AI Magazine Volume 4 Number 3 (1983) (© AAAI)

THE DISTRIBUTED VEHICLE MONITORING TESTBED:

A Tool For Investigating Distributed Problem Solving Networks

Victor R. Lesser
Daniel D. Corkill

Computer and Information Science Department
University of Massachusetts
Amherst, MA 01003

Abstract

Cooperative distributed problem solving networks are distributed networks of semi-autonomous processing *nodes* that work together to solve a *single* problem. The Distributed Vehicle Monitoring Testbed is a flexible and fully-instrumented research tool for empirically evaluating alternative designs for these networks. The testbed simulates a class of a distributed knowledge-based problem solving systems operating on an abstracted version of a vehicle monitoring task.

There are two important aspects to the testbed: (1) it implements a novel generic architecture for distributed problem solving networks that exploits the use of sophisticated local node control and meta-level control to improve global coherence in network problem solving; (2) it serves as an example of how a testbed can be engineered to permit the empirical exploration of design issues in knowledge-based AI systems. The testbed is capable of simulating different degrees of sophistication in problem solving knowledge and different focus-of-attention mechanisms, for varying the distribution and characteristics of error in its (simulated) input data, and for measuring the progress of problem solving. Node configurations and communication channel characteristics can also be independently varied in the simulated network.

A project as large and complex as the Distributed Vehicle Monitoring Testbed involved a number of individuals and became itself a distributed problem solving task. The efforts of Richard Brooks, Eva Hudlicka, Larry Lefkowitz, Raam Mukunda, Jasmina Pavlin, and Scott Reed contributed to the success of the testbed. We would also like to acknowledge Lee Erman's collaboration on the initial formulation of the Functionally Accurate, Cooperative approach and his work on the pilot experiments. This research was sponsored, in part, 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.

THERE ARE TWO MAJOR THEMES of this article. First, we introduce readers to the emerging subdiscipline of AI called *Distributed Problem Solving*, and more specifically the authors' research on Functionally Accurate, Cooperative systems. Second, we discuss the structure of tools that allow more thorough experimentation than has typically been performed in AI research. An example of such a tool, the Distributed Vehicle Monitoring Testbed, will be presented. The testbed simulates a class of distributed knowledge-based problem solving systems operating on an abstracted version of a vehicle monitoring task. This presentation emphasizes how the testbed is structured to facilitate the study of a wide range of issues faced in the design of distributed problem solving networks.

Characteristics of Distributed Problem Solving.

Distributed Problem Solving (also called Distributed AI) combines the research interests of the fields of AI and Distributed Processing (Chandrasekaran 1981; Davis 1980, 1982; Fehling & Erman 1983). We broadly define distributed problem solving networks as distributed networks of semi-autonomous problem solving *nodes* (processing elements) that are capable of sophisticated problem solving and cooperatively interact with other nodes to solve a *single* problem. Each node can itself be a sophisticated *problem solving system* that can modify its behavior as circumstances change

and plan its own communication and cooperation strategies with other nodes.

Distributed problem solving is an important research area for several reasons First, hardware technology has advanced to the point where the construction of large distributed problem solving networks is not only possible, but economically feasible. While the first networks may consist of only a small number of nodes, distributed problem solving networks may eventually contain hundreds or thousands of individual nodes. We are nearing a situation of exciting hardware possibilities unaccompanied by the problem solving technology required for their effective utilization. Second, there are AI applications that are inherently spatially distributed A distributed architecture that matches their spatial distribution offers many advantages over a centralized approach. Third, understanding the process of cooperative problem solving is an important goal in its own right. Whether the underlying system is societal, managerial, biological, or mechanical, we seem to understand competition far better than cooperation It is possible that the development of distributed problem solving networks may serve the same validating role to theories in sociology, management, organizational theory, and biology as the development of AI systems have served to theories of problem solving and intelligence in psychology and philosophy.

Although this new area borrows ideas from both AI and Distributed Processing, it differs significantly from each in the problems being attacked and the methods used to solve these problems

Distributed Problem Solving and Distributed Processing

Distributed problem solving networks differ from distributed processing systems in both the style of distribution and the type of problems addressed (Smith & Davis 1981). These differences are most apparent when we study the interactions among nodes in each of the types of networks A distributed processing network typically has multiple, disparate tasks executing concurrently in the network. Shared access to physical or informational resources is the main reason for interaction among tasks The goal is to preserve the illusion that each task is executing alone on a dedicated system by having the network operating system hide the resource sharing interactions and conflicts among tasks in the network. In contrast, the problem solving procedures in distributed problem solving networks are explicitly aware of the distribution of the network components and can make informed interaction decisions based on that information. This difference in emphasis is, in part, due to the characteristics of the applications being tackled by conventional distributed processing methodologies These applications have permitted task decompositions in which a node rarely needs the assistance of another node in carrying out its problem solving function. Thus, most of the research as well as the paradigms of distributed processing do not directly address the issues of cooperative interactions of tasks to solve a single problem. As will be discussed later, highly cooperative task interaction is a requirement for many problems that seem naturally suited to a distributed network.

Distributed Problem Solving and Artificial Intelligence

Distributed problem solving also differs from much of the work in AI because of its emphasis on representing problem solving in terms of asynchronous, loosely-coupled process networks that operate in parallel with limited interprocess communication Networks of cooperating nodes are not new to artificial intelligence. However, the relative autonomy and sophistication of the problem solving nodes, a direct consequence of limited communication, sets distributed problem solving networks apart from most others, including Hewitt's work on the actor formalism, Kornfeld's ETHER language, Lenat's BEINGS system, and the augmented Petri nets of Zisman (Hewitt 1977, Kornfeld 1979, Lenat 1975, Zisman 1978) The requirement for limited communication in a distributed network has also led to the development of problem solving architectures that can operate with possibly inconsistent and incomplete data and control information. In many applications, communication delay makes it impractical for the network to be structured so that each node has all the relevant information needed for its local computations and control decisions. Another way of viewing this problem is that the spatial decomposition of information among the nodes is ill-suited to a functionally distributed solution Each node may possess the information necessary to perform a portion of each function, but insufficent information to completely perform any function.

The Uses of Distributed Problem Solving.

Most initial work in distributed problem solving has focused on three distributed air traffic control, and distributed robot systems (Davis 1980, 1982; Fehling 1983). All of these applications need to accomplish distributed interpretation (situation assessment) and distributed planning. Planning here refers not only to planning what actions to take (such as changing the course of an airplane), but also to planning how to use resources of the network to carry out the interpretation and planning task effectively This latter form of planning encompasses the classic focus-of-attention problem in AI

In addition to the commonality in terms of the generic tasks being solved, these application domains are characterized by a natural spatial distribution of sensors and effectors, and by the fact that the subproblems of both the local interpretation of sensory data and the planning of effector actions are interdependent in time and space For example, in a distributed sensor network tracking vehicle movements, a vehicle detected in one part of the sensed area implies that a

vehicle of similar type and velocity will be sensed a short time later in an adjacent area (Figure 1). Likewise, a plan for guiding an airplane must be coordinated with the plans of other nearby airplanes in order to avoid collision. Interdependency also arises from redundancy in sensory data Often different nodes sense the same event due to overlaps in the range of sensors and the use of different types of sensors that sense the same event in different ways. Exploiting these redundant and alternative views and the interdependencies among subproblems require nodes to cooperate in order to interpret and plan effectively. This cooperation leads to viewing network problem solving in terms of a single problem rather than a set of independent subproblems.

It is difficult to develop a distributed problem solving architecture that can exploit the characteristics of these applications to limit internode communication, to achieve real-time response, and to provide high reliability. Nodes must cooperate to exploit and coordinate their answers to interdependent subproblems, but must do so with limited interprocessor communication. This requires the development of new paradigms that permit the distributed system to deal effectively with environmental uncertainty (not having an accurate view of the number and location of processors, effectors, sensors, and communication channels), data uncertainty (not having complete and consistent local data at a node) and control uncertainty (not having a completely accurate model of activities in other nodes).

We see the development of these paradigms as drawing heavily on the work in knowledge-based AI systems and, simultaneously, making contributions to AI. As Nilsson has noted, the challenges posed by distributed Artificial Intelligence will contribute to (and may even be a prerequisite for) progress in "ordinary" artificial intelligence (Nilsson, 1980) One example of this interaction is the problem of controlling semi-autonomous problem solving agents possessing only a local and possibly errorful view of the global state of problem solving. Solutions being developed for this problem have involved the use of meta-level control, integrated data-directed and goal-directed control, and focus-of-attention strategies based on reasoning about the state of local problem solving (Corkill 1983). Approaches similar to these are being used to solve the control problems that are faced in the development of a new generation of centralized knowledge-based problem solving systems, which have significantly larger and more diverse knowledge bases

In the remainder of this article we first describe the Functionally Accurate, Cooperative distributed problem solving paradigm and pilot experiments that explored the viability of this approach After describing the issues we wish to explore using the Distributed Vehicle Monitoring Testbed, we present the vehicle monitoring task, followed by a detailed discussion of the testbed. Later sections describe how we have quantified system behavior and the use of these measures for simulating and evaluating the performance of various system components, overview the tools that help a user define experiments and analyze their output, review the current status of the testbed implementation, and outline future research directions.

Functionally Accurate, Cooperative Distributed Problem Solving

Our research has focused on the design of distributed problem solving networks for applications in which there is a natural spatial distribution of information and processing requirements, but insufficient information for each processing node to make completely accurate control and processing decisions without extensive internode communication (used to acquire missing information and to determine appropriate node activity). An example of this type of application is distributed vehicle monitoring. Vehicle monitoring is the task of generating a dynamic, area-wide map of vehicles moving through the monitored area. Distributed vehicle monitoring typically has a number of processing nodes, with associated acoustic sensors (of limited range and accuracy), geographically distributed over the area to be monitored (Lacoss 1978, Smith 1978). Each processsing node can communicate

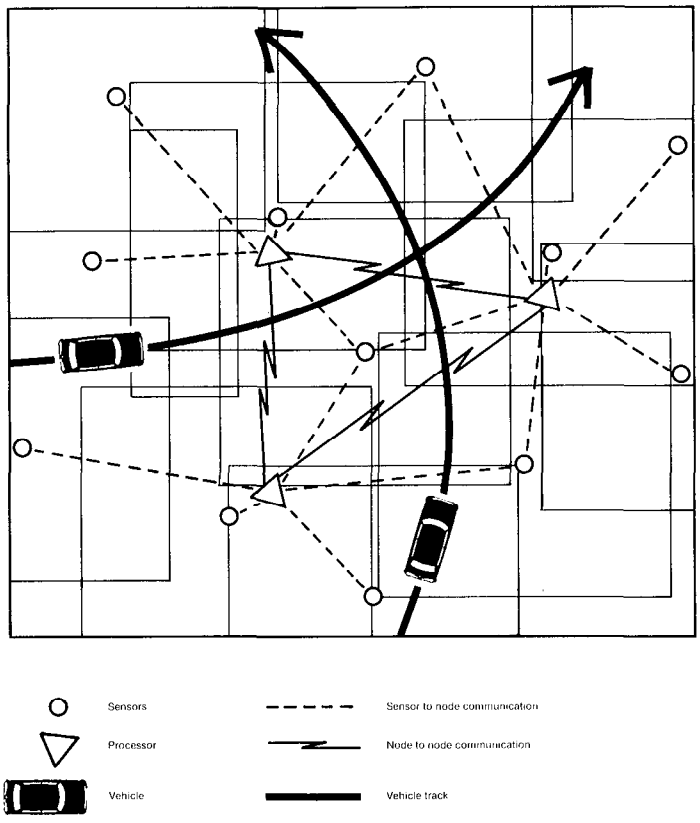

Figure 1
Tracking Vehicle Movements in a Distributed Sensor Network

with other nearby nodes over a packet radio communication network (Kahn 1978). Each sensor includes the actual acoustic transducer, low-level signal processing hardware and software, and communication equipment necessary to transmit the processed signals to a high-level (symbolic) processing site.

As a vehicle moves through the monitoring area, it generates characteristic acoustic signals. Some of these signals are recognized by nearby sensors which detect the frequency and approximate location of the source of the signals. An acoustic sensor has a limited range and accuracy, and the raw data it generates contains significant error. Using data from only one sensor can result in "identification" of non-existent vehicles and ghosts, missed detection of actual vehicles, and incorrect location and identification of actual vehicles. To reduce these errors, information from various sensors must be correlated over time to produce the answer map. The amount of communication required to redistribute the raw sensory data necessary for correct localized processing makes such an approach infeasible.

One way to reduce the amount of communication and synchronization is to loosen the requirement that nodes always produce complete and accurate results. Instead, each node produces tentative results which may be incomplete, incorrect, or inconsistent with the tentative partial results produced by other nodes. For example, a node may produce a set of alternative partial hypotheses based on reasonable expectations of what the missing data might be. In the vehicle monitoring task, each node's tentative vehicle identification hypotheses can be used to indicate to other nodes the areas in which vehicles are more likely to be found and the details (vehicle type, rough location, speed, etc.) of probable vehicles. This information help a node to identify the actual signals in its noisy sensory data. In addition, consistencies between these tentative identification hypotheses serve to reinforce confidence in each node's identifications. Such cooperation is not only appropriate for vehicle identification, but also potentially useful in other stages of processing (identification of raw signals, groups of harmonically related signals, patterns of vehicles, etc.).

This type of node processing requires a distributed problem solving structure in which the nodes cooperatively converge to acceptable answers in the face of incorrect, inaccurate, and inconsistent intermediate results. This is accomplished using an iterative, coroutine type of node interaction in which nodes' tentative partial results are iteratively revised and extended through interaction with other nodes. A network with this problem solving structure is called *Functionally Accurate, Cooperative* (FA/C) (Lesser 1981). "Functionally accurate" refers to the generation of acceptably accurate solutions without the requirement that all shared intermediate results be correct and consistent (as is the case with conventional distributed processing). "Cooperative" refers to the iterative, coroutine style of node interaction in the network. The hope of this approach is that much less communication is required to exchange these high-level, tentative results than the communication of raw data and processing results that would be required using a conventional distributed processing approach. In addition, synchronization among nodes can also be reduced or eliminated entirely, resulting in increased node parallelism. Finally, this approach leads to a more robust network since errors resulting from hardware failure are potentially correctable in the same fashion as errors resulting from the use of incomplete and inconsistent local information.

A Pilot Experiment in Distributed Interpretation

A set of pilot experiments was performed to investigate the suitability of the FA/C approach using a network of complete HEARSAY-II interpretation systems (Lesser 1980). The HEARSAY-II architecture appeared to be a good structure for each node because it incorporates mechanisms for dealing with uncertainty and error as an integral part of its basic problem solving. Further, the processing can be partitioned or replicated naturally among network nodes because it is already decomposd into independent and self-directed modules called, *knowledge sources*, which interact anonymously and are limited in the scope of the data they need and produce. For further information about the HEARSAY-II architecture see Erman, et al (1980).

Experiments were performed to determine how the problem solving behavior of a network of HEARSAY-II nodes compared to a centralized system. Each node was completely self-directed in its decisions about what work it should perform and what information it should transmit to other nodes. The aspects of behavior studied included the accuracy of the interpretation, the time required, the amount of internode communication, and network robustness in the face of communication errors. These experiments simulated only the distributed hardware – they used an actual HEARSAY-II speech understanding system analyzing real data. A spatial distribution of sensory data was modelled by having each node of the distributed speech understanding network sample one part (time-contiguous segment) of the overall speech signal.

The experiments showed that a network of three HEARSAY-II speech understanding nodes performs well as a cooperative distributed network even though each node has a limited view of the input data and exchanges only high-level (phrasal) partial results with other nodes. In an experiment with errorful communication, network performance degraded gracefully with as much as 50% of the messages lost, indicating that the system can often compensate automatically for the lost messages by performing additional computation.

Although these experiments were extremely positive, they did point up a key issue in the successful application of the FA/C approach. This issue, which we feel is also important for the design of any complex distributed problem solving network, is that of obtaining a sufficient level of cooperation and coherence among the activities of the semi-autonomous, problem solving nodes in the network (Davis &

Smith 1982, Corkill 1982). If this coherence is not achieved, then the performance (speed and accuracy) of the network can be significantly diminished as a result of lost processing as nodes work at cross-purposes with one another, redundantly applied processing as nodes duplicate efforts, and misallocation of activities so that important portions of the problem are either inaccurately solved or not solved in timely fashion.

In the pilot experiments with the three-node network, we observed that the simple *data-directed* and *self-directed* control regime used in these experiments can lead to noncoherent behavior (Lesser 1980). Situations occurred when a node had obtained a good solution in its area of interest and, having no way to redirect its attention to new problems, simply produced alternative but worse solutions. Another problem occurred when a node had noisy data and could not possibly find an accurate solution without help from other nodes. In this situation, the node with noisy data often quickly generated an inaccurate solution which, when transmitted to the nodes working on better data, resulted in the distraction of these nodes. This distracting information in turn caused significant delay in the generation of accurate solutions by nodes with accurate as well as noisy data. We believe that development of appropriate network coordination policies (the lack of which resulted in diminished network performance for even a small network) will be crucial to the effective construction of large distributed problem solving networks containing tens to hundreds of processing nodes.

The Need for a Testbed

Although these experiments provided intitial empirical validation for the FA/C approach and pointed out an important set of issues that needed to be solved, they were just a first step. These experiments were not based on a realistic distributed task, and more importantly were limited in the scope of issues that could be addressed. Thus, a more extensive set of empirical investigations was necessary in order to better understand the utility and limitations of the FA/C approach Empirical performance measures were needed for a wide range of task and problem solving situations in order to evaluate and analyze the following issues:

- *Self-correcting computational structures.* What and how much uncertainty and error can be handled using these types of computational structures? What are the costs (and trade-offs) in processing and communication to resolve the various types of errors? How does the quality of knowledge used in the network affect the amount of uncertainty and error that can be accomodated?
- *Task characteristics and the selection of an appropriate network configuration:* What characteristics of a task can be used to select the network configuration appropriate for it? When should problem solving among nodes be organized hierarchically? What type of authority relationship should exist among nodes? Should nodes be completely self-directed or should there be certain nodes that decide explicitly what other nodes should do, or should there be a negotiation structure among nodes (Smith & Davis 1981)? Similarly, should information be transmitted on a voluntary basis or only when requested or some mixture of these policies?

The candidate task characteristics to evaluate included the size of the network and the communication topology; the type, spatial distribution, and degree of uncertainty in information; the quality of knowledge in the network;interdependencies among subproblems; and the size of the search space.

Unfortunately, it was difficult to extend the distributed HEARSAY-II speech understanding system for these studies. There were two major reasons for this difficulty: the computation time needed to run experiments and inflexibilities in the design of the system. We discuss these reasons because they point out why extensive experimentation with large knowledge-based AI systems is very difficult.

The use of an existing knowledge-based system as the basic underlying problem solving system in the experiments lent credibility to the simulation results and also avoided the extensive knowledge engineering that normally would have been required. The importance of having a concrete framework to explore ideas cannot be underestimated. Not until the problems of getting the HEARSAY-II speech understanding network to work appropriately in a distributed setting were confronted did many of our intuitions about how to design distributed problem solving networks evolve. However, there were major negative implications of using the real HEARSAY-II speech understanding system. First, it was extremely time consuming to run the multi-node simulations since the underlying problem solving system was large and computationally slow. Second, the speech understanding system did not naturally extend to larger numbers of nodes and more complex communication topologies without significant changes to the system. In part, this is because the speech task is not a realistic distributed processing task and its sensory data is one-dimensional (the time dimension). Third, efficiency considerations in the design of the speech understanding system led to a tight coupling among knowledge sources and the elimination of data-directed control at lower blackboard levels. This tight coupling precluded the exploration of many interesting network architectures. It was not possible to configure nodes with only a partial set of knowledge sources without significant modifications to the knowledge source interaction patterns. Fourth, the sheer size and complexity of knowledge source code modules made modification a difficult and time consuming process.

Basically, the flexibility of the HEARSAY-II speech understanding system (in its final configuration) was sufficient to perform the pilot experiments, but was not appropriate for more extensive experimentation. Getting a large knowledge based system to turn over and perform creditably requires

a flexible initial design but, paradoxically, this flexibility is often engineered out as the system is tuned for high performance. Extensive experimentation, if not originally conceived and maintained as a goal of the system design, is a difficult task.

The Distributed Vehicle Monitoring Testbed

This section introduces the distributed vehicle monitoring testbed, a flexible and fully-instrumented research environment constructed for the empirical evaluation of alternative designs for functionally accurate, cooperative distributed problem solving networks. The concept of the testbed evolved from:

- An understanding of both the difficulties and importance of an empirical approach to issues in distributed problem solving;[1]
- The need for a realistic environment for exploring new paradigms for obtaining global coherence

Here, the motivation for the testbed, its basic structure, and its parameterization and measurement capabilities are described.

Motivation

Our approach to designing the testbed was to:

1 Take a realistic distributed problem solving task and appropriately abstract it to reduce the problems of knowledge engineering, to speed up problem solving, and to make it a more generic and parameterizable task;

2 Develop for this abstracted task a distributed problem solving system that can model (through appropriate parameter settings and pluggable modules of code) a wide class of distributed problem solving architectures;

3 Create a simulation system that can run this distributed problem solving system under varying environmental scenarios, different node and communication topologies, and different task data.

We feel that this approach is the only viable way to gain extensive empirical experience with the important issues in the design of distributed problem solving systems. In short, distributed problem solving networks are highly complex. They are difficult to analyze formally and can be expensive to construct, to run, and to modify for empirical evaluation.

Real distributed problem solving applications are difficult to construct due to the large knowledge acquisition and engineering effort required, and once built, they are difficult to instrument and modify for experimentation. Thus, it is difficult and expensive to gain these experiences by developing a "real" distributed problem solving application in all its detail.

Likewise, we see the formal modelling route as not viable. The research in distributed problem solving is still in its infancy and formal analytic approaches are not yet available. Underlying, the development of analytical approaches are intuitions gained from experiences with actual systems. Without sufficient intuitions for appropriately simplifying and abstracting network problem solving, the development of a model that is both mathematically tractable and accurate is difficult.

Our hope is that the testbed will provide the appropriate environment for acquiring this experience and will eventually be useful in evaluating the accuracy of the analytical models.[2] Especially important are experiences with large distributed problem solving networks of ten to hundreds of nodes. It is with networks of this size the we expect to see the problems of cooperation and coherence dominate and where important intuitions about how to design distributed problem solving networks will arise.

In summary, the empirical approach taken here represents a compromise between the reality of an actual system and simplicity of an analytical model. We have abstracted the task and simplified the knowledge but still are performing a detailed simulation of network problem solving. It should be mentioned that even with significant simplifications the building of the testbed was a substantial implementation effort. However, in contrast to the construction of a "real" application where considerable effort must be spent in knowledge engineering, our efforts have been spent in parameterizing the problem-solving architecture and making the testbed a useful experimental tool

Why Distributed Vehicle Monitoring?

Distributed vehicle monitoring has four characteristics that make it an ideal problem domain for research on distributed problem solving.

First, distributed vehicle monitoring is a natural task for a distributed problem solving approach since the acoustic sensors are located throughout a large geographical area. The massive amount of sensory data that must be reduced to a highly abstract, dynamic map seems appropriate for a distributed approach.

Second, distributed vehicle monitoring can be formulated as an interpretation task in which information is incrementally aggregated to generate the answer map Nilsson

[1] We had, in fact, earlier embarked on the development of such an environment, based on what we called the Distributed Processing Game (Lesser & Corkill, 1978), but failed This venture failed because we had chosen an application for which the knowledge engineering was so complex and our understanding of the task was so vague that we could not develop sufficient knowledge for the system to turn over

[2] The testbed is already beginning to be used in this manner See the work by Pavlin (1983) on initial attempts at formulating a model for distributed intrepretation systems

has termed systems with this characteristic commutative (Nilsson 1980).

Commutative systems have the following properties:

1. Actions that are possible at a given time remain possible for all future times.
2. The system state that results from performing a sequence of actions that are possible at a given time is invariant under permutations of that sequence.

Commutativity allows the distributed vehicle monitoring network to be liberal in making tentative initial vehicle identifications, since generation of incorrect information never precludes the later generation of a correct answer map. Without commutativity, the basic problem solving task would be much more difficult.

Although the generation of the answer map is commutative, controlling node activity is not. Here we enter the realm of limited time and resources. If a crucial aspect of the answer map is not immediately undertaken by at least one node in the network, the network can fail to generate the map in the required time In the determination of node activities, mistakes cause the loss of unrecoverable problem solving time and can therefore eliminate the possibility of arriving at a timely answer map. If the nodes and sensors are mobile, their placement adds another non-commutative aspect to the distributed vehicle monitoring task; a misplaced node or sensor can require substantial time to be repositioned. (We are currently limiting our investigations to stationary nodes and sensors.)

Third, the complexity of the distributed vehicle monitoring task can be easily varied. For example:

- Increasing the density of vehicle patterns in the environment increases the computational and communication load on the network.
- Increasing the similarity of the vehicles and patterns known to the network increases the effort required to distinguish them
- Increasing the amount of error in the sensory data increases the effort required to discriminate noise from reality.

Fourth, the hierarchical task processing levels coupled with the spatial and temporal dimensions of the distributed vehicle monitoring task permit a wide range of spatial, temporal, and functional network decompositions. Node responsibilities can be delineated along any combination of these dimensions.

An important decision in the design of the testbed was the level at which the network would be simulated. An abstract modeling level, such as the one used by Fox (1979), that represents the activities of nodes as average or probabilistic values accumulated over time would not capture the changing intermediate processing states of the nodes. It is precisely those intermediate states that are so important in both building and evaluating in a realistic way different network coordination strategies. Instead, the testbed duplicates (as closely as possible) the data that would be generated in an actual distributed vehicle monitoring network as well as the effect of knowledge and control strategies on that data This approach also allows users of the testbed to receive concrete feedback about how their algorithms are performing However, because the purpose of building the testbed is to evaluate alternative distributed problem solving network designs rather than to construct an actual distributed vehicle monitoring network, a number of simplifications of the vehicle monitoring task were made (Table 1). The goal of these simplifications was to reduce the processing complexity and knowledge engineering effort required in the testbed

The major task simplifications in the Distributed Vehicle Monitoring Testbed include:

- The monitoring area is expressed as a two-dimensional square grid, with a maximum spatial resolution of one unit square.
- The environment is not sensed continuously Instead, it is sampled at discrete time intervals called *time frames*
- Frequency is represented as a small number of *frequency classes*
- Communication from sensor to node uses a different channel than internode communication.
- Internode communication is subject to random loss, but if a message is received by a node it is received without error
- Sensor to node communication errors are treated as sensor errors.
- Signal propagation times from source to sensor are processed by the (simulated) low-level signal processing hardware of the sensor;
- Sensors can make three types of errors: failure to detect a signal; detection of a non-existent signal; and incorrect determination of the location or frequency of a signal.
- Sensors output signal events, which include the location of the event (resolved to a unit square), time frame, frequency (resolved to a single frequency class), and belief (based on signal strength)
- Incompletely resolved location or frequency of a signal is represented by the generation of multiple signal events rather than a single event with a range of values
- Nodes, sensors, and internode communication channels can temporarily or permanently fail without warning

Table 1.
The Simplified Vehicle Monitoring Task

without significantly changing the basic character of the distributed interpretation task.

A second design decision was to instrument the testbed fully. The testbed includes measures that indicate the quality of the developing solution at each node in the network, the quality of the developing solution in the network as a whole, and the potential effect of each transmitted message on the solution of the receiving node. This is made possible through the use of an *oracle* containing the structure of the actual problem solution.

A third decision in the design of the testbed was to make it parameterized. Experience with complex artificial intelligence systems demonstrated the difficulty of experimenting with alternative knowledge and control strategies. As a result, potential experimentation with the system is often not performed. Incorporated into the testbed are capabilities for varying:

- The knowledge sources available at each node, permitting the study of different problem solving decompositions;
- The accuracy of individual knowledge sources, permitting the study of how different control and communication policies perform with different levels of system expertise;[3]
- Vehicle and sensor characteristics, permitting control of the spatial distribution of ambiguity and error in the task input data;
- Node configurations and communication channel characteristics, permitting experimentation with different network architectures;
- Problem solving and communication responsibilities of each node, permitting exploration of different problem solving strategies;
- The authority relationships among nodes, permitting experimentation with different organizational relationships among nodes.

The result is a highly flexible research tool which can be used to explore empirically a large design space of possible network and environmental combinations.

Testbed Node Architecture

The Distributed Vehicle Monitoring Testbed simulates a network of HEARSAY-II nodes working on the vehicle monitoring task. Each node is an architecturally-complete

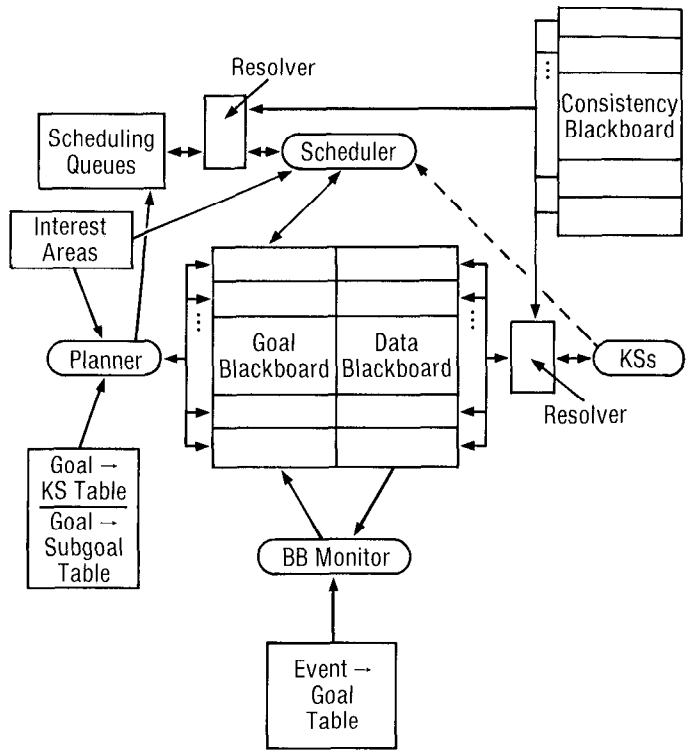

Figure 2: Testbed Node Architecture.

HEARSAY-II [4] system (Erman *et al* 1980), capable of solving the entire vehicle monitoring problem if it is given all of the sensory data and makes use of all of its knowledge. This permits any subset of the knowledge sources to be used at a node and allows the simulation of a single node (centralized) system to provide a benchmark for various distributed networks monitoring the same environment.

The basic HEARSAY-II architecture has been extended in each testbed node to include the capability of communicating hypotheses and goals among nodes, more sophisticated local control, and an interface to meta-level network coordination components (Corkill 1981, 1982, 1983). In particular, communication knowledge sources, a goal blackboard, a planning module, and a meta-level control blackboard have been added (Figure 2). The testbed also has several components that are used to measure the performance of each node and the overall network and to vary the "intelligence" of each node's knowledge sources and scheduler. These components are the *consistency blackboard* and the knowledge source and scheduler *resolvers*.

The Structure of the Data Blackboard

Hypothesized vehicle movements are represented on the

[3]The quality of the knowledge used by each node to distinguish between consistent and inconsistent data plays a major role in the success of a functionally accurate, cooperative approach A network using low quality knowledge is unable to detect subtle inconsistencies among tentative partial results and may be unable to arrive at an acceptable solution As the quality of knowledge used in the network is improved, the network should generate an answer with greater accuracy in less time.

[4]With knowledge sources appropriate for the task of vehicle monitoring

data blackboard. This blackboard is partitioned into four task abstraction levels: signal, group, vehicle, and pattern (Figure 3). *Signals* are at the lowest abstraction level and are the output of low-level analysis of sensory data. Each signal includes the frequency, approximate position, time frame of detection, and belief (based partly on signal strength and sensor quality) of the acoustic signal as well as the identity of the detecting sensor. Signals are the basic input to the problem solving network.

At the next level in the data hierarchy are signal groups. A *group* is a collection of harmonically related signals (emanating from a common source). Each group includes the fundamental frequency of the related signals and its approximate position, time frame, and belief (a function of the beliefs and characteristics of the related signals).

Vehicles are the next level in the data hierarchy. A *vehicle* consists of a collection of groups associated with a particular vehicle. Vehicles include the identity of the vehicle and its time frame, approximate position, and belief.

At the highest level of processing are vehicle patterns. A *pattern* is a collection of particular vehicle types with a particular spatial relationship among them. Patterns were included in the testbed to investigate the effects of strong constraints between distant nodes. A pattern includes the identity of the pattern and its time frame, approximate position, and belief. A single vehicle can be a pattern.

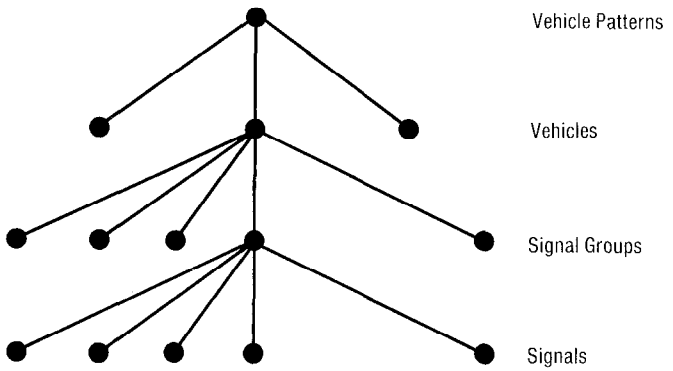

Figure 3 Vehicle Monitoring Task Processing Levels
Forming a vehicle pattern from sensory signals involves combining harmonically related signals into signal groups Various signal groups can collectively indicate a particular type of vehicle Specific vehicle types with a particular spatial relationship among themselves form a vehicle pattern.

The desired solution, or answer map, is produced from the vehicle patterns based upon their beliefs and continuity over time. There are two types of answer map distribution: one where a complete map is to be located at one or more answer sites within the monitored area and one where a partial (spatially relevant) map is to be located at numerous sites within the area In distributed vehicle monitoring tasks such as air or ship traffic control, both distributions of the answer map may be required. Each node might use its portion of the distributed map to control nearby vehicles, while the complete map might be produced for external monitoring of the network.

Each of these four abstraction levels is further divided into two levels, one containing location hypotheses and one containing track hypotheses. A *location hypothesis* represents a single event at a particular time frame A *track hypothesis* represents a connected sequence of events over a number of contiguous time frames.

These orthogonal partitionings result in the eight blackboard levels shown in Figure 4. Location hypotheses are formed from location hypotheses at the next lower abstraction level. Track hypotheses can be formed from location hypotheses at the same abstraction level or from track hypotheses at the next lower level. The task processing level most appropriate for shifting from location hypotheses to track hypotheses is dependent on the problem solving situation.

The relationships among the hypotheses at each level is supplied to the testbed as part of a testbed *grammar.* Changing the grammar automatically varies behavior throughout the testbed. By increasing the size and connectivity of the grammar, the interpretation task can be made more difficult Another aspect of a testbed grammar that specifies the difficulty of the interpretation task involves tracking vehicle movement. The tracking component of a testbed grammar contains two values: the maximum velocity of a vehicle (and implicitly, events at all levels) and the maximum acceleration of a vehicle. These values are used in the creation and extension of track hypotheses. By reducing the constraints on vehicle movement, the tracking task becomes more difficult.

Knowledge Source Processing

An important consideration in developing the set of knowledge sources for the testbed was to structure processing so that information could be asynchronously transmitted and received at any blackboard level. This permits exploration of a wide range of different processing decompositions based on partially configured nodes (nodes without all knowledge sources) without modifying the knowledge source modules and local control structures.

There are six basic problem solving activities performed by the processing knowledge sources in the testbed. They are:

Location Synthesis – Abstracting location hypotheses at one level of the blackboard into a new location hypothesis at the next higher location level

Track Synthesis – Abstracting track hypotheses at one level of the blackboard into a new track hypothesis at the next higher track level

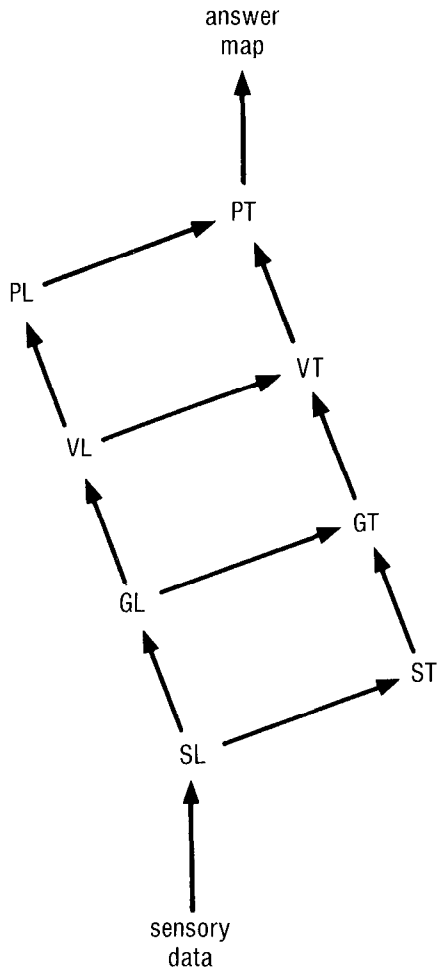

Figure 4 Blackboard Levels in the Testbed
The eight blackboard levels in the testbed are: location (SL) signal track (ST) group location (GL) group track (GT) vehicle location (VL) vehicle track (VT) pattern location (PL) pattern track (PT) The arrows indicate the four possible synthesis paths from sensory data to generation of the answer map

Track Formation - Combining a location hypothesis in one time frame with a "matching" location hypothesis in an adjacent time frame to form a one-segment track hypothesis

Track Extension – Extending a track hypothesis into an adjacent time frame by combining it with a "matching" location

Location-to-Track Joining Taking a location hypothesis and combining it with a "matching" track hypothesis that begins or ends in an adjacent time frame

Track Merging - Merging two overlapping or abutting track hypotheses into a single track hypothesis at the same abstraction level

Details of Goal Processing in a Node

In order to permit more sophisticated forms of cooperation among nodes in the system, we have integrated goal-directed control into the data-directed control structure of the basic HEARSAY-II architecture This has been accomplished through the addition of a goal blackboard and a planner.

The *goal blackboard* mirrors the structure of the data blackboard. Instead of hypotheses, the basic data units are *goals*, each representing an intention to create or extend a hypothesis with particular attributes on the data blackboard For example, a simple goal would be a request for the creation of a vehicle location hypothesis above a given belief in a specified area of the data blackboard.[5]

Goals are created on the goal blackboard by the *blackboard monitor* in response to changes on the data blackboard These goals explicitly represent the node's intention to abstract or extend particular hypotheses. Goals received from another node may also be placed on the goal blackboard Placing a high-level goal onto the goal blackboard of a node can effectively bias the node toward developing a solution in a particular way.

The *planner* responds to the insertion of goals on the goal blackboard by developing plans for their achievement and instantiating knowledge sources to carry out those plans The *scheduler* uses the relationships between the knowledge source instantiations and the goals on the goal blackboard as a basis for deciding how the limited processing and communication resources of the node should be allocated

Communication Knowledge Sources

Internode communication is added to the node architecture by the inclusion of *communication knowledge sources* These knowledge sources allow the exchange of hypotheses and goals among nodes in the same independent and asynchronous style used by the other knowledge sources There are six types of communication knowledge sources in the testbed:

Hypothesis Send – Transmits hypotheses created on the blackboard to other nodes based on the level, time frame, location, and belief of the hypothesis

Hypothesis Receive – Places hypotheses received from other nodes onto the node's blackboard Incoming hypotheses are filtered according to the characteristics of the received hypothesis to ensure that the node is interested in the information Hypothesis Receive uses a simple model of the credibility of the sending node to possibly lower the belief of the received hypothesis before it is placed on the blackboard

[5]An important aspect in the structure of the integrated control architecture is a correspondence between the blackboard area covered by the goal and the blackboard area of the desired hypothesis This correspondence allows the planner to relate goals and hypotheses quickly

Goal Send – Transmits goals created on the goal blackboard to other nodes based on the level, time frames, regions, and rating of the goal Goal Send transmits goals based on meta-level control information – whether or not the node is to attempt to achieve the goal locally

Goal Help – Transmits goals that the node's planner has determined cannot be satisfied locally (possibly after executing a number of local problem solving knowledge sources)

Goal Receive – Places goals received from other nodes onto the node's goal blackboard. Incoming goals are filtered according to the characteristics of the received goal to ensure that the node is interested in receiving goals of that type Goal Receive uses a simple model of the node's authority relationship with the sending node to possibly lower the rating of the received goal before it is placed on the blackboard

Goal Reply – Transmits hypotheses created on the blackboard in response to a received goal requesting information from the node

Experimentation with more complex versions of these communication knowledge sources is easily accomplished by simulating a more sophisticated knowledge source by:

- modifying its power (*cf* **Modifying Knowledge Source Power**);
- modifying the code of the knowledge source to use more sophisticated knowledge in its choices (this can done by adding code that filters the input or output of the knowledge source);
- completely replacing a knowledge source with an alternative module

Measuring Node and Network Performance

An important aspect of our use of the testbed is measuring the relative performance of various distributed problem solving configurations and strategies. For example, we conjecture that in a network with accurate knowledge and with input data that has low error, organizing the system hierarchically and using an explicit control and communication strategy would be effective Likewise, we conjecture that in systems with weaker knowledge sources and with more errorful input data, more cooperative and implicit control/communication strategies are desirable

In order to understand the reasons for differences in the performance characteristics of alternative systems organizations, dynamic measures are needed that take into account the intermediate state of system processing and thus permit observations of performance over time. For example, one way of measuring the effectiveness of different communication strategies is to develop measures that evaluate the effect of each transmitted message on the current processing state of the receiving node. The need for measuring the intermediate states of processing have led us to develop a semi-formal model for analyzing how a HEARSAY-II-like system constructs an accurate solution and resolves the uncertainty and error in its input data (Lesser 1980) This measure increases as the system becomes more certain of the consistency of "correct" hypotheses and decreases as the system becomes more certain of the consistency of "incorrect" hypotheses The "correctness" of hypotheses is obtained from a hidden data structure called the *consistency blackboard*, which is precomputed from the simulation input data. This blackboard holds what the interpretation would be at each information level if the system worked with perfect knowledge. This blackboard is not part of the basic problem solving architecture of a node but rather is used to measure problem solving performance from the perspective of the simulation input data. The consistency blackboard is also used to mark consistent and false hypotheses (and the activities associated with them) in system output

Modifying Knowledge Source Power

One parameter that can have a significant effect on the performance of the network is the problem solving expertise of the nodes. The ability of a knowledge source to detect local consistencies and inconsistencies among its input hypotheses and to generate appropriate output hypotheses is called the *power* of the knowledge source Knowledge source power ranges from a perfect knowledge source able to create output hypotheses with beliefs that reflect even the most subtle consistencies among its input hypotheses down to a knowledge source that creates syntactically legitimate output hypotheses without regard to local consistency and with beliefs generated at random. Note that a perfect knowledge source is not the same as an omniscient one. A perfect knowledge source can still generate an incorrect output hypothesis if supplied with incorrect, but completely consistent, input hypotheses

The testbed can modify the power of a knowledge source to be anywhere along this range. This is achieved by separating each knowledge source into two stages: a candidate generator and a resolver. The *candidate generator* stage produces plausible hypotheses for the output of the knowledge source and assigns each hypothesis a tentative belief value. The candidate generator stage for each knowledge source in the testbed incorporates relatively simple domain knowledge There are two types of knowledge used in the candiate generator to form possible output hypotheses based on patterns of input hypotheses. One type of knowledge derives patterns from the particular testbed grammar and knowledge of sensor error characteristics. The other type of knowledge is used to compute a belief for each output hypothesis using the beliefs of the input hypotheses and knowledge about the relative consistency of the input pattern All of the knowledge used by the candidate generator is easily varied through either parameter settings or pluggable code modules

The next stage, the *resolver*, uses information provided

by the consistency blackboard to minimally alter the initial belief values of these plausible hypotheses to achieve, on the average, a knowledge source of the desired power. The hypotheses with the highest altered beliefs are then used by the resolver stage as the actual output hypotheses of the knowledge source.

The alteration of hypothesis belief values by the resolver stage can be used to simulate the detection of more subtle forms of local consistency than is provided by the candidate generator's knowledge (and thereby increase the apparent power of the knowledge source). Hypothesis belief alteration can also be used to degrade the performance of the candidate generator (and thereby reduce the apparent power of the knowledge source) [6]

Even with the flexibility and detail of our approach, there are limitations:

- Our simulation of knowledge source resolving power is based on a combination of simple knowledge about local consistency and reference to an oracle, while real knowledge sources attempt to infer truth from local consistency alone (and falsehood from local inconsistency) [7]
- The behavior of different simulated knowledge sources sharing similar errors in knowledge will not be correlated due to our statistical approach to knowledge source simulation

Given these limitations, we do not expect a simulated knowledge source to behave exactly as a real knowledge source We feel, however, that the essential behavior of each knowledge source has been captured so that system phenomena are adequately simulated.

Local Node Control in the Testbed

An important capability of the testbed is the ease with which alternate control and communication strategies can be explored. This exploration has two aspects. The first is the ability to perform experiments comparing the performance of different control strategies (for example, the performance of a hierarchical network versus a laterally organized network) The second aspect is the ability to augment the basic testbed node architecture with additional control components (for example, adding a meta-level control component that varies the organizational relationships among nodes dynamically). Both types of experimentation are possible with the testbed. This section discusses how the local node control architecture has been structured to accomplish both types of experimentation.

Interest Areas

A key aspect of the control framework implemented in the testbed is the use of a nonprocedural and dynamically variable specification of the behaviors of each local node's planner, its scheduler, and its communication knowledge sources. Called *interest areas*, these data structures reside on the meta-level control blackboard and are used to implement particular network configurations and coordination policies.

There are six sets of interest areas for each node in the testbed:

Local Processing Interest Areas – Influence the local problem solving activities in the node by modifying the priority ratings of goals and knowledge source instantiations and the behavior of the node's planner and scheduler

Hypothesis Transmission Interest Areas – Influence the behavior of HYP-SEND knowledge sources in the node

Hypothesis Reception Interest Areas – Influence the behavior of HYP-RECEIVE knowledge sources in the node

Goal Transmission Interest Areas – Influence the behavior of GOAL-SEND knowledge sources in the node

Goal Help Transmission Interest Areas – Influence the behavior of GOAL-HELP knowledge sources in the node

Goal Reception Interest Areas – Influence the behavior of GOAL-RECEIVE knowledge sources in the node.

Each interest area is a list of regions of the data or goal blackboard.

Each local processing interest area has a single parameter associated with it: a weight specifying the importance of performing local processing within the interest area. Transmission interest areas (hypothesis transmission, goal transmission, and goal help transmission) are specified for one or more lists of nodes that are to receive information from the node. Similarly, reception interest areas (hypothesis reception and goal reception) are specified for lists of nodes that are to transmit information to the node Each transmission interest area has a weight specifying the importance of transmitting hypotheses or goals from that area (to nodes specified in the node-list) and a threshold value specifying the minimum hypothesis belief or goal rating needed to transmit from that area Each reception interest area has a weight specifying the importance of receiving a hypothesis or goal in

[6] The work by Paxton on the SRI speech understanding system (Paxton 1978) comes closest to our approach He used ground consistency information to simulate statistically the output of the low level acoustic processor in the SRI speech system Our approach differs from Paxton's in that it dynamically relates characteristics of the inputs of a knowledge source to the characteristics of its outputs, while Paxton's does not The output of his model depends on precomputed behavioral statistics which are independent of the belief values and consistency values of its inputs Because of this difference, we are able to simulate any or all knowledge sources in our system, while Paxton's model is valid only for front-end processing of input data similar to those used to compute the statistics

[7] In order to capture more closely the notion of local consistency, we can include on the consistency blackboard false hypotheses that would appear to be consistent by even a perfect knowledge source operating at that blackboard level The resolver judges the consistency of these false hypotheses (termed "correlated-false" hypotheses) in the same way as it does true hypotheses

that area (from a node specified in the node-list), a minimum hypothesis belief or goal rating needed for the hypothesis or goal to be accepted, and a credibility weight. The credibility weight parameter is used to change the belief of received hypotheses or the rating of received goals. A node can reduce the effect of accepting messages from a node by lowering the belief or rating of messages received from that node Each hypothesis reception interest area also has a focusing weight parameter that is used to determine how heavily received hypotheses are used in making local problem solving focusing decisions.

Rating Goals and Subgoaling

Goal ratings specify the importance of creating hypotheses with particular attributes on the data blackboard. They influence the behavior of the planner, the scheduler, and the goal communication knowledge sources. The knowledge source instantiation rating calculation is basically a weighted sum of a data-directed and a goal-directed component. The data-directed component captures the expected belief of an output hypothesis (as specified in the knowledge source instantiation's output-set attribute) The goal-directed component measures the ratings of goals that would be satisfied (at least in part) by an output hypothesis. The goal-weighting parameter can be adjusted to change the importance given to producing strongly believed hypotheses versus satisfying highly-rated goals. Gaussian noise is added to the rating calculation to simulate knowledge source precondition procedures with imperfect output hypothesis estimation capabilities

In addition to instantiating knowledge sources to achieve a goal, the planner can also create subgoals that reflect the importance of lower-level data in achieving the original goal and that, if satisfied, increase the likelihood of achieving the original goal. Subgoaling is an effective means of focusing low-level synthesis activities based on high-level expectations.[8]

The knowledge needed to perform subgoaling is based on the behavior of the testbed knowledge sources and is parameterized by the grammar. Because subgoaling requires some effort, its use needs to be controlled. In the testbed, subgoaling is controlled in two ways: by restricting subgoaling to particular levels and by a minimum rating threshold for a goal to be subgoaled. The relative settings of these parameters strongly influence the balance between local and external direction Examples of how specific control and communication relationships are specified in the testbed are presented in a recent paper (Corkill & Lesser 1983).

[8]There are no prediction knowledge sources in the testbed Predictive knowledge is used by the planner to generate predictive goals that can be subgoaled to focus activity on lower blackboard levels

Knowledge Source Precondition Procedures

The overall performance of each node depends on the ability of its planner and scheduler to correctly estimate which of the potential knowledge source actions is most likely to improve the current problem solving state as well as the cost of performing that action. In "real" systems, this estimation is based in part on information provided by each knowledge source to the scheduler about the output the knowledge source is likely to produce given particular input hypotheses (the knowledge source *response frame* (Hayes-Roth & Lesser 1977). This estimation is usually fast and approximate - it is made without a detailed analysis of the knowledge source's input data. Increasing uncertainty in this estimation makes it less likely that the planner and scheduler will appropriately decide what knowledge source actions to perform.

In order to investigate the effects of this uncertainty the testbed simulation *preexecutes* the entire knowledge source as the precondition procedure. The knowledge source does not actually create any hypotheses or goals, but instead places an exact specification of their attributes in the *output-set* attribute of the knowledge source instantiation. The output-set provides an exact description of what the knowledge source instantiation will do if executed (The output-set is updated if the input context of the knowledge source instantiation is modified while it is awaiting execution) The actual hypotheses or goals are created when the knowledge source instantiation executes.

The information contained in the output-set allows the knowledge source instantiation rating to be made with perfect knowledge of the knowledge source instantiation's behavior. Precondition procedures with less than perfect estimation luffi abilities are simulated by perturbing these perfect ratings. The details are described in the next section

Rating Knowledge Source Instantiations

The knowledge source instantiation rating calculation is basically a weighted sum of a data-directed and a goal-directed component The data-directed component captures the expected belief of an output hypothesis (as specified in the knowledge source instantiation's output-set attribute). The goal-directed component measures the ratings of goals that would be satisfied (at least in part) by each output hypothesis. The goal-weighting parameter adjusts the importance given to satisfying highly-rated goals versus producing strongly believed hypotheses The weighted sum of these two components is computed for each output hypothesis in the knowledge source instantiation's output-set attribute and the maximum value (multiplied by the knowledge source efficiency estimate) is used as the base rating for knowledge source instantiation

Since the testbed precondition procedures precompute the actual output hypotheses of the knowledge source instantiation, the scheduler's base rating calculation uses the

exact beliefs of the output hypotheses and the goals that they satisfy Gaussian noise can be added to this base rating to simulate the effects of knowledge source precondition procedures that are imperfect in their estimation of output hypotheses's beliefs and of goal satisfaction.

The knowledge sources' precondition procedures use information localized to a particular region of the data blackboard in estimating the belief values of output hypotheses. On the other hand, the scheduler is in a position to determine how a knowledge source instantiation's expected output hypotheses fit into the overall developing solution at the node. This difference in viewpoint leads to an interesting engineering issue. Should the scheduler rely solely on the myopic estimations of the precondition functions in rating a knowledge source instantiation or should it be given domain-dependent knowledge of its own to determine consistencies between knowledge source instantiations? To experiment with this issue, an oracle weighting in the data-directed component can be used to introduce the consistency of each output hypothesis (as specified on the consistency blackboard) into the rating calculation As with the knowledge source instantiations themselves, this consistency information is used to simulate the effects of developing additional knowledge which can better detect the consistencies among hypotheses

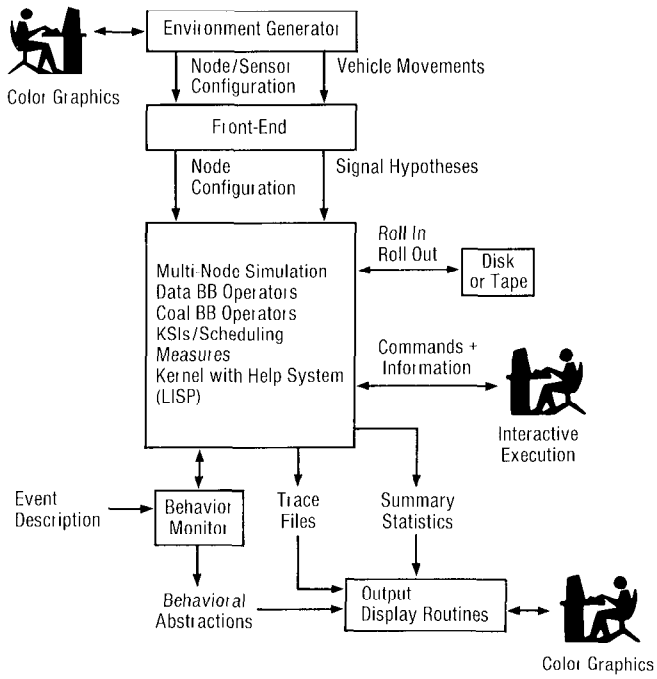

Figure 5: Testbed Kernel and Related Subsystems

Facilities for Experimentation

The testbed kernel is surrounded by a number of other subsystems to facilitate experimentation by making it easy to vary the parameters of an experiment and to analyze the results of an experiment (Figure 5)

FRONTEND knowledge source is the special, simulation-level knowledge source used to initialize the testbed network. It is always the first knowledge source executed in an experiment. The FRONTEND reads a complete specification of the run from an input file called the *environment file.* The environment file contains all the input data for the testbed, and consists of system, structural, and environmental data *System data* denotes basic parameters of the simulated vehicle monitoring system: a seed for random number generation, the minimum and maximum location and time ranges, and the numbers of nodes and sensors. *Structural data* denotes the spatial relationships among nodes and the grammar used by knowledge source candidate generators By varying this grammar, the number of legal patterns of hypotheses can be varied. The most constrained grammar would be one that only allowed the particular scenario for the experiment in question to be recognized. Thus, the nature and the scope of consistency constraints used by knowledge sources to resolve errors can be altered This ability to modify the grammar combined with the ability to vary the local resolving power of knowledge source provides a powerful tool for varying the knowledge expertise in the simulated system. *Environmental data* denotes the actual environment for the vehicle monitoring system: locations of patterns and vehicles at various time frames, and information concerning missing and false patterns, vehicles, groups, and signals at various time frames Environmental data is used in conjunction with the structural data by the FRONTEND to create the consistency blackboard.

The environment file has gone through several design iterations as we have recognized the interdependencies among the parameters that must be specified for a testbed experiment and the difficulties of correctly specifying these parameters for networks of more than a few nodes. In its present form, it allows the specification of generic classes of node types, local problem solving capabilities, authority relationships, communication policies, and sensor characteristics. These classes are then instantiated to individual nodes and sensors in the network.

The FRONTEND, in its generation of sensor data, can introduce controlled error (noise) to model imperfect sensing Noise is added to the location and signal class and the distance of the signal from the sensor FRONTEND processing is also parameterized so that either these signals can be introduced into the nodes all at once or at the time they are sensed The former provision allows exploration of systems in which there are burst receptions of sensor data

To facilitate the inclusion of additional control, display, and measurement routines into a particular experiment, the testbed has a number of programming "hooks" available to the experimenter Each hook consists of a dummy module that can be easily redefined to include calls to the experimeter's procedures. In the testbed, there is a hook

at the beginning of the simulation, another hook following the FRONTEND (when all sensory data and the consistency blackboard have been determined), one prior to each knowledge source execution at each node, one when messages are transmitted or received, and one when the simulation is finished. Each hook has sufficient information available (such as the current node that is executing, the type of knowledge source to be executed, the simulation time, etc) to allow the experimenter's procedures to decide whether or not they are interested in being executed. The experimenter's procedures have complete access to all information in the testbed.

In order to help in the analysis of the results of an experiment, a number of tools have been developed: a selective trace facility, a summary statistics facility, an interactive, menu-driven debugging facility, an event monitoring facility, and a color-graphics display facility. Each of these tools use the information on the consistency blackboard to highlight their presentations. For example, the trace facility marks knowledge source instantiations based on the correctness (consistency) of their input and output hypotheses This permits the experimenter to quickly scan a large amount of data for unexpected phenomena.

The trace facility presents a chronological trace of the knowledge sources creation and execution and the associated creation of hypotheses and goals and a run The user can vary the level of details of the internal operations of the systems that are to be traced.

The summary statistics facility is used at the end of a run to generate a set of measures that indicate the performance of various aspects of the systems. These statistics are both on a node and system basis

In addition to these fairly common analysis tools, we feel that there is need for tools that permit a more dynamic and high-level view of the distributed and asynchronious activity of the simulated nodes. An event monitoring facility, which has not yet been fully implemented, will permit a user to define and gather statistics on such user-defined events as the average time it takes for a node to receive a hypothesis and incorporate the received information into a message to be transmitted to another node (Bates & Wileden, 1982).

Another facility which is currently operational in a limited form is a color-graphics output facility. The current output display provides dynamic visual representations of the distribution of hypotheses in the x-y space of the Distributed Sensor Network during a simulation Location and track hypotheses are displayed as symbols and paths connecting symbols, respectively, in the physical x-y space The level, node, belief, and type of event of each hypothesis is encoded in its representation. Through this display, it is possible to get a high-level view of the relationship among the nodes' current interpretations and their relationship to the actual monitored tracks. The hypotheses displayed can be selected according to the characteristics of any of their attributes For example, it is possible to display only those hypotheses above some belief value or those on a certain level, etc. In addition, an ordering function exists to rank the hypotheses to be displayed according to several attributes (node, level, type of event, and end-time) allowing less important hypotheses to be replaced (painted over) by more important ones. We are also working on other display formats that show more abstract measures of system performance such as the transmission rate among nodes, the current reliability of nodes, etc.

Testbed Status, Uses, and Future Directions

The testbed, which has been operational since January of 1982, has been a much larger system building effort than was originally anticipated at the onset of the project. The current size of the testbed, which is written in CLISP (Corkill 1980) running under VMS, including support facilities is approximately 500K bytes of compiled lisp code Over the three year development period, between fifteen and twenty man-years of effort have gone into the construction of the testbed

This extensive construction effort has come in part from the large number of major design iterations. The basic concept of the testbed has stayed intact through these iterations but significant modifications to all aspects of the testbed have been required as we came to understand how to better parameterize the various components.

It should also be mentioned that even though the task knowledge was simplified, considerable effort was still required to get the planner and knowledge sources to work effectively together The testbed uses a very general mechanism for knowledge source interaction, and a number of interaction patterns that would not occur in a centralized system do occur in distributed networks.

The saving grace of all these redesign efforts was that it lead us to a better understanding of how knowledge-based AI systems and, more specifically, knowledge-based distributed problem solving systems operate In short, designing a knowledge-based AI system remains an art and requires considerable iteration.

A key concern that we still have about the testbed design, which cannot be answered without extensive use of the testbed, is the range of issues that can be effectively explored in the testbed. So far, only one extensive set of experiments have been run in the testbed. These experiments emphasized the use of the testbed to explore the effects of different network problem solving strategies (Corkill & Lesser, 1983b). Characteristics that were varied included:

- whether communication is *voluntary* (a node transmits hypotheses at its pleasure), *requested* (a node transmits hypotheses only when that information is requested by another node), or a *mixed initiatve* combination of voluntary and requested hypotheses (a node volunteers only its highest rated hypotheses and awaits requests before transmitting any other hypotheses);
- whether a node is *self-directed* or *externally-directed* in its activities (or a combination of both);

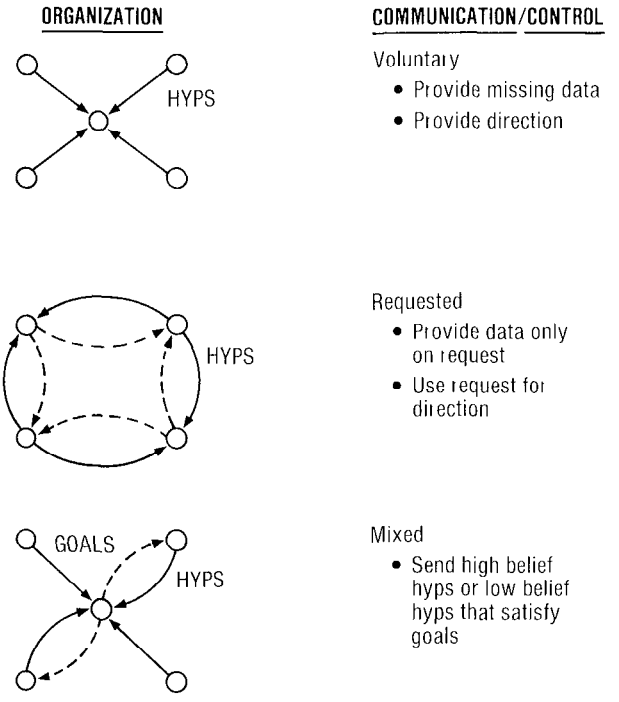

Figure 6 Alternative Distributed Problem Solving Strategies

- whether hypotheses, goals, or both hypotheses and goals are used for internode coordination

The organizational strategies were evaluated using two different network architectures: a laterally-organized, four-node network with broadcast communication among nodes and a hierarchically-organized, five-node network in which the fifth node acts as an integrating node (Figure 6). In both architectures, the network is structured so that the nodes cooperate by exchanging partial and tentative high-level hypotheses.

Although these experiments did not explore all the parameters in the testbed, they do provide evidence of the utility and flexibility of the testbed as a research tool The different network problem solving strategies and environmental configurations were easily expressed, and interesting empirical results indicating the performance of the different strategies were obtained. The most interesting of these results were how different organizational and control strategies performed in a noisy input environment that created the potential for the exchange of distracting information among the nodes.

As part of these initial experiments, we had planned to explore larger node configurations (with 10 to 20 nodes). However, only a few of these larger test cases were run Between 3 and 5 hours of CPU time were required to simulate one of these larger experiments The efficiency of the simulation is crucial to exploring large node configurations. We are now beginning the process of selectively tuning the testbed but do not have a feel for the potential speedup We are also beginning work on modifying the testbed to run as a parallel simulation system on a local area network of VAX 11/750s [9]

In setting up larger and more complex configurations, a large number of interrelated parameters needed to be specified. This specification process was both time consuming and error-prone. To remedy this problem, we are now building additional graphical support tools to allow an experimenter to design and view the network configuration Additionally, we are developing tools allowing complex node topologies to be specified in a generic way, independent of any specific number of nodes (Corkill & Pattison, 1983c)

We now firmly believe that no matter how flexible and general a research tool is, if it is not convenient to use, or if the empirical results are not easy to understand, only a small subset of its capabilities will be exploited.

Conclusion

In this article we have described the area of distributed problem solving and discussed some of the important issues that must be addressed. We also introduced the Functionally Accurate, Cooperative approach with its emphasis on dealing with uncertain data and control information as an integral part of network problem solving

The need for an empirical investigation of distributed problem solving was discussed, especially with regard to network coordination. Such an investigation requires a flexible experimental tool. The Distributed Vehicle Monitoring Testbed was presented as an example of such a tool.

The testbed facilitates the exploration of the following factors in distributed problem solving:

- node-node and node-sensor configurations;
- mixes of data- and goal-directed control in the system;
- distributions of uncertainty and error in the input data;
- distributions of problem solving capability in the system;
- types of communication policies used;
- communication channel characteristics;
- the problem solving and communication responsibilities of each node; and
- the authority relationships among nodes

The multiple dimensions of independent control and the detailed level of simulation in the testbed provide what we feel is a very useful environment for experimentation.

[9] We had initially hoped to solve the effeciency problem through the use of two different testbeds, one written in LISP as the development system and the other in PASCAL as the production system Unfortunately, with the extensive design iterations that occurred during the building of the testbed, it was impossible to keep the PASCAL implementation current and eventually it was dropped

There is a need for more extensive experimentation with AI systems All too often getting a large knowledge-based AI system to work at all is the major goal. Extensive experimentation with the system over a range of conditions is rarely done. The testbed is one of the few exceptions In this presentation we have emphasized what makes the testbed a flexible experimental tool Many of these techniques are appropriate for any large knowledge-based AI system.

References

Bates, P. C & Wileden J C (1982) EDL: *A Basis for distributed system debugging tools.* Proceedings of the Fifteenth Hawaii International Conference on System Science, pages 86–93

Chandrasekaran B (1981) *Natural and social system metaphors for distributed problem solving: Introduction to the issue* IEEE Transactions on Systems, Man, & Cybernetics, SMC-11(1):1-5.

Corkill, D D (1980) CLisp reference manual (an interactive help facility) Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts

Corkill, D D & Lesser, V R (1981) A goal-directed HEARSAY-II architecture: Unifying data and goal directed control. Technical Report 81-15, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts

Corkill, D D., Lesser, V R., & Hudlicka. E (1982) *Unifying data-directed and goal-directed control An example and experiments* In Proceedings of the Second National Conference on Artificial Intelligence, Pittsburgh, PA, 143–147

Corkill, D. D (1983) A Framework for Organizational Self-Design in Distributed Problem Solving Networks PhD Thesis, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts (Available as Technical Report 82-33, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts, 1982)

Corkill, D D & Lesser, V R (1983) The use of meta-level control for coordination in a distributed problem solving network In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence,* Washington, D.C., in press.

Corkill, D D & Pattison E. Specifing organizational relationships Technical report, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts, in preparation

Davis, R (1980) Report on the workshop on distributed artificial intelligence *SIGART Newsletter,* (73):43-52

Davis, R & Smith R. G (1981) Negotiation as a metaphor for distributed problem solving AI Memo 624, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts

Davis, R. (1982) Report on the second workshop on distributed artificial intelligence. SIGART *Newsletter,* (80):13-23

Erman, L D., Hayes-Roth, F , Lesser, V R , & Reddy, D.R. (1980) The HEARSAY-II speech understanding system: Integrating knowledge to resolve uncertainty *Computing Surveys* 12(2):213-253.

Fehling, M & Erman L (1983) Report on the third annual workshop on distributed artificial intelligence SIGART *Newsletter* (84):3-12

Fox, M S. (1979) Organization structuring: Designing large complex software Technical Report CMU-CS-79-155, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania

Hayes-Roth, F. & Lesser, V R (1977) Focus of attention in the HEARSAY-II speech understanding system In IJCAI-77, 27-35

Hewitt, C (1977) Viewing control structures as patterns of passing messages. *Artificial Intelligence* 8(3):323-364

Kahn, R E , S A. Gronemeyer, J Burchfiel, & R. C Kunzelman (1978) Advances in packet radio technology *Proceedings of the* IEEE 66(11):1468-1496

Kornfeld, W A (1979) ETHER: A parallel problem solving system *In IJCAI-79,* 490-492

Lacoss, R., & Walton, R (1978) Strawman design of a DSN to detect and track low flying aircraft. *Proceedings of the Distributed Sensor Nets Workshop* 41-52 Copies may be available from the Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 15213

Lenat, D. B (1975) Beings: Knowledge as interacting experts. *In IJCAI-75,* 126-133

Lesser, V. R. & Corkill, D D (1978) Cooperative distributed problem solving: A new approach for structuring distributed systems Technical Report 78-7, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts.

Lesser, V R , Pavlin, J , & Reed, S (1980) Quantifying and simulating the behavior of knowledge-based interpretation systems *In Proceedings of the First National Conference on Artificial Intelligence.* Stanford, CA, 111-115.

Lesser, V. R (1980) Cooperative distributed problem solving and organizational self-design In "Reports on the MIT Distributed AI Workshop", SIGART *Newsletter* (73):46 Also in the same issue: "Models of problem-solving," page 51.

Lesser, V R. & Erman, L. D. (1980) Distributed interpretation: A model and experiment *IEEE Transactions on Computers,* C-29(12):1144-1163

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

Nilsson, N J (1980) Two heads are better than one *SIGART Newsletter* (73):43.

Pavlin, J (1983) Task allocation in distributed problem solving systems. *In Proceedings of the Third National Conference on Artificial Intelligence* Washington, D C., in press

Paxton, W H (1978) The executive system In D E Walker, (Ed) *Understanding spoken language* Elsevier, North Holland

Smith, R G. (1978) A Framework for Problem Solving in a Distributed Processing Environment PhD thesis, Stanford University Available as Technical Report STAN-CS-78-800, Computer Science Department, Stanford University, Stanford, California

Smith, R. G & Davis R (1981) Frameworks for cooperation in distributed problem solving IEEE *Transactions on Systems, Man, and Cybernetics* SMC-11(1):61-70

Zisman, M D (1978) Use of production systems for modeling asynchronous, concurrent processes In D A Waterman & Frederick Hayes-Roth, (eds) *Pattern-Directed Inference Systems.* 53-68 Academic Press