Learning Reliability Models of Other Agents in a Multiagent System

Costas Tsatsoulis[1] and Grace Yee[2]

[1] Center for Excellence in Computer-Aided Systems Engineering
Department of Electrical Engineering and Computer Science
The University of Kansas
Lawrence, KS 66045
tsatsoul@eecs.ukans.edu

[2] Lockheed Martin Missiles and Space
Organization 96-10, Building 255
3251 Hanover St.
Palo Alto, CA 94304
yee@stc.lockheed.com

Abstract

This paper describes how agents in a multiagent system can use local domain information to learn reliability models of other agents. Using these reliability models agents can evaluate the quality and validity of messages communicated to them. The dynamically learned, explicit knowledge of the quality of transmitted information is significant in an intelligent system where the autonomous agents have only partial and incomplete view of solutions and the state of problem solving activities, and where any cooperation or coordination behavior relies on communication. We describe how agents learn reliability models, and how these models can be used in multiagent problem solving coordination and coherence.

1. Introduction

In a distributed agent system the ability of an agent to learn can be viewed as learning as a group, and learning to adjust its views or actions (Shaw & Whinston 1990). In order to learn as a group, agents learn to be more cooperative to achieve a global solution. Classical Distributed AI (DAI) systems like the DVMT (Lesser & Corkill 1983), the Contract Net (Davis & Smith 1983), and the Open System architecture (Hewitt 1986), improve their performance by better coordination or more efficient task allocation. On the other hand, intelligent agents in MACE (Gasser, Braganza & Herman 1988) learn to adjust their local views to achieve a better global solution.

The work presented in this paper concentrated on a different aspect that improves the performance of a multiagent system, that of data reliability. In general, in a distributed environment, the data should be consistent and correct to avoid erroneous interpretation and lack of problem solving coordination. A multiagent system must be able to identify agents that contribute inconsistent data, dynamically assign them a degree of reliability, and keep updating this value, all these while the multiagent system is operating.

Because of the need for coordination, DAI systems are facing more conflicts and inconsistencies then stand-alone systems. Most blackboard frameworks have disparity problems when nodes interpret the global data differently (Erman et al. 1980; Durfee & Lesser 1988). In the DVMT, network nodes conflict because they only have partial viewpoints of the world (Durfee & Lesser 1987; Durfee & Lesser 1991). In RESUN and DRESUN (Carver, Cvetanovic & Lesser 1991) the possibility of alternative explanations of data can result in contradicting hypotheses. Knowledge base incompatibilities can also induce inconsistencies (Lenat 1975).

We have developed a new approach to the problem of learning to deal with inconsistent data. Our approach concentrates on the communication between agents, and our systems learn to inhibit or allow inter-agent message passing based on probabilities. Our system is called the EACCS, Error Adaptation Communication Control System, and it helps agents learn to avoid erroneous data, thus optimizing the global problem solving process.

The EACCS is composed of three main components: a dependency trace, a blame assignment mechanism, and a contradiction resolution algorithm. The dependency trace keeps track of the path of a received message, including both the source of the message and the reasoning that led to it. The blame assignment mechanism assigns blame to the paths supplying inconsistent data, and is the major learning component that designates reliability values to the agents communicating data. The contradiction resolution algorithm chooses between any contradicting information received by an agent. With these three features, an agent in EACCS builds a dynamic reliability model of its neighbors which help it resolve both local and global inconsistencies.

2. Data Inconsistency in Multiagent Systems

In a multiagent system messages come from two different sources: facts generated from the local problem solving agent (the internal facts), and facts received from the neighboring agents (the external facts). Three types of inconsistencies arise in a multiagent system: inconsistencies involving only internally deduced facts, inconsistencies involving both external and internal facts, and inconsistencies involving only external facts (Bridgeland & Huhns 1990). In our work we treat all three inconsistencies similarly: since internally deduced facts are supported by external facts, all inconsistencies can be expressed as type 3 errors, i.e. conflicts between externally received facts[1].

We propose a new classification of data inconsistency in multiagent systems: sensor errors, knowledge base errors, and communication link errors.

As a "sensor" we define any process that provides the multiagent system with input data. A sensor can be an actual real-time environment sensing device or a user. A sensor is assumed to

[1] We are assuming that an agent believes completely in the correctness of its problem solving knowledge, thus it will always assume that incorrect or inconsistent inferences are the result of erroneous data that was received from outside sources.

generate or collect a certain class or classes of facts. We define a sensor error as the generation of a datum from another class of facts.

Knowledge base errors arise from an incomplete or overlapping knowledge base. In a rule-based system an overlapping knowledge base can be caused contradiction, subsumption and redundancy in rules. Subsuming and redundant rules only induce inefficiency, while contradicting rules produce inconsistencies.

Communication errors arise from unreliable communication channels. Data are corrupted when the content is changed during transmission. Redundant information and communication are traditionally used to recreate the lost and corrupted messages (Lesser & Erman 1980).

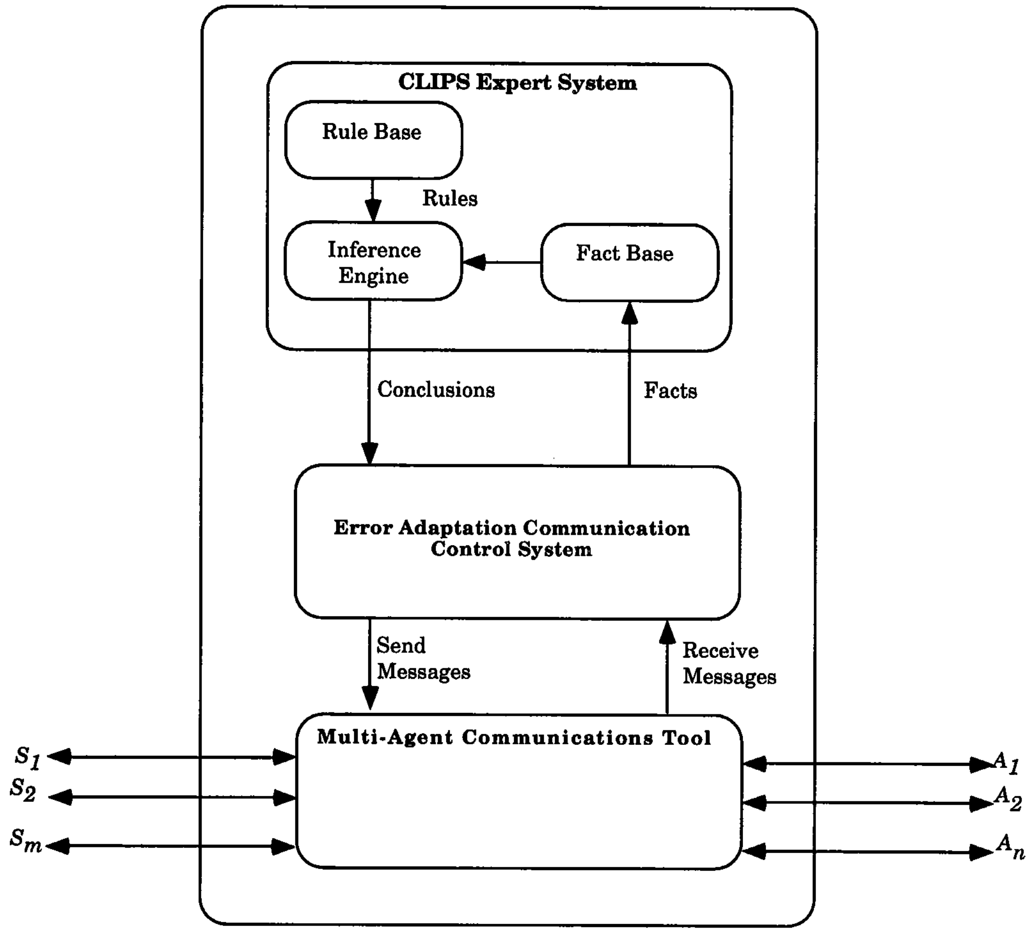

Figure 1: Architecture of an agent

3. Structure of a Multiagent System

In our system an "agent" consists of three parts, as shown in Figure 1. The top level of an agent is its problem solving component, implemented as a CLIPS rule base in our experiments. The lowest level is the communications component which deals with the receipt and transmission of messages. The intermediate component, and the one on which this paper concentrates, is the EACCS, which determines data inconsistencies, learns reliability models, and resolves data inconsistencies based on these models.

A sensor is a simple agent which only collects or produces a class or classes of facts, and then communicates them. This implies that facts are grouped into classes which are recognizable by all agents. This common vocabulary is important for learning, since conclusions made about a single datum are rarely useful unless they can be extended to the larger class to which the datum belongs. Figure 2 shows such a sensor.

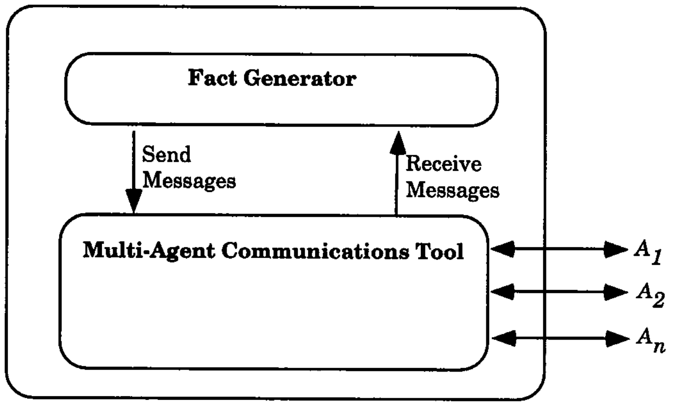

Figure 2: Architecture of a sensor

The multiagent system consists of agents and sensors, connected by communication lines. Figure 3 shows such a system. Our representational convention is that agents, i.e. the problem solving components of the system, are represented by ovals, while sensors are represented by rectangles. Note that only the agents directly connected to a sensor have access to it; other agents need to have sensor information routed to them through other members of the multiagent system.

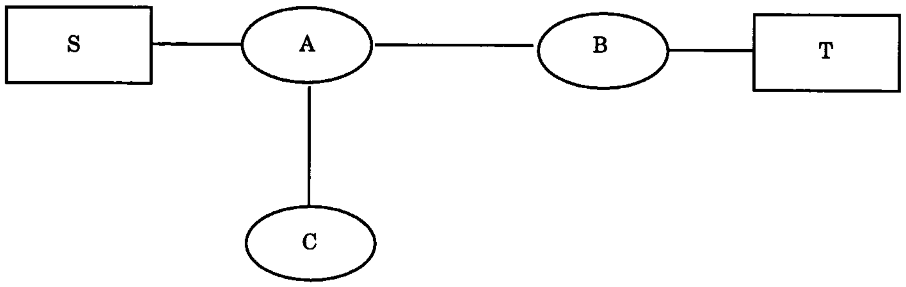

Figure 3: An example multiagent system

4. Learning Reliability Models in a Multiagent System

Traditionally, a fact is believed if its supporting justifications are believed. This requires either a fundamental fact or circular arguments to ground the beliefs (Doyle 1979). Distributed Truth Maintenance Systems support incomplete and non-monotonic reasoning by incremental consistency checking and labeling and by management of a shared datum labeling (Huhns & Bridgeland 1991), or by sharing partial results to retain overall consistency in the data labelings (Mason 1988). To implement such truth maintenance in a multiagent system requires that every agent has complete knowledge of all facts in the system or that all partial results are shared between all agents, both restricting assumptions. The goal of a distributed truth maintenance system is to *maintain* a consistent world upon every interaction; our goal is to accept inconsistency as unavoidable and to allow the multiagent system to learn how to react to it and how to resolve it. To do this in the EACCS we take a different view of the validity of a fact: all facts are assumed to be believed without reason. When inconsistencies arise, neither of the contradictory facts has ground knowledge to support it; thus, the agent disbelieves all of them. The inconsistency is resolved by the *reliability model*, a model that each agent learns and modifies dynamically.

The reliability model is a probabilistic model that leads an agent to believe a fact "more" than a contradictory one. This implies that the EACCS must first identify when two facts are contradictory and then to build the reliability model of the sensors and agents that transmit data that cause contradictions.

4.1. Identifying Contradictions

Contradicting facts are facts that cannot co-exist at the same time in the same system, and, both facts must be part of the problem solving[2]. In a real-time environment, a fact can be valid at certain times and incorrect in other periods. For example, the direction of a vehicle may change over time, and the existence of two different vehicle directions in the same database is not necessarily an error. On the other hand, a vehicle cannot have two different directions within some Δt, and the existence of such facts would indicate a data inconsistency. The EACCS component of an agent uses domain-constraint knowledge to restrict the facts that can co-exist in that agent in a window of time and to recognize when the multiagent system is attempting to solve problems in two different, contradicting contexts. The domain-constraint knowledge knows about contradicting solution paths and will identify such inconsistencies. For example, suppose that based on sensor inputs, one agent identified an airplane as enemy and planned hostile actions. At the same time, another sensor supplied information that the airplane is friendly; based on this sensor, some other agents follow a "friendly" problem solving context. This will lead to a failure when some agent in the system receives facts generated by the two contradictory solution contexts.

4.2. Learning and Updating Reliability Models

To recover from inconsistent information an agent has to develop a mechanism to deal with the agents that transmit data that cause contradictions. However, the agent which generates the inconsistency may not be the one responsible for the corrupted data, since its reasoning may be

[2] If a fact does not contribute to the problem solving activities of an agent, then we assume that the agent is not concerned with the fact's validity.

sound but based on erroneous information *it* received. Thus, an agent has to trace back to all the agents supplying data or justifications that led to the inconsistencies. Since the inconsistencies arise from erroneous data affecting a small portion of the belief space (i.e. it is more likely that the valid fact of the contradicting pair is more consistent with the world than the erroneous one), we can use probability to lower the belief in the agents providing these data. As the agents establish a reliability model of their neighbors, the system can isolate the error-generating agents without harming the overall problem solving.

An agent records all incoming messages, including both the supporting justifications and the sources. Using its domain-constraint knowledge it checks for the validity of all incoming messages. When conflicts arise, the agent stops its problem solving, traces back, and takes off ***reliability credits*** from the agents responsible. An agent will only lower the reliability in the erroneous fact class received by an agent, allowing the reliability of other facts received from this agent to remain unaffected. All these operations are performed by the Error Adaptation Communication Control System, the EACCS.

The EACCS consists of three main components: the dependency trace, the blame assignment mechanism and the contradiction resolution algorithm.

The dependency trace helps an agent keep track of all incoming messages and their sources. This structure includes the source of a message, that is, the agent which transmitted the fact, and a list of all justifications, the facts supporting the one transmitted. Since there is no ground fact to support any of the data, all the antecedents of the facts may be involved in the inconsistent data. Thus, we have to record the whole path history of all justifications, from the beginning of the sensor sampling. Following is the format of a received message:

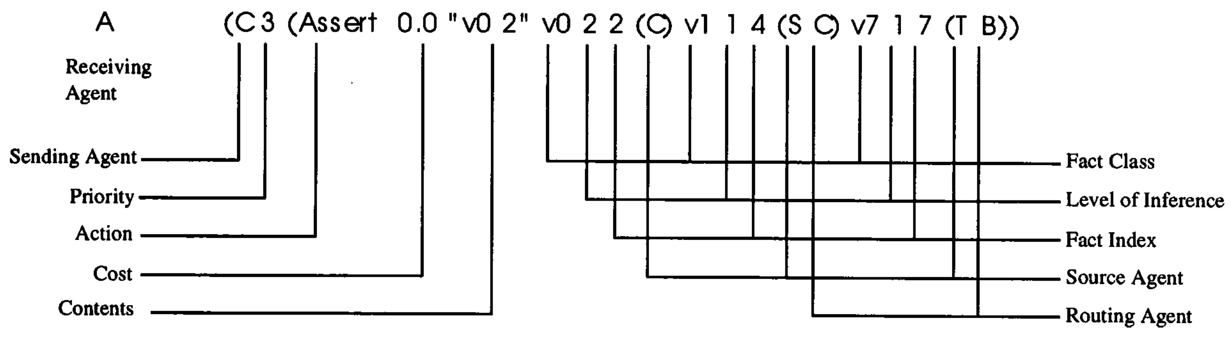

Figure 4: Message format

Notes about the message:

• The message has an Action denoting the procedure that the Receiving agent should take. An Action can be Assert, Send, Sample, etc. It also has a priority value, used when storing incoming messages in the agent's priority first-in-first-out queue, and a cost value representing how expensive it was to generate and route the message (see (Kinney 1992) for detail).

• The message contains the complete path of the information. Whenever inferencing is performed, the "level of inference" is incremented.

• Routing Agent is the agent that routes the fact to another agent but is not responsible for its generation.

The message of Figure 4 means a fact of class v7 and Fact ID 1 is generated from sensor T and is routed to agent B and from there to C. Another fact of class v1 and Fact ID 1 is sampled from sensor S and is also routed to agent C. Agent C uses these two facts to fire a rule which generates a new fact with class v0 and Fact ID 2. Finally, fact 2 was sent to agent A with priority 3, and the actual fact was asserted in the rule base of agent A.

In large multiagent systems the dependency trace may become very long, and the communication bandwidth needed will overwhelm the system. To avoid this the user of the system can define a threshold that will limit the length of the dependency path inside the message. This may degrade the perfrormance of the reliability algorithm, but allows the EACCS to scale up well.

After a fact is received, the domain-constraint knowledge checks the consistency of this fact with the other data known by the agent. If a fact generates an inconsistency, the EACCS activates its blame assignment mechanism and updates its reliability models.

Since there are no ground facts to support any of the reasoning, both of the contradicting facts have equal probability of being wrong; in fact, the whole antecedent paths of both facts are suspected. The receiving agent will first retract all suspected facts, and then retract *reliability credit* from all the agents involved in generating the message. The reliability credit, or simply the RC, is a degree of trust an agent should show towards fact classes received by other agents. Each agent has a table of RC values for all the neighbors for all fact classes. Figure 5 shows a piece of such an RC table held by an agent.

	Fact Class 1	Fact Class 2	Fact Class 3	...
Agent A	1	0	0.6	
Agent B	0.8	1	1	
Agent C	1	0.9	0.1	
...				

Figure 5: Table with reliability credits.

Obviously our approach is sensitive to the classification of facts, since a single, consistently erroneous fact would lower the reliability of a whole class. Having many small classes helps avoid penalizing too many facts at the same time. On the other hand, many classes increase the size of the reliability table stored and maintained by an agent, thus increasing memory and computing requirements.

Reliability credit values range from [0,1], where “0” means that a fact of this class received from a particular agent is never trusted and believed, while “1” denotes a totally reliable fact class. So, in the table of Figure 5, a fact belonging to class 2 and received from agent A will never be believed.

Initially, when the multiagent system is first activated, all RC values are 1, since no agent has received contradictory information. When a contradiction is identified, suspected agents have their RC reduced. The RC value is decreased by the following equation:

$$\Delta RC_T(a,b) = RC_T(a,b) \times TP_T \times \frac{1}{RS_b}$$

where:
$\Delta RC_T(a,b)$ is the change in the reliability credit of class T transmitted from agent a to b;
RS_b is the rule base size of agent b;
TP_T is the normalized transmission path length of class T.

In other words, the RC is decremented inversely proportionally to the rule base size of the receiving agent, since the larger the rule base size the easier it is for an agent to produce contradicting facts. Note also that we use the normalized length of the transmission chain defined as the transmission length to an agent in the path divided by the total path length[3]. The assumption is that the agents deeper in the transmission path are more likely to be the causes of the inconsistent data, since, otherwise, the error should have been detected earlier. At the same time, we do not completely absolve agents early in the path, since it is possible that their domain-constraint knowledge simply failed to identify the inconsistency.

Note that reliability cannot be restored in our current methodological formulation. One-time errors, though, will not affect the system greatly, since the equation is not sensitive to single errors. Some penalty will be applied, but it takes many inconsistencies generated by the same agent or path to impact the reliability credit substantially.

4.3. Resolving Contradictions

After an inconsistency is detected, the agent must decide which of the contradicting facts to believe and use. If the RC value of one fact is above a maximum threshold and the other one below a minimum one, this implies that one fact can be trusted while the other one is unreliable. The agent will then penalize the path submitting the unreliable fact. In any other case (if the RC values of both facts are below the minimum threshold or above the maximum one, or one of them is between the maximum and the minimum thresholds) it implies that both facts are either too unreliable or too reliable. In either case both data are disbelieved and discarded, and both paths supplying them are penalized and have their RC values reduced.

5. Experimental Results

We ran experiments of the EACCS on various multiagent architectures using different values for the probability of failure of a sensor, an agent, and a communications link. Since we concentrated on studying the experimental performance of our model, we did not use any real domain knowledge. Our systems contained randomly generated rule-based and sensors.

[3] A fact transmitted by a sensor is assumed to have transmission path length of 1.

Our results have shown that the system is able to identify and isolate the source of error. Interestingly, the larger the probability of error, the faster the system converges towards a reliability model. Our experiments also showed that if there is more than one alternative path to acquire the same knowledge, the agent is able to identify the corrupted path and choose to collect data from the other paths. The cost of the additional information that needs to be transmitted (i.e. the complete data routing information) is more than balanced by the increased accuracy and coherence, which mean that the system spends less time in contradictory problem solving contexts.

Our experiments also showed that our scheme is sensitive to the number of transmission lines between agents and to the location of the conflict detection rules. For example, if a class of data can be retrieved from a single sensor, and this sensor is isolated because it is generating erroneous data, we are effectively isolating any correct data of this class that the sensor may generate. Also, if the conflict detection rules are not located in the agents which will be receiving the contradictory data, the system may take a long time to converge to a reliability model.

6. Conclusions

Our method for learning and updating reliability models of agents offers numerous advantages. First, it addresses a fundamental issue in agent communication, leading to better coordination and coherence of the whole system through local only decision making. Second, our model makes no assumptions about the agents or the data. We are assuming that errors can happen anywhere, that agents do not need a global view of all the data, and that inconsistencies are not necessarily recognized at the point of occurrence. This is a general model for dealing with data inconsistencies in distributed systems, and is not constrained by any domain-specific limitations.

The major issues we will be studying in the future are the trade-offs between the number and size of fact classes versus system performance, and the definition of a reliability model that can restore reliability credits and be less sensitive to one-time errors. Finally, we will also be looking at errors generated by the communication links.

7. Acknowledgments

We would like to thank the anonymous reviewers for many valuable suggestions that helped improve the presentation of our work.

8. References

Bridgeland, D.M. and Huhns, M.N. 1990. Distributed Truth Maintenance. Proceedings of the National Conference on Artificial Intelligence (*AAAI-90*).

Carver, N., Cvetanovic, Z. and Lesser, V.R. 1991. Sophisticated Cooperation in FA/C Distributed Problem Solving Systems. Proceedings of the National Conference on Artificial Intelligence (*AAAI-91*), 191-198.

Davis, R. and Smith, R.G.. 1983. Negotiation as a Metaphor for Distributed Problem Solving. *Artificial Intelligence,* Vol. 20, 63-109.

Doyle, J. 1979. A Truth Maintenance System, *Artificial Intelligence* 12 (3), 231-272.

Durfee, E.H. and Lesser, V.R. 1987. Using Partial Global Plans to Coordinate Distributed Problem Solvers. Proceedings of the International Joint Conference on Artificial Intelligence (*IJCAI-87*), Milan, Italy, 875-883.

Durfee, E.H. and Lesser, V.R. 1988. Incremental Planning to Control a Time-Constrained, Blackboard-Based Problem Solver. *IEEE Transactions on Aerospace and Electronic Systems* 24(5), 647-662.

Durfee, E.H. and Lesser, V.R. 1991. Partial Global Planning: A Coordination Framework for Distributed Hypothesis Formation. *IEEE Transactions on Systems, Man, and Cybernetics* 21(5), 1167-1183.

Erman, L.D., Hayes-Roth, F.A., Lesser, V.R. and Reddy, D.R. 1980. The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty. *Computing Surveys* 12(2), 213-253.

Gasser, L., Braganza, C. and Herman, N. 1988. Implementing Distributed AI Systems Using MACE. in: *Readings in Distributed Artificial Intelligence*, A.H. Bond and L. Gasser (Eds.), Morgan Kaufmann, San Mateo, CA, 445-450.

Hewitt, C.E. 1986. Offices are Open Systems. *ACM Transactions on Office Information Systems* 4(3), 271-287.

Huhns, M.N. and Bridgeland, D.M. 1991. Multi-Agent Truth Maintenance. *IEEE Trans. on Systems, Man and Cybernetics* 21 (6), 1437-1445.

Kinney, M.D. 1992. *A Multiagent Communications Tool Using Remote Procedure Calls.* CECASE-TR-8640-12, Center for Excellence in Computer Aided Systems Engineering, The University of Kansas.

Lenat, D.B. 1975. Beings: Knowledge's Interacting Experts. Proceedings of the International Joint Conference on Artificial Intelligence (*IJCAI-75*), 126-133.

Lesser, V.R. and Corkill, D.D.. 1983. The Distributed Vehicle Monitoring Testbed: A Tool For Investigating Distributed Problem Solving Networks. *Artificial Intelligence,* 15-33.

Lesser, V.R. and Erman, L.D. 1980. Distributed Interpretation: A Model and Experiment. *IEEE Trans. on Computers* C-29 (12), 1144-1163.

Shaw, M.J. and Whinston, A.B. 1990. Learning and Adaptation in Distributed Artificial Intelligence Systems. in: L. Gasser and M.N. Huhns (Eds.), *Distributed Artificial Intelligence* Vol. II, Pitman Publishing/Morgan Kaufmann Publishers, Los Altos, CA, 413-429.