An Update on the Distributed Vehicle Monitoring Testbed

Victor R. Lesser, Daniel D. Corkill,
and Edmund H. Durfee

COINS Technical Report 87-111

October 1987

AN UPDATE ON THE DISTRIBUTED VEHICLE MONITORING TESTBED: A Tool For Investigating Distributed Problem Solving Networks

Victor R. Lesser, Daniel D. Corkill, and Edmund H. Durfee
Department of Computer and Information Science
University of Massachusetts
Amherst, Massachusetts, 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 distributed knowledge-based problem solving systems operating on an abstracted version of a vehicle monitoring task.

There are two important aspects to the testbed. First, 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. Second, 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, of processing (simulated) input data with varying amounts of distribution and ambiguity, and of testing different mechanisms for controlling local and network problem solving (including organizational structuring, local planning, and partial global planning). Node configurations and communication channel characteristics can also be independently varied in the simulated network.

This paper updates and extends our previous description of the testbed [36].

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, Joe Hernandez, David Hildum, Eva Hudlicka, Larry Lefkowitz, Raam Mukunda, Jasmina Pavlin, Ed Pattison, Scott Reed, Joe Walters, and David Westbrook have 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-8306327, by the National Science Foundation under Support and Maintenance Grant DCR-8318776, by the National Science Foundation under CER Grant DCR-8500332, and by the Defense Advanced Research Projects Agency (DOD), monitored by the Office of Naval Research under Contract N00014-79-C-0439. Edmund Durfee also received support from an IBM Graduate Fellowship.

Contents

1. Introduction

Distributed Problem Solving (also called Distributed AI) combines the research interests of the fields of AI and Distributed Processing [3,12,11,23,26,47]. 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.

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 (see Figure 1). 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 [32,46]. Each processing node can communicate with other nearby nodes over a packet radio communication network [31]. 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 a significant amount of 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

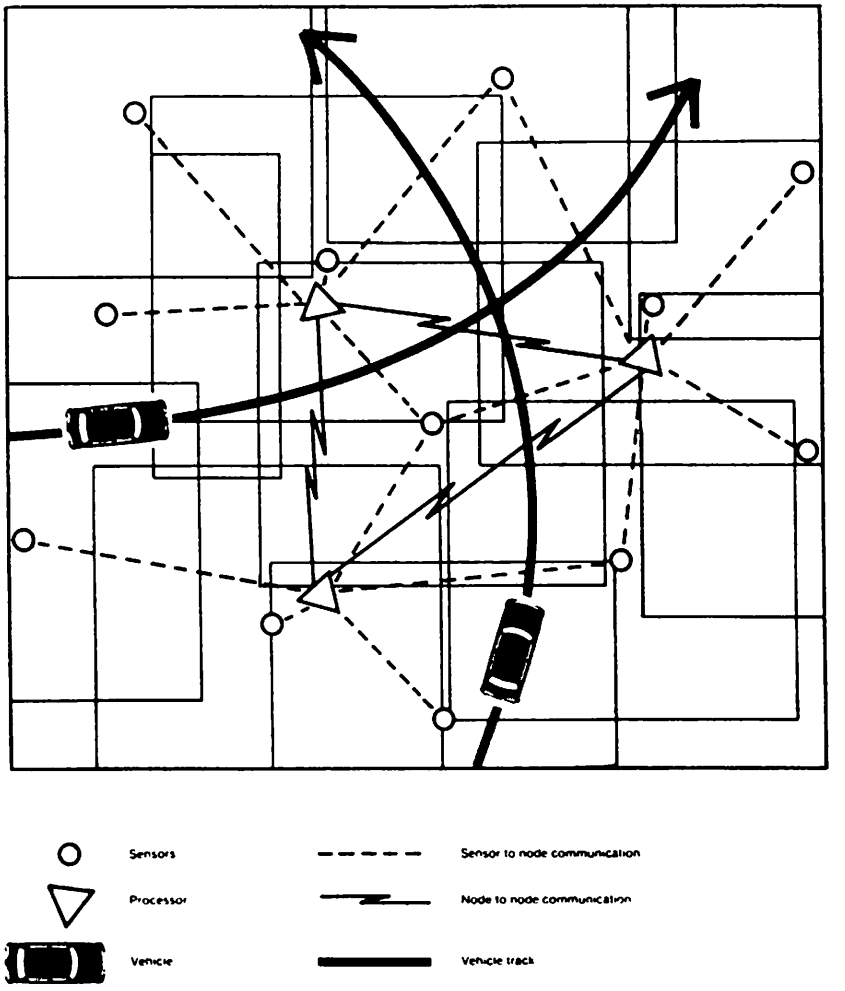


Figure 1: Tracking Vehicle Movements in a Distributed Sensor Network.

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) [37]. 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 the amount of communication required to exchange these high-level, tentative results is much less than the communication of raw data and processing results which 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.

2. A Pilot Experiment in Distributed Interpretation

A set of pilot experiments were performed to investigate the suitability of the FA/C approach using a network of complete Hearsay-II interpretation systems [38]. 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 basic problem solving. Further, the processing can be partitioned or replicated naturally among network nodes because it is already decomposed into independent and self-directed modules, called **knowledge sources** (KSs), 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. [22]).

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 modeled 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 [10,9,17]. 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 non-coherent behavior [34]. 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 both nodes with noisy and accurate 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.

3. The Need For A Testbed

Although these pilot experiments provided initial 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 were 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 accommodated?

- **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 [45]? Similarly, should information be transmitted on a voluntary basis or only when requested or some mixture of these policies?

The candidate task characteristics that we wanted 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 the 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 actual 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 that had led to a tight coupling among KSs 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 KSs without significant modifications to the KS interaction patterns. Fourth, the sheer size and complexity of KS 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.

4. 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.

4.1 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 a distributed problem solving system for this abstracted task that can model (through appropriate parameter settings and pluggable code modules) 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 beasts. 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 modeling route as not viable. The research in distributed problem solving is still in its infancy and formal analytic approaches are rudimentary. 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.

[1] We had, in fact, earlier embarked on the development of such an environment, based on what we called the Distributed Processing Game [35], 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 coordination effects were secondary to developing appropriate gaming strategies.

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.

4.2 Why Distributed Vehicle Monitoring?

Distributed vehicle monitoring has four characteristics making 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 has termed systems with this characteristic **commutative** [39]. 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 very liberal in making tentative initial vehicle identifications, since generation of incorrect information never precludes the 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 limited 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 adequately 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 (x,y) 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 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 [24,25], 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 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. The goal of these simplifications was to reduce the processing complexity and knowledge engineering effort required in the testbed without significantly changing the basic character of the distributed interpretation task. The major task simplifications in the Distributed Vehicle Monitoring Testbed include:

- The monitoring area is represented 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.

A second design decision was to fully instrument the testbed. 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 KSs available at each node, permitting the study of different problem solving decompositions;

- the accuracy of individual KSs, permitting the study of how different control and communication policies perform with different levels of system expertise;[2]

- 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;

- the problem solving and communication responsibilities of each node, permitting exploration of different problem solving strategies;

[2]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. The effects of varying knowledge quality have not yet been extensively explored.

- the authority relationships among nodes, permitting experimentation with different organizational relationships among nodes;
- the sophistication of local and network control mechanisms, permitting the evaluation of the benefits of making more intelligent and informed control decisions, and also the costs in communication and computation overhead.

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

The distributed vehicle monitoring testbed embodies these design criteria. In the next section, the simplified version of the distributed vehicle monitoring task used in the testbed is presented. In subsequent sections, the basic architecture of the testbed will be described.

5. Testbed Node Architecture

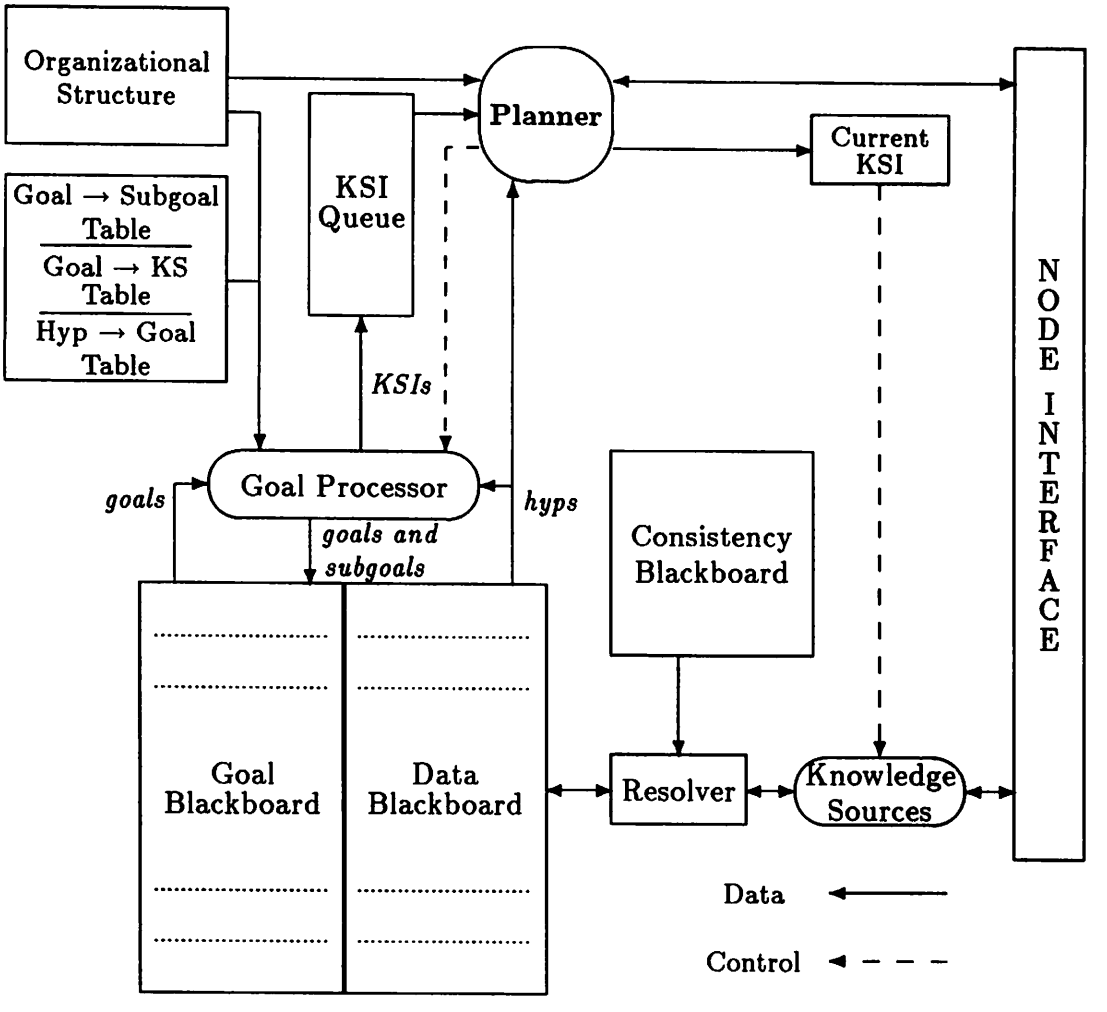
The Distributed Vehicle Monitoring Testbed simulates a network of Hearsay-II nodes working on the vehicle monitoring task. Each node has a blackboard-based, Hearsay-II architecture [22] with KSs and levels of abstraction appropriate for the task of vehicle monitoring. A node is capable of solving the entire vehicle monitoring problem by itself if it is given all of the sensory data and a full complement of KSs. This permits any subset of the KSs 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 architecture of a node is represented graphically in Figure 2, and its various components are described in turn.

5.1 The Data Blackboard and Knowledge Sources

Hypothesized vehicle movements are represented on the **data blackboard**, and KSs perform the basic problem solving tasks of extending and refining these **hypotheses**. An important consideration in developing the set of KSs 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 KSs) without modifying the KS modules and local control structures.

A hypothesis is characterized by one or more **time-locations** (the vehicle's locations at various time frames), by an **event-class** (the frequency class or vehicle identity information), and by a **belief** (the confidence in the accuracy of the hypothesis). The data blackboard organizes the hypotheses based on four levels of abstraction: **signal** (for low-level analyses of the sensory data), **group** (for collections of harmonically related signals), **vehicle** (for collections of groups that correspond to given vehicle types), and **pattern** (for collections of spatially related vehicle types such as vehicles moving in a formation). Each of these levels is split into a level for location hypotheses (which have one time-location) and a level for track hypotheses (which have a sequence of time-locations). In total, nodes



A KS forms hypotheses on the data blackboard, which in turn trigger the formation of goals on the goal blackboard. The goal processor forms KSIs to satisfy the goals, which are placed on the KSI queue. The planner then chooses the current KSI, triggers the corresponding KS, and the cycle repeats. In its simplest form, the planner simply invokes the KSI at the top of the queue, but with more sophisticated local control it forms plans, exchanges plans with other nodes, considers organizational information, and controls goal processing in order to make more coordinated decisions about which KSI to pursue next.

Figure 2: The Node Problem Solving Architecture.

have eight blackboard levels with appropriate KSs for combining hypotheses on one level to generate more encompassing hypotheses on the same or on a higher level (Figure 3).

The relationships among the hypotheses at each level is supplied to the testbed as part of a testbed **grammar** (see Figure 4 for an example). When a node generates a hypothesis at a higher level, the belief it assigns the hypothesis depends on how many of the possible *supporting* hypotheses at lower levels it actually finds and on their beliefs. Changing the grammar automatically varies behavior throughout the testbed; for example, increasing the size and connectivity of the grammar makes the interpretation task more difficult. Another modifiable parameter in the testbed limits allowable vehicle movements, and is specified as two values: the maximum velocity of a vehicle (and implicitly, events at all levels) and the maximum acceleration of a vehicle. A node uses these values when creating and extending track hypotheses. For example, a node will have higher belief in a track hypothesis indicating a vehicle moving at moderate speeds and accelerations (relative to the maxima) than it will in a track hypothesis indicating a vehicle making extreme movements. Reducing the constraints on vehicle movement makes tracking more difficult because a node must form tracks that satisfy tighter constraints.

The 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.

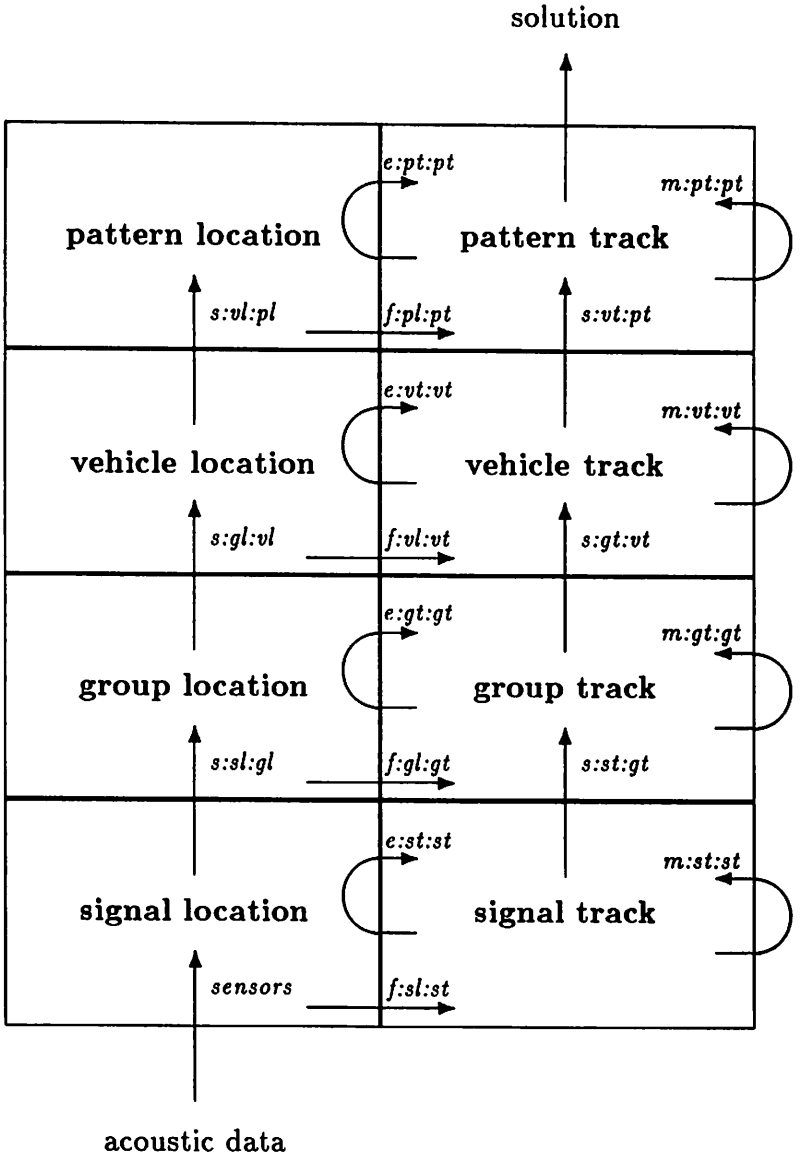
5.2 Communication Knowledge Sources

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

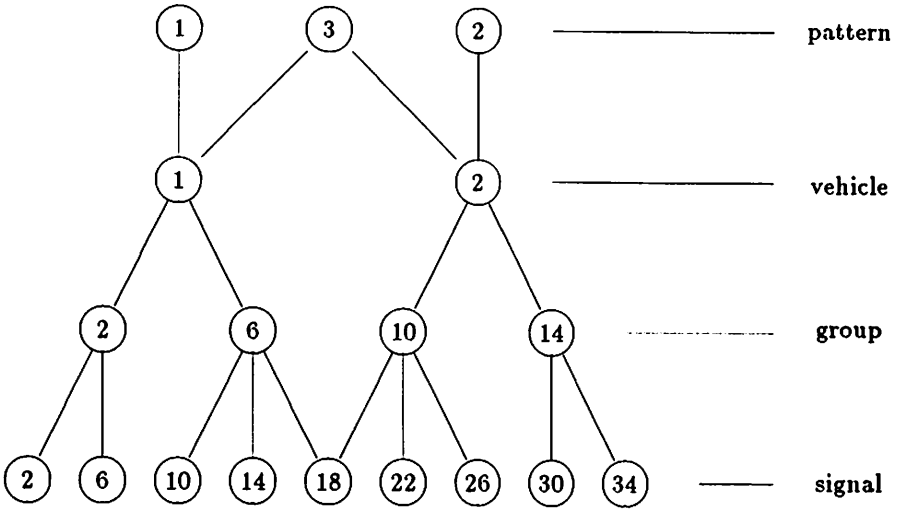
Hypothesis Receive — Places hypotheses received from other nodes onto the node's blackboard. Incoming hypotheses are filtered according to the characteristics of the received hypothesis to ensure that the node is truly interested in the information. Hypothesis Receive uses a simple model of the credibility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

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 meta-level control information — whether or not the node is to attempt to achieve the goal locally.



KSs combine hypotheses to form more encompassing hypotheses on the same or higher levels. Synthesis (s:) KSs generate higher level hypotheses out of compatible lower level hypotheses. Formation (f:) KSs form a track hypothesis from two combinable location hypotheses. Extension (e:) KSs combine a track hypothesis with a compatible location hypothesis to extend the track. Merge (m:) KSs combine two shorter track hypotheses that are compatible into a single longer track. The sensors KS creates signal location hypotheses out of sensed data. Communication KSs are not shown.

Figure 3: Levels of Abstraction and Knowledge Sources.



A simple, three pattern grammar specifying the relationship among the event classes at the various levels in the data hierarchy. There are two different vehicle types (1 and 2) each formed from two different signal groups. Each vehicle can form a single vehicle pattern (1 and 2 respectively) or both together (at a specific distance not illustrated here) can form a two-vehicle pattern (3). Signal class 18 is included among the supports of both vehicle types and adds confusion in this grammar.

Figure 4: An Example Signal Grammar

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

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 truly 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 KSs is easily accomplished by simulating a more sophisticated KS by:

- modifying its power (see Section 5.5);
- modifying the code of the KS to use more sophisticated knowledge in its choices (this can done by adding code that filters the input or output of the KS);
- completely replacing a KS with an alternative module.

5.3 Goal Processing

A **knowledge source instantiation** (KSI) represents the potential application of a particular KS to specific hypotheses. Each node maintains a queue of pending KSIs and, at any given time, must rank the KSIs to decide which one to invoke next. To improve these decisions, we have extended the Hearsay-II architecture (Figure 2) so that nodes can reason more fully about the intentions or **goals** of the KSIs [8]. Rated and stored on a separate goal blackboard, goals are explicit representations of the node's intentions to abstract and extend hypotheses. 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.[3]

The goal processor generates goals for each new hypothesis based on the table that maps hypotheses to goals. Having formed a goal to further develop new data, the goal processor can relate the new goal to existing goals in order to alter the relative importance of pursuing various KSIs [28]. For example, if one KSI might simultaneously achieve several goals, then the relative importance of that KSI could increase.

The goal processor might also form subgoals of goals at high levels of the goal blackboard. These subgoals reflect the importance of lower-level data in achieving the original goal, and if they are 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.[4]

The knowledge needed to perform subgoaling is based on the behavior of the testbed KSs and is parameterized by the grammar. Because subgoaling requires some effort, its use needs to be controlled. In the testbed as originally implemented, subgoaling was 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 [7]. With the improvements to local planning (see Section 6.3), subgoaling can be controlled so that it is only applied when ambiguity in ways of pursuing a plan cannot otherwise be resolved [14].

5.4 Forming and Rating Knowledge Source Instantiations

The goal processor forms KSIs to achieve goals by first using its table that maps goals to KSs and then by executing KS precondition procedures that estimate the potential benefits and costs of performing each of those KSs. In "real" systems, this estimation is based in part on information provided by each KS to the scheduler about the output the KS is likely to produce given particular input hypotheses (the KS **response frame** [27]). This estimation is usually fast and approximate — it is made without a detailed analysis

[3] 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 quickly relate goals and hypotheses to one another.

[4] There are no prediction KSs 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.

of the KS's input data. Increasing uncertainty in this estimation makes it less likely that the planner and scheduler will appropriately decide what KS actions to perform.

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

The KSI 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 KSI'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 KSI's output-set attribute and the maximum value (multiplied by the KS efficiency estimate) is used as the base rating for the KSI.

Since the testbed precondition procedures precompute the actual output hypotheses of the KSI, 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 KS precondition procedures that are imperfect in their estimation of output hypotheses's beliefs and of goal satisfaction.

The KSs' 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 KSI'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 KSI or should it be given domain-dependent knowledge of its own to determine consistencies between KSIs? To experiment with this issue, an oracle weighting in the data-directed component can be used to introduce the consistency of each output hypothesis into the rating calculation. As with the KSIs themselves, this consistency information is used to simulate the effects of developing additional knowledge which can better detect the consistencies among hypotheses.

The consistency of a hypothesis 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. Finally, the consistency blackboard provides an oracle that tells the system when

all solutions have been found so that problem solving should terminate.[5]

5.5 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 each KS to detect local consistencies and inconsistencies among its input hypotheses and to generate appropriate output hypotheses is called the **power** of the KS. KS power ranges from a perfect KS able to create output hypotheses with beliefs that reflect even the most subtle consistencies among its input hypotheses down to a KS which creates syntactically legitimate output hypotheses without regard to local consistency and with beliefs generated at random. Note that a perfect KS is not the same as an omniscient one. A perfect KS can still generate an incorrect output hypothesis if supplied with incorrect, but completely consistent, input hypotheses.

The testbed can modify the power of a KS to be anywhere along this range. This is achieved by separating each KS into two stages: a candidate generator and a resolver. The **candidate generator** stage produces plausible hypotheses for the output of the KS and assigns each hypothesis a tentative belief value. The candidate generator stage for each KS in the testbed incorporates relatively simple domain knowledge. There are two types of knowledge used in the candidate generator to form possible output hypotheses based on patterns of input hypotheses. These patterns are derived from the particular testbed grammar and knowledge of sensor error characteristics. The second 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 KS 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 KS.

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 KS). Hypothesis belief alteration can also be used to degrade the performance of the candidate generator (and thereby reduce the apparent power of the KS).[6]

---

[5] We have recently begun to explore alternative mechanisms for making termination decisions that compare the quality and scope of hypotheses that have been formed already with the attributes of hypotheses that could potentially be formed in the future [15].

[6] The work by Paxton on the SRI speech understanding system [44] comes closest to our approach. He used ground consistency information to statistically simulate the output of the low level acoustical processor in the SRI speech system. Our approach differs from his in that it dynamically relates characteristics of the inputs of a KS to the characteristics of its outputs, while Paxton's model does not. The output of his model depends on precomputed behavioral statistics which are independent of the belief- and consistency-values of its inputs. Because of this, we are able to simulate any or all KSs in our system, while Paxton's model is

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

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

Given these limitations, we do not expect a simulated KS to behave exactly the same as a real KS. We feel, however, that the essential behavior of each KS has been captured so that system phenomena are adequately simulated, although as yet we have not extensively explored the effects of varying KS powers.

5.6 Other Features

The node architecture also includes other components that predominantly influence local control decisions to promote coordinated interactions among agents. The organizational structure specifies general roles and responsibilities of nodes in a declarative and infrequently changing data structure. This information affects local control by influencing the ratings of goals and KSIs (see Section 6.1). The planner reasons about the alternative KSIs on the KSI queue to determine which KSI should be invoked next. In its most primitive state, the planner simply invokes the most highly-rated KSI, but our more recent research has concentrated on developing much more sophisticated planning (see Sections 6.2 and 6.3).

6. 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 within an existing control framework. The second aspect is the ability to augment the basic testbed node architecture with additional control components to explore the benefits and costs of more sophisticated control. Both types of experimentation are possible with the testbed.

6.1 The Organizational Structure

An **organizational structure** specifies a set of long-term responsibilities and interaction patterns for the nodes. This information guides the local control decisions of each

---

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 KSs 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.

node and increases the likelihood that the nodes will cooperate coherently by providing a global strategy for network problem solving. 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 node's goal processor and communication KSs.

The organizational structure is implemented in the DVMT as a set of data structures called **interest areas**. As implied by its name, an interest area specifies the node's interests (represented as a set of parameters) in developing and communicating about hypotheses in particular areas of the data blackboard. 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 KSIs and the behavior of the node's planner and scheduler.

Hypothesis Transmission Interest Areas — Influence the behavior of HYP-SEND KSs in the node.

Hypothesis Reception Interest Areas — Influence the behavior of HYP-RECEIVE KSs in the node.

Goal Transmission Interest Areas — Influence the behavior of GOAL-SEND KSs in the node.

Goal Help Transmission Interest Areas — Influence the behavior of GOAL-HELP KSs in the node.

Goal Reception Interest Areas — Influence the behavior of GOAL-RECEIVE KSs in the node.

Each interest area is a list of regions of the data or goal blackboard. The goal processor uses the interest area to modify the ratings of goals; goals to generate hypotheses in desirable areas of the blackboard would have their ratings increased. Goals to transmit or receive information similarly have their ratings modified based on the interest areas of both the sending and receiving nodes. Since the goal rating is a factor in ranking KSIs, the interest areas can influence node activity; but because there are other factors in ranking KSIs (such as the expected beliefs of the output hypotheses), a node still preserves a certain level of flexibility in its local control decisions. The organizational structure thus provides guidance without dictating local decisions, and can be used to control the amount of overlap and problem solving redundancy among nodes, the problem solving roles of the nodes (such as "integrator", "specialist", and "middle manager"), the authority relations between nodes, and the potential problem solving paths in the network [7].

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 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. Since a node may modify the ratings of received goals or goals stimulated by received hypotheses, it has flexibility in how it responds to received information. If the ratings of these goals are increased, the node is said to be *externally-biased* and is essentially subservient to others; if the ratings of these goals are decreased, the node is *locally-biased* and prefers its own activities over the suggestions of others; and if the ratings of these goals are not modified at all, the node is *unbiased*.

By providing even a simple network with some organization, better use of communication and computation resources can be achieved. An organizational structure that appropriately balances the amount of interest that nodes have in various problem solving activities and the degree to which they may be externally biased (based on the relative completeness of their views) can result in consistently acceptable network problem solving behavior over the long term. However, an organization that is specialized for one short-term situation may be inappropriate for another [17]. Because network reorganization may be costly and time consuming, and since specific problem characteristics cannot be predicted beforehand, an organizational structure should thus be chosen which can achieve acceptable and consistent performance in the long-term rather than being very good in a limited range of situations and very bad in others.

6.2 Local Planning and Communication: A Preliminary Approach

To dynamically tailor an "all-purpose" organizational structure for specific network needs, we developed planning mechanisms to explicitly represent and reason about sequences of related actions. In our initial implementation, the planner formed a very simple abstraction of the node's problem solving state and plans relatively short sequences of KSs to achieve the goals it recognizes [19,17]. A plan, as a representation of some sequence of related (and sequential) activities, indicates the specific role that a node will be playing in the organization over a certain time interval.

A node can use information about its plans to improve its coordination knowledge by dynamically refining its interest areas. For example, consider a node that is responsible both for low-level synthesis activity in a small region and for integrating partial tracks over a much larger region. At a given time, the node might develop and begin to execute a plan to perform synthesis. Given this plan, the node could recognize that it is not integrating partial tracks, and therefore knows that any locally generated or received tracks should be forwarded to other nodes for integration. On the other hand, if the node were executing a plan to integrate partial tracks, it might decide that it should not transmit the larger tracks that it forms because this might occupy bandwidth that could better be used by other nodes to send additional partial tracks to it for integration.

When nodes exchange information about their plans, they can build a more complete and current model of each other's current roles within the organizational structure. This *meta-level* communication improves coordination further by allowing nodes both to make

more relevant local control decisions (for example, to dynamically avoid redundant effort) and to make more informed communication decisions (for example, to communicate more locally complete hypotheses) [17,18]. However, once again we found limitations in this approach to improving coordination, stemming predominantly from the inability of plans to represent more than only near-future activities, from the lack of explicit representations for concurrent, cooperative activities in the network, and from the limited set of responses that nodes could make in cooperative situations.

6.3 Partial Global Planning: A Unified Approach

Our current approach provides mechanisms for nodes to develop much richer representations for plans by forming a more expressive abstraction of its problem solving situation [14,15,21]. A node's planner abstracts the local state to recognize possible competing and compatible solutions and roughly predicts the importance and expense of developing these solutions. With this information, the planner sketches out sequences of major plan steps that it expects will most efficiently resolve its uncertainty about which of the possible solutions to work toward. The planner only details actions for the near future because the results of these actions will influence how (and whether) a plan should be pursued. As problem solving proceeds, the planner adds new details to the plan incrementally, and monitors and repairs the plan to insure it achieves its goals whenever possible. Our experimental results indicate that the planner substantially improves local problem solving without introducing excessive overhead.

Using this more powerful local planner as a foundation, we have developed **partial global planning** as a new and versatile approach to dynamically coordinating problem solvers [16,21]. The problem solvers exchange summaries of their local plans to develop network awareness, and when they see that they have pieces of larger goals, they each form and follow **partial global plans** (PGPs) that specify cooperative actions and interactions. Since a node's plans can change and communication takes time, nodes may base decisions on incomplete, out-of-date, and inconsistent views of the network. Our partial global planning approach lets nodes plan satisfactory, though not necessarily optimal, cooperation even in dynamic domains. The extent and quality of a node's network model and how it is formed depends on the **meta-level organization**: the communication topology, capacity, delay, and reliability; the coordination responsibilities of different nodes; the credibility that a node has in coordination information from other nodes (which determines their authority relationships); and so on. The distributed problem solving network is therefore organized both in terms of problem solving responsibilities (the domain-level organization) [9,7] and in terms of coordination responsibilities (the meta-level organization).

Prescribing appropriate meta-level organizations and styles of cooperation for generic problem situations depends on the evaluation criteria considered, and remains an important avenue for future research. However, the partial global planning framework is an important step toward understanding organizational issues because it lets us explore different styles of cooperation using one set of mechanisms. We can have nodes pass around node-plans and plan for themselves, or send node-plans to a node that plans for everyone, in which

case we can even force them to sit idle waiting for these PGPs by having them give no credibility to their locally developed PGPs. Or we could let nodes exchange PGPs so that they negotiate on a consistent global view by adopting the most highly-rated version of a PGP. Nodes can also use node-plans and PGPs to form contracts. A node with a particular task can generate a PGP activity-map that has the task being performed in the future at some remote nodes.[8] The PGP is sent to these nodes which locally develop node-plans representing these potential future tasks. These node-plans are modified and rerated to reflect the node's view of its own activities; for example, if it may form an activity-map for the node-plan indicating that the received tasks could not be performed until much later in the future. The nodes return these node-plans to the original node, which adopts the best one and follows it by sending task information (data) to the chosen node. In essence, one node requests bids for a cooperative plan and each of the others bids on how it expects it could cooperate. The node that likes the interaction most (or dislikes it least) is awarded the task. Moreover, because the returned node-plan conveys information about how the task fits into the bidding node's more global view, the originating node could use the node-plans it gets back to recognize that the task is unimportant and should not be awarded at all.

We have experimentally evaluated the partial global planning approach to examine its benefits in improving coordination, its costs in additional overhead, and its versatility in allowing cooperation in different styles to achieve a variety of goals. Our research shows that partial global planning is a flexible and practical approach for coordinating problem solving networks and distributed computing systems in general.

7. 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.) The FRONTEND KS is a special, simulation-level KS used to initialize the testbed network. It is always the first KS 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 KS 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 KSs to resolve errors can be altered. This ability to modify the grammar combined with the ability to vary the

[8] The node's planner thus not only reorders activities but also finds possible reassignments that move computational load from a bottleneck node to nodes that are not participating in highly-rated PGPs. Making such reassignments is a complex problem for which we as yet have only primitive mechanisms.

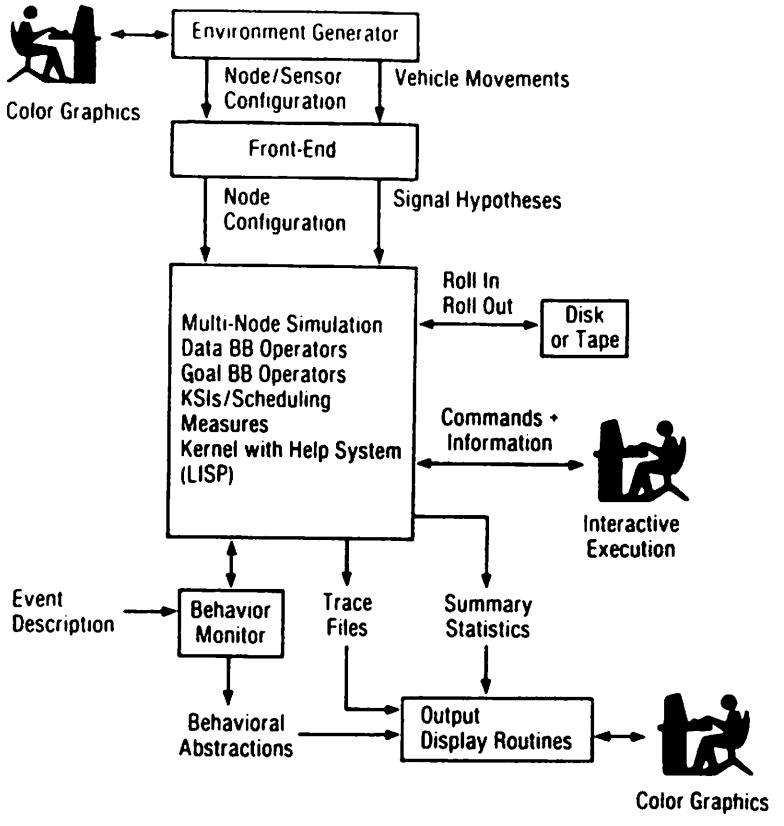


Figure 5: Testbed Kernel and Related Subsystems.

local resolving power of KS 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 of each true and/or consistent-false signal according to the sensor 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 experimenter's procedures. In the testbed, there is a hook at the beginning of the simulation, another following the FRONTEND (when all sensory data and the consistency blackboard have been determined), one prior to each KS 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 KS 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 KSIs 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 KSs' 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 asynchronous activity of the simulated nodes. In a related research project, an event monitoring facility has been developed and has successfully been applied to our testbed, permitting the 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 [1,2].

Another facility which has been developed and continues to evolve 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.

To increase the rate at which the testbed can simulate network activities, we implemented a distributed version of the testbed to run on a network of VAX 11/750s [13,20]. When beginning a simulation, the user can specify which machines to use and how the simulated nodes should be distributed among those machines. The testbed then runs in parallel on the various machines, allowing a certain degree of asynchrony but ensuring deterministic results. Therefore, how (and whether) the simulation was distributed does not affect the simulated results but only influences the time needed to perform the simulation.

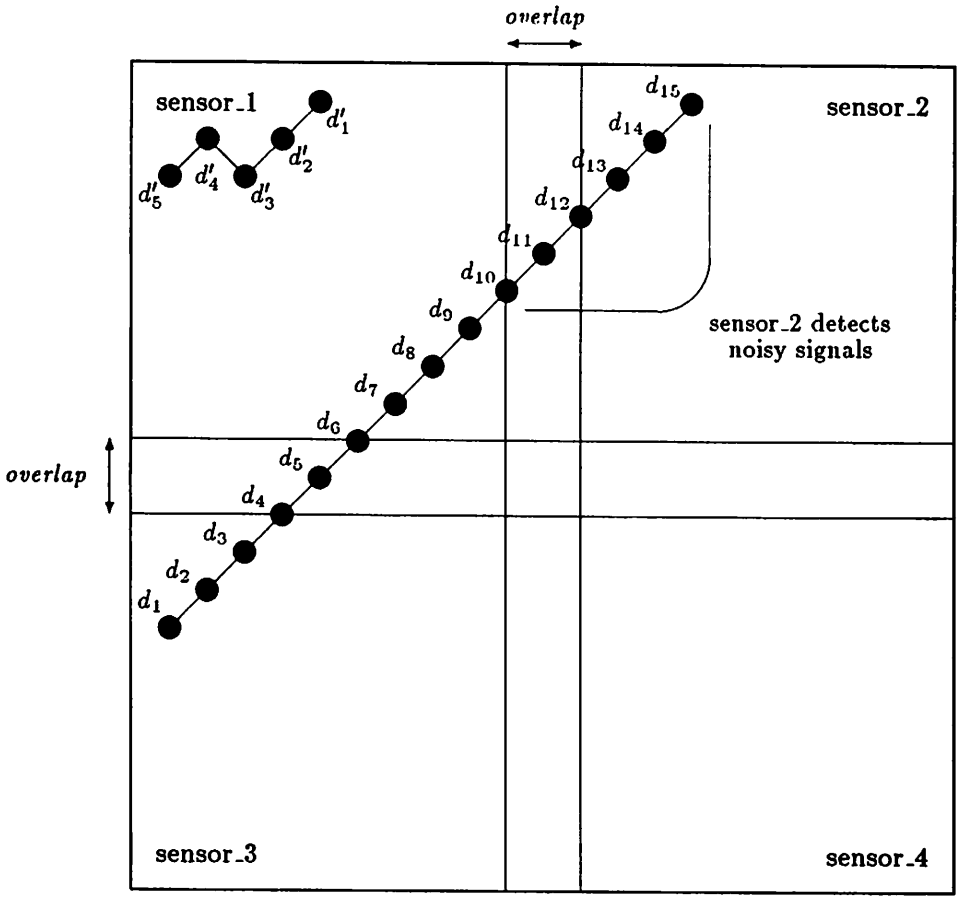
8. Testbed Status, Experiments, and Future Directions

The testbed, which has been operational since January of 1982, has undergone continual evolution, refinement, and extension over the years. Most recently, a recoding effort has converted the majority of the system from CLISP [4] to Common Lisp [48]. In the course of the recoding, our previous experience in developing blackboard-based systems was consolidated into the Generic Blackboard Development System (GBB) [6,5], which in turn served as a foundation upon which the other testbed mechanisms could be reimplemented.

Throughout its lifetime, the testbed has been and continues to be a rich environment in which to explore issues and techniques in cooperative problem solving. Because we have abstracted the vehicle monitoring task, we are not limited to only studying a small range of scenarios: instead, we can experiment with situations that would be unlikely to occur in the vehicle monitoring task but which do involve important issues in cooperation. As a consequence, we purposely construct and evaluate experimental situations that are contrived to elucidate our knowledge of coordination and to illustrate the benefits, costs, and limitations of our approaches to cooperation.

For example, consider a problem situation (Figure 6) with four nodes, each connected to a different sensor (node i to sensor i). When all the data is present, then from a local perspective: node 1 plans to develop two tracks (d'_1–d'_5 and d_4–d_{12}); 2 plans for one track (d_{10}–d_{15}), but because its sensor is faulty it will need much time to filter out noisy data; 3 plans for one track (d_1–d_6); and 4 has no local plans. From a global perspective: node 1 should first work on track d_4–d_{12} since it has more global significance (joining the tracks of 2 and 3), and also should quickly generate and send a predictive result (like the short track d_8–d_9 which borders on node 2's view) to help 2 disambiguate its noisy data; 2 should expect this predictive information; 3 should take responsibility for the data that both it and 1 sense (d_4–d_6) since 1 has more data to process; and 4 should take on some tasks, either by getting data from other nodes, or by acting as network coordinator, or both. The nodes therefore need to negotiate about task assignments, exchange partial results to converge on global solutions, and plan interactions that will help each other perform their tasks better.

Given this type of situation, we use the testbed to experiment with different ways of controlling local and network activity. It is through such experiments that we have gained an appreciation for how difficult and knowledge-intensive cooperation actually is, and how



The four overlapping sensors detect signal data at discrete sensed times (the dots with associated times). Sensor 2 is faulty and not only generates signal data at the correct frequencies but also detects noisy signals at spurious frequencies.

Figure 6: Four-Sensor Configuration with Sensed Data.

it requires powerful and versatile mechanisms. Our experiments and observations have provided crucial feedback as we have developed novel approaches to coordination that are of use in networks for vehicle monitoring and, more generally, for a wide range of cooperative tasks.

The testbed has allowed us to explore issues such as:

- should communication be **voluntary** (a node transmits hypotheses at its pleasure), **requested** (a node transmits hypotheses only when that information is requested by another node), or through **mixed initiative**, combining voluntary and requested hypotheses (a node volunteers only its highest rated hypotheses and awaits requests before transmitting any other hypotheses);

- should a node be locally-biased or externally-biased in its activities (or a combination of both);

- should hypotheses, goals, or both hypotheses and goals be used for internode coordination;
- should the problem solving organization be lateral (allow each node to have equal authority and problem solving capabilities), hierarchical (where some nodes coordinate and control others and integrate their results), or some combination of lateral and hierarchical;
- should nodes transmit all potentially relevant partial results or should they reduce communication by waiting until they have formed their most encompassing partial results;
- what aspects of node and network activities should be monitored to detect and diagnose incorrect problem solving behavior;
- should nodes change strategies for local problem solving and/or network coordination as the overlap between their activities changes (making duplication of effort more or less likely);
- should nodes change strategies for local problem solving and/or network coordination as the rate at which data arrives and must be processed changes;
- should nodes change strategies for local problem solving and/or network coordination when deadlines for generating solutions are imposed;
- should nodes change strategies for local problem solving and/or network coordination when node reliability (probability of failure) is changed;
- how do various mechanisms for controlling local and network problem solving scale-up as network size (number of nodes) increases;
- when do the overhead costs of sophisticated control outweigh its benefits.

The testbed has therefore been a fruitful tool for exploring a range of important issues in distributed problem solving, and many of its capabilities have yet to be exploited. In addition, it has provided a context in which to explore other important issues in distributed problem solving such as developing approaches for abstracting and understanding the behavior of such systems [42,43,41], and for diagnosing inappropriate system activities [29,30].

One of the most important goals we are working toward is exploring larger node configurations: to date, the largest networks that we have simulated and analyzed results for have consisted of 13 nodes. Several obstacles remain before we can easily explore networks of 20 to 100 nodes. One of these is simply the limitations of our computing resources: simulating very large networks requires more computation time and more memory than we currently have available. The expected addition of two Sequent multiprocessors with a total of 24 individual processors to our computing resources should allow us to simulate much larger networks. However, even when we simulate larger networks we will still be limited

by our ability to analyze results. Automated tools for analyzing system performance based on a simple, semi-formal model of how blackboard systems construct solutions and resolve uncertainty [33] have had only limited success because they cannot model the diverse set of complex interactions exhibited in the testbed. Our current interactive tools, on the other hand, help users to identify and understand complex aspects of problem solving in small networks where users themselves keep track of the simultaneous activities of nodes, but these tools need substantial improvement if they are to help users make sense of data from 20 or more nodes without being overwhelmed.

In setting up these 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. We are also developing tools allowing complex node topologies to be specified in a generic way, independent of any specific number of nodes [40]. We have come to firmly believe that no matter how flexible and general a research tool is, if it is not convenient to use and the empirical results easy to understand, only a small subset of its capabilities will be exploited.

Our other principal goal has been and continues to be developing, evaluating, and extending mechanisms for control in distributed problem solving networks. Over the past years we have made significant progress, to where nodes are now able to cooperate in ways that were previously beyond reach. We expect that our mechanisms will continue to evolve, allowing nodes to communicate a wider variety of information, to reason about each other more fully, and to develop views of the network so that they can eventually reorganize themselves in suitable ways as situations change.

9. Conclusion

In this article we have briefly 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.

Next, 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;
- the authority relationships among nodes;
- the sophistication of control mechanisms for both local and network problem solving;

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.

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

[1] Peter C. Bates and Jack C. Wileden.
Event Definition Language: An aid to monitoring and debugging of complex software systems.
In *Proceedings of the Fifteenth Hawaii International Conference on System Science*, pages 86–93, January 1982.

[2] Peter Charles Bates.
Debugging Programs in a Distributed System Environment.
PhD thesis, University of Massachusetts, February 1986.
(Also published as Technical Report 86-05, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, February 1986.).

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

[4] Daniel D. Corkill.
CLisp Reference Manual.
Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts, 1980.
(Periodically revised through 1984).

[5] Daniel D. Corkill, Kevin Q. Gallagher, and Philip M. Johnson.
Achieving flexibility, efficiency, and generality in blackboard architectures.
In *Proceedings of the National Conference on Artificial Intelligence*, pages 18–23, Seattle, Washington, July 1987.
(Also published as Technical Report 87-37, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, April 1987.).

[6] Daniel D. Corkill, Kevin Q. Gallagher, and Kelly E. Murray.
GBB: A generic blackboard development system.
In *Proceedings of the National Conference on Artificial Intelligence*, pages 1008–1014, Philadelphia, Pennsylvania, August 1986.
(Also to appear in *Blackboard Systems*, Robert S. Engelmore and Anthony Morgan, editors, Addison-Wesley, in press, 1987. Also published as Technical Report 86-67, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, April 1986.).

[7] Daniel D. Corkill and Victor R. Lesser.
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*, pages 748–756, Karlsruhe, Federal Republic of Germany, August 1983.
(Also appeared in *Computer Architectures for Artificial Intelligence Applications*, Benjamin W. Wah and G.-J. Li, editors, IEEE Computer Society Press, pages 507–515, 1986).

[8] Daniel D. Corkill, Victor R. Lesser, and Eva Hudlicka.
Unifying data-directed and goal-directed control: An example and experiments.
In *Proceedings of the Second National Conference on Artificial Intelligence*, pages 143–147, August 1982.

[9] Daniel David Corkill.
A Framework for Organizational Self-Design in Distributed Problem Solving Networks.
PhD thesis, University of Massachusetts, February 1983.
(Also published as Technical Report 82-33, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, December 1982.).

[10] Randall Davis and Reid G. Smith.
Negotiation as a metaphor for distributed problem solving.
Artificial Intelligence, :63–109, 1983.

[11] Randy Davis.
Report on the second workshop on Distributed AI.
SIGART Newsletter, 80:13–23, April 1982.

[12] Randy Davis.
Report on the workshop on Distributed AI.
SIGART Newsletter, (73):42–52, October 1980.

[13] Edmund H. Durfee, Daniel D. Corkill, and Victor R. Lesser.
Distributing a distributed problem solving network simulator.
In *Proceedings of the Fifth Real-Time Systems Symposium*, pages 237–246, December 1984.

[14] Edmund H. Durfee and Victor R. Lesser.
Incremental planning to control a blackboard-based problem solver.
In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 58–64, August 1986.

[15] Edmund H. Durfee and Victor R. Lesser.
Incremental planning to control a time-constrained, blackboard-based problem solver.
IEEE Transactions on Aerospace and Electronics Systems, In press.
(Also published as Technical Report 87-07, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, February 1987.).

[16] Edmund H. Durfee and Victor R. Lesser.

Using partial global plans to coordinate distributed problem solvers.
In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 875–883, August 1987.

[17] Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill.
Coherent cooperation among communicating problem solvers.
IEEE Transactions on Computers, 1987.
(Accepted for publication. Also published as Technical Report 85-15, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, April 1985).

[18] Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill.
Cooperation through communication in a distributed problem solving network.
In Michael N. Huhns, editor, *Distributed Artificial Intelligence*, chapter 2, pages 29–58, Pitman, 1987.
(Also to appear as Chapter 7 in Scott P. Robertson, Wayne Zachary, and John Black, editors, *Cognition, Computing, and Cooperation: Collected works on cooperation in complex systems*, in press).

[19] Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill.
Increasing coherence in a distributed problem solving network.
In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 1025–1030, Los Angeles, California, August 1985.
(Also published as Technical Report 85-14, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, April 1985.).

[20] Edmund Howell Durfee.
A Parallel Simulation of a Distributed Problem Solving Network.
Master's thesis, University of Massachusetts, Amherst, Massachusetts 01003, September 1983.

[21] Edmund Howell Durfee.
A Unified Approach to Dynamic Coordination: Planning Actions and Interactions in a Distributed Problem Solving Network.
PhD thesis, University of Massachusetts, Amherst, Massachusetts 01003, September 1987.
(Also published as Technical Report 87-84, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, September 1987.).

[22] Lee D. Erman, Frederick Hayes-Roth, Victor R. Lesser, and D. Raj Reddy.
The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty.
Computing Surveys, 12(2):213–253, June 1980.

[23] Michael Fehling and Lee Erman.
Report on the third annual workshop on Distributed Artificial Intelligence.
SIGART Newsletter, (84):3–12, April 1983.

[24] Mark S. Fox.
Organization Structuring: Designing large complex software.
Technical Report 79-155, Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania, December 1979.

[25] Mark S. Fox.
An organizational view of distributed systems.
IEEE Transactions on Systems, Man, and Cybernetics, 11(1):70–80, January 1981.

[26] Les Gasser.
Report on the 1985 workshop on distributed artificial intelligence.
AI Magazine, 8(2):91–97, Summer 1987.

[27] Frederick Hayes-Roth and Victor R. Lesser.
Focus of attention in the Hearsay-II system.
In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 27–35, Tiblisi, Georgia, USSR, August 1977.

[28] Joseph A. Hernandez, Daniel D. Corkill, and Victor R. Lesser.
Details of Goal Processing in Blackboard Architectures.
Technical Report 87-24, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, May 1987.

[29] Eva Hudlicka.
Diagnosing Problem-Solving System Behavior.
PhD thesis, University of Massachusetts, February 1986.
(Also published as Technical Report 86-03, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, February 1986.).

[30] Eva Hudlicka and Victor Lesser.
Modeling and diagnosing problem-solving system behavior.
IEEE Trans. on Systems, Man, and Cybernetics, SMC-17(3):407–419, May/June 1987.

[31] R. E. Kahn, S. A. Gronemeyer, J. Burchfiel, and R. C. Kunzelman.
Advances in packet radio technology.
In *Proceedings of the IEEE*, pages 1468–1496, November 1978.

[32] R. Lacoss and R. Walton.
Strawman design of a DSN to detect and track low flying aircraft.
In *Proceedings of the Distributed Sensor Nets Workshop*, pages 41–52, 1978.
Published by the Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania.

[33] V. R. Lesser, S. Reed, and J. Pavlin.
Quantifying and simulating the behavior of knowledge-based interpretation systems.
In *Proceedings of the First Annual National Conference on Artificial Intelligence*, pages 111–115, Stanford, California, August 1980.

[34] Victor R. Lesser.
Cooperative distributed problem solving and organizational self-design.
SIGART Newsletter, :46, October 1980.
(Part of the "Report on the Workshop on Distributed AI").

[35] Victor R. Lesser and Daniel D. Corkill.
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 01003, May 1978.

[36] Victor R. Lesser and Daniel D. Corkill.
The Distributed Vehicle Monitoring Testbed: A tool for investigating distributed problem solving networks.
AI Magazine, 4(3):15–33, Fall 1983.
(Also to appear in *Blackboard Systems*, Robert S. Engelmore and Anthony Morgan, editors, Addison-Wesley, in press, 1987 and in *Readings from AI Magazine 1980–1985*, in press,

1987).

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

[38] Victor R. Lesser and Lee D. Erman.
Distributed interpretation: A model and experiment.
IEEE Transactions on Computers, C-29(12):1144–1163, December 1980.

[39] Nils J. Nilsson.
Two heads are better than one.
SIGART Newsletter, (73):43, October 1980.

[40] H. Edward Pattison, Daniel D. Corkill, and Victor R. Lesser.
Instantiating descriptions of organizational structures.
In Michael N. Huhns, editor, *Distributed Artificial Intelligence*, chapter 3, pages 59–96, Pitman, 1987.
(Also published as Technical Report 85-45, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, November 1985.).

[41] Jasmina Pavlin.
A Model for Prediction and Description of Knowledge Based System Behavior.
PhD thesis, University of Massachusetts, May 1985.

[42] Jasmina Pavlin.
Predicting the performance of distributed knowledge-based systems: A modeling approach.
In *Proceedings of the Third National Conference on Artificial Intelligence*, pages 314–319, August 1983.

[43] Jasmina Pavlin and Daniel D. Corkill.
Selective abstraction of AI system activity.
In *Proceedings of the Fourth National Conference on Artificial Intelligence*, pages 264–268, August 1984.

[44] W. H. Paxton.
The Executive System.
D. E. Walker, Elsevier, North-Holland, 1978.

[45] Reid G. Smith and Randall Davis.
Frameworks for cooperation in distributed problem solving.
IEEE Transactions on Systems, Man, and Cybernetics, SMC-11(1):61–70, January 1981.

[46] Reid Garfield Smith.
A Framework for Problem Solving in a Distributed Processing Environment.
PhD thesis, Stanford University, December 1978.
A revised version was published by UMI Research Press.

[47] N. S. Sridharan.
Report on the 1986 workshop on distributed artificial intelligence.
AI Magazine, 8(3):75–85, Fall 1987.

[48] Guy L. Steele Jr.
Common Lisp: The language.
Digital Press, 1984.