DESIGN OF A KNOWLEDGE-BASED FAULT DETECTION AND DIAGNOSIS SYSTEM

Eva Hudlicka and Victor Lesser

Department of Computer and Information Science
University of Massachusetts
Amherst, Massachusetts, 01003
TR # 84-03.

CSnet address: eva.umass-cs Csnet-relay

20 March 1984

This research was sponsored by the National Science Foundation under Grant MCS-8006327 and by the Defense Advanced Research Projects Agency (DOD), monitored by the Office of Naval Research under Contract NR049-041.
This paper appeared in the Proceedings of the 17th Hawaii International Conference on System Sciences, Vol. 1, pp. 224-230.

ABSTRACT

This paper presents a design of a knowledge-based fault detection and diagnosis (FDD) component intended to function as part of an intelligent processor (agent) in a distributed problem solving system. This component will permit the system to monitor its own behavior, detect an abnormal system state, and identify the fault that caused it. Once the fault is identified, the system can repair it or reconfigure itself, thus improving its performance. The heart of the FDD component is a model of the system it is monitoring. This model represents the correct behavior of the system and the criteria for detecting deviations from this behavior. Thus it integrates information necessary for detection and diagnosis into a uniform structure. Detection is accomplished by monitoring the system state and detecting situations which violate the predefined expectations. Diagnosis is accomplished by constructing a representation of the current system state and determining the earliest points of departure from the expected behavior as represented by the correct system model. These points constitute the identified faults.

I INTRODUCTION

In large, complex systems, such as distributed problem solving-systems, independent intelligent agents (processors) cooperate on the overall solution. In order to function in this manner the agents must decide which subproblem in their area to work on, whether to satisfy other agents' demands for information, and what type of information to communicate and how often. The information upon which these decisions are based is distributed among the different agents; there is no centralized control or global database. Due to this lack of a coherent global database it is common for a given agent's model of the world to be incomplete or inaccurate. For example,

- the agent's model of the system organization is wrong and this results in communicating irrelevant information;
- the agent's model of the environment is wrong (for example, due to a faulty sensor) and this results in wasted effort as bad data is processed;
- the agent's model of its own role in the overall system is wrong resulting in its working on a subproblem that has already been solved by another agent.

Such errors at best result in degraded system performance and at worst in the system not achieving its goal at all. We call such errors *problem solving control errors*, because the control decisions made by the agents are inappropriate due to their inaccurate models of the world. Note that problem solving control errors can be due to hardware faults (as in the sensor failure in the example above).

There are two general ways to deal with faults and the errors they cause. One way is to eliminate the faults a priori and the other is to accept them as unavoidable and design the system so it can deal with them. These approaches have been termed fault intolerance and fault tolerance respectively by Avizienis [1] in the context of hardware and software. In the context of distributed problem solving there are two alternative approaches to making the appropriate control decisions to guarantee correct problem solving which are analogous to the fault intolerant and the fault tolerant methods mentioned above. One is to insure that each agent always has complete and accurate information. Unfortunately, for many distributed problem solving tasks, maintenance of such a database would require extensive synchronization and communication among the agents and consequently decrease system performance. The other way to deal with a lack of coherent global database is to accept the fact that there will be inconsistencies in the agents' knowledge and deal with the consequences as they arise.

One approach which falls within the category of fault tolerance has been described by Lesser [5] and termed the functionally accurate/cooperative (FA/C) approach. This approach is based on the philosophy that a system must deal with errors as part of its problem solving strategy rather than considering them as an abnormal situation. In order to achieve this, an FA/C system utilizes redundant views of the data as well as multiple paths to the solution. Because of this built-in redundancy an FA/C system tolerates some errors implicitly if given sufficient amount of overlapping and redundant information. It is thus suited to tolerating some types of faults such as transient faults or faults which can be overcome by using redundant information. However, there are faults which the FA/C approach cannot deal with. In situations where a fault is permanent (for example, a sensor continually distorting its data) the system wastes effort dissipating the effects of such erroneous data because it has no explicit awareness of the underlying fault. In cases where redundant information does not exist the FA/C approach clearly cannot help either. For example, suppose there is only one channel through which two agents may communicate. If this channel fails communication becomes impossible. In both these cases the system performance could be improved if it could detect such a problem state, identify the fault that caused it, and correct it or reconfigure the system.

Existing fault tolerance (FT) techniques for detection and diagnosis are not directly applicable to the distributed problem solving domain. Error detection in both hardware and software FT systems is usually accomplished by replication and voting. This technique is expensive since the entire system needs to be replicated several times and processes the same data. Thus we need a new type of detection method which is not based on complete replication of the system in order to recognize an error. Diagnosis in hardware FT systems is performed by running a battery of tests on the failed circuit or system. This approach often fails when faced with complex, non-permenent or multiple, interacting faults. Thus hardware diagnostic techniques are inadequate for systems where many complex faults may occur simultaneously. In software FT systems automated diagnosis following a failure is generally not attempted at all. Some work has been done in the automated diagnosis of faults in hardware systems using artificial intelligence techniques. Shubin et al. [8] developed a tool for automatically identifying a failed unit in PDP 11/03's. His system assumes a single fault. Genesereth [4] and Davis et al. [3] have developed systems which can identify multiple faults in a digital circuit using models of the

correct circuit structure and function.

This paper discusses a design of a component capable of fault detection and diagnosis in a FA/C distributed interpretation system. The interpretation system consists of a number of intelligent agents distributed over a two-dimensional area. The task of each agent is to track moving objects through the part of the area that it can sense via its acoustic sensors. The task of the system as a whole is to create a complete map of the environment indicating the location and type of each object.

The rest of the paper is divided into three sections. Section 2 describes the types of faults the system can handle, the models needed for detection and diagnosis, and the use of these models. Section 3 presents an example of a problem detection and fault diagnosis. Section 4 concludes the paper and discusses the contributions we expect from this research.

II MODELS FOR DETECTION AND DIAGNOSIS

A. Types of Faults FDD can Handle

The fault detection and diagnosis (FDD) component detects *problem solving control errors* which are inadequate control strategies adopted by a problem solving component as a result of incomplete or inaccurate information. Because such an error can result either from hardware failures (failed sensor or channel) or from faults specific to a problem solving system, (such as wrong values of internal parameters), our model allows the identification of both types of faults by representing the system at the appropriate level of detail. The FDD component can deal with multiple, possibly interacting, faults as well as with complex failures such as a sensor uniformly distorting its data or bursty faults.

B. Knowledge Needed for Detection

The problem of detection is to *recognize an abnormal system state.* To do this, the system must have an idea of what constitutes a normal system state. The obvious way to detect an abnormal system state would be to compare the system behavior with the correct behavior, i.e., behavior expected if the system was working in an optimal manner on deriving the correct answer. Since we are dealing with a problem solving system neither the correct answer nor the optimal approach is available a priori. Therefore we must have other methods of detecting unsatisfactory states that do not involve comparing the system's progress with the answer it is attempting to derive. There are two such methods:

1. one uses general knowledge about appropriate system behavior, about the task, and about the environment;
2. the other uses internal consistency standards based on redundancy within the system.

General knowledge about appropriate system behavior comes from expectations of what a good interpretation system should do. For example, we expect the system's confidence in its results to increase with time. We also expect it to make reasonable progress through the environment, accounting for more and more of its data as time goes on. Other system behavior criteria may include time or accuracy limits imposed by the user. In this case the system would be functioning abnormally if it did not derive an answer within a specified time limit or with specified degree of accuracy.

The system can also use domain specific knowledge which can be applied to detect an abnormal system state. In our case, the domain of acoustic sensing, we can put constraints on the types and motion of the vehicles which can function as expectations whose violation constitutes an abnormal system state. For example, vehicles can move only within some range of velocities or can change direction within some range of angles.

Finally, the system can use some general knowledge about the specific environment it is dealing with. This knowledge can be obtained by some preliminary, rough measurements of the data. For example, before the detailed, intelligent interpretation, the system can build a density map of the data. This would then serve as an expectation of the load distribution. For example, if an agent in an area known to have high density of raw data is idle, this is an abnormal system state.

Internal consistency standards can be utilized in a system which contains redundant information. This redundancy can be in terms of multiple views of the same data or in terms of multiple ways to process the data to derive the answer. Using the internal consistency measures consists of making sure that these multiple sources of information agree. Our system is ideally suited for this method since it contains both multiple view of the same data (by agents in different locations) and multiple methods of deriving the final answer (different knowledge sources taking different paths to the solution). Examples of measures expectations based on internal consistency are:

- Predictions based on partial results should be fulfilled. For example, in the distributed interpretation system, predictions are represented as goals. A goal is an expectation that a vehicle of a particular type will be in a particular region at a particualr time. Such expectations are produced by agents in the course of processing and are either fulfilled locally by the same agent or sent out to another agent which can fulfill them.

- The system parameters should have values consistent with each other and with the current operating environment of the interpretation system, (i.e., with respect to the current system configuration, data distribution, and external expectations placed on the system). For example, the interpretation system's knowledge is contained in task processing modules (knowledge sources) which are rated based on their effectiveness, the strength of the data they will utilize, and the importance of the results they will produce. These knowledge sources are rated and if they exceed some predefined threshold they are put on a queue. The highest rated knowledge source is put on the queue each system cycle. If the data is very weak and the threshold does not reflect this, it may be set so high that very few or no knowledge sources are invoked and as a result the agent is under-utilized.

- Communication areas among the different agents should be consistent and utilized. In order for two agents to communicate, their communication areas must match: the sending agent must know it should send messages to the receiving agent and vice versa. If the communication areas do not match, then an agent

may continue sending messages which, although they get over the channel, are ignored by the receiving agent because it does not know that it should be accepting them.

A distributed problem solving system contains many such examples of internal consistency and we plan to exploit them in detecting an abnormal system state.

C. Knowledge Needed for Diagnosis

The problem of diagnosis is to *identify the faults causing a given abnormal system state*. There are two approaches to this problem:

1. using a malfunction model or
2. using a model of the correct system behavior.

A *malfunction model* consists of a map between the set of possible symptoms and the set of possible faults. This model is appropriate for systems where such a symptom-fault map is simple or for black box systems whose internal structure is not understood and which can only be described by such stimulus-response type maps. Such models can be very efficient (diagnosis consists of table lookup) but with any non-trivial systems it is difficult to build a complete malfunction model. In other words, there is always one more way in which the system could fail that has not yet been catalogued. A malfunction model would fail in such a case because it would not find a match in its set of symptom-fault pairs. Malfunction models are also not feasible for detecting interacting faults because of the combinatorial explosion of the number of symptoms that would need to be catalogued.

An alternative to the malfunction model is a model of the system's intended structure and function. This model has been termed function-oriented model by Nelson [7]. Such model is appropriate if the system's structure is known, as is the case, at least in principle, in man-made systems, or if the construction of a malfunction model is unfeasible. An example of a good application for this type of diagnostic model is a man-made system intended to be self-sufficient, such as a satellite. It would be unfeasible to test the satellite for long enough time to catalogue all possible malfunctions. Even if that could be done, it would not solve all the problems since the actual working conditions of the satellite would presumably be different from the testing conditions and unpredicted faults could occur later. Since the satellite was designed by humans a complete description of its structure and function can be made available to a detection/diagnostic system and function as the correct system model.

The disadvantage of this approach is its inefficiency. Suppose that a given symptom always indicates only one fault. In a malfunction-model-based diagnosis this fault would be identified very fast. In a correct-system-model-based diagnosis this fault might take a long time to identify, depending on how far the initial symptom state was from the fault state. Furthermore, the diagnostic module would repeat its entire set of actions every time that symptom appeared although the symptom could never imply any other fault. Ideally, a fault tolerant system would start out with the complete, correct model of the system and adaptively build up a malfunction model which could then be used for more efficient diagnosis. In cases where malfunction model was inadequate the system could always fall back on the underlying model of the correct system behavior. Experienced diagnosticians, such as physicians, seem to use this type of strategy [9]. As a first step towards this approach to diagnosis we have developed a method based on the correct system model.

We model the distributed problem solving system as a set of objects and the set of states these objects undergo during processing (see Figure 1). The states of the objects are represented as nodes in a graph and are connected by state transition arcs which capture the causality of events; i.e., if state A preceeds state B then this means that state A must

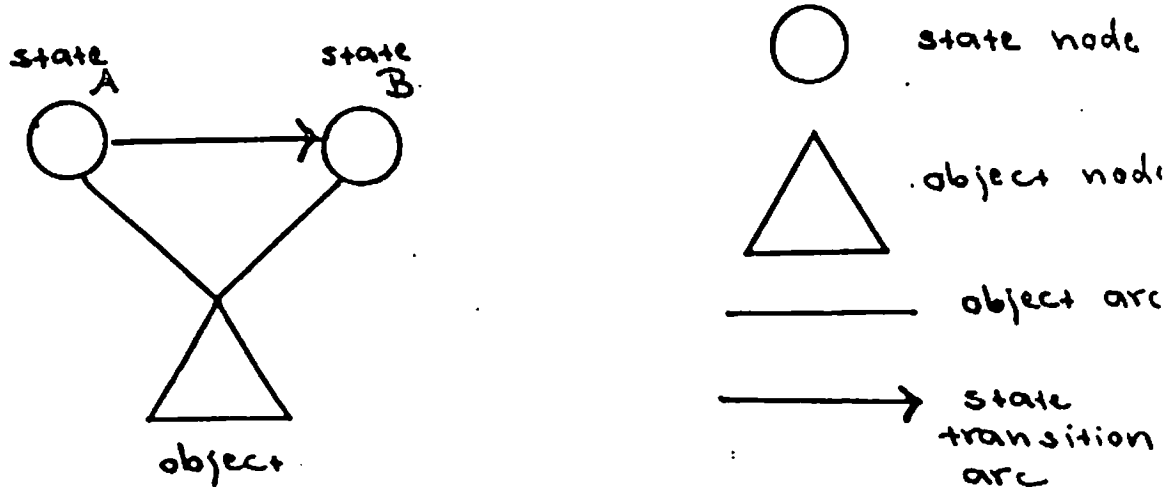


Figure 1: System Model.

be true before state B can take place. Different states of the same object are connected together via a node in the graph representing the object. Examples of objects in our system are hypotheses (representing results derived by the system from the initial sensor data), goals (representing predictions of future hypotheses based on hypotheses constructed so far), knowledge sources (pieces of code actually performing the interpretation task), and knowledge source queues. The graph representing the objects and their states is organized into small clusters for manageability. These clusters are then organized in a hierarchy corresponding to a top-down view of the system. For example, a high level cluster might represent an event as two states, state A followed by state B. A cluster at a lower level of the hierarchy would contain other events which occur between state A and state B. Such a hierarchical structure makes the detection more flexible and the diagnosis more efficient.

Detection flexibility is achieved by monitoring the system state at the level necessary rather than always monitoring the system at the lowest level of detail, as would be necessary without a hierarchical representation. The lower the level of detail at which the monitoring takes place, the less the errors are able to propagate and the less damage they do. Thus depending on the accuracy and speed required for a given task, the monitoring can be moved up or down the hierarchy.

Efficiency of diagnosis is achieved by using discrepancies found at the higher level of representation as a means of focusing into the general problem area and only then examining the low level states. This is clearly more efficient than starting the diagnosis at the lowest level of detail and examining every node until the problem is found. Figure 2 shows a cluster in the system model representing a high level view of communication between two agents. The objects modeled are messages (node MESSAGE-OBJECT in figure). Two states of this object are represented in the sending agent: node MESSAGE-EXISTS and node MESSAGE-SENT. In order for a message to be sent it must exist and the sending area must be consistent (node SEND-AREA-OK). If a message is sent and if the channel is ok the message is received by the agent at the other end of the channel. Here two states of the object message are represented: MESSAGE-RECEIVED and MESSAGE-INCORPORATED. If the message is received and the receiving area is consistent at the receiving agent the message is incorporated into the receiving agent's database. The next section discusses the integration and use of the

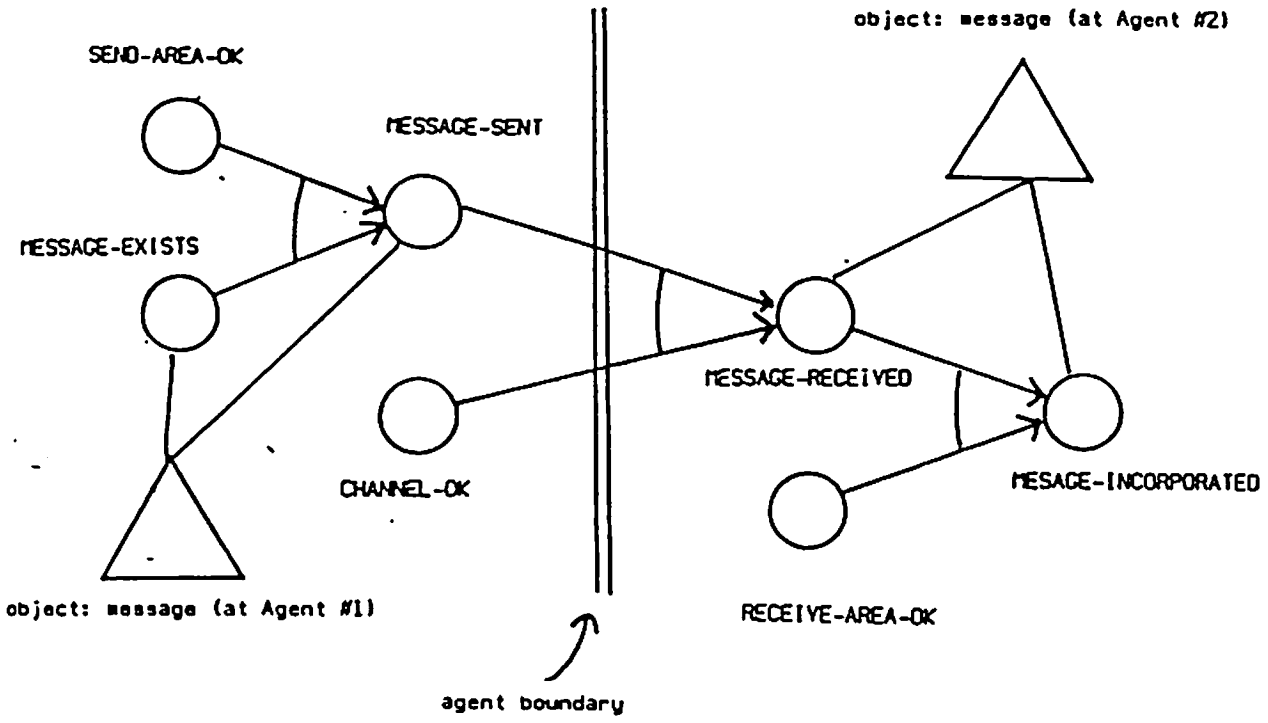


Figure 2: A Cluster Representing a High Level View of Communication.

detection and diagnostic knowledge.

D. Integrating and Using the Detection and Diagnostic Knowledge

The discussion in the previous section described how the model of the correct behavior represents the structure and causality relationships of the system. This section discusses how we can associate the detection information with arbitrary states in this model and thus provide a structure relating the different detection criteria with the diagnostic knowledge. Any state in the system model can be tagged and associated with a detection criteria data structure. All tagged states are watched and updated by the monitor at some predefined interval. The detection criteria data structure contains a list of the statistics that should be collected by the monitor, a place to store the collected data, and a threshold value indicating when the diagnostic component should be invoked. This diagnostic threshold is expressed in terms of the statistics collected. For example, the *MESSAGE-INCORPORATED* state in the high level communication model shown in Figure 2, has associated with it the following information:

1. percent of messages rejected so far (say 30%);

2. allowed percent of messages rejected before diagnosis is invoked (the diagnostic threshold) (50%);

3. description of the types of messages watched for (for example, we might only watch for goals sent from one particular agent).

This information indicates that the agent expects 50 percent of the messages that arrive over the channel from another agent to fall within its receive area. So far, only 30 percent of these messages were rejected which is well within the tolerated range. However, should this value exceed the diagnostic threshold (set to 50 percent), the diagnostic component would be invoked.

When invoked, the diagnostic component is given a description of the violated expectation which caused the invocation. This description consists of a specific object (in this case, say Message #51) and the state this object did not go through as expected (in this case *MESSAGE-INCORPORATED* was the state not satisfied). Once the diagnostic component is given a symptom it begins the construction of the current system state model starting with the symptom nodes and going backwards in time (i.e., following the state transition arcs backwards). The current system state model is created by determining for each node in the model whether it is true or false with respect to the object being investigated. Each node in the model has associated with it information describing how to determine whether it is true or false. This information is obtained from the rich set of the interpretation system's data structures maintaining the system's history. (Data structures such as the hypotheses about the environment, predictions about further hypotheses (goals), different knowledge source queues, and sets of already invoked knowledge sources).

Nodes in the model continue to be evaluated until a state is found which still satisfies the expectations (the last good state). Once this state is found, we know that the fault occured between it and the false state which immediately follows. At the high levels this discrepancy, (the last good state followed by the first bad state), serves as a focusing point guiding the diagnosis into the correct part of the system model. Only the relatively low states in the model constitute reportable failures. These states are marked as primitive. The diagnostic component will in practice evaluate several clusters, descending down the hierarchy, before a primitive false state is found.

By analyzing why a particular state of some object was not satisfied (in this case why particular message was rejected) the diagnostic component may discover a fault in the receiving area and thus account for the rejection of the other messages. Depending on how crucial it is for the system to catch faults, either lower level states or higher level states can be monitored. The lower the level monitored, the less the fault has a chance to propagate and the less damage it does. But there are more of these states and therefore their monitoring is more expensive.

III EXAMPLE OF DETECTION AND DIAGNOSIS OF A PROBLEM SOLVING ERROR

In this section an example of fault detection and diagnosis is presented in the context of the system we will be monitoring: the distributed vehicle monitoring testbed (DVMT). Each agent is a sophisticated problem-solver capable of making local decisions and carrying out the entire task, given the data. Briefly, an agent functions as follows:

1. A piece of data (a hypothesis describing a vehicle of certain type at some location at some time (time frame)) arrives.
2. This stimulates the creation of goals. Goals are descriptions of the types of hypotheses whose creation is expected based on the newly arrived hypothesis. Goals are thus predictions of future data based on current data.
3. A goal together with a hypothesis which caused its creation stimulate a knowledge source that is capable of satisfying the goal.
4. The knowledge source is put on the scheduling queue.
5. The highest rated knowledge source on the scheduling queue is invoked, creating more hypotheses and the cycle begins again.

Each agent in the testbed will contain an FDD component responsible for fault detection and diagnosis. The individual FDD components will engage in distributed problem solving with its counterparts at other agents as it performs the detection and diagnosis, much as the underlying interpretation agents engage in distributed problem solving as they perform the task of acoustic signal interpretation. The testbed will eventually include a component responsible for reconfiguring the system in response to faults reported by the individual FDD components [2].

Consider the following scenario. Two agents are cooperating on an overall interpretation of the environment (see Figure 3). Agent #1 receives data from region sensed by Sensor A (left half of the environment); Agent #2 receives data from region sensed by Sensor B (right half of the environment). Both Agent #1 and Agent #2 process the data at the lower levels of abstraction (up to vehicle tracks) but only Agent #1 can process the data to the highest level of abstraction (the pattern tracks) and is responsible for integrating information over the entire environment. Since Agent #1 can only directly sense data covered by its sensor (Sensor A) it must receive hypotheses falling within Sensor B's area from Agent #2. The agents need to communicate in order to achieve the solution and therefore their communication areas must be matched: Agent #1's receiving area must cover area sensed by Sensor B as does Agent #2's sending area.

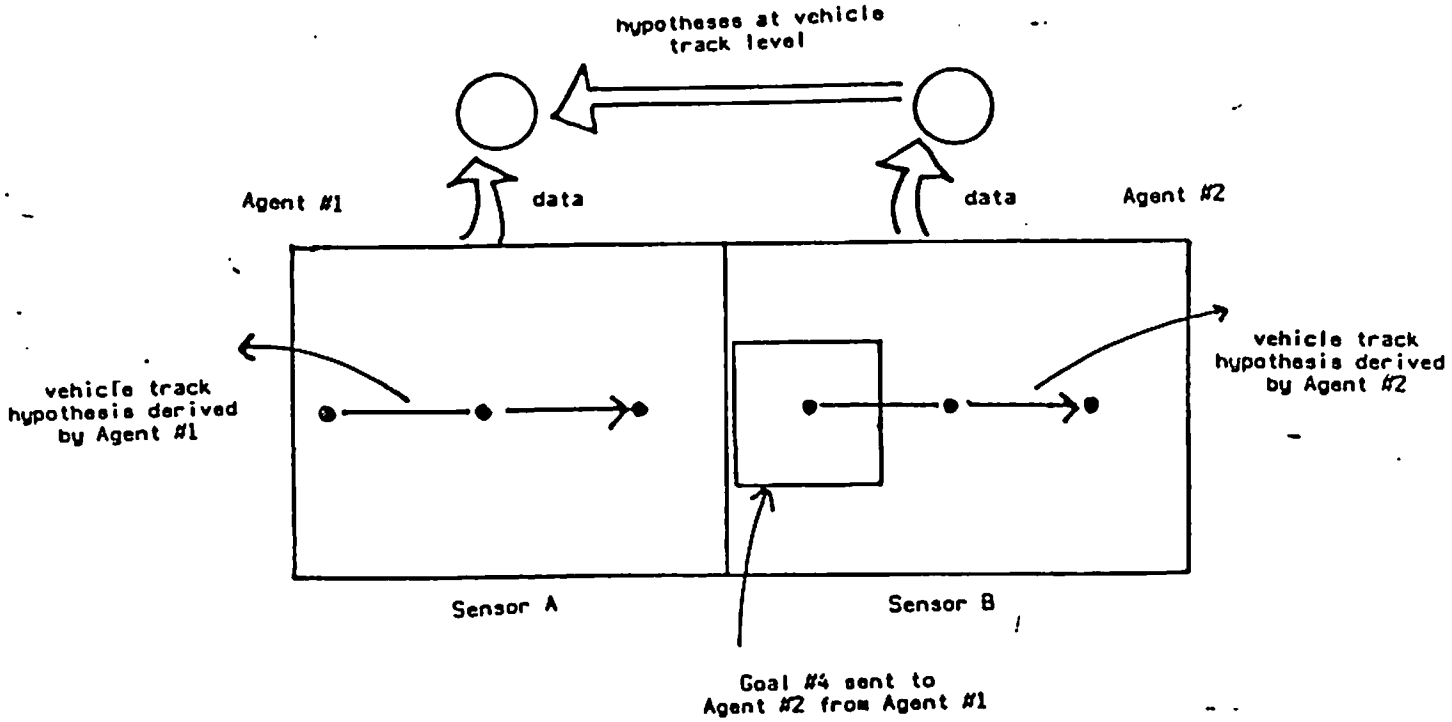


Figure 3: Two Processor Failure Scenario.

Suppose that Agent #2's receiving area is wrong and does not specify Sensor B's area. Suppose that Agent #1 has successfully tracked a vehicle for some time and has a vehicle track hypothesis ranging from time frame 1 to time frame 3. It creates a goal for time frame 4, Goal #4, but since this goal is out of Agent #1's sensed area, it sends this goal out to Agent #2 and awaits an answer (a hypothesis that would satisfy the goal). Meanwhile, Agent #2 has derived such a hypothesis (one covering time frames 4 through 6). Agent #2 sends its hypothesis to Agent #1 but because Agent #1's receiving area is wrong, even though the hypothesis gets over the channel, it is never actually incorporated into Agent #1's database and Goal #4 remains unsatisfied. Suppose there is a limit on the amount of time within which a goal must be satisfied. The detection component notices that Goal #4 has violated this time limit and invokes the diagnostic component.

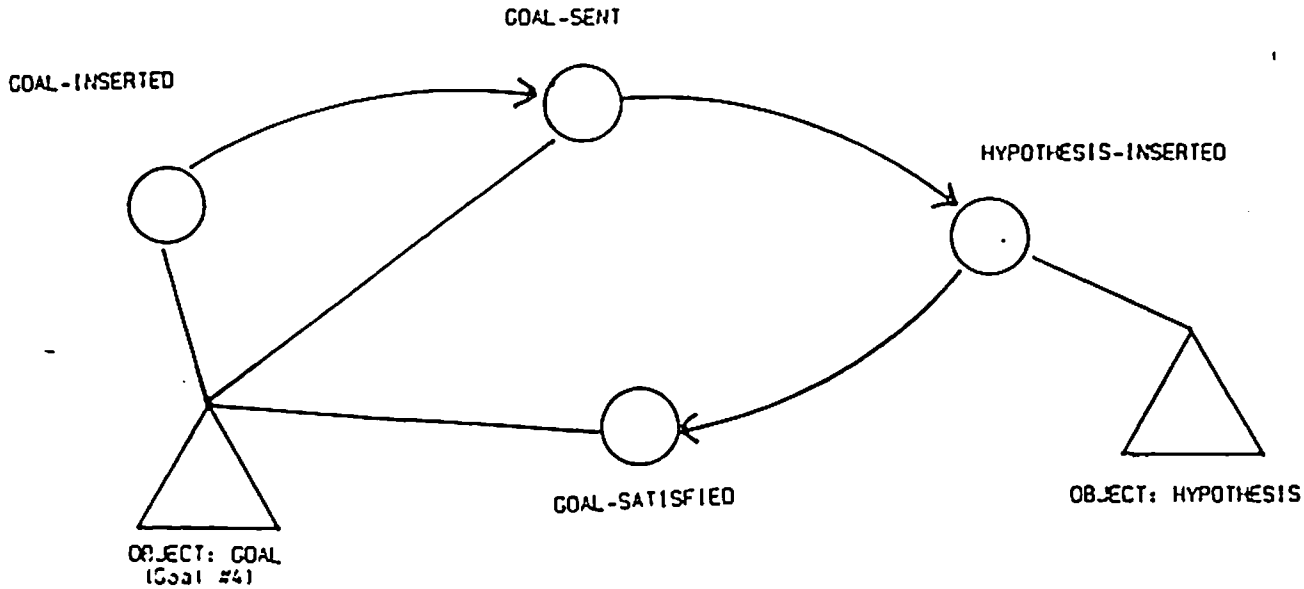
Investigation begins in the part of the system model which represents the symptom: in this case, a cluster which includes the state *GOAL-SATISFIED* whose violation caused the trigerring of the diagnostic component (see Figure 4 a). The aim is to find the value (true or false) of each state node in this cluster. The values of the nodes are found by evaluating the description of the value associated with each node. Once these values are found, the true-false pair (a true node followed by a false node) can be identified. (In the figure, true nodes are filled in and false nodes have a box around them.) This is the point at which correct processing stopped. If the false node is a primitive node, then it can be reported as the identified fault. Since in this cluster no nodes are primitive, the true-false pair will be used to focus the diagnosis into the correct part of a more detailed model. The true-false pair in this case is found at *GOAL-SENT - HYPOTHESIS-INSERTED*. Since goal was sent out but no hypothesis was received the fault must have occured in one of the states occuring in between *GOAL-SENT* and *HYPOTHESIS-INSERTED*. The node *GOAL-SENT* points to another cluster in the model which represents the events that occur between it and the next node, *HYPOTHESIS-INSERTED*. This is the cluster represented in Figure 4 b. Again, the values of all nodes in the cluster are determined as before. In this cluster there is a primitive node, *RECEIVE-AREA-OK*. This node is found to be false with respect to Goal #4. In other words, Goal #4 does not fall within the receive area. Because a primitive false node has been found, a failure has been identified and can be reported to the component responsible for reconfiguration and repair. Any pending symptoms are scanned to see which ones can be accounted for by the identified fault. This allows the system to function more efficiently since it reduces the number of symptoms that have to be diagnosed in full.

Several issues arise:

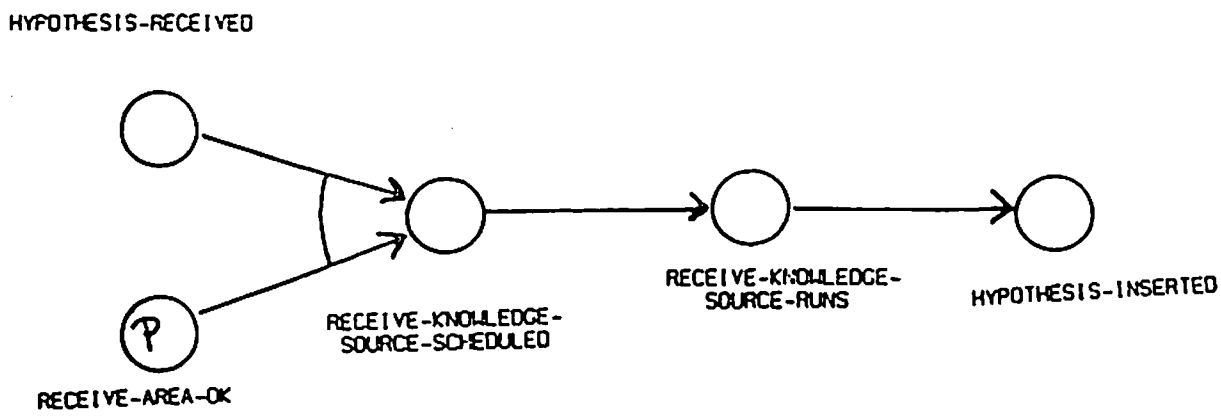
1. Since all the information that may be needed for diagnosis cannot be stored there will be cases when the system cannot determine the value of a state. It will have to try and determine the value indirectly, by analyzing surrounding states.

2. Should the diagnostic component continue and search for other faults or should it stop with the first one found.

3. Finally, how can the FDD know that it is the receive area that is wrong and not the goal (i.e., suppose Goal #4 should never have been created by Agent #1 or should never have been sent to Agent #2). Deciding which of the two is wrong requires other information, such as, who can sense the area of Goal #4 and whether it is reasonable for Agent #1 to require its satisfaction from Agent #2.



a



b

Figure 4: Diagnostic System Model.

We are just beginning to address these issues.

IV CONCLUSIONS

This paper presented a design for a distributed knowledge-based fault-detection and diagnosis component (FDD) intended to monitor and correct the behavior of a distributed problem solving system. We described the detection criteria and the model used for diagnosis. We presented an example demonstrating the use of the detection and diagnostic knowledge in the identification of a fault. FDD performs error detection in a complex problem solving system without resorting to replication and voting as is the case in hardware fault tolerant systems. Unlike existing hardware systems it can deal with complex, multiple failures. FDD is a first step in providing fault-tolerant control in a problem solving system.

V REFERENCES

1. Algirdas Avizienis.
   Fault-Tolerance and Fault-Intolerance: Complementary Approaches to Reliable Computing.
   ACM Sigplan Int. Conference on Reliable Software, pages 458-463, April 1975.

2. Daniel D. Corkill.
   A Framework for Organizational Self-Design in Distributed Problem Solving Networks.
   COINS Technical Report 82-33.

3. R. Davis, H. Shrobe, W. Hanscher, K. Wieckert, M. Shirley, and S. Polit.
   Diagnosis based on descriptions of structure and function.
   In *Proceedings of AAAI*, pages 137-142, August 1982.

4. M. Genesereth.
   Diagnosis using hierarchical design models.
   In *Proceedings of AAAI*, pages 278-283, August 1982.

5. Victor R. Lesser and Daniel D. Corkill.
   Functionally-accurate, cooperative distributed systems.
   IEEE Transactions on Systems, Man, and Cybernetics, SMC-11(1):81-96, January 1981.

6. Victor Lesser, Daniel Corkill, Jasmina Pavlin, Larry Lefkowitz, Eva Hudlicka, Richard Brooks, and Scott Reed.
   A high-level simulation testbed for cooperative distributed problem solving.
   Proceedings of the Third International Conference on Distributed Computer Systems, pages 341-349, October 1982.

7. William R. Nelson.
   Reactor: An expert system for diagnosis and treatment of nuclear reactor accidents.
   In *Proceedings of the Second National Conference on Artificial Intelligence*, pages 296-301, August 1982.

8. H. Shubin, and John W. Ulrich.
   IDT: An Intelligent Diagnostic Tool.
   In *Proceedings of the Second National Conference on Artificial Intelligence*, pages 290-295, August 1982.

9. P. Szolovits, and S.G. Pauker.
   Categorical and Probablistic Reasoning in Medical Diagnosis.
   Artificial Intelligence, Vol. 11, #1&2, 1978, pages 115-144.