer the values of the input ports in terms of the values of other I/O ports. This is important if hypothetical reasoning is used for fault diagnosis. Though at this stage, VMES only uses the functional description to calculate values at output ports, our representation scheme can be used both ways.

As shown in Figure 1(b), the functional descriptor of a port contains a pointer to its functional description as well as other information concerning the use of the functional description. The functional description itself is implemented as a LISP function (see Figure 1(c)), which calculates the desired port value in terms of the values of other ports. Every port of a component type has such a function associated with it. Some more discussion about functional representation is given in Section 4..



out1 = inp1*inp2 + inp1*inp3
out2 = inp1*inp3 + inp2*inp3

Figure 1(a). Level-1 abstraction of component *type* M3A2.

```
(build
   avb $x
   ant (build object *x type M3A2 state TBI-L1A)
   cq  ((build inport-of *x inp-id 1) = INP1
        (build inport-of *x inp-id 2) = INP2
        (build inport-of *x inp-id 3) = INP3
        (build outport-of *x out-id 1) = OUT1
        (build outport-of *x out-id 2) = OUT2
        (build port *OUT1 f-rule M3A2out1
               pn 3 p1 *INP1 p2 *INP2 p3 *INP3)
        (build port *OUT2 f-rule M3A2out2
               pn 3 p1 *INP1 p2 *INP2 p3 *INP3]
```

Figure 1(b). Instantiation rule for the level-1 abstraction of component *type* M3A2: I/O ports and functional descriptors. Variables are shown in italics, and "*" is a SNePS macro for variable value substitution.

```
(defun M3A2out1 (inp1 inp2 inp3)
   (plus (product inp1 inp2)
         (product inp1 inp3]

(defun M3A2out2 (inp1 inp2 inp3)
   (plus (product inp1 inp3)
         (product inp2 inp3]
```

Figure 1(c). Functional description of component *type* M3A2.

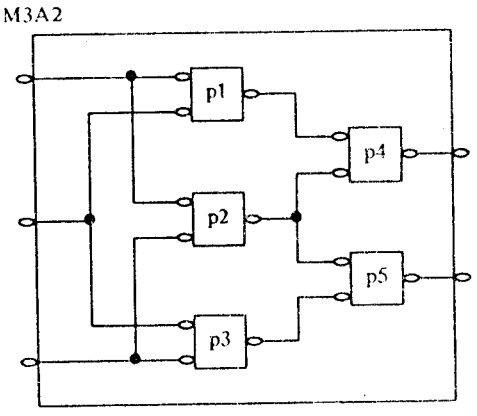
Level-2 Abstraction: Structural Template for Subparts and Wire Connections

At the level-2 abstraction, a structural template, which is implemented as a SNePS assertion, is used to describe the sub-parts of the object at the next hierarchical level, and the wire connections between the object and its sub-parts, as well as those among the sub-parts themselves. In figure 2(a), the abstraction of component *type* M3A2 at this level is illustrated. Note that the sub-parts are abstracted at their own level-1 abstraction, i.e., modeled as modules with I/O ports. The component *types* of sub-parts are also indicated.

The structural template representation is shown in Figure 2(b). The representation can be viewed as consisting of three parts. The first part, which is the second line of Figure 2(b), denotes that the representation is the structural template (ST) for component *type* M3A2 at level-2 abstraction (L2A). The second part describes the sub-parts. Associated with each sub-part are a part-id, an ext-name, and a class indicator. The part-id identifies the sub-part of the component *type*. The ext-name is for name extension, and class is the component *type* of the sub-part. This information is used for instantiating a sub-part. For example, if when diagnosing a device D1 of *type* M3A2, the second sub-part (with part-id M3A2-p2 inside the structural template) is found suspicious, then an object is created with a name of D1-M2 and a type of MULT. The last part of the structural template specifies the wire connections shown in Figure 2(a).

A structural template provides the necessary knowledge about the sub-structure of all objects of the same component *type* without representation overhead. Unlike instantiation rules, structural templates are never executed (fired) to produce a representation for any specific object. When reasoning on the sub-structure of an object is required, instead of instantiating the sub-structure (all the sub-parts and wire connections) and then reasoning on the resulted representation, we do it directly on the structural template of the object. If suspicious sub-parts are located, they (but not all sub-parts) are instantiated at the level-1 abstraction by the instantiation rules for further examination.

Device representation by instantiation rule and structural template is very compact and effective. In the next section, an application example of using this representation scheme is demonstrated.



Multiplier: p1, p2, p3
Adder: p4, p5

Figure 2(a). Level-2 abstraction of component *type* M3A2.

```
(build
  type M3A2 state ST-L2A
  sub parts
    ((build part-id M3A2-p1 ext-name M1 class MULT)
     (build part-id M3A2-p2 ext-name M2 class MULT)
     (build part-id M3A2-p3 ext-name M3 class MULT)
     (build part-id M3A2-p4 ext-name A1 class ADDER)
     (build part-id M3A2-p5 ext-name A2 class ADDER))
  connections
    ((build from (build inport-of M3A2 inp-id 1)
          to   ((build inport-of M3A2-p1 inp-id 1)
                (build inport-of M3A2-p2 inp-id 1)))
     (build from (build inport-of M3A2 inp-id 2)
          to   ((build inport-of M3A2-p1 inp-id 2)
                (build inport-of M3A2-p3 inp-id 1)))
     (build from (build inport-of M3A2 inp-id 3)
          to   ((build inport-of M3A2-p2 inp-id 2)
                (build inport-of M3A2-p3 inp-id 2)))
     (build from (build outport-of M3A2-p1 out-id 1)
          to   (build inport-of M3A2-p4 inp-id 1))
     (build from (build outport-of M3A2-p2 out-id 1)
          to   ((build inport of M3A2-p4 inp-id 2)
                (build inport-of M3A2-p5 inp-id 1)))
     (build from (build outport-of M3A2-p3 out-id 1)
          to   (build inport-of M3A2-p5 inp-id 2))
     (build from (build outport-of M3A2-p4 out-id 1)
          to   (build outport-of M3A2 out-id 1))
     (build from (build outport-of M3A2-p5 out-id 1)
          to   (build outport-of M3A2 out-id 2))]
```

Figure 2(b). Structural template for the level-2 abstraction of component *type* M3A2: sub-parts and wire connections.

AN APPLICATION EXAMPLE

We are developing a versatile maintenance expert system (VMES) for digital circuit trouble-shooting [8]. VMES is intended to be versatile in several senses: good for a wide range of devices in the domain; good for most common faults in the domain; and able to communicate with the user by several media [9]. VMES consists of two major modules: an expandable component library for device representation and an inference engine for diagnostic reasoning. The representation scheme described above is used for the current implementation of the component library.

Inference Engine of VMES

The inference engine for fault diagnosis follows a simple control structure. It starts from the top level of the structural hierarchy of the device and tries to find output ports that violate an expectation. "Violated expectation" is defined as a mismatch between the expected (calculated) value and the observed (measured) value at some output. After detecting a violated expectation, the system reasons on the structural template to find a subset of components at the next lower hierarchical level which might be responsible for the bad outputs. This process is then continued with the suspicious parts. A part is declared faulty if it shows some violated expectation at its output port and it is at the bottom level of the structural hierarchy. The bottom of the hierarchy will contain the smallest replaceable units for the intended maintenance level. In other words, if a device can be replaced but not repaired in a certain situation, then there is no need to represent its internal structure.

The inference engine is a rule-based system implemented in SNePS. The control flow is enforced by a LISP driving function called "diagnose". SNePS can do both forward and backward inference, and is capable of doing its own reasoning to diagnose a fault. The LISP driving function has been introduced for efficiency reasons only.

A small set of SNePS rules is activated at every stage of the diagnosis. For example, three rules are activated when reasoning about a possible violated expectation of a specific port of a device. One rule is to deduce the measured value of the port. If the value can not be deduced from the wire connections, the rule would activate a LISP function which asks the user to supply one. A similar rule is activated for the calculated value, and the last rule is used to compare the two values to decide if there is a violated expectation. Figure 3 shows the last rule in both SNePS code and in English.

In SNePS code:

```
(build
  avb (Sp $vc $vm)
  &ant ((build port *p value *vc  source calculated)
        (build port *p value *vm source measured))
  cq (build
       min 1 max 1
       arg (build name: THEY-MATCH p1 *vc p2 *vm)
       arg (build port *p state vio-expct))
```

In English:

If the calculated and measured values of port p are vc & vm, one and only one of the following statements is true:

(1) vc and vm agree;

(2) port p displays a violated expectation.

Figure 3. SNePS rule for detecting violated expectation at output ports.

The diagnosis strategy along with the combination of a LISP driving function and SNePS rules turns out to be very efficient. The diagnosis can be monitored by the SNePS text or graphic inference trace.

A Diagnostic Example

Figure 4 shows the representation scheme used by VMES in diagnosing a multiplier/adder board. Again, the component *type* M3A2 is used as our example.

We first name the board D1. Figure 4(a) shows the result of instantiating D1 using the instantiation rule for M3A2 *type* device (see Figure 1(b)). After the instantiation, D1 has its own I/O ports and functional descriptors, and thus its I/O values can be assigned. The result of value assignments is also shown in Figure 4(a). Then the inference engine begins to check the outputs of D1 by using the functional description of D1. It concludes that there is a violated expectation at the first output port of D1 as shown in Figure 4(b), since the expected value, which is calculated using the functional description, should be a "4" instead of the observed "2".

At this stage, it is necessary to check the substructure of D1 to locate the faulty parts. Thus VMES turns to the structural template for M3A2 (see Figure 1(c)), i.e., the component *type* of D1. From the wire connection depicted by the structural template, VMES determines that sub-parts p1, p2, and p4 of D1 may be responsible for the malfunctioning of D1. This is shown in Figure 4(c). Note that sub-part p2 may be excluded if a "single fault assumption (SFA)" is made for diagnosis. The reason is that a faulty sub-part p2 is inconsistent with the observed behavior of D1 under SFA. In other words, a bad output of sub-part p2 should cause violated expectation at both output ports of D1, but this is not the case.

Suppose SFA is not made for this example. The next step is to instantiate all suspicious sub-parts of D1, and move the diagnosis process to those sub-parts. Figure 4(d) shows the instantiation of these sub-parts. Sub-part p1 is instantiated as D1-M1, p2 as D1-M2, and p4 as D1-A1 using the information supplied by the structural template of the component *type* of D1, i.e., M3A2. Note that sub-parts p3 and p5 are not touched at all. This is the main advantage of this representation scheme.

Now the diagnostic reasoning process moves to D1-M1, D1-M2, and D1-A1 with the same inference strategies used for diagnosing D1. As shown in Figure 4(e), D1-M1 and D1-M2 show no problem, but D1-A1 shows a violated expectation at its output. The process will turn to the structural template of the component *type* of D1-A1 (an ADDER) if D1-A1 is not an object at the bottom level of the structural hierarchy. This is not the case in this simple example, where D1-A1 is an SRU (smallest replaceable unit) for the intended maintenance level. Therefore, D1-A1 is finally identified as the faulty part.

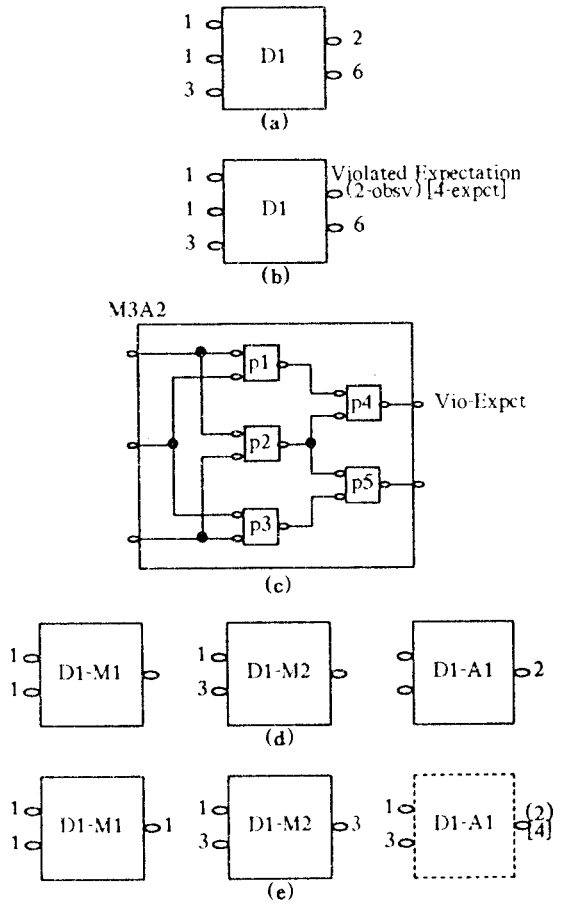


Figure 4. A diagnostic example for the device representation scheme. (D1 is an M3A2 *type* device).

DISCUSSION

Hand coding every detail, i.e., all sub-parts at all hierarchical levels, of a device is inefficient. It results in unnecessary redundancies in device representation since many parts of a device may be identical in the domain of electronic circuit boards. For instance, the six PCM chips on a six channel PCM board are exactly the same.

Representing a device as a hierarchically arranged set of objects, each of which is modeled as a module, is hardly a new idea. What is significant in our representation scheme are the clear distinction between the two levels of the abstraction and the use of an instantiation rule and a structural template to represent the different levels. The representation scheme along with an expandable component library leads to several important advantages: compact representation and system efficiencies in both system development and operating phases.

We first claim that a clear distinction between the two level abstractions of an object is desirable. In some points during diagnosis, we would like to treat an object as a complete black box — that means only the knowledge from the level-1 abstraction, which consists of I/O ports and a functional description of the object, is needed. To represent the sub-structure of the object, which is the level-2 abstraction, together with the level-1 abstraction is inefficient, since the sub-structure of the object may never be needed.

The use of structural templates to represent the substructure of objects of a component *type* has advantages over a procedural representation which uses a procedure or an instantiation rule for it [1,9]. Whenever it is needed to reason about the substructure of an object, it is carried out on the unique structural template for the component *type* of the object. Only the subparts that requires further examination will be instantiated (by the proper instantiation rules for them). Unlike the structural template representation, a procedural representation is used to instantiate "all" sub parts of an object, and then the reasoning is carried out over the resulting substructures. This creates unnecessary representation, and thus is memory inefficient. This is also execution inefficient due to the overhead of instantiating all sub-parts. An extreme example is an object with one hundred sub-parts at the next hierarchical level, only three of them needing further investigation. The advantage of the structural template is quite significant in memory and processor critical environments, such as the widespread microprocessor based computers.

Since different ports of different component *types* might have the same function, some functions can be shared. For instance, the simple function "ECHO" defined as:

(defun ECHO (inp) inp)

is shared by several different component *types* namely by the type "super-buffer", the type "wire" and the type "1 to-1 transformer". All these component *types* show the same behavior at our level of component abstraction: they echo the input to the output. As depicted above, the functional description is versatile in that it supports the simulation and the inference of the device behavior; it also supports hypothetical reasoning and the representation scheme is quite simple.

Along with the representation scheme using instantiation rules and structural templates is the idea of an expandable component library. This makes life very easy in adapting VMES to other devices. All that the user has to do is to add the structural and functional information of the "new" component *types* to the component library. A new component *type* is defined as a component *type* which has not been described to the component library. The new device itself is a new component type by our definition. The effort required to adapt the system to new devices should be minimal since digital circuit devices have a lot of common components, and the structural and functional description are readily available at the time a device is designed.

In order to test this idea as well as the suitability of hierarchical structural representation, we invented another artificial device *type* called XM3A2 and entered its description into the system. The XM3A2 *type* has three inputs and two outputs exactly like the M3A2 *type*, but it only has a single subpart which is of M3A2 *type*. Actually, it is a device which has an extra layer of packaging on top of an M3A2 *type* device. Given that the M3A2 *type* has been known to the system, only the XM3A2 *type* had to be added, which was done by adding a simple instantiation rule and a simple structural template. There was no need for a new functional description since the function of XM3A2 is the same as the function of M3A2. The XM3A2 device has three levels of structural hierarchy, and our test successfully found the faulty part at the lowest level. Though the example of XM3A2 is somewhat simplistic, it shows the capability of our system to deal with a wide range of devices in the domain with arbitrary complexity. Actually, a real six channel PCM board has been represented and a fault has been successfully located.

Acknowledgement

The authors would like to thank Scott S. Campbell, Dale Richards and Norman Sturdevant for their help and useful comments on the development of VMES.

References

1. R. Davis and H. Shrobe, "Representing Structure and Behavior of Digital Hardware," *Computer*, pp. 75-82 (Oct. 1983).
2. R. Davis, "Diagnostic Reasoning Based on Structure and Behavior," *Artificial Intelligence* **24** pp. 347-410 (1984).
3. R. O. Duda, P. E. Hart, N. J. Nilson, and G. L. Sutherland, "Semantic Network representations in Rule Based Inference Systems," in *Pattern-Directed Inference Systems*, ed. F. Hayes-Roth,Academic Press, New York (1978).
4. M. R. Genesereth, "The Use of Design Descriptions in Automated Diagnosis," *Artificial Intelligence* **24** pp. 411-436 (1984).
5. R. T. Hartley, "How Expert Should an Expert System Be?," pp. 862-867 in *Proceedings of the 7th International Joint Conference on AI*, (August 1981).
6. J. J. Richardson, R. A. Keller, R. A. Maxion, P. G. Polson, and K. A. DeJong, "Artificial Intelligence in Maintenance: Synthesis of Technical Issues," AFHRL-TR 85-7, Airforce Systems Command, Air Force Human Resources Laboratory, Brooks Air Force Base, TX 78235 (1985).
7. S. C. Shapiro, "The SNePS Semantic Network Processing System," pp. 179-203 in *Associative Networks: The Representation and Use of Knowledge by Computers*, ed. Nicholas V. Findler,Academic Press, New York (1979).
8. S. C. Shapiro, S. N. Srihari, J. Geller, and M. R. Taie, "A Fault Diagnosis System Based on an Integrated Knowledge Base," *IEEE Software* 3(2)(March, 1986).
9. S. C. Shapiro, S. N. Srihari, M. R. Taie, and J. Geller, "VMES: A Network-Based Versatile Maintenance Expert System," pp. (to appear) in *Proc. of 1st International Conference on Applications of AI to Engineering Problems*, . Southampton, U.K. (April 1986).
10. E. H. Shortliffe, *Computer-Based Medical Consultations: MYCIN*, American Elsevier North Holland, New York (1976).
11. Z. Xiang and S. N. Srihari, "A Strategy for Diagnosis Based on Empirical and Model Knowledge," pp. (to appear) in *Proc. of Sixth Int. Workshop on Expert Systems and their Applications*, . Avignon, France (April 1986).