A High-Level Simulation Testbed for Cooperative Distributed Problem-Solving*

V. Lesser, D. Corkill, J. Pavlin,
L. Lefkowitz, E. Hudlicka, R. Brooks, and S. Reed

COINS Technical Report 81-16**

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

*This research was sponsored by the National Science Foundation under Grant MCS-8006327 and by the Defense Advanced Research Projects Agency (DOD), monitored by the Office of Naval Research under Contract NR049-041.

**This is a revised version of a report published in March, 1981.

Abstract

The high-level simulation testbed discussed in this paper provides a research tool for empirically evaluating alternative designs for cooperative distributed problem-solving systems. The testbed simulates a network of nodes, each of which is an architecturally-complete Hearsay-II-like system, extended to include goal-directed control. The nodes attempt to identify, locate, and track patterns of vehicles moving in a two-dimensional space.

Incorporated into the testbed are capabilities for varying the accuracy of individual knowledge sources. This is accomplished through the use of an oracle which can compare the developing interpretations with the interpretation that would be produced if the system had perfect knowledge. These capabilities permit the study of how different control and communication policies perform under varying distributions of uncertainty and error in the intermediate states of processing. Additionally, both vehicle and sensor characteristics are variable, permitting control of the spatial distribution of ambiguity and error in the task input data. Node configurations and communication channel characteristics can also be independently varied in this simulated system.

Table of Contents

I. Introduction*

We have been exploring a new paradigm for distributed problem solving systems in which the distributed system is able to function effectively even though processing nodes have inconsistent and incomplete views of the data bases necessary for their computations. This paradigm is appropriate for distributed applications in which the data necessary to achieve a solution cannot be partitioned in such a way that a node can complete a subtask without seeing the intermediate state of processing at other nodes.

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. In one distributed version of this task processing nodes with their associated acoustic sensors (of limited range and accuracy) are geographically distributed over the area to be monitored [Lacoss and Walton 1978; Smith, 1980]. Each processsing node can communicate with other nearby nodes over a packet radio communication network [Kahn 1978]. Because acoustic sensors characteristically produce a significant amount of error, purely localized processing of sensory data would result in "identification" of non-existent vehicles, missed detection of actual vehicles, and incorrect location and identification of actual vehicles. In this application, the amount of communication required to redistribute the raw sensory data necessary for correct localized processing would be significant.

An alternative approach for resolving these errors is for processing nodes to interact in a highly cooperative way, exchanging tentative partial results with one another. For example, each node's tentative vehicle identifications 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. In addition, consistencies between these tentative identifications serve to reinforce confidence in each node's identifications. Such cooperation is not only appropriate for vehicle identification, but also potentially useful in other stages of processing (identification of raw signals, groups of harmonically related signals, patterns of vehicles, etc.).

In order to perform this cooperative style of distributed processing, and thereby extend the range of applications to which distributed processing can be applied effectively, we have been developing a new approach to distributed system design. We call this

---

*Section I is primarily excerpted from [Lesser and Corkill 1981]. Readers familiar with the functionally accurate, cooperative paradigm and Hearsay-II can skip to Section II directly.

new approach functionally accurate, cooperative (FA/C) [Lesser and Corkill 1979, 1981; Lesser and Erman 1980].

In order to evaluate this new approach, we have designed a parameterized simulation testbed. This testbed permits the evaluation of different control/communication strategies under different distributions of problem-solving expertise in the system and a wide rage of task characteristics. The testbed simulates a model of a distributed, knowledge-based, problem-solving architecture applied to an abstracted version of a vehicle monitoring task. We anticipate the parameterized task and architecture together will provide a very powerful tool for investigating the utility and limitations of the FA/C approach to the design of distributed problem-solving systems.

We first briefly describe our distributed problem-solving paradigm and pilot experiments undertaken to ascertain the viability of an FA/C approach. After describing the issues we wish to explore using the testbed we present the simulated traffic monitoring task, followed by a detailed discussion of the simulated traffic monitoring system. Later sections describe how we have quantified system behavior and the use of these measures for simulating and evaluating the performance of various system components, overview tools that help a user define experiments and analyze their output, review the current status of the testbed implementation, and outline future research directions. There are several appendices A thru E that, respectively, present the following details of our implementation: a sample trace of a KS execution, the implementation of the system performance measures, the format of the summary statistics, the knowledge used by the KS candidate generators, and a description of the Front-End with sample input-file. Additional papers in this volume discuss Organizational Self-Design [Corkill 1980], Goal-Processing [Corkill and Lesser 1981], and Distributed Debugging [Bates, Wileden, and Lesser 1981].

I.1. A Paradigm for Distributed Systems

In the FA/C approach, the distributed system is structured so that each node can perform useful processing using incomplete input data while simultaneously exchanging the intermediate results of its processing with other nodes to construct cooperatively a complete solution. The hope is that the amount of communication required to exchange these results is much less than the communication of raw data and processing results which would be required using a conventional approach.

The FA/C style of processing can be characterized as an approach to problem-solving in the presence of uncertainty. A node may be uncertain as to what input data it is missing, the missing values of the data, and the correctness, completeness, and consistency of the results of its

processing and of the processing results received from other nodes. In order to resolve these data uncertainties, a node must be able to:

1. detect inconsistencies between its tentative partial results and those received from other nodes;

2. integrate into its local database those portions of other nodes' results which are consistent with its results;

3. use the newly integrated results to make up for its missing input data so that its tentative partial results can be revised and extended.

In FA/C distributed systems, it also may be difficult to determine which alternative tasks are globally the most beneficial to perform without extensive internode communication. This control uncertainty is due to differences between the natural distribution of control information among the nodes in the network and the distribution of where the control decisions are made. The existence of data uncertainty (discussed above) and uncertainty as to whether information transmitted by a node is correctly received further exacerbates this difficulty*.

One way to allow the distributed system to make control decisions without complete control information is to have node activity be self-directed. Each node uses its local estimate of the state of network problem-solving to control its processing (i.e., what new information to generate) and its transmissions to other nodes [Lesser and Erman 1980]. The degree of self-directed activity in an FA/C system is potentially quite large because a node is able to choose a processing direction for which all the necessary data may not be available or consistent with other nodes. For instance, if a node does not receive an appropriate partial result in a given amount of time, it has the option to continue processing, utilizing whatever data are available at that time, or to choose some other processing direction which appears to be more beneficial. This flexibility in node processing allows node interactions to be asynchronous and permits significant decoupling of node activity.

The self-directed control decisions made by each node may lead to unnecessary, redundant, or incorrect processing. The hope is that the system still produces acceptable answers (within allowable time constraints) and that the amount of additional communication resulting

---

*In some distributed communication networks, the usable capacity of the communications channel is significantly degraded if the correct reception of all messages needs to be verified. Therefore, systems that can function effectively without the acknowledgment of messages may be advantageous.

from incorrect local control decisions is less than the additional communication required to provide complete control information. This hope is not unreasonable, given that the additional data uncertainty caused by incorrect local control decisions may be resolvable by the same mechanisms used to resolve data uncertainties caused by incomplete local databases. Self-directed control has the added benefit of increased system robustness in the face of communication and node failure and increased system responsiveness to unexpected events. Based on this form of node activity, it is more appropriate to view an FA/C distributed system as being synthesized from individual local systems operating at each node as opposed to the decomposition viewpoint described above that is normally taken of a conventional distributed system.

I.2. A Pilot Experiment in Distributed Interpretation

A set of pilot experiments were performed to begin evaluating the effectiveness of this approach using a network of nodes, each of which was a complete Hearsay-II interpretation system [Lesser and Erman 1980]. The Hearsay-II architecture appears to be a good structure for an FA/C system because it incorporates mechanisms for dealing with uncertainty and error as an integral part of the problem-solving approach. 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. It was also our hypothesis that the control and data structures of Hearsay-II could be distributed effectively because there were already existing mechanisms within its problem-solving structure for resolving uncertainty caused by incomplete or incorrect input data and KS processing. (For further information about the Hearsay-II architecture see [Erman, Hayes-Roth, Lesser, and Reddy 1980].)

Our approach to developing a distributed interpretation architecture based on the Hearsay-II model was to organize the network into nodes operating on partial and possibly inconsistent views of the current interpretation and system state. This has led to a distributed interpretation architecture structured as a network in which each node in the network is an architecturally complete Hearsay-II system. "Architecturally complete" means that each node could function as a complete Hearsay-II system if it were given all of the sensory data and the required KSs. However, due to the distribution of sensory data and limited internode communication, each node has a limited view of the complete problem-solving database and, in effect, a limited set of KSs.

Experiments were performed to determine how the problem-solving behavior of such a network of Hearsay-II systems compares to a centralized system. The aspects of behavior studied include the accuracy

of the interpretation, time required, amount of internode communication, and robustness in the face of communication errors. These experiments were simulations only in part, since they used an actual interpretation system analyzing real data; i.e., the Hearsay-II speech understanding system. In these experiments, we modelled a spatial distribution of sensory data by having each node of the distributed speech understanding network sample one part (time-contiguous segment) of the speech signal.

The experiments showed that the network of Hearsay-II speech understanding systems, with only minor changes involving the addition of mechanisms for internode cooperation, performs well as a cooperative distributed network even though each node has a limited view of the input data and exchanges only high-level partial results with other nodes. In an experiment with errorful communication, system 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. These results support our general model of distributed system design. They also indicate that the Hearsay-II architecture is a good one to use as a basis for this approach.

II. Important Issues for FA/C Problem-Solving

The pilot experiments gave exciting results, but were restricted to a small part of the design space of system organizations and were not based on a realistic distributed task. We now feel it is appropriate to expand our research focus in order to 1) empirically measure the performance of FA/C systems to determine their limitations and utility on a more realistic task 2) develop new strategies for control and communication to extend the range of applications appropriate for the FA/C paradigm.

Empirical performance measures will be taken for a range of task and problem-solving system situations in order to evaluate and analyze the approach:

1. Self-correcting Computational Structure - What and how much uncertainty and error can be handled using these types of computational structures, and what are the costs (and tradeoffs) in processing and communication to resolve the various types of errors?

2. Task Characteristics and the Selection of an Appropriate Network Configuration - What characteristics of a task can be used to select a network configuration appropriate for it? When can implicit control and information flow

structures be used? Similarly, when should flat, hierarchical, matrix organizations (see Figure 1) or mixtures of them be used? Candidate task characteristics include the patterns of node interaction; the type, spatial distribution, and degree of uncertainty in information; interdependences of partial interpretations; size of the search space; desired reliability, accuracy, responsiveness, throughput, and available computing resources.

Strategies for communication and control are vitally important because of the potential for resource overloads, imbalances and lack of coherence when there is no global view or control of the problem-solving system:

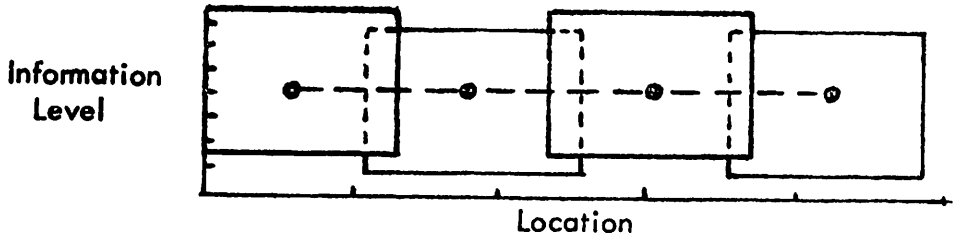
1. Communication Policies - What type of information and meta-information should be transmitted among nodes, what should be the criteria for deciding when to transmit this information and to whom?

2. Control of Processing - What type of control and organization relationships should there be among nodes? How to decide in a decentralized manner what tasks each node should focus its limited resources on? This is the problem of dynamic allocation of information and processing within the network so as to adapt to the varying state of the system and the environment.

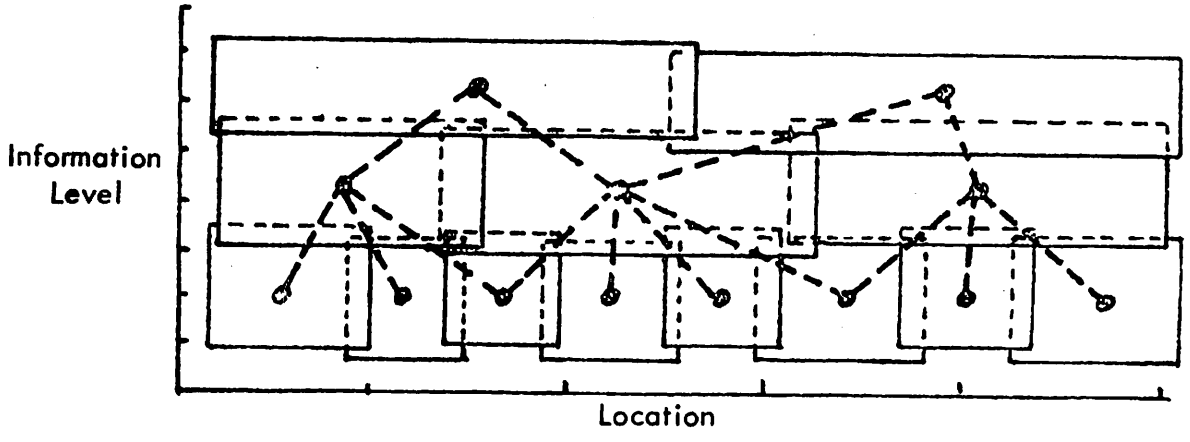
This last issue, control of processing, is a major focus of our research. We feel a key issue in the successful application of an FA/C approach is the attainment of high-level cooperation and global coherence in the network of autonomous, self-directed problem-solvers while still maintaining system robustness and limited communication bandwidths. We are especially interested in considering not only implicit, self-directed forms of decentralized control, such as used in the pilot experiments, but also more explicit, externally-directed control structures. We have been developing new approaches that combine both these types of control and discuss this in the following section. Communication policies are discussed more fully following that.

II.1. Explicit versus Implicit Approaches to Control

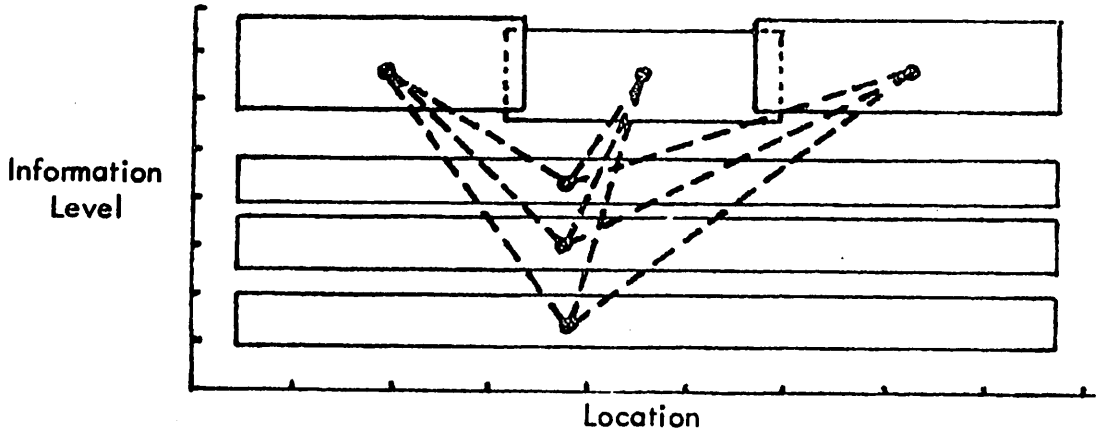
In our previous discussions of FA/C distributed systems, we have emphasized an "implicit" form of decentralized control. The degree of implicit versus explicit control in internode coordination can be characterized by the precision with which a node can specify the nature of the tasks that are to be executed by another node and the degree to which those tasks must be performed by the other node. From a



a: Schematic of a "flat" configuration.



b: Schematic of an overlapping hierarchical configuration.



c: Schematic of a matrix configuration.

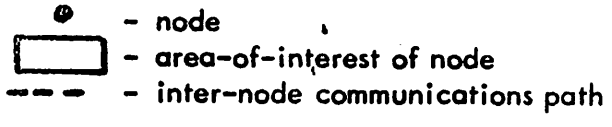


Figure 1: Schematics of some network configurations.

communication perspective, this control spectrum takes the form of a range of assumptions a node can make about: who is going to receive its messages, how its messages are going to be processed, who is going to send it messages, the nature of the information contained in these messages, what processing is expected by the transmitters of these messages, and what responses they expect to receive.

We have emphasized implicit and decentralized control for FA/C distributed systems because in this type of control a node has fewer built-in assumptions about the nature of internode coordination. This permits nodes to be more adaptive and flexible in the face of data and control uncertainty. We have implemented this form of control by having self-directed nodes which are activated in a data-directed manner. In this control regime, nodes interact only through the transmission of data. When a node receives information, it must decide whether or not to accept the information, what credibility to associate with it, what processing results (goals) it should achieve in light of this information, and what processing tasks it should execute to accomplish these goals. Because these decisions are made locally, node processing is entirely self-directed. In a similar self-directed manner, a node decides what and when information should be transmitted, based on the state of its local processing and its perception of the state of problem-solving in the network.

This data-directed and self-directed approach to control can be contrasted with approaches where either goals or tasks are explicitly transmitted and with approaches where nodes are externally-directed. By "externally-directed" we mean that a node is required to perform some action in response to the receipt of a message. In these other approaches to control, nodes have less flexibility in their processing strategies. These alternative control regimes are illustrated in Figure 2. The more specific the message (i.e., tasks are more specific than goals and goals are more specific than data) and the more externally-directed a node is, the more explicit the form of control.

In the pilot experiments with the distributed Hearsay-II architecture, we have observed that the data-directed and self-directed control regime used in this architecture can potentially lead to redundant and unnecessary processing. For example, in the pilot experiments 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. The problem of enlarging or changing a node's area of interest is nontrivial since it may become necessary later to go back and re-solve the old problem using new information. Another problem occurs when a node has bad data and cannot possibly find a good solution without help from other nodes. How can such a situation be detected and, if detected, what should be done? Similarly, when nodes are producing solutions at different rates, nodes with bad solutions will distract

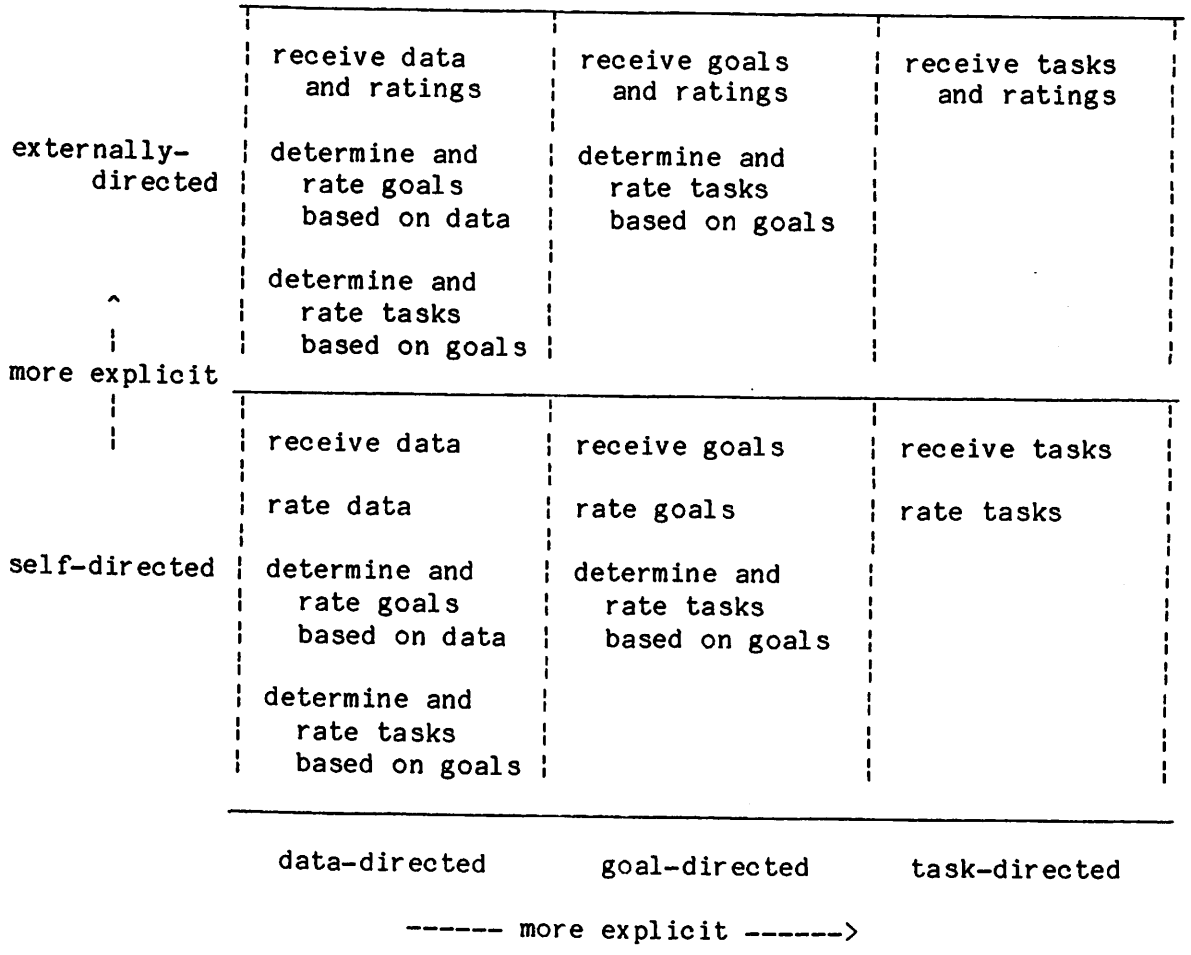


Figure 2: Implicit and explicit forms of control.

other nodes by communicating their results*. Thus, it appears that the data-directed, self-directed form of control may not always provide sufficient global coherence among the nodes. It is also our conjecture that in situations of dynamically changing task characteristics and changing processing capacities, additional control mechanisms will be necessary for maintaining global coherence.

There are two approaches for obtaining increased global coherence. The first is to provide each node with a better view of the state of problem-solving in the network so that its data-directed and self-directed control decisions are more informed and consistent. This can be accomplished by having nodes exchange detailed meta-information about the state of their local problem-solving and what they have learned about the states of other nodes. Another approach, which is compatible with the first, is to integrate more explicit forms of control into network problem-solving. These types of control can be used to institute more non-local and precise control over the activities of individual nodes.

One example of a more explicit approach to decentralized control is the work of Smith and Davis on the contract-net formalism [Smith 1980; Smith and Davis 1981]. In this approach, nodes coordinate their activities through contracts to accomplish specific goals. These contracts are elaborated in a top-down manner; at each stage, a node decomposes its contracts into sub-contracts to be accomplished by other nodes. This process uses a bidding protocol based on a two-way transfer of information to establish the nature of the sub-contracts and which node will perform a particular sub-contract. This elaboration procedure continues until a node can complete its contract without assistance. From an FA/C perspective, the disadvantages of this approach are that it is difficult to quickly refocus the system to new events (because of the hierarchical nature of control) and that it does not really address the issue of coordinating the iterative, coroutine exchange of partial and tentative intermediate results between nodes.

We believe that more sophisticated control regimes that incorporates the full range of implicit to explicit control may be required for effective problem-solving in some FA/C distributed systems. This type of control regime can provide the flexibility to handle control and data uncertainty while still maintaining a sufficient level of global coherence to guarantee that acceptable solutions will be generated within given resource constraints. This approach to control implies, as will be discussed later, that a node possesses a more

---

*Even in a situation of bad data, a node may be able to find some incorrect partial solutions that appear reasonable and in fact finding them more quickly than a node with good data because the search space of plausible alternatives is much smaller.

sophisticated form of local control; it should possess capabilities to plan sequences of activities and to adapt its plan based on its self-awareness of its previous activities, its problem-solving role in the network, and the status and role of other nodes in the network.

Given these planning and meta-information gathering capabilities, it is natural to think about organizational self-design [Corkill 1980] as a framework for obtaining node cooperation and coherence. Organization self-design is the explicit planning by the system of the information and control relationships that exist among the nodes in the network and the distribution of problem solving expertise among the nodes in the network. As the problem solving environment changes, the distributed system may need to change its organizational structure to maintain its effectiveness. In order to effect such a change, the distributed system must (1) detect the decreased effectiveness of its present organizational structure; (2) propose alternative organizational structures; (3) evaluate the cost of continuing with its current organizational structure versus reorganizing itself into a more appropriate structure; and (4) carry out the reorganization if appropriate.

Our initial approach to exploring a more sophisticated control regime has been to extend the distributed Hearsay-II architecture by adding a planning module to each node and permit communication of goals [Corkill and Lesser 1981]; this planner can adapt local node activity to respond to both externaly-directed requests by other nodes (goals) and the processing needs of the node based on the data it is receiving and the results it has so far produced. Furthermore, the planner can bias scheduling of these activities based on the organizational role the node is currently playing. In such an approach, the priority given to goals received from other nodes versus local data-directed activity determines the degree of explicit versus implicit control present in the system. Our approach is to permit both types of coordination, and to develop adaptive mechanisms for the system that dynamically determine an appropriate combination [Corkill and Lesser 1981]. An important part of the development of this approach will be empirical studies to understand the appropriate balance between data-directed and goal-directed activity.

II.2. Communication Policies

In Lesser and Erman (1980), a wide range of alternative policies for inter-node communication were laid out. One of the major focuses in the use of the testbed is the empirical evaluation of the appropriatenes of these policies under different task and system characteristics. In the context of these different policies, we would like to explore various high level trade-offs such as: timeliness versus reliability of communicated information, communication intensity and detail of

information versus system reliability, and fixed versus adaptive communication policies.

The alternative policies for communication that we want to examine can be classified along the following general dimensions:

1. The type and level of detail of information communicated.

2. The type of control relationships among the producer and consumer nodes used to initiate the transmission of information.

3. The type of meta-knowledge that the producer and consumer have of each other and how that knowledge is acquired and used.

The level of detail of communicated information can range from a hypothesis with complete substructure (all hypotheses that support it) to one with no substructure. The level of detail is closely related to the purpose of transmission, and though for most cases a "single hypothesis" seems to represent the right level, this may not always be the case. The increased cost of transmitting the hypothesis's substructure can be justified if the receiver is able to use this additional level of detail to avoid redundant computation and to more effectively integrate the hypothesis into its current problem-solving state. For example, while a received hypothesis may be inconsistent with hypotheses already in a node, some of the support may be consistent and can be used if it is transmitted. Also, the manner in which a hypothesis is merged with similar hypotheses depends on whether the substructure is the same or different for each hypothesis. In addition, the transmission of substructure allows the receiver to deduce more detailed meta-information about the quality of processing in the transmiting node. The experiments with the testbed will reveal more about the reliability and cost-of-transmission trade-off involved here.

The type of control that is used to initiate communication can be characterized along an implicit/explicit dimension. In pure implicit policies a node decides what and when information is transmitted in a self-directed manner, based on the state of its local processing and its perception of the state of problem-solving in the network. In this pure form, there is no use of knowledge about who will receive the information nor what type of processing the receiver will perform. Additionally there is no explicit acknowledgment that the information has been received. An example of a simple implicit strategy is the local broadcasting of information to all nodes that are directly connected. Implicit policies begin to move in the direction of explicit policies as information about whom to transmit to is incorporated into the policy. In the pure explicit communication policy, all transmission is in response to an explicit request for that information by another node.

In implicit policies, some of the important criteria for what and when information is transmitted are timeliness, accuracy of information and amount of local processing resources available. The timeliness criterion, for example, can be translated into a "best-first" type of communication policy, in which as soon as a relatively high belief hypothesis is produced, it is transmitted. However, if guided solely by the accuracy criterion, a hypothesis will not be transmitted before local processing on it is exhausted ("locally complete" policy). The pilot experiment shows that when input data reliability is generally low, the locally complete policy seems to be a better choice, but we plan to examine this issue in detail using the testbed.

The emphasis in the availability of resources criterion is a result of our intuitions that the system should adapt to environmental change. "Exceptional events" such as heavy traffic or processor failure could be treated directly in the network hardware configuration, for example by adding "slack" processing resources in every node. However, it seems more reasonable to design a network for average conditions, and then let the communication policy compensate for overload, missing KSs and data etc. by redistributing information in the network.

We are also interested in exploring transmission criteria based on domain-dependent characteristics. For example, transmitting location hypotheses on the "edges" of node's sensing area, or the track hypothesis for a vehicle that will be soon leaving the node's area of interest.

The type of meta-information used is another important aspect of both implicit and explicit communication policies. There are four classes of meta-information about the network problem-solving structure that are potentially useful for each node: the topology of the network, the processing capability of nodes, the dynamic state of problem-solving in the nodes and other node's meta-information. The lack of appropriate meta-information contributes to the introduction of four possible types of uncertainty into internode cooperation: consumer, producer, functional and result uncertainty [Fox 1981]. Determining an effective balance between decreases in system reliability and inefficient uses of system resources (caused by the lack of appropriate meta-information) against the cost of acquiring and maintaining meta-information is one of the key issues that needs to be explored.

Meta-information can be transmitted explicitly or implicitly. An example of implicit transmission occurs when the hypotheses received from other nodes are used to construct a model of the state of problem-solving in the network. (This is the model used in the pilot experiments.) Certain types of meta-information that a receiver wants to have are known to the sender and it may be more economical for this type of meta-information to be explicitly communicated rather than being deduced by the receiver. There are further variations of the explicit scheme: a node can decide locally when to send meta-information (when

it changes significantly) or as a result of a request to do so. Most of the considerations about implicit/explicit communication of problem-solving information apply to meta-information, and we would like to examine the factors that determine which scheme of meta-information acquisition is appropriate in a given situation.

Let us now show in detail some of the different policies we want to examine in terms of the type of uncertainty they produce.

Consumer uncertainty: who will receive the message? In a simple implicit communication policy where there is no model of network topology and a local broadcast strategy is used, there is high consumer uncertainty. However, this simple policy is quite robust since if the network structure is changed (as in the case of displaced or inoperative node) no changes to the policy need be made. Implicit policies can be augmented by "murmuring" or retransmitting high-impact hypothesis [Lesser and Erman 1980]. This policy also increases robustness in case of errorful communication channels between nodes. An incremental transmission mechanism (with processing at each step) also serves this purpose, by ensuring the spread of important information through the network.

In more explicit communication mechanisms, dynamic communication paths are established between the nodes. Consumer uncertainty can be reduced in two ways: by negotiating a consumer [Smith 1980; Lesser and Erman 1980] or locating a consumer based on a model of processing of other nodes. The first method is potentially time and resource consuming, while the second can impose considerable processing overhead for maintaining the model information.

Producer uncertainty: who contributed in generating this information? This type of uncertainty can be resolved if processing history, represented by KSs that have worked on a hypothesis, is sent with the hypothesis. The additional knowledge represents another aspect of level of detail in information, and has similar impact as hypothesis' substructure: it may save unnecessary processing, locate sources of problems, redirect control, etc.

Functional uncertainty: what function (in our case, KS) will work on the information? If this is not known, it can not be predicted how long it will take a KS to process the information and how good the produced result will be. Also, the proper level of detail and accuracy of transmitted information may be unknown.

Response uncertainty (corresponds to Fox's result uncertainty): what will be the response of the receiver to transmitted information? It depends on the cause for transmission. For example, if a node sends a high belief hypothesis, it assumes that nodes that can make use of it will receive it. On the other hand, if a hypothesis is about the vehicle moving out from node's sensing area, a node will probably want

an acknowledgment that someone else picked it up. In certain cases, the acknowledgment would not be enough; for example, if a node is lacking a KS, it might want to send the best hypothesis on its input level, and have the results of its processing sent back. We plan to augment the nodes with a dialogue-game structure [Mann 1979]. A small set of games with corresponsing rules for their acceptance or refusal are defined (information offer, information seeking, request). This approach, besides reducing the result uncertainty, increases the robustness of the system, since a node always anticipates a response of a certain type, and if expectations are not satisfied an error has occurred.

III. A High Level Testbed for Interpretation Systems

In order to empirically evaluate the alternative policies for control and communication we have developed a high-level simulation testbed. This testbed is based on a distributed interpretation task that, as well as being realistic, will allow us to explore empirically a much larger part of the design space than the task in the pilot experiments. In the following sections, we first discuss the task and how it has been abstracted and parameterized and then discuss the problem-solving framework that has been developed to solve the task.

III.1. The Parameterized Traffic Monitoring Task

The interpretation task that we have chosen for the testbed is distributed traffic monitoring. The problem is to identify, locate, and track patterns of vehicles moving in a two-dimensional space. Figure 3 shows a typical layout of a task environment, with sensors, processing nodes, and vehicles. As a result of normal operation, vehicles generate acoustic signals, some of which are received by sensors reporting to a monitoring system. The monitoring system produces and maintains a map of traffic in the environment. (This task is similar to one used by Smith [Smith 1980].)

The traffic monitoring task has been chosen for several reasons. It is a natural task for a distributed approach, since sensors are in different geographical locations. Also, the task can be easily varied in terms of signal complexity, temporal and spatial constraints on possible interpretations, and configurations of sensors and nodes in the monitoring system (see Table 1). This variability permits us to consider the effects on system behavior of different types, spatial distributions, and degrees of uncertainty in the task. The size of the search space is adjusted by varying the density of vehicles and the shape of their trajectories. The number and size of the vehicle patterns can also be varied in order to modify the strength of the

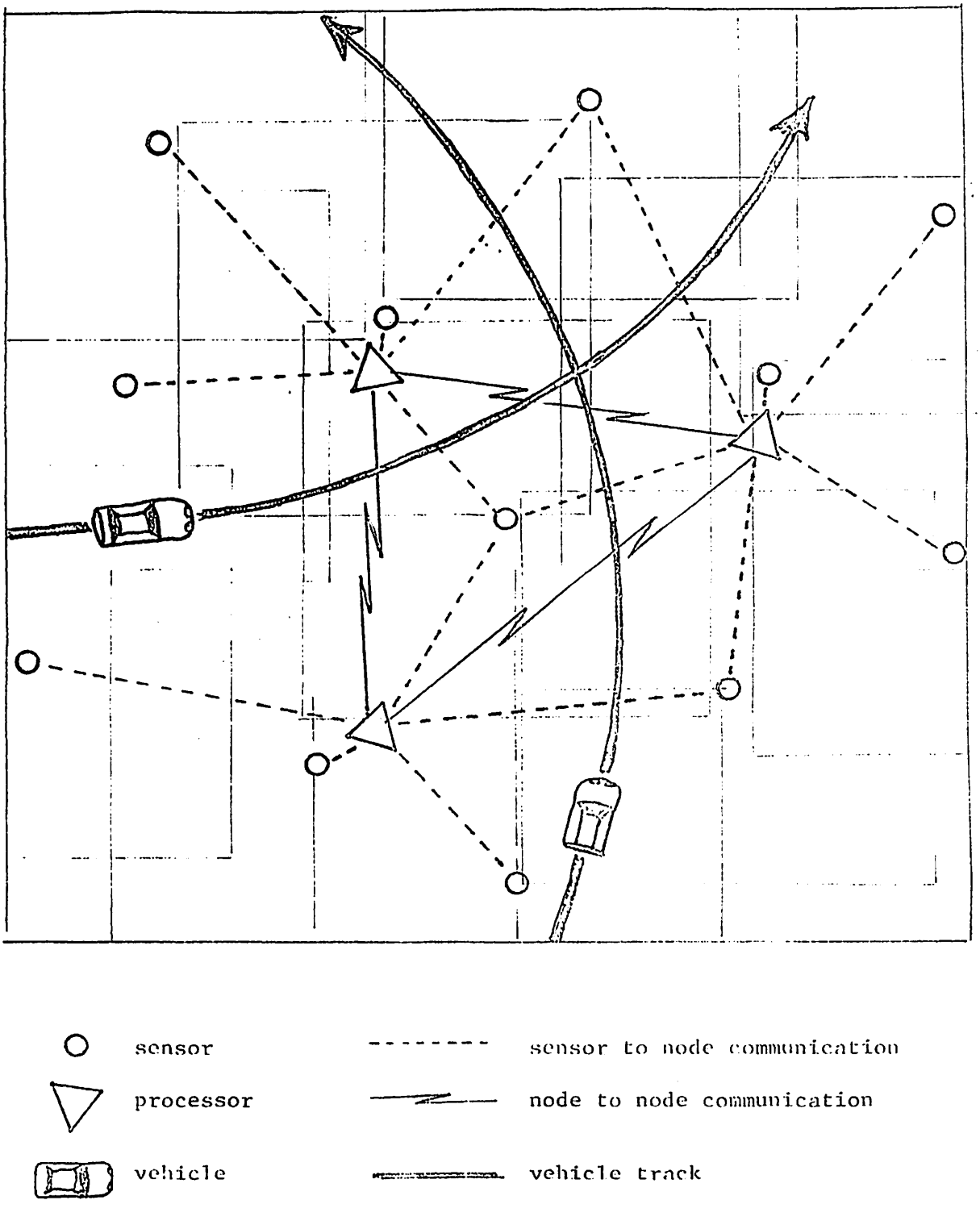


Figure 3: Vehicle monitoring task.

spatial interdependence of events. Additionally, the type of processing required to accomplish the task leads to a decomposition into disjoint levels of abstraction, with independent processing possible at each level. Another important aspect of this task is that multiple, diverse sources of knowledge must be brought to bear in order to solve it (i.e., a harmonic-set-formation expert, a tracking expert, etc.)

The task and sensors are abstracted in several ways, shown in Table 2. The viability of our task abstraction was confirmed during discussions with the Lincoln Lab Distributed Sensor Net Team, which is designing a real signal interpretation system [Green 1979]. This task is a representative example of distributed interpretation problems, which we feel are appropriate for our approach.

III.2. The Parameterized Distributed Interpretation System

The testbed simulates a network of extended Hearsay-II systems applied to the monitoring task. In addition to abstracting the task, we have also made simplifying assumptions about the knowledge processing required to solve the problem. We have chosen these abstractions and simplifications because the actual task is highly complex and thus requires much effort to successfully engineer the required knowledge and excessive processing resources to simulate. Parameterization of the actual task is also much more difficult.

We have at the same time chosen not to simulate a highly abstracted version of the task. A more abstracted version of the task, while efficient to simulate and flexible, would not capture all the knowledge and task-related behavior we want to study. Specifically, a highly abstract model that represents processing only as manipulations of average belief-values (see [Fox 1979]) is inappropriate for our purposes. We feel the task should be detailed enough to distinguish a system's answer (the interpretation) and all intermediate processing states on the basis of:

1. the location of hypothesized events in time and space;
2. the nature of hypothesized events (signal frequency, harmonic set, vehicle, pattern);
3. the consistency among hypotheses (the degree of correctness);
4. the belief-values (credibility-ratings) of hypotheses.

This level of detail permits the system to produce actual interpretations, and thus we can be more sure from its results of the nature and interactions of knowledge needed to solve an actual problem.

ENVIRONMENT VARIABLES:
- -number of events per time frame
- -size of monitored region
- -size of a unit square

SENSOR VARIABLES:
- -number, size and resolution of sensor zones
- -probability of not sensing an existing signal
- -probability of sensing a non-existent signal
- -number and relative positions of sensors
- -fuzziness in signal location and frequency
- -belief assigned to signal hypotheses

VEHICLE VARIABLES:
- -number and density of vehicles
- -speeds
- -shapes of trajectories
- -number and size of vehicle patterns
- -branching factor relationships among patterns, vehicles, harmonic groups, and signals
- -'confusability': number of overlaps in frequencies and harmonic sets between the vehicles of different types

KNOWLEDGE SOURCE VARIABLES:
- -knowledge source power
- -generator stage threshold on output hypothesis belief-values
- -processing time

NODE VARIABLES:
- -number and types of knowledge sources in a node
- -blackboard levels present in a node
- -sensors reporting to a node
- -processing speed of a node
- -area of interest (part of the interpretation that node is responsible for)
- -scheduler power

SYSTEM VARIABLES:
- -number and relative positions of nodes and sensors
- -overlap of areas of interest
- -types of communication links
- -characteristics of a communication channel (error, bandwidth)

Table 1: Testbed Variabilities

1. Space is a two-dimensional square grid, where the size of the unit square determines maximum spatial resolution.

2. The system does not observe its environment continuously, but at discrete points in time (time-frames). The phenomena which occur due to the time lags of signal propagation are ignored.

3. Sensors indicate signals' locations and discretized frequencies.

4. Sensors can make three types of errors:
   -failure to detect a signal,
   -detection of a nonexistent signal, and
   -inaccurate determination of location and/or frequency of signal

Table 2: Abstraction of Task and Sensors

For example, without this detail we can not explore the effects of different transmission policies (what information to transmit, how to treat received information, etc.). In order to understand the effect of these different policies, we need to understand how the system behaves when there is multiple processing of the same information or processing that increases certainty due to multiple views of the same event, which depend on the details of the information in the nodes' data bases and their interactions.

The remainder of this section of the paper describes a node in the system. This includes a high level description of knowledge source (KS) processing in a node, the basic architecture of a node, the blackboard structure of a node, the details of KS and goal processing in a node and a description of our method of parameterizing and measuring problem-solving expertise in the system.

3.2.1 KS Processing in a Node

The following considerations were used as criteria for constructing KS processing structures at each node:

1. KS processing should permit the effective utilization of information asynchronously generated or received at any level of abstraction (data-directed at all levels).

2. KS processing should be uniform at all levels and should permit information to be generated bottom-up, top-down or any combination of the two.

3. KS processing should allow a node to work in a self-directed manner, in that it does the best possible processing with available information. KSs should be able to skip over information that is unavailable and continue processing.

4. Lower-level hypotheses should concern a small region of space and time, while abstract hypotheses at higher blackboard levels should have greater scope. This type of hierarchical structure in the blackboard makes it possible to construct "realistic" heirarchical node organizations in terms of both information and control flow.

A system with node processing that meets these criteria permits exploration of a wide range of different processing decompositions (flat, hierarchical, matrix, etc.) based on partially configured nodes (those without all necessary KSs and with limited areas of spatial/temporal interest) without modifying the KS modules and local control structures.

3.2.2 Architecture of a Node

Our KS processing criteria are met by nodes structured according to a generalized and extended Hearsay-II architecture with KSs, blackboards, KS scheduler, and planner. Interprocessor communication can be naturally included in the architecture by adding KSs on each blackboard level to send and receive messages (hypotheses and goals). Our simulation model permits arbitrary node-node, node-sensor connectivity and arbitrary distributions of task processing capability across the nodes in the system.

We also have extended the capabilities of the Hearsay-II architecture so that the full range of distributed processing issues previously detailed can be effectively explored by introducing a form of explicit coordination among nodes through the use of goals [Corkill and Lesser 1981]. A diagram of the problem-solving architecture of a node is given in Figure 4.

3.2.3 Blackboard Partitions in a Node

There are two parallel blackboard structures in each node of our system: data and goal*.

The data blackboard corresponds to the blackboard of an actual monitoring system. It is the only blackboard from which KSs can obtain input and is the site of all KS stimulus events.

---

*In our original implementation there was a third partition called the potential blackboard. The potential blackboard partition held hypotheses that did not have enough credibility to currently justify further processing. The potential blackboard avoided the building of a more complex KS model in which the KS acts as an incremental hypothesis candidate generator based on threshhold controls. In this latter case, a KS would generate a few most plausible hypotheses and, if necessary, be later re-invoked to generate less plausible hypotheses. Instead, we initially decided to structure a KS so that it generates all the plausible hypotheses on the potential blacboard and then have a focus-of-attention mechanism control how many and when these hypotheses will be moved onto the data blackboard. However in practice, we found that the use of this intermediate blackboard was unnecessary because the number of hypotheses KSs created was not large and thus the processing costs of placing all the KSs hypotheses directly on the data blackboard was not unreasonable.

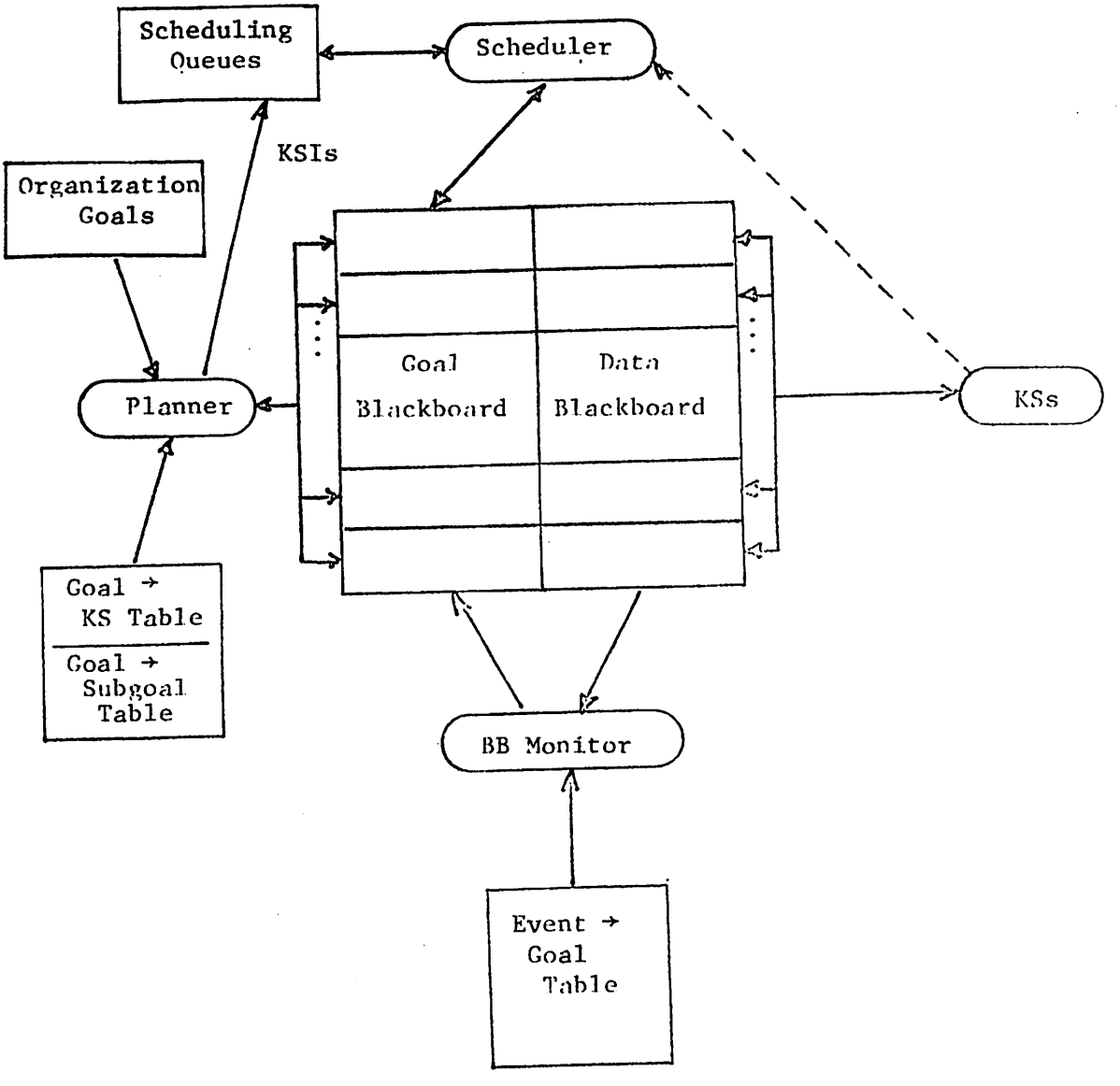


Figure 4: Goal-directed Hearsay-II architecture.

The goal blackboard holds goals, each representing a request to create a set of hypotheses with specific attributes on the data blackboard in the (corresponding) area covered by the goal. For example, a simple goal would be a request for the creation of a vehicle track hypothesis above a given belief in a specified area of the data blackboard. The goal blackboard is used by the scheduling and planning mechanisms to decide which KSs should be instantiated.

The blackboard of a node can be thought of as having four-dimensions, because hypotheses can be addressed by their level of abstraction, the two spatial (x,y) coordinates of the hypothesized event, and the time of the event. Information classes in the system, in order of increasing abstraction, are signal, group (harmonic set), vehicle and vehicle formation (pattern). For each information class, there are two blackboard levels, one representing hypotheses about the location of a single event, and the other representing a track made from events which are consecutive in time and space (see Figure 5). Location blackboard levels are: signal-location (SL), group-location (GL), vehicle-location (VL) and pattern-location (PL). Track blackboard levels are: signal-track (ST), group-track (GT), vehicle-track (VT) and pattern-track (PT). Tracks can be formed from various combinations of locations and tracks of the same information class, or from the tracks of the lower abstraction information class. Locations can be formed from the locations of the lower abstraction information class. An example of a simple grammar for specifying these groupings is specified in Figure 6.

3.2.4 Details of KS Processing in a Node

The patterns of KS processing arise from the interaction of three distinct types of KSs: synthesizers, mergers, and extenders (see Figure 5).

A synthesizer KS creates higher-level hypotheses by abstracting the information contained in a group of lower level hypotheses. There are three types of synthesizer KSs in our system which work from location to location, location to track, and track to track levels on the data blackboard. An example of the first type, S:SL:GL KS, is triggered by the creation or modification of a signal-location (SL) hypothesis, and generates group-location (GL) hypotheses, combining the signals belonging to the same harmonic set. An example of the second type, S:SL:ST KS, is triggered by the creation or modification of a signal-location hypothesis, and generates signal-track (ST) hypotheses combining consecutive signal-location hypotheses of similar frequencies. An example of the third type, S:ST:GT KS, is triggered by the creation or modification of a signal-track hypothesis, and generates group-track (GT) hypotheses combining the tracks of signals in the same harmonic set.

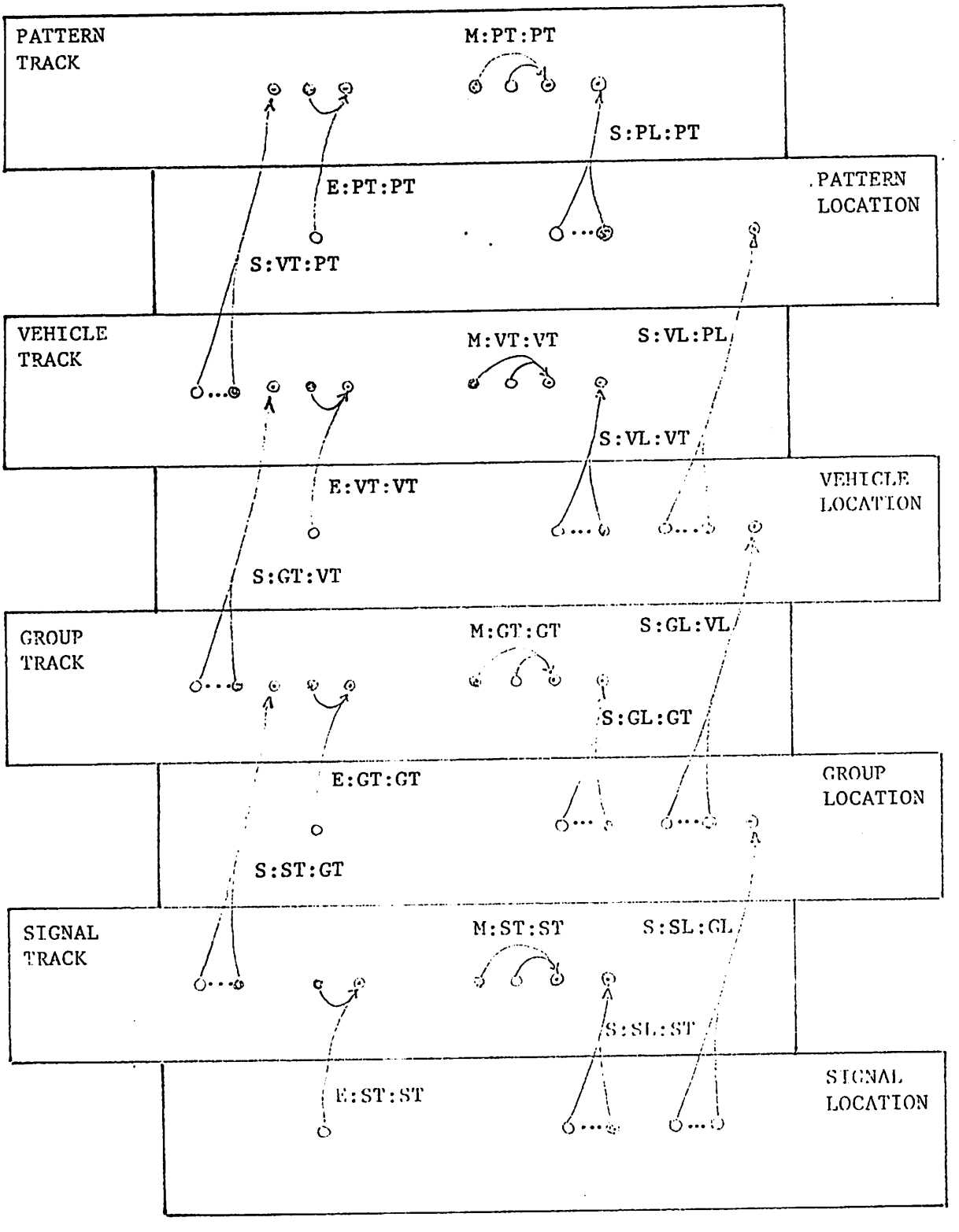


Figure 5: Data blackboard showing levels and KSs.

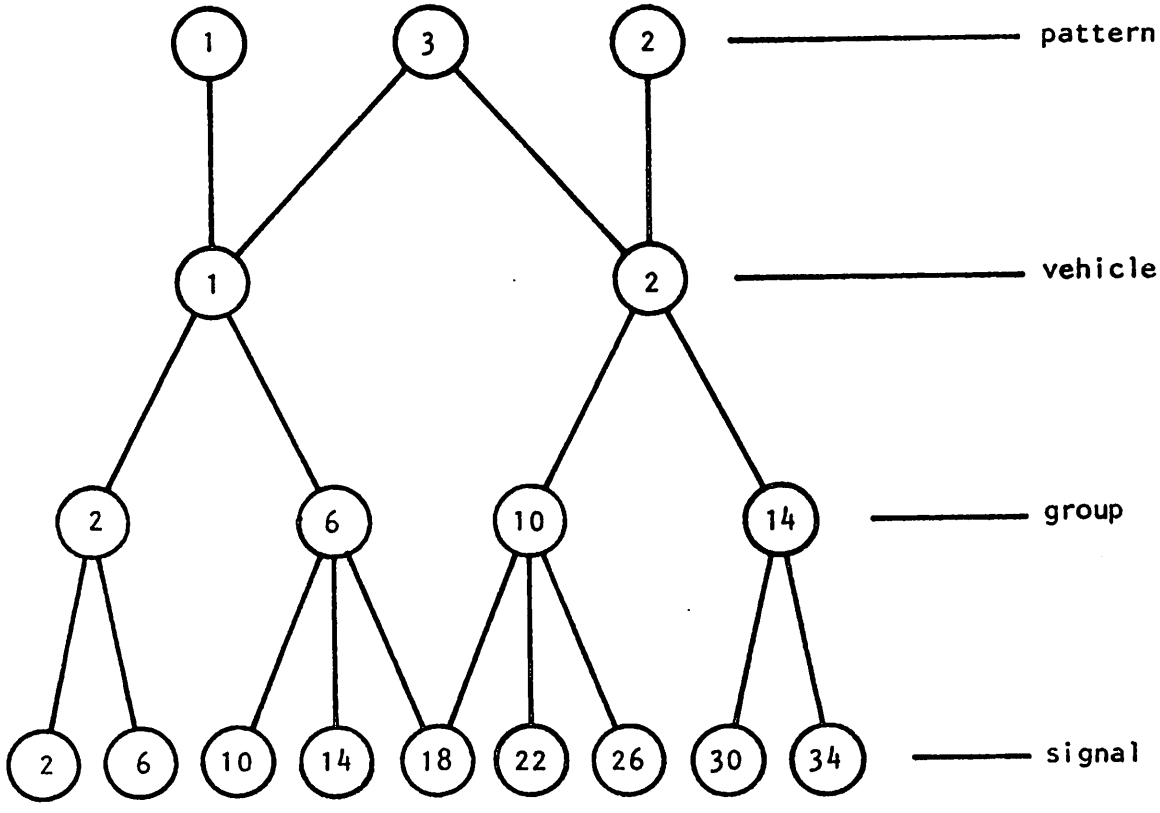


Figure 6: Example of a simple grammar.

The merge KS is different from synthesis KSs because it does not produce a hypothesis at a higher level of abstraction than its input but rather at the same level. There is a merge KS for each track level on the blackboard. The criteria for merging two tracks of the same event class is that the tracks overlap in time and location. The resulting merged track is supported by the two overlapping tracks. This type of KS is necessary for integrating track hypotheses received from other nodes into the blackboard of the receiving node. This KS is triggered by the creation or modification of a track hypothesis.

The extender KS creates track hypotheses from an existing track hypothesis and location hypotheses on the same abstraction level by extending the track forward or backward in time. These KS's are triggered by the creation or modification of a track hypothesis. There is an extender KS for each level of abstraction. Appendix D contains a more complete desciption of these different types of KSs.

We have not included among these basic types of KSs those that do top-down elaboration or "analysis by synthesis" action, such as decomposing a group location hypothesis into the set of possible signal location hypotheses that could potentially support it. As will be discussed in the paper on goal-processing in this volume, we feel this type of use of hypotheses to force the verification of model-directed expectations is inappropriate. Rather, this type of control should be implemented in the planner through subgoal-processing on the goal blackboard (see [Corkill and Lesser 1981]).

3.2.5 Details of Goal Processing in a Node

In order to permit more sophisticated forms of cooperation among nodes in the system, we have integrated goal-directed control into the data-directed control structure of the Hearsay-II architecture; this has been accomplished through the addition of a planning module and a goal blackboard. This permits nodes to effect the processing of other nodes not only through the transmission of hypotheses but also through transmission of goals which are requests for the creation of hypotheses with certain attributes.

The integration of data-directed and goal-directed control into a single framework is based on the following observation:

> The stimulation of a KS in the data-directed architecture not only indicates that it may be possible to execute the KS, but that it may be desirable to do so in order to achieve the goal implicit in the output generated by the KS.

In order to make these implicit goals explicit, we split the event -> KSs mapping into two steps: event -> goals and goals -> KSs. The

blackboard monitor watches for the occurrence of a data blackboard event, but instead of placing KS instantiations on the scheduling queue (if the KS preconditions are satisfied), it uses the event -> goals mapping to determine the appropriate goals to generate from the event and inserts them onto the goal blackboard. These goals represent the implicit goals contained in the event -> KSs mapping used by the original blackboard monitor (see Appendix A for an example of this process).

A new control component, the planner, is also added to the architecture. The planner responds to the insertion of goals on the goal blackboard by developing plans for their achievement. The goal -> KSs mapping is used by the planner to instantiate one or more KSs which can potentially satisfy the goals. The scheduler then uses the relationships between the KS instantiations and the goals on the goal blackboard as a basis for its scheduling decisions.

The planner also has the responsibility of integrating externally received goals into the node's current goal structure. This integration is accomplished by linking together external and internal goals through goal-subgoal relationships. Local node activity can then be biased so as to satisfy external goals by rating higher those internal goals that help achieve the external goals. The higher the rating attached to these internal goals, the more explicit the control relationship among nodes can be made.

The planner also bases its goal rating decisions on 1) its view of the current state of processing in the node, 2) the "organizational" goals it is currently adhering to, and 3) the importance attached to the goal by the sender. In our current implementation, the following factors are used to define the organizational goals of a node:

1. the organizational importance of having the node generate hypotheses at particular levels, times, areas, and event-classes;

2. the organizational importance of having the node send to and accept from particular nodes hypotheses and goals at particular levels, times, areas, and event-classes.

The more importance the planner gives to "organizational goals" the more coordinated the system can be made. A more comprehensive description of goal processing and the role of organizational design in FA/C system is given, respectively, in companion papers in this volume [Corkill and Lesser 1981] and [Corkill 1980].

We have currently implemented a simple version of the send goal and receive goal KSs based on the following algorithm. A goal becomes a candidate for transmission if its characteristics match those specified by the node's organizational role. The priority of sending this

candidate goal is then determined by the following factors:

1. the local rating associated with the goal;

2. the importance that is attached to sending this type of goal as specified by the node's organizational role;

3. whether the goal can be satisfied locally (i.e., there are KSs that could be run which can potentially satisfy this goal); and,

4. whether the goal is linked to any subgoals which could be potentially satisfied as a result of further node processing.

Also associated with a node's organizational role is a minimal threshold for transmission; if the goal is below this threshold, it is not transmitted. Otherwise, the goal is placed on a priority rated queue and is transmitted when all more highly rated goals have been transmitted. A similar scheme is used to rate hypotheses for transmission, and hypotheses that are candidates for transmission are placed on the same priority rated queue as is used for goals. This use of a single queue for transmission permits the easy specification of mixed initiative transmission policies.

In the case of a receiving node, if an externally received goal is satisfied as a result of local processing, the node based on the external goal's attributes can respond in one of the following ways:

1. it can do nothing;

2. it can transmit an acknowledgement to the sender of the goal that the goal is satisfied; or,

3. it can transmit the satisfying hypotheses and possibly the acknowledgement of satisfaction to the sender of the goal.

As additional options, all the nodes that have seen this goal can be notified, or just a selected set of nodes (e.g., the original node that generated the goal).

In the situation in which a node cannot satisfy an externally received goal it may be able, based on the model of other nodes processing contained in its organizational goals, to pass on this goal to other nodes in the network for possible satisfaction. In this way a store-and-forward goal processing network can be built-up. If appropriate organizational goals are set-up, this network can be made to simulate the hierarchical flow of goals in the contract-net approach. However, currently there is no bidding protocol, like that in the contract-net approach, for dynamically choosing which nodes should

receive a goal. As we increase the sophistication of the send/receive goal KSs and transmit organizational goals among nodes, we hope to be able to simulate more sophisticated coordination and communication policies including the use of negotiation.

IV. Resolving Power - An Independent Variable*

An important aspect of our empirical evaluation of alternative FA/C system organizations is the measurement of how their effectiveness changes as the distribution of problem-solving expertise is varied. For example, we conjecture that in a system with KSs that are very reliable and with input data that has low error, organizing the system hierarchically and using an explicit control and communication strategy is more effective. Likewise, it is conjectured that in systems with less reliable KSs and with more errorful input data, more cooperative and implicit control/communication strategies are desirable. We would also like to examine the effects in performance caused by the trade-off between reliability of KSs versus processing time, i.e., low reliable and fast KSs versus highly reliable and slow KSs.

In order to understand the reasons for differences in the performance characteristics of alternative systems organizations, it is necessary to have dynamic measures that take into account the intermediate state of system processing and thus permit observations of performance over time. For example, one way of measuring the effectiveness of different communication strategies is to develop measures that evaluate the effect of each transmitted message on the current processing state of the receiving node. The need for parameterization of KS power and measurement of intermediate states of processing, plus the desire to simplify KS engineering have led us to develop a semi-formal model for analyzing how a Hearsay-II-like system constructs an accurate solution and resolves the uncertainty and error in its input data.

In the Hearsay-II paradigm for problem solving a complete and consistent interpretation is incrementally constructed by aggregating lower-level hypotheses into more abstract and encompassing higher-level hypotheses (partial interpretations). The lower-level hypotheses are said to support the higher-level hypotheses. This aggregation process involves the detection of local consistency (or inconsistency) relationships among hypotheses. In KS processing, the belief-values of supporting hypotheses are not changed as a result of detection of local

---

*This section is an excerpted and revised version of a paper by [Lesser, Reed, and Pavlin 1980].

consistency, as would be the case in a relaxation process. The construction of a higher-level hypothesis and its associated belief-value is an explicit encoding of the nature and degree of consistency found among its supporting hypotheses. The hope is that this incremental aggregation process resolves the uncertainty among competing interpretations and, simultaneously, distinguishes between correct and incorrect interpretations.

We have not discussed KSs as solely reducing the uncertainty in the system, because uncertainty is a measure of the distribution of belief-values and does not reflect the accuracy of hypotheses. We feel that KS processing causes changes in both certainty and accuracy in a system's database and we have developed a measure, called reliability, that combines the two. A good KS will produce higher-level hypotheses which are more reliable than the lower-level hypotheses supporting them.

IV.1. Measuring System State

Basic to our view of processing in knowledge-based systems is the concept of system state. The system state is the current set of hypotheses and their relationships to the input data. It is through measures taken of the system state that we can talk in a uniform manner about the performance of KSs and the scheduler. It also permits us to define the power of a node and to develop measures for the effects of communication among nodes.

For purposes of measuring reliability, we associate a hidden attribute with each hypothesis which we call its consistency-value. This attribute measures the closeness of the hypothesized event to an event that a KS with perfect knowledge would consider consistent (correct). For task domains in which a solution is either totally consistent or inconsistent, we quantify consistency-value as either 1 (consistent) or 0 (inconsistent), while in domains in which there are solutions of varying degrees of acceptability, consistency-values range between 1 and 0. In the testbed, the consistency value is either 0 or 1 and is detected by checking whether a hypotheses on the data blackboard also exists on a hidden blackboard structure, called the consistency blackboard, that will shortly be discussed.

One way of evaluating an intermediate state of processing is by measuring the reliability of the set of all possible complete interpretations (final answers) that are supported (at least in part) by hypotheses in the current state. We feel that a direct measure of this sort is not feasible because it is very difficult in general to relate the set of partial interpretations to the very large set of complete interpretations they can support.

We take an alternative approach, called reflecting-back, which is based on two premises. First, the creation of a high-level hypothesis is a result of detecting the constitency among its supporting hypotheses. This creation process is an alternative to actually changing the belief-values of the supporting hypotheses, as occurs in the relaxation paradigm. Thus, the creation of a high-level hypothesis implicitly changes the reliability of its supporting hypotheses. This change can be traced down to the input hypotheses whose reliability is implicitly improved to the extent that they are aggregated into reliable high-level hypotheses. Second, we assume that processing which implicitly improves the reliability of the input hypotheses also improves the reliability of the complete interpretations supported by these hypotheses.

In the version of reflecting-back approach used in the testbed, we associate with each hypothesis the average of its own reliability with the reliability of hypotheses it supports, where reliability for an individual hypothesis is defined as the belief-value for consistent hypotheses and one minus the belief-value for inconsistent hypotheses.

Our measure for the reliability of an intermediate system state is based on a measure of the reliability of the input (sensor level) competitor sets computed from the reflected-back belief- and consistency-values. A competitor set contains all alternative hypotheses describing possible interpretations for mutually-exclusive aspects of the sensory data. For example, all acoustic signals sensed in the same point in space but impossible to be generated by the same vehicle. As will be discussed later, the competitor set structure for all sensor level hypotheses in the testbed is pre-computed based on oracle knowledge before an experimental run is begun. Intuitively, a measure of reliability for a competitor set should have the following properties, based on both the belief- and consistency-values of its hypotheses:

1. With respect to accuracy, reliability should be high if a consistent hypothesis has high belief-value or if a inconsistent hypothesis has a low belief-value, while reliability should be low if a consistent hypothesis has low belief-value or a inconsistent hypothesis has high belief-value;

2. With respect to uncertainty, reliability should be high if one hypothesis in a competitor set has a high belief-value and the rest have low belief-values, while reliability should be low if all the hypotheses have similar belief-values.

A measure for the reliability, RC(S), of a competitor set, S, that captures and adequately combines both of these properties is:

$$RC(s) = \frac{Bc(s)}{\#C(s)} - f_1 * \frac{Bf(s)}{1+f_2 * \#F(s)},$$

where $Bc(s) = \sum_{c \in C(s)} [\, sign(BV(c)) * BV(c)^2 \,]$,

$C(s)$ is the set of consistent hypotheses in s,

$Bf(s) = \sum_{f \in F(s)} [\, sign(BV(f)) * BV(f)^2 \,]$,

$F(s)$ is the set of false hypotheses in s,

$\#C(s)$ is the number of consistent hypotheses in s; note that there may be more than one if false-consistent hypotheses lie near true ones,

$\#F(s)$ is the number of false hypotheses in s,

f_1 determines the relative influence of the consistent hypothesis term versus the false hypothesis term,

f_2 determines the effect of the number of false hypotheses, intended to prevent a large number of relatively poor false hypotheses from swamping a few good true hypotheses,

$BV(h)$ is the belief-value of hypothesis h, and

$sign(v)$ is 1 if $v \geq 0$ and -1 otherwise.

Other measures may also be applicable, but of those we considered this one best captures our intuitive notion of reliability*.

---

*In the original paper the following formulas were defined:

$$RC(S) = 1 - \underset{h \text{ in } S}{avg} \; |CV(h)-BV(h)|$$

which is equivalent, in the case of binary consistency-values, to the correlation of consistency- and belief-values:

$$RC(s) = \underset{h \text{ in } S}{avg} \; [BV(h)*CV(h) + (1-BV(h))*(1-CV(h))]$$

where $BV(h)$ is the belief-value of hypothesis h and $CV(h)$ is the consistency-value of hypothesis h. In experiments with these formulas, it was found that this measure was too sensitive to set size, and the reliability of false data dominated the measure.

Based on this measure of competitor set reliability, we can construct, for instance, a measure of processing effectiveness associated with the current intermediate system state. This measure is the average over the input competitor sets of the difference between the initial reliability and the reliability of the intermediate state. The initial reliability is the average reliability of the competitor sets formed from input hypotheses, using the initial belief- and consistency-values. The reliability of the intermediate state is the average reliability of the competitor sets formed from the input hypotheses, where the reflected-back consistency- and belief-values of hypotheses are used in place of the original ones. A positive processing effectiveness value indicates that some uncertainty and error has been resolved. The larger the value, the more resolution, the more effective is the processing.

IV.2. Measuring the Resolving Power of a KS

We define the instantaneous resolving power of a KS as a change in reliability due to a single KS execution. This change is measured on competitor sets constructed from the KS input hypotheses. Thus, instead of calculating the reflected-back reliability of the entire system state, the procedure is localized only to the subset of the state directly affected by the KS execution. We measure the change in reliablilty as a result of KS execution by measuring before KS processing the reliability of the KS input competitor sets using their belief-values, then measuring after KS processing the reliability of these sets based on values reflected-back from the KS output hypotheses*, and finally taking the difference between the results obtained divided by the original input set reliability. Since this measure is normalized and uses only KS output hypotheses in the reflected-back calculation, it is not effected by other KS actions on this and other blackboard levels.

The resolving power of a KS can now be defined as its average instantaneous resolving power over a series of system executions. This view of KS resolving power does not take into account the global impact of KS execution on the entire system state and on future processing. Rather, it is a local, instantaneous measure of the effects of KS execution. The global effects occur through KS interactions, which we believe should be separated from our measure of the resolving power of a single KS.

---

*These reflected-back values do not include the effect of previously generated hypotheses that are supported by KS input hypotheses.

We understand that reliability is a crude measure of the goodness of solutions or parts of solutions and recognize that this is reflected in our measure of KS resolving power. Experience with the testbed and with actual knowledge-based systems will tell us whether the approach described here is viable.

IV.3. Simulating a KS

We can simulate KSs of any desired power in the testbed through the use of an oracle which knows how to judge the closeness of a hypothesized interpretation to a consistent interpretation. Each node in the system has access to 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 it is used by the testbed oracle to regulate the power of KSs and the scheduler, and to dynamically measure the problem solving performance of the simulated system with respect to the simulation data.

A KS is simulated in two stages: candidate generator and resolver. The candidate generator produces plausible hypotheses for KS output. It can be implemented either by using a real KS whose performance needs to be upgraded or by constructing a simplified KS incorporating relatively simple domain knowledge. In many cases, it is relatively easy to design a KS which provides moderate accuracy. Most of the effort in knowledge engineering is spent in increasing this accuracy to gain superior performance. The next stage, the resolver, uses information provided by the oracle to minimally alter the belief-values of the hypotheses output by the candidate generator to achieve, on the average, a desired KS resolving power. With this alteration process, we can simulate the detection of more sophisticated forms of local consistency than is provided by the candidate generator. This technique allows us to simulate, in a real system, either an upgraded version of a KS (where the old KS is used as candidate generator) or, as we have done in the testbed, a totally new KS (with partially developed knowledge), in order to understand the system performance implications of a proposed change. (See Figure 7 for a diagram of the system with resolver and consistency blackboard.) A similar approach has been used to introduce a parameterized scheduler into the testbed.

A simple resolver we have been experimenting with operates (to increase KS power, for example) by 1) raising the belief-value of each consistent output hypothesis a fixed percentage of the maximum possible increment for each hypothesis, and 2) lowering the belief-value of inconsistent hypotheses the same fixed percentage of the maximum possible decrement for each hypothesis. In the testbed, we have chosen to use this simpler and less precise way of dynamically calibrating the

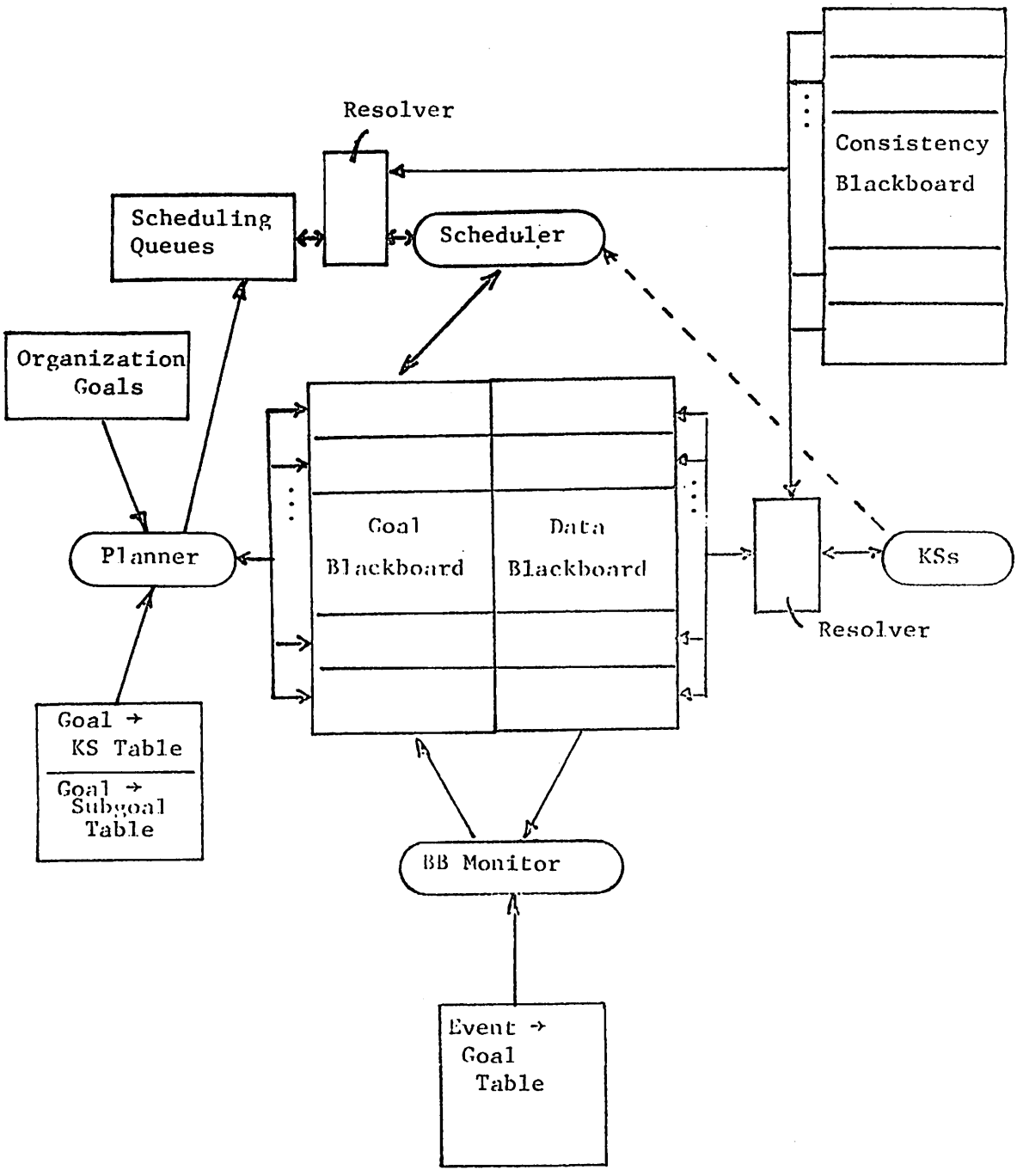


Figure 7: Architecture with resolver and consistency blackboard.

resolving power of KS because of the processing costs of a more complex version and also a precisely calibrated resolver is not really needed. In order to do our empirical evaluation, we feel that a resolver which will be able to generate a low, medium, and high power KSs will be sufficient. In some experimental runs already completed, we have found that by adjustments of percentage change factors in the simple resolver we can produce these three types of KS power.

We believe that our approach to parameterizing the power of a KS, which makes heavy use of the consistency blackboard, is a novel and useful idea. Consistency information has been used in other knowledged-based systems but only for in limited ways. Hearsay-I [Reddy et al. 1973] and Hearsay-II [Erman and Lesser 1978] use this type of information for tracing and debugging, while HWIM [Wolf and Woods 1980] used it to calibrate KSs offline to obtain a mechanism for uniformly combining KS ratings. To our knowledge, the work by Paxton on the SRI speech understanding system [Walker 1978] comes closest to our approach. He used ground consistency information to statistically simulate the output of the low level accoustical processor in the SRI speech system. Our approach differs from his in a number of ways:

1. Our model 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 valid only for front-end processing of input data similar to those used to compute statistics.

2. Our model introduces the idea of a candidate generator stage based on simple, abstracted versions of domain knowledge. This permits further statistically-based simulation to operate within the context of the local consistency relationships of the task domain. Paxton's model does not capture processing behavior at this level of detail.

Even with the flexibilty and detail of our approach, there are some apparent limitations:

1. 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)*. 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 hope, however, the essential behavior of a KS has been captured so that system phenomena are adequately modelled.

IV.4. Simulation of Accuracy in the Scheduler

Reliability measures can also be used in the simulation of a scheduler of a specific accuracy. The task of a scheduler is choosing a KS instantiation for execution. A KS instantiation is a KS-stimulus pair, where the stimulus is the set of hypotheses which caused the KS to be considered for scheduling. The scheduler evaluates alternative instantiations according to its knowledge of the characteristics of the KSs, the stimuli, and the current state of processing. The effects of future processing are not factored into this model of scheduling; we take an instantaneous view of scheduling decisions. Because of this, we are unable to model scheduling algorithms such as the "shortfall density scoring method" [Wolf and Woods 1980] which use information about future processing. We hope to develop a formulation that includes this type of information.

A good scheduler chooses for execution the KS instantiation that will most improve the reliability of the current system state. The accuracy of a single scheduling decision is defined relative to the performance of an optimum scheduler, which uses accurate information about the resolving power of the KSs and the reliability of the KS stimuli and system state. The accuracy of a scheduler is the average of the accuracy of many scheduling decisions.

We view the optimum scheduling process in two steps:

---

*In order to capture more closely in our model the notion of local consistency, we can, through the consistency blackboard, introduce false hypotheses where falseness cannot be detected by KSs operating at that level, i.e., the KS candidate generator and resolver will judge the consistency of these false hypotheses in the same way as it does true hypotheses. We call these correlated false hypotheses. In this way, we can more effectively model the performance of a real KS.

1. For each KS instantiation on the scheduling queue, make accurate predictions concerning its instantaneous resolving power. These predictions involve determining the reliability of its output hypotheses based on the reliability of its input hypotheses (using the oracle and knowledge of the resolving power of the KS).

2. Make accurate predictions as to the global system state which would result from scheduling each instantiation given the predictions of step 1. These predictions will determine optimum ratings for the instantiations and result in an optimum schedule.

We feel it would be an error to model scheduling only as a function of the consistency-value of stimulus hypotheses. Real schedulers do not have access to the consistency-values of hypotheses, but only infer truth from belief-values and processing history. The point is that two instantiations of the same KS, whose stimulus hypotheses have equivalent characteristics (same belief-value, level of abstraction, database region, processing history, etc.) except for their consistency-values would be rated the same by even the best scheduler. Additionally, in order to determine the rating of a KS instantiation, real schedulers consider other factors, besides the characteristics of the stimulus hypotheses. For example, schedulers take into account such factors as the balance between depth-first versus breadth-first processing or between executing KSs that work in areas with rich processing history versus executing KSs that work where little processing has been done. These additional considerations are, in fact, heuristics which attempt to capture the concept of improvement in the reliability of the system state and to be conservative in case of scheduler error. Thus, in our view, a scheduler should be characterized in terms of its ability to estimate the improvement in system state reliability, rather than its ability to detect the truthfulness of the instantiation's stimulus hypotheses.

An approach to modelling the scheduler (which we do not currently use) is to obtain statistically accurate ratings for the instantiations, based on the optimum schedule, and then choose for execution an instantiation from within the ordering which results. The position in the ordering of the chosen instantiation depends on the desired accuracy of the scheduler being modelled; the closer to the top of the order, the more accurate the scheduler. In the testbed we have been exploring a simpler and less expensive scheme for simulating schedules of more or less power.

The currently implemented scheduler utilizes both goal priorities and data reliability to determine the rating of a KS instantiation. The goal priority component of the scheduler rating is a measure, from a system-wide perspective, of the importance of the potential KS output while the data reliability component is a measure, from a local

perspective based on the KS inputs, of the reliability of the potential KS output. The scheduler evaluates a KS instantiation by first considering the rating of the goal(s) that the instantiation could potentially satisfy; working on more important goals is preferable. Next, it uses the quality of the data the KS would produce, the goodness of the KS, and the extent to which the KS would increase its goals' degree of satisfaction. Since the execution of a KS is relatively inexpensive, the hypotheses that it would create if invoked (and their beliefs) may be calculated before the actual invocation of the KS. This permits the scheduler to have a more accurate picture of the (local) effects of invoking the KS. Thus, it may determine the improvement in reliability by determining the local effect the created hypotheses would have. This also permits determination of what portion of its goal(s) will be satisfied. A greater increase in a goal's degree of satisfaction makes the KS more desirable. The goodness of a KS is a utility function that incorporates its power and its processing cost. More powerful, less expensive KSs are given preference.

A more powerful scheduler may be simulated by invoking the oracle to obtain the truth value of the created data. The degree to which it uses this truth value (as opposed to just the belief) determines the additional power of the scheduler. Since our current scheduler is using the created hypotheses of a KS to determine its rating, it may be considered to have a perfect model of the KS. It may be desirable to simulate a less powerful scheduler. This is done by adding Gaussian noise to the rating the scheduler assigns to the KS so that the idealized ordering of potential KS instantiations is degraded. The amplitude (and standard deviation) of this noise determines the amount of reduction in scheduler power.

IV.5. Extensions to a Multi-Node System

There are a number of extensions to these measurements and simulation concepts that are needed for a multi-node problem-solving architecture. Most important is a way of measuring the intermediate state of processing of the entire system. It is inappropriate for this measurement to be just an average of the local node states since this does not take into account the relationships among areas of processing interest of local nodes. For example, the average of the state measures of local nodes could have a reasonable value but from a system-wide state perspective be poor in situations where each of the nodes have a high state measure in overlapping interest areas and have a very low state measure in non-overlapping areas.

A more accurate measurement of system state can be introduced by having a system-wide notion of competitor sets. Node states having overlapping processing interest areas would be indicated by having those nodes share some of the same global competitor sets. A system-wide

reflecting-back process is introduced in which the reflection process is not stopped at the sensor level hypotheses of the local node but is continued for one additional level. This additional level groups sensor level hypotheses from different nodes which relate to the same processing area into the same global competitor set. These global competitor sets, which are based on the sensory level hypotheses that will be generated during a run, can be pre-computed because we are dealing with a simulation not a real system*. From an implementation perspective, we associate with each sensor level hypothesis a hidden attribute which indicates its global competitor set number and its entry number in the set. Two nodes receiving the same sensor hypothesis would have two different local names for this hypotheses but each local hypothesis would have the same hidden attribute.

This concept of global competitor set also has the advantage of making it straightforward to measure the effect of transmitted hypotheses on a local node state. This is accomplished by associating with each transmitted hypothesis a hidden support structure attribute based on the global competitior set attributes of its supporting sensory level hypotheses. When a hypothesis is to be transmitted, the testbed kernel will automatically trace through the hypothesis support structure down to its sensor hypothesis support in order to calculate the hidden attribute.

The effect of a transmitted hypothesis on a receiving node's state can then be calculated even though the transmitted hypotheses may not be able to be linked into the existing hypothesis structure of the receiving node. This calculation is accomplished by using the hidden structure attribute to directly reflect the belief of the transmitted hypothesis into the appropriate entries in the node's local competitor set structure. The local node competitor set structure is a subset of the global competitor set structure based on the pre-defined area of interest of the node.

Given this ability to evaluate the effect of a transmitted hypothesis on the state of the receiving node, we can use the same methodology to simulate send KSs as we used to simulate schedulers. The more powerful a send KS, the more accurate is its evaluation of the potential effect on the receiving node of each of the candidate hypotheses that can be transmitted. A "real" send KS would use among other things meta-information about the state of processing in the

---

*We compute global competitor sets by first making a separate competitor set for each sensor signal on the consistency blackboard. For each of these signals we include in its associated competitor set those errorful signals that can be confused with it. Errorful signals that are not included in a competitor as a result of this process are used as seeds for additional competitor sets.

system and specific nodes, and knowledge about the relationship of the consistency of its hypotheses to their belief in order to evaluate the potential impact of a transmission. Instead, we can simulate the use of this additional knowledge by using the oracle to evaluate the reliability change in the state of node processing if a candidate hypotheses was transmitted. The ratings generated by this process then indicates an optimal priority ordering for transmission of hypothesis to specific nodes. A more or less powerful send KS can be simulated by appropriately modifying the priority transmission order specified by an optimal transmission policy, as was done with our simulation of a scheduler.

We can also extend our measurement concepts to define a measure for the power of a node. We can define this measure by reflecting-back a node's output data (transmitted hypotheses) to its input data (sensor data and received hypotheses). In this way, we can measure the effective processing that has resulted from the combined activity of a node's KSs.

Appendix B contains a description of the testbed implementation of the measures described in this section.

V. Facilities for Experimentation

The testbed kernel is surrounded by a number of other subsystems (see Figure 8) to facilitate experimentation by making it easy to vary the parameters of an experiment and to analyze the results of an experiment.

The Front-End subsystem is responsible for initializing the testbed with a particular configuration of nodes and sensors and a particular scenario of patterns and vehicles moving in the environment. This initialization process involves reading an input file to set up local and global data structures, creating the consistency blackboard, and placing at appropriate times the raw signal data on the data blackboards of appropriate nodes. Each configuration-scenario combination is stored compactly in an input file. A sample input file, with detailed comments, is presented in Appendix E. This input file can be either constructed by hand or through an interactive, menu driven graphics facility. The advantages of the graphics facility is that it permits a user to see and manipulate the 2-dimensional layout of the configuration-scenario that is desired for experimentation (see Figure 9).

The input file contains all the input data for the testbed, and consists of system data, grammar definition, node definitions, sensor definitions, communication reliability data, and environment data.

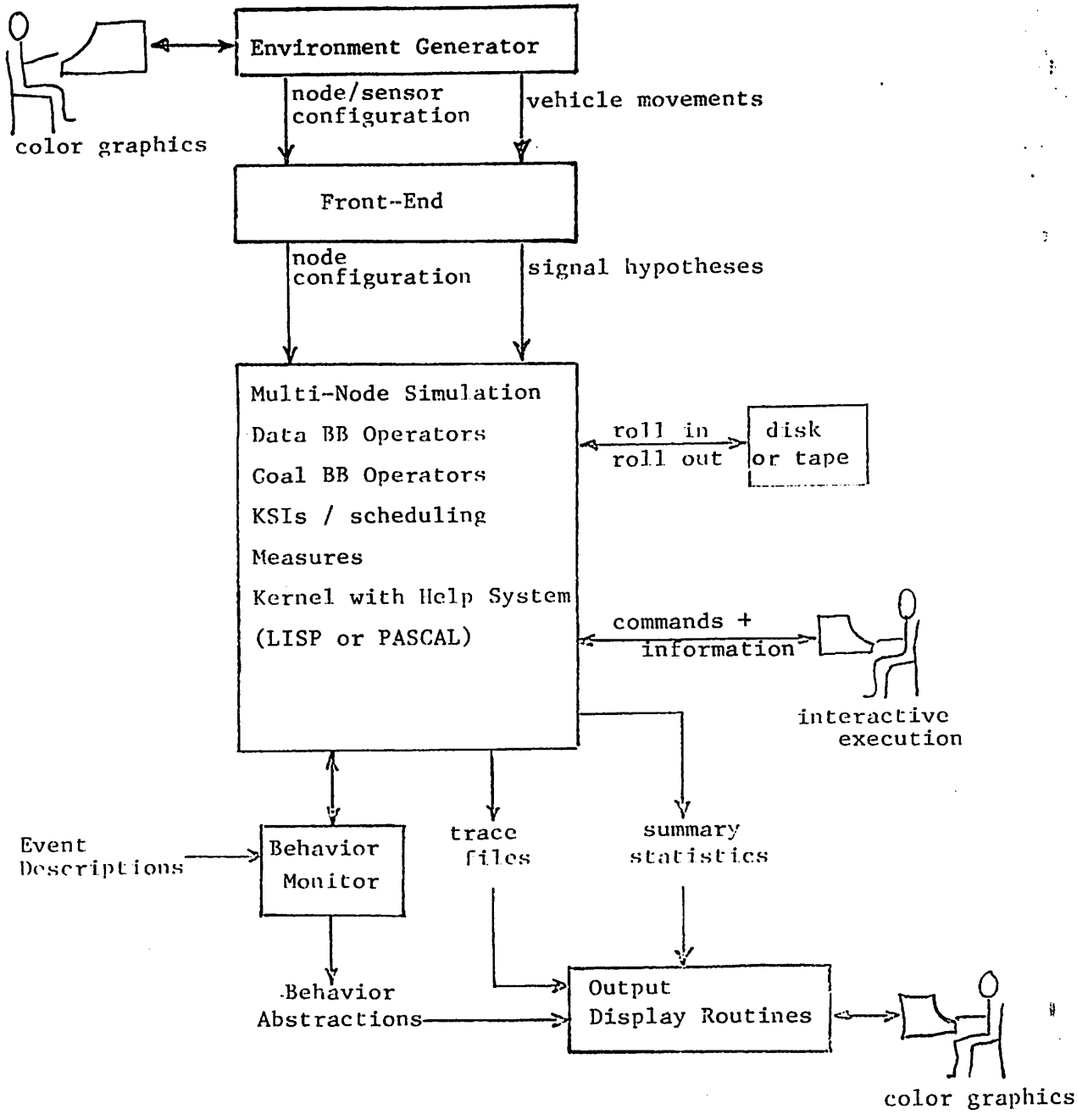


Figure 8: Diagram of testbed structure and facilities

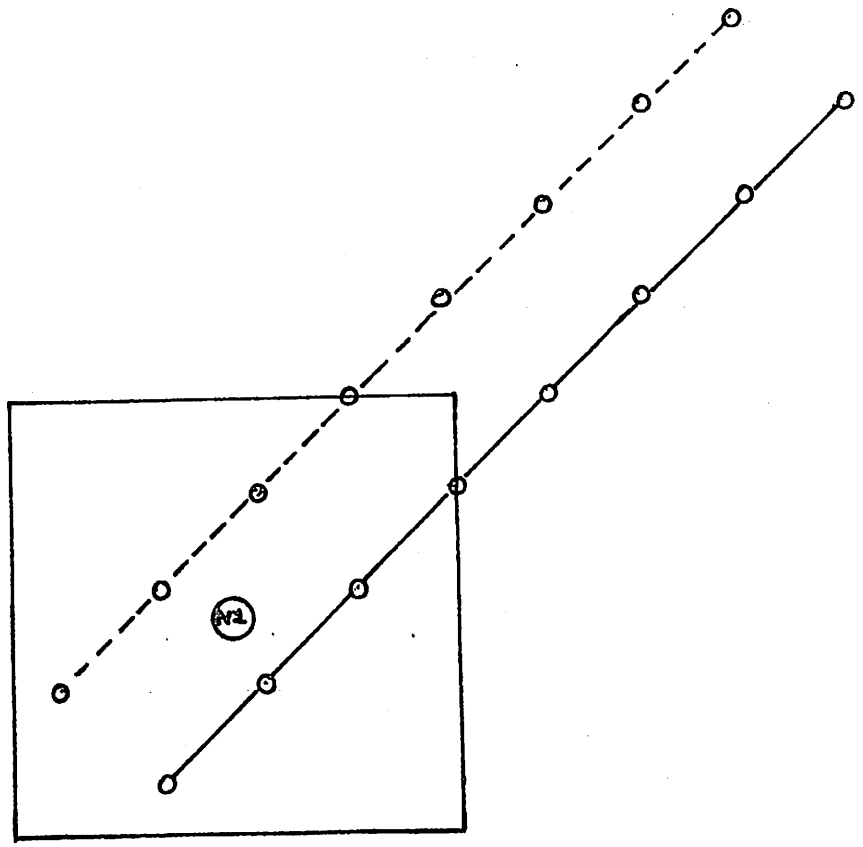


Figure 9: Sample 2-dimensional layout of scenario

System data denotes basic parameters of the simulated vehicle monitoring system: location and time ranges, a seed for random number generation, maximum velocity and maximum velocity change, and some global tuning parameters. Grammar definition specifies the relationship among patterns, vehicles, groups, and signals used by KS candidate generators. Node definitions contains information concerning nodes: their locations, interest areas, KSs, etc. Sensor definitions contains information concerning sensors: their locations, ranges, accuracies, etc. Communication reliability data specifies the reliability of communication links. Consistency data supplies the oracle with information indicating what a perfect knowledge source would perceive as consistent. And, environmental data denotes the actual environment for the vehicle monitoring system: locations of patterns and vehicles at various time frames excluding patterns, vehicles, groups, and signals to be missed and including false patterns, vehicles, groups, and signals.

By varying the 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 local resolving power of KS provides a powerful tool for varying the knowledge expertise in the simulated system. The Front-End, in the 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. Front-End processing is also parametized 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.

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, and an interactive, menu-driven debugging facility, an event-monitoring facility, and a color-graphics display facility.

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. An annotated listing of a detailed trace of one KS execution is presented in Appendix A.

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. Appendix C presents an annotated listing of this output. As we

get more experience with the use of the testbed, we expect these summary measures to undergo significant modifications and additions.

The menu-driven interactive debugging facility permits a user, during and after a run, to examine details of system performance that were either not chosen to be displayed in the trace or are difficult to understand from the linear, chronological flow of the trace. During the execution of the testbed very few of the intermediate data structures are destroyed. This record keeping is feasible because of the virtual memory machine architecture (DEC VAX 11/780) underlying our implementation. Thus a user can, through a menu-driven facility, go back to look at the detailed characteristics of the system at the end of a run or when the system has crashed, and even calculate new summary statistic measures on the performance of this system. We expect this ability for interactively queryng the system at the end of a run will eliminate the need for the recording of a detailed trace of system activity.

In addition to these three fairly common analysis tools, we feel that there is need for tools that permit a more dynamic and high-level view of the distributed and asynchronious activity of the simulated nodes. An event monitoring facility, which has not yet been fully implemented, will permit a user to define and gather statistics on such user-defined events as the average time it takes for a node to receive a message (i.e., hypothesis) and incorporate the received information into a message to be transmitted to another node. A description of this facility is contained in companion paper [Bates, Wileden, and Lesser 1981] in this volume.

Another facility which is currently operational in a limited form is a color-graphics output facility. The current output display provides dynamic visual representations of the distribution of hypotheses in the x-y space of the Distributed Sensor Network during a simulation. Presently, two display formats exist.

The first format shows location and track hypotheses as symbols and paths connecting symbols, respectively, in the physical x-y space (see Figure 10). The level, node, belief, and event-class 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 second format shows the reliability of data at any position in x-y space for each node (not shown). This permits the user to quickly determine the success of the processing of a node in any area. This display also shows the areas of processing interest for each node.

The hypotheses displayed in either of the display formats may be selected according to the characteristics of any of their attributes. For example, it is possible to display only those hypotheses above some

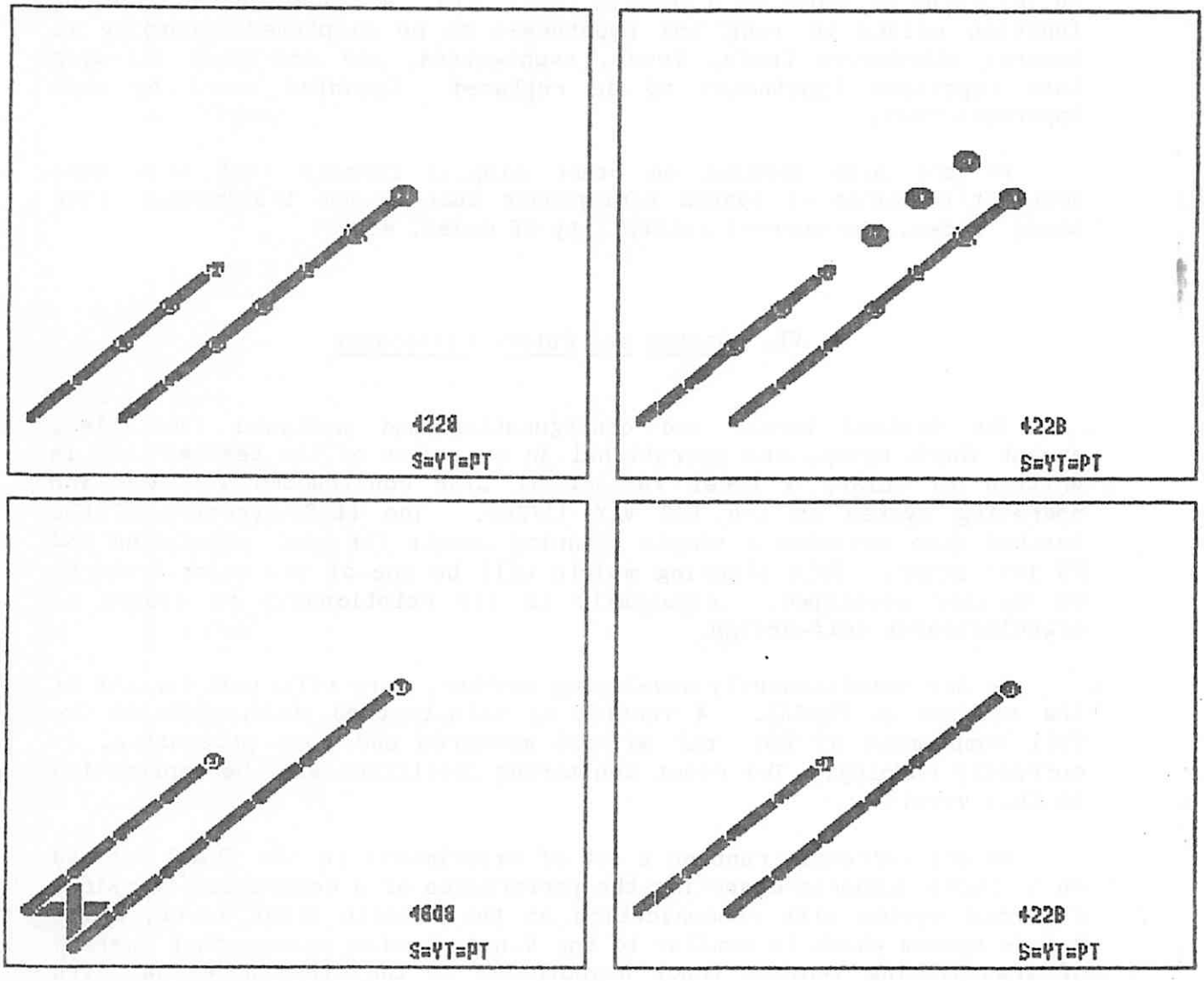


Figure 10: Snapshot of dynamic display of location and track hypotheses

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, event-class, 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.

VI. Status and Future Directions

The testbed kernel and configuration and analysis facilities, except where noted, are operational in a version of the testbed that is written in CLISP, a local variant of LISP constructed for the VMS operating system on the DEC VAX 11/780. The CLISP version of the testbed also contains a simple planning module for goal processing and KS invocation. This planning module will be one of the major areas to be further developed, especially in its relationship to issues of organizational self-design.

We are simultaneously developing another, more efficient version of the testbed in PASCAL. A version of this testbed which contains the full complement of KSs, but without measures and goal processing, is currently running. The event monitoring facilities will be implemented in this version.

We are currently running a set of experiments in the CLISP testbed on a simple scenario comparing the performance of a centralized version, a 4-node system with communication at the vehicle track level, and a 5-node system which is similar to the 4-node system except that instead of transmitting vehicle track huypotheses to the other nodes, a fifth node without sensors is inserted which receives all the communications. These experiments have already been very instructive in helping us in debugging and modifying our KS measures, scheduler, and planner.

VII. Summary

We have described a high level simulation testbed for evaluating alternative cooperative problem-solving strategies applied to a flexibly parameterized task domain. The task is monitoring of moving vehicles being sensed by a set of acoustic sensors with overlapping ranges. Both vehicle and sensor characteristics are variable to permit control of the spatial distribution of ambiguity and error in the task input

data. We have developed a parameterized, distributed interpretation system based on the Hearsay-II architecture with the addition of mechanisms for goal-directed control. Node configurations and communication channel characteristics can be independently varied in this simulated system. The monitoring task KSs and KS scheduler used in the nodes of the testbed system have resolution capabilities which can be varied independently. This permits control of the distribution of problem-solving expertise across all the nodes in the system at a detailed level.

The testbed provides a means for exploring the importance and interrelationships of the following factors in cooperative problem-solving:

1. node-node and node-sensor configurations,
2. mixes of data- and goal-directed control in the system,
3. distributions of uncertainty and error in the input data,
4. distributions of problem-solving capability in the system,
5. types of communication policies used.
6. communication channel characteristics,

The testbed is used by specifying each of these factors, independently, for each of a set of test cases and comparing the results of testbed simulation of each case. The multiple dimensions of independent control and the detailed level of simulation in the testbed provide what we feel is a very useful vehicle for exploring high-level issues in cooperative distributed problem-solving.

Acknowledgements

We would like to acknowledge the contribution to the design and construction of the testbed by Peter Bates, Robert Markey, and Sheryl Franklin.

VIII. References

Bates, P.C. (1981). "Event Definition Language: An Aid to Monitoring and Debugging Complex Software Systems", Technical Report 81-17, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts, June.

Bates, P.C., J.C. Wileden, and V.R. Lesser (1981). "A Language to Support Debugging in Distributed Systems", Technical Report 81-7, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts, February.

Corkill, D.D (1980). "An Organizational Approach to Planning Distributed Problem-Solving Systems", Technical Report 80-13, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts, May.

Corkill, D.D., and V.R. Lesser (1981). "A Goal-Directed Hearsay-II Architecture: Unifying Data-Directed and Goal-Directed Control", Technical Report 80-15, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachussets, June.

Erman, L.D., F. Hayes-Roth, V.R. Lesser and D.R. Reddy (1980). "The Hearsay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty", Computing Surveys, Vol.12, No.2, pp.213-253, June.

Erman, L.D., and V.R. Lesser (1978). "System Engineering Techniques for Artificial Intelligence Systems", In A.R. Hanson and E.M. Riseman (Editors), Computer Vision Systems, pp.37-45, Academic Press, New York, New York.

Fox, M.S. (1979). "Organization Structuring: Designing Large Complex Software Programs", Technical Report, Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvannia.

Fox, M.S. (1981). "An Organizational View of Distributed Systems", IEEE Transactions on Systems, Man and Cybernetics, Vol.SMC-11, No.1, pp.70-80, January.

Green, P. (1979). Private communications. Lincoln Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Kahn, R.E., and S.A. Gronemeyer, J. Burchfiel, and R.C. Kunzelman (1978). "Advances in Packet Radio Technology", Proceedings of the IEEE, Vol.66, No.11, pp.1468-1496, November.

Lacoss, R., and R. Walton (1978). "Strawman Design of a DSN to Detect and Track Low Flying Aircraft", Proceedings of the Distributed Sensor Nets Workshop, Carnegie-Mellon University, Pittsburgh, Pennsylvannia, pp.41-52, December.

Lesser, V.R., and D.D. Corkill (1979). "The Application of Artificial Intelligence Techniques to Cooperative Distributed Processing", Proceedings of the Sixth International Joint Conference on Artificial Intelligence, 1979, pp.537-540, Tokyo, Japan, August.

Lesser, V.R., and D.D. Corkill (1981). "Functionally Accurate, Cooperative Distributed Systems", IEEE Transactions on Systems, Man and Cybernetics, Vol.SMC-11, No.1, pp.81-96, January.

Lesser, V.R. and L.D. Erman (1980). "Distributed Interpretation: A Model and an Experiment", IEEE Transactions on Computers, Special Issue on Distributed Processing Systems, Vol.C-29, No.12, pp.1144-1162, December.

Lesser, V.R., J. Pavlin, and S. Reed (1980). "Quantifying and Simulating the Behavior of Knowledge-Based Interpretation Systems", Proceedings of the First Annual National Conference on Artificial Intelligence, Stanford University, August 18-21, pp.111-115.

Mann, W.C. (1979). "Dialogue Games", Information Science Institute of University of Southern California, Marina del Ray, California, Technical Report.

Reddy, D.R., L.D. Erman, and R.B. Neely, (1973). "A Model and a System for Machine Recognition of Speech", IEEE Transactions on Audio and Electro-acoustics, AU-21, pp.229-238.

Smith, R.G. (1980). "A Framework for Problem Solving in a Distributed Processing Environment", IEEE Transactions on Computers, Special Issue on Distributed Processing Systems, Vol.C-29, No.12, pp.1104-1113.

Smith, R.G., and R. Davis (1981). "Frameworks for Cooperation in Distributed Problem-Solving", IEEE Transactions on Systems, Man and Cybernetics, Vol.SMC-11, No.1, pp.61-69, January.

Walker, D.E. [Ed.], (1978). Understanding Spoken Language, Elsevier, North-Holland, New York.

Wesson, R.B., F. Hayes-Roth, J.W. Burge, C. Stacy, and C.A. Sunshine (1979). "Network Structures for Distributed Situation Assessment", IEEE Transactions on System, Man and Cybernetics, Vol.SMC-11, No.1, pp.5-23, January.

Wolf, J.J. and W.A. Woods (1980). "The HWIM Speech Understanding System", in W.A. Lea [ed.], Trends in Speech Recognition, Chapter 14, Prentice-Hall, Englewood Cliffs, N.J., 1980.

Appendix A. A KS Execution Trace with Descriptions

This trace is represented in figure 11. Thick lines correspond to the structure present on data blackboard before ksi32 executes. The comment lines are indented and enclosed in {}. The character * indicates that the event is dealing with true data. Similarly, + indicates consistent and - indicates false data. The fields in KSI, HYP and GOAL line are as follows:

KSI: (1) - name of KS instantiation (KSI)
(2) - ks name (type:input-level:output-level)
(3) - stimulating goals
(4) - stimulating hypotheses
(5) - ksi rating

HYP: (1) - hypothesis name
(2) - level
(3) - event class
(4) - begin and end time
(5) - belief

GOAL:(1) - goal name
(2) - level
(3) - event class list
(4) - begin and end time
(5) - goal rating

```
-------------------- Executing Node 1 -- Cycle  7 --------------------

* INVOKED -------> ksi32 s:g1:v1 (g602) (h348) {3160}

        {KSI:      (1)     (2)     (3)     (4)     (5)}

        {The KS that synthesizes v1 hypotheses from g1 hypotheses is
         executing. It attempts to satisfy goal g602. It is a goal to
         create a vehicle of event class 1 at time 2.}

* CREATED -------> h355 v1 1 2..2 {2988}

        {HYP:      (1)  (2)(3)(4)  (5)  }

* INSERTED ------> h355 v1 1 2..2 {2988}

* ---LINKED------> h355 v1 1 2..2 (g602)

        {h355 is linked as a satisfying hyp to the existing goal g602}

* CREATED -------> g624 p1 (1 3) 2..2 {-2988}
```

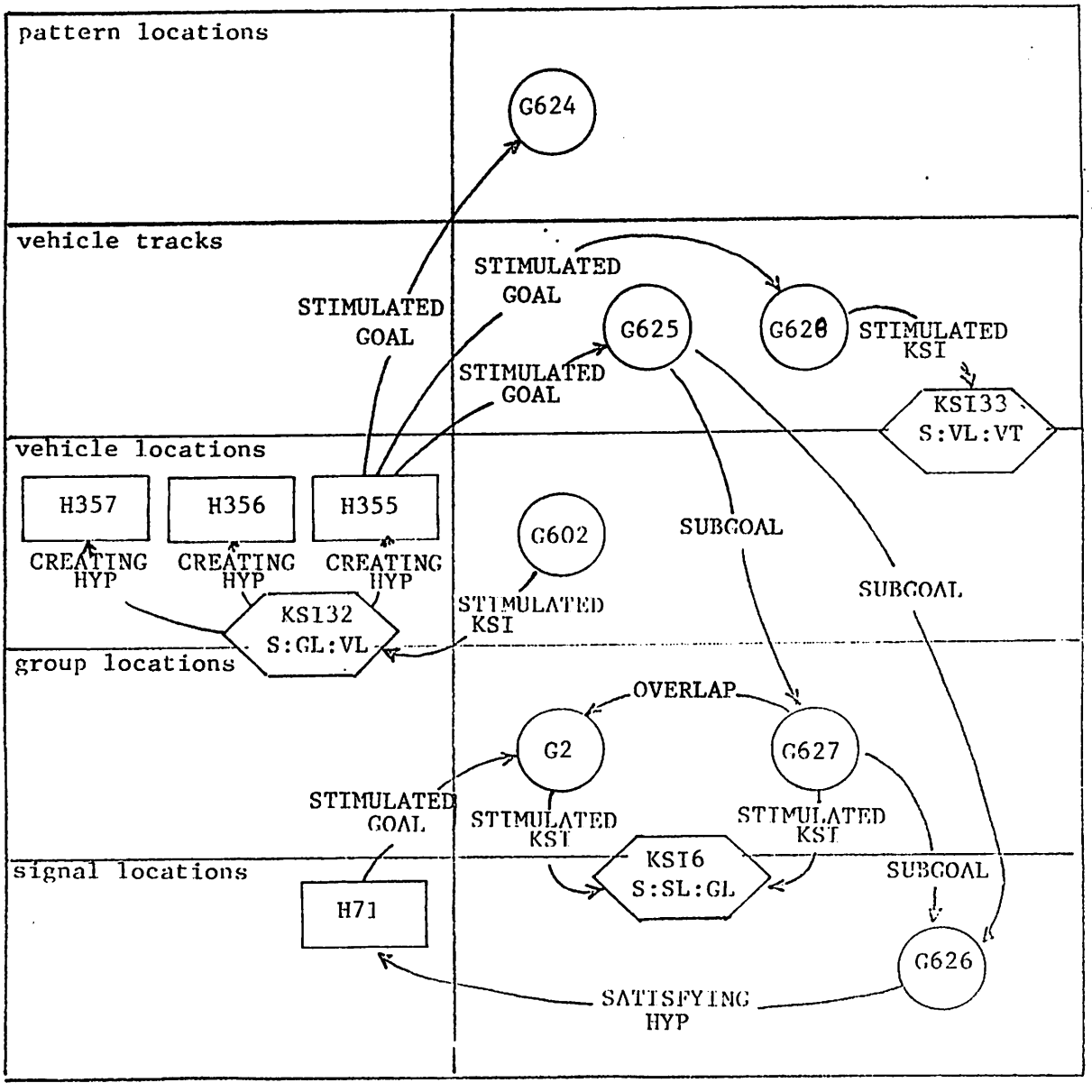


Figure 12: Blackboards for execution trace.

{h355 causes the creation of a number of goals. g624 is a goal of two patterns, of classes 1 and 3 respectively at time 2. There are two pattern classes because vehicle of class 1 can belong to both patterns, as defined by the grammar.}

* CREATED -------> g625 vt (1) 1..1 {2988}

{Goal g625 is a vehicle track (vehicle class 1) that includes time 1 (extending h355 in backward direction).}

* SUBGOALED -----> g625

---CREATED-----> g626 sl (2 6 10 14 18) 1..1 {2988}

{As a result of subgoaling g625, g626 is created. In this example, the planner is instructed to subgoal only at the vehicle track and pattern track levels. The level of subgoaling is controlled because only expectations from reliable hypotheses should be used to drive further processing.}

* ---LINKED------> g626 sl 2 1..1 (h71)

{g626 is linked to an existing satisfying hypothesis, h71.}

---CREATED ----> g627 gl (2 6) 1..1 {2988}

{Since there is a hypothsis which satisfies g626, the next higher level goal with the same characteristics as g626 is created on gl level, g627. For efficiency reasons, we don't create the intermediate subgoals until there is data that makes the intermediate goal feasible.}

* ---LINKED -----> g2 gl to g627 gl

{Goal g2 is found, which overlaps g627. As a result, an overlap link is created between g2 and g627.}

---LINKED -----> g627 gl {2988} between g625 vt 2988 and g626 sl 2988

{g627 is linked as a subgoal of g625. g626 is linked as a subgoal of g627.}

* ---LINKED -----> ksi6 s:sl:gl (g2) to g627 gl

{ksi6, which was stimulated by g2, is found to satisfy g627. It is now linked as stimulated by g627.}

* ---RERATED ----> ksi6 s:sl:gl (g2 g627) (h77) {1194 to 2613}

{As a result of connecting ksi6 to g627, ksi6 is rerated

from 1194 to 2613.}

* CREATED -------> g628 vt (1) 3..3 {2988}

{Goal g628 is also created from h355. It is a goal for a vehicle track which includes time 3 (extending h355 in forward direction). }

* INSTANTIATED --> ksi33 s:vl:vt (g628) (h355) {4624}

{This time a KSI gets instantiated, because the precondition part of the KS found data for its execution. After this point, the sequence is very similar to the one explained. A sophisticated planner would in this case probably avoid subgoaling on g628 until it runs ksi33 and finds out whether it can satisfy this goal. However, the planner may choose to subgoal without waiting for this instantiation, because it is low rated due to poor inputs. The action of subgoaling in this case would hopefully lead to better input data for the instantiation, or another way of deriving the goal.}

Appendix B. Formulas for Performance Measures

This appendix contains

- o KSI Rating Scheme
- o Reliability of a Competitor Set
- o System State Measure
- o Global KSI Power
- o Local KSI Power

KSI Rating Scheme

Rating(KSI) =

$$N(s, \underset{o \in O(KSI)}{MAX} [w_1 * data\_part(KSI,o) + (1-w_1) * goal\_part(KSI,o)])$$

where $data\_part(KSI,o) = Goodness(KSI) * [(1-w_2) * BV(o) + w_2 * CV(o)]$

$$goal\_part(KSI,o) = \underset{g \in G(KSI)}{MAX} Goal\_Rating(g)$$

w_1 is a constant in [0,1] that determines the relative influence of the reliability of the data versus that of the goals,

Goodness(KSI) indicates the efficiency (cost versus power) of the KS executing the KSI,

w_2 is a constant in [0,1] that determines the amount of oracle used in the rating function,

O(KSI) is the set of output hypotheses of the KSI,

BV(h) is the belief value of output hypothesis h,

CV(h) is 1 if output hypothesis h is consistent and 0 otherwise,

$N(s,m) = GN(s) * (1-Measure(KSI)^2)$,

GN(s) is a random variable in [-1,1] drawn from a gaussian} distribution with mean 0 and standard deviation s,

G(KSI) is the set of goals satisfied by the KSI, and

Goal_Rating(g) is the rating assigned by the goal planning mechanism to goal g.

Reliability of a Competitor Set

The measure for the reliability, RC(s), of a competitor set, s, is:

$$RC(s) = \frac{Bc(s)}{\#C(s)} - f_1 * \frac{Bf(s)}{1+f_2 * \#F(s)},$$

where $Bc(s) = \sum_{c \in C(s)} [\, sign(BV(c)) * BV(c)^2 \,]$,

C(s) is the set of consistent hypotheses in s,

$Bf(s) = \sum_{f \in F(s)} [\, sign(BV(f)) * BV(f)^2 \,]$,

F(s) is the set of false hypotheses in s,

#C(s) is the number of consistent hypotheses in s; note that there may be more than one if false-consistent hypotheses lie near true ones,

#F(s) is the number of false hypotheses in s,

f_1 determines the relative influence of the consistent hypothesis term versus the false hypothesis term,

f_2 determines the effect of the number of false hypotheses, intended to prevent a large number of relatively poor false hypotheses from swamping a few good true hypotheses,

BV(h) is the belief-value of hypothesis h, and

sign(v) is 1 if $v \geq 0$ and -1 otherwise.

System State Measure

$$\text{System\_State} = \underset{s\ \text{CSETS}}{\text{AVG}}\ \text{RRC}(s)$$

where $$\text{RRC}(s) = \frac{Bc'(s)}{\#C(s)} - f_1 * \frac{Bf'(s)}{1+f_2 * \#F(s)},$$

where $$Bc'(s) = \underset{c \in C(s)}{\text{SUM}}\ [\ \text{sign}(BV'(c)) * BV'(c)^2\ ],$$

$C(s)$ is the set of consistent hypotheses in s,

$$Bf'(s) = \underset{f \in F(s)}{\text{SUM}}\ [\ \text{sign}(BV'(f)) * BV'(f)^2\ ],$$

$F(s)$ is the set of false hypotheses in s,

$\#C(s)$ is the number of consistent hypotheses in s; note that there may be more than one if false-consistent hypotheses lie near true ones,

$\#F(s)$ is the number of false hypotheses in s,

f_1 determines the relative influence of the consistent hypothesis term versus the false hypothesis term,

f_2 determines the effect of the number of false hypotheses, intended to prevent a large number of relatively poor false hypotheses from swamping a few good true hypotheses,

$R(h) = RR(h)$ if h is consistent and $-RR(h)$ if h is inconsistent

$$RR(h) = \text{AVG}(\ \underset{x \in IS(h)}{RR(x)}\ ,\ R(h)\ ),$$

sign(v) is 1 if $v \geq 0$ and -1 otherwise, and

CSETS is the set of all competitor sets.

Global KSI Power

The global power of a KSI is the change in node state resulting from executing the KSI.

$$\text{Global\_Power(KSI)} = \text{Node\_State}^{\text{before\_KSI}(n)} - \text{Node\_State}^{\text{after\_KSI}(n)}$$

where $\text{Node\_State}^{t}(n) = \underset{c \in \text{Csets}(n)}{\text{AVG}} \text{RRCn}(c)$ at time t,

RRCn(c) = is the reflected reliability of a competitor set, c, restricting references only to hypotheses in node n and using the following slightly modified reflected reliability of a hypothesiss:

$$\text{RR'}(h) = \begin{cases} \text{RR}(h) & \text{if there exists a link between h and at least one hypothesis in node n,} \\ 0 & \text{if there are no links between h and some hypothesis in node n and the hypothesis h is consistent,} \\ 1 & \text{if there are no links between h and some hypothesis in node n and the hypothesis h is inconsistent.} \end{cases}$$

Csets(n) is the set of global competitor sets which are in the area of processing interest of node n.

Local KSI Power

$$\text{Local\_Power(KSI)} = \underset{s\in S(KSI)}{\text{AVG}} \left(\underset{o\in O(KSI,s)}{\text{AVG}} \left(\underset{i\in IS(o)}{\text{AVG}} \left(LRRC(i)-RC(i) \right) \right) \right)$$

where S(KSI) is the set of stimulus hypotheses for the KSI,

O(KSI,s) is the set of output hypotheses of the KSI which are supported by s,

IS(o) is the set of input hypotheses to KSI which immediately support o,

LRRC(i) is the reflected reliability of the competing set containing hypothesis i, but reflecting back from only the hypotheses in O(KSI,s), s in S(KSI), and

RC(i) is the reliability of the competitor set containing hypothesis i.

$$\text{Local\_Receive\_Power(RKSI)} = \underset{r\in R(RKSI)}{\text{AVG}} \left(\underset{s\in CS(r)}{\text{AVG}} \left(RR(r)-R(s) \right) \right)$$

where RKSI is a "receive" KSI

R(RKSI) is the set of hypotheses received and inserted by RKSI

CS(r) is the competitor set hypotheses with hidden links to the sensor supports of hypothesis r in the sending node

$$\text{Local\_Send\_Power(SKSI,h)} = \underset{n\in N(SKSI)}{\text{AVG}} \left(\underset{g\in G(SKSI,n)}{\text{AVG}} \left(RR(h)-R(g) \right) \right)$$

where SKSI is a "send" KSI

N(SKSI) is the set of nodes to which a hypothesis is being sent

G(SKSI,n) is the set of unsupported hypotheses at node n pointed to by the competitor set hypotheses support

RR(h) is the reflected reliability of hypothesis h

R(g) is the reliability of hypothesis g

Appendix C. Sample Summary Statistics

The following is a sample of the summary statistics for a run:

```
Environment:  b2224nj
Run  Started:  2-JUL-1982 04:37:34.61
Last Stopped:  2-JUL-1982 04:57:59.17
Kernel Date:   1-JUL-1982 12:20:59.17

**************************************************************************
env file modified 17-may-1982
complete grammar, no noise, false and missing signals
single vehicle pattern
four node, sensor mask=1, power=0, total belief=500
goal weight=8000, goal send/hyp reply/ no subgoaling
**************************************************************************

                  ==================
                  Summary Statistics
                  ==================

                  System Cycles: 41
```

Configuration

```
Nodes:    4
Sensors:  4

Grammar:  grammar-1x

Patterns:  1
Vehicles:  1
Groups:    1
Signals:   1

Scheduler Power:    10000
Scheduler Belief Weight: 10000

Initial System State: 1555
Final System State:   1429
```

Input Environment

Level	Created	True	Consistent
pt	1	1	0
pl	8	8	0
vt	1	1	0
vl	8	8	0
gt	1	1	0
gl	8	8	0
st	1	1	0
sl	8	8	0

Sensor Output

Sensor	True	Consistent	False
1	50%	0%	50%
2	0%	0%	100%
3	100%	0%	0%
4	50%	0%	50%

Communication Topology

Topology Level	Nodes	Up-level Paths	Same-level Paths	Down-level Paths
1	(1 2 3 4)		vt 3.0	

Resolver Power

Topological Level	pt	pl	vt	vl	gt	gl	st	sl
1	0	0	0	0	0	0	0	0

System Summary Statistics

Nodes: (1 2 3 4)

			No. True	Cons.	False	
Initial Node State:	288	Hyps Created:	161	57%	0%	42%
Final Node State:	288	Hyps Transmitted:	46	56%	0%	43%
Competitor Sets:	16	Hyps Received:	46	56%	0%	43%
GD Goals Created:	291	DD Goals Created:	313	58%	0%	41%
		DD Goals Transmitted:	297	61%	0%	38%
		DD Goals Received:	291	62%	0%	37%
Global KS Power:	0	KSs Instantiated:	167	58%	0%	41%
Local KS Power:	0	KSs Invoked:	160	58%	0%	41%

	Created Hyps				Transmitted Hyps				Received Hyps			
Level	No.	T	C	F	No.	T	C	F	No.	T	C	F
pt	54	59%	0%	40%	0	0%	0%	0%	0	0%	0%	0%
pl	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
vt	54	59%	0%	40%	46	56%	0%	43%	46	56%	0%	43%
vl	16	56%	0%	43%	0	0%	0%	0%	0	0%	0%	0%
gt	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
gl	17	52%	0%	47%	0	0%	0%	0%	0	0%	0%	0%
st	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
sl	20	50%	0%	50%	0	0%	0%	0%	0	0%	0%	0%

Created DD Goals Transmitted DD Goals Received DD Goals

Level	No.	T	C	F	No.	T	C	F	No.	T	C	F
pt	178	58%	0%	41%	0	0%	0%	0%	0	0%	0%	0%
pl	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
vt	98	61%	0%	38%	297	61%	0%	38%	291	62%	0%	37%
vl	17	52%	0%	47%	0	0%	0%	0%	0	0%	0%	0%
gt	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
gl	20	50%	0%	50%	0	0%	0%	0%	0	0%	0%	0%
st	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
sl	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%

Created GD Goals

pt	pl	vt	vl	gt	gl	st	sl
0	0	291	0	0	0	0	0

Level	Instantiated KSs No.	T	C	F	Invoked KSs No.	T	C	F	Local KS Power	Global KS Power
pt	50	64%	0%	36%	50	64%	0%	36%	0	0
pl	0	0%	0%	0%	0	0%	0%	0%	0	0
vt	80	58%	0%	41%	77	57%	0%	42%	0	0
vl	17	52%	0%	47%	16	56%	0%	43%	0	0
gt	0	0%	0%	0%	0	0%	0%	0%	0	0
gl	20	50%	0%	50%	17	52%	0%	47%	0	0
st	0	0%	0%	0%	0	0%	0%	0%	0	0
sl	0	0%	0%	0%	0	0%	0%	0%	0	0

Topology Level 1 Summary Statistics

Nodes: (1 2 3 4)

		No. True	Cons.	False
Initial Node State: 288	Hyps Created:	161	57%	0%	42%
Final Node State: 288	Hyps Transmitted:	46	56%	0%	43%
Competitor Sets: 16	Hyps Received:	46	56%	0%	43%
GD Goals Created: 291	DD Goals Created:	313	58%	0%	41%
	DD Goals Transmitted:	297	61%	0%	38%
	DD Goals Received	291	62%	0%	37%

Global KS Power:	0	KSs Instantiated:	167	58% 0% 41%
Local KS Power:	0	KSs Invoked:	160	58% 0% 41%

	Created Hyps				Transmitted Hyps				Received Hyps			
Level	No.	T	C	F	No.	T	C	F	No.	T	C	F
pt	54	59%	0%	40%	0	0%	0%	0%	0	0%	0%	0%
pl	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
vt	54	59%	0%	40%	46	56%	0%	43%	46	56%	0%	43%
vl	16	56%	0%	43%	0	0%	0%	0%	0	0%	0%	0%
gt	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
gl	17	52%	0%	47%	0	0%	0%	0%	0	0%	0%	0%
st	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
sl	20	50%	0%	50%	0	0%	0%	0%	0	0%	0%	0%

	Created DD Goals				Transmitted DD Goals				Received DD Goals			
Level	No.	T	C	F	No.	T	C	F	No.	T	C	F
pt	178	58%	0%	41%	0	0%	0%	0%	0	0%	0%	0%
pl	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
vt	98	61%	0%	38%	297	61%	0%	38%	291	62%	0%	37%
vl	17	52%	0%	47%	0	0%	0%	0%	0	0%	0%	0%
gt	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
gl	20	50%	0%	50%	0	0%	0%	0%	0	0%	0%	0%
st	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
sl	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%

Created GD Goals							
pt	pl	vt	vl	gt	gl	st	sl
0	0	291	0	0	0	0	0

	Instantiated KSs				Invoked KSs					
Level	No.	T	C	F	No.	T	C	F	Local KS Power	Global KS Power
pt	50	64%	0%	36%	50	64%	0%	36%	0	0
pl	0	0%	0%	0%	0	0%	0%	0%	0	0
vt	80	58%	0%	41%	77	57%	0%	42%	0	0
vl	17	52%	0%	47%	16	56%	0%	43%	0	0
gt	0	0%	0%	0%	0	0%	0%	0%	0	0
gl	20	50%	0%	50%	17	52%	0%	47%	0	0
st	0	0%	0%	0%	0	0%	0%	0%	0	0
sl	0	0%	0%	0%	0	0%	0%	0%	0	0

Node 1 Summary Statistics

Topology Level: 1

			No.	True	Cons.	False
Initial Node State:	157	Hyps Created:	46	43%	0%	56%
Final Node State:	157	Hyps Transmitted:	12	50%	0%	50%
Competitor Sets:	8	Hyps Received:	6	33%	0%	66%
GD Goals Created:	78	DD Goals Created:	72	40%	0%	59%
		DD Goals Transmitted:	63	38%	0%	61%
		DD Goals Received	78	67%	0%	32%
Global KS Power:	0	KSs Instantiated:	46	47%	0%	52%
Local KS Power:	0	KSs Invoked:	42	42%	0%	57%

	Created Hyps				Transmitted Hyps				Received Hyps			
Level	No.	T	C	F	No.	T	C	F	No.	T	C	F
pt	12	41%	0%	58%	0	0%	0%	0%	0	0%	0%	0%
pl	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
vt	12	41%	0%	58%	12	50%	0%	50%	6	33%	0%	66%
vl	7	42%	0%	57%	0	0%	0%	0%	0	0%	0%	0%
gt	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
gl	7	42%	0%	57%	0	0%	0%	0%	0	0%	0%	0%
st	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
sl	8	50%	0%	50%	0	0%	0%	0%	0	0%	0%	0%

	Created DD Goals				Transmitted DD Goals				Received DD Goals			
Level	No.	T	C	F	No.	T	C	F	No.	T	C	F
pt	36	38%	0%	61%	0	0%	0%	0%	0	0%	0%	0%
pl	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
vt	21	38%	0%	61%	63	38%	0%	61%	78	67%	0%	32%
vl	7	42%	0%	57%	0	0%	0%	0%	0	0%	0%	0%
gt	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
gl	8	50%	0%	50%	0	0%	0%	0%	0	0%	0%	0%
st	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
sl	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%

Created GD Goals

pt	pl	vt	vl	gt	gl	st	sl
0	0	78	0	0	0	0	0

	Instantiated KSs				Invoked KSs					
Level	No.	T	C	F	No.	T	C	F	Local KS Power	Global KS Power
pt	11	45%	0%	54%	11	45%	0%	54%	0	0
pl	0	0%	0%	0%	0	0%	0%	0%	0	0
vt	20	50%	0%	50%	17	41%	0%	58%	0	0
vl	7	42%	0%	57%	7	42%	0%	57%	0	0
gt	0	0%	0%	0%	0	0%	0%	0%	0	0
gl	8	50%	0%	50%	7	42%	0%	57%	0	0
st	0	0%	0%	0%	0	0%	0%	0%	0	0
sl	0	0%	0%	0%	0	0%	0%	0%	0	0

Node 2 Summary Statistics

Topology Level: 1

			No.	True	Cons.	False
Initial Node State:	-1135	Hyps Created:	34	52%	0%	47%
Final Node State:	-1135	Hyps Transmitted:	11	27%	0%	72%
Competitor Sets:	2	Hyps Received:	14	71%	0%	28%
GD Goals Created:	72	DD Goals Created:	77	58%	0%	41%
		DD Goals Transmitted:	75	56%	0%	44%
		DD Goals Received	72	65%	0%	34%
Global KS Power:	0	KSs Instantiated:	36	52%	0%	47%
Local KS Power:	0	KSs Invoked:	36	52%	0%	47%

	Created Hyps				Transmitted Hyps				Received Hyps			
Level	No.	T	C	F	No.	T	C	F	No.	T	C	F
pt	14	64%	0%	35%	0	0%	0%	0%	0	0%	0%	0%
pl	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
vt	14	64%	0%	35%	11	27%	0%	72%	14	71%	0%	28%
vl	2	0%	0%	100%	0	0%	0%	0%	0	0%	0%	0%
gt	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
gl	2	0%	0%	100%	0	0%	0%	0%	0	0%	0%	0%
st	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
sl	2	0%	0%	100%	0	0%	0%	0%	0	0%	0%	0%

Level	Created DD Goals				Transmitted DD Goals				Received DD Goals			
	No.	T	C	F	No.	T	C	F	No.	T	C	F
pt	48	64%	0%	35%	0	0%	0%	0%	0	0%	0%	0%
pl	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
vt	25	56%	0%	44%	75	56%	0%	44%	72	65%	0%	34%
vl	2	0%	0%	100%	0	0%	0%	0%	0	0%	0%	0%
gt	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
gl	2	0%	0%	100%	0	0%	0%	0%	0	0%	0%	0%
st	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
sl	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%

	Created GD Goals							
	pt	pl	vt	vl	gt	gl	st	sl
	0	0	72	0	0	0	0	0

Level	Instantiated KSs				Invoked KSs				Local KS Power	Global KS Power
	No.	T	C	F	No.	T	C	F		
pt	13	69%	0%	30%	13	69%	0%	30%	0	0
pl	0	0%	0%	0%	0	0%	0%	0%	0	0
vt	19	52%	0%	47%	19	52%	0%	47%	0	0
vl	2	0%	0%	100%	2	0%	0%	100%	0	0
gt	0	0%	0%	0%	0	0%	0%	0%	0	0
gl	2	0%	0%	100%	2	0%	0%	100%	0	0
st	0	0%	0%	0%	0	0%	0%	0%	0	0
sl	0	0%	0%	0%	0	0%	0%	0%	0	0

==========
Node 3 Summary Statistics
==========

Topology Level: 1

Initial Node State: 1450
Final Node State: 1450
Competitor Sets: 2

GD Goals Created: 72

	No.	True	Cons.	False
Hyps Created:	34	70%	0%	29%
Hyps Transmitted:	5	100%	0%	0%
Hyps Received:	6	56%	0%	43%
DD Goals Created:	77	68%	0%	31%
DD Goals Transmitted:	75	72%	0%	28%
DD Goals Received	72	59%	0%	40%

Global KS Power:	0	KSs Instantiated:	35	71%	0%	28%		
Local KS Power:	0	KSs Invoked:	35	71%	0%	28%		

	Created Hyps				Transmitted Hyps				Received Hyps			
Level	No.	T	C	F	No.	T	C	F	No.	T	C	F
pt	14	64%	0%	35%	0	0%	0%	0%	0	0%	0%	0%
pl	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
vt	14	64%	0%	35%	5	100%	0%	0%	16	56%	0%	43%
vl	2	100%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
gt	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
gl	2	100%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
st	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
sl	2	100%	0%	0%	0	0%	0%	0%	0	0%	0%	0%

	Created DD Goals				Transmitted DD Goals				Received DD Goals			
Level	No.	T	C	F	No.	T	C	F	No.	T	C	F
pt	48	64%	0%	35%	0	0%	0%	0%	0	0%	0%	0%
pl	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
vt	25	72%	0%	28%	75	72%	0%	28%	72	59%	0%	40%
vl	2	100%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
gt	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
gl	2	100%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
st	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
sl	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%

Created GD Goals

pt	pl	vt	vl	gt	gl	st	sl
0	0	72	0	0	0	0	0

	Instantiated KSs				Invoked KSs					
Level	No.	T	C	F	No.	T	C	F	Local KS Power	Global KS Power
pt	14	64%	0%	35%	14	64%	0%	35%	0	0
pl	0	0%	0%	0%	0	0%	0%	0%	0	0
vt	17	70%	0%	29%	17	70%	0%	29%	0	0
vl	2	100%	0%	0%	2	100%	0%	0%	0	0
gt	0	0%	0%	0%	0	0%	0%	0%	0	0
gl	2	100%	0%	0%	2	100%	0%	0%	0	0
st	0	0%	0%	0%	0	0%	0%	0%	0	0
sl	0	0%	0%	0%	0	0%	0%	0%	0	0

Node 4 Summary Statistics

Topology Level: 1

			No.	True	Cons.	False
Initial Node State:	682	Hyps Created:	47	63%	0%	36%
Final Node State:	682	Hyps Transmitted:	18	66%	0%	33%
Competitor Sets:	8	Hyps Received:	10	50%	0%	50%
GD Goals Created:	69	DD Goals Created:	87	65%	0%	34%
		DD Goals Transmitted:	84	75%	0%	25%
		DD Goals Received:	69	57%	0%	42%
Global KS Power:	0	KSs Instantiated:	50	64%	0%	36%
Local KS Power:	0	KSs Invoked:	47	68%	0%	31%

	Created Hyps				Transmitted Hyps				Received Hyps			
Level	No.	T	C	F	No.	T	C	F	No.	T	C	F
pt	14	64%	0%	35%	0	0%	0%	0%	0	0%	0%	0%
pl	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
vt	14	64%	0%	35%	18	66%	0%	33%	10	50%	0%	50%
vl	5	80%	0%	20%	0	0%	0%	0%	0	0%	0%	0%
gt	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
gl	6	66%	0%	33%	0	0%	0%	0%	0	0%	0%	0%
st	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
sl	8	50%	0%	50%	0	0%	0%	0%	0	0%	0%	0%

	Created DD Goals				Transmitted DD Goals				Received DD Goals			
Level	No.	T	C	F	No.	T	C	F	No.	T	C	F
pt	46	63%	0%	36%	0	0%	0%	0%	0	0%	0%	0%
pl	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
vt	27	74%	0%	25%	84	75%	0%	25%	69	57%	0%	42%
vl	6	66%	0%	33%	0	0%	0%	0%	0	0%	0%	0%
gt	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
gl	8	50%	0%	50%	0	0%	0%	0%	0	0%	0%	0%
st	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%
sl	0	0%	0%	0%	0	0%	0%	0%	0	0%	0%	0%

Created GD Goals

pt	pl	vt	vl	gt	gl	st	sl
0	0	69	0	0	0	0	0

	Instantiated KSs				Invoked KSs					
Level	No.	T	C	F	No.	T	C	F	Local KS Power	Global KS Power
pt	12	75%	0%	25%	12	75%	0%	25%	0	0
pl	0	0%	0%	0%	0	0%	0%	0%	0	0
vt	24	62%	0%	37%	24	62%	0%	37%	0	0
vl	6	66%	0%	33%	5	80%	0%	20%	0	0
gt	0	0%	0%	0%	0	0%	0%	0%	0	0
gl	8	50%	0%	50%	6	66%	0%	33%	0	0
st	0	0%	0%	0%	0	0%	0%	0%	0	0
sl	0	0%	0%	0%	0	0%	0%	0%	0	0

Appendix D. KS Descriptions

Candidate generator parts of KSs (refered to as KSs in this section) are procedures that use hypotheses as input to produce new hypotheses on their output. Each KS consists of a precondition and an action part. The precondition part forms the input set based on the stimulus hypothesis. The stimulus is a hypothesis that causes a KS to be invoked. The input set is formed by the precondition part from the hypotheses that represent the context of a KS. If the input set is empty, the action part of a KS is not executed. The action part uses the hypotheses in the input set to create output hypotheses. The output hypotheses produced by the action part are supported by the particualr input hypotheses used in their creation. The output hypotheses are those that can be derived from the input hypotheses by the grammar.

The belief of an output hypothesis is proportional to the degree of consistency in the input, i.e. to the number and beliefs of its supporting hypotheses. The belief of an output hypothesis is obtained from input beliefs by the rule for combining the probabilities of independent events. Since the sensors are error-prone, KSs have to incorporate slightly inconsistent input hypotheses in their output. A KS can form an output hypothesis supported by inputs that have locations and signal frequencies slightly off from one another (within the specified location and signal frequency deviation tolerance). Such an offset is reflected in the lower belief of the output hypothesis. The belief of an output hypothesis is also weighted by the ratio of the number of supporting hypotheses to the maximum number of supporting hypotheses (as defined in the grammar).

The name of the KS is given in the form:

type : input-blackboard-level : output-blackboard-level.

There are seven types of KSs: form-track (FT), synthesize-locations (SLL), synthesize-tracks (STT), extend-track (ET), merge-tracks (MT), hypothesis-send (HS) and hypothesis-receive (HR). Blackboard levels are: signal location (SL), signal track (ST), group location (GL), group track (GT), vehicle location (VL), vehicle track (VT), pattern location (PL) and pattern track (PT). The following KSs are defined in the system:

1. FT:SL:ST, FT:GL:GT, FT:VL:VT, FT:PL:PT,

2. SLL:SL:GL, SLL:GL:VL, SLL:VL:PL,

3. STT:ST:GT, STT:GT:VT, STT:VT:PT,

4. ET:ST:ST, ET:GT:GT, ET:VT:VT, ET:PT:PT,

5. MT:ST:ST, MT:GT:GT, MT:VT:VT, MT:PT:PT,

6. HS:SL:SL, HS:ST:ST, HS:GL:GL, HS:GT:GT, HS:VL:VL, HS:VT:VT, HS:PL:PL, HS:PT:PT,

7. HR:SL:SL, HR:ST:ST, HR:GL:GL, HR:GT:GT, HR:VL:VL, HR:VT:VT, HR:PL:PL and HR:PT:PT.

Since KSs of the same type operate similarly on different levels, only one KS of a type will be presented here. The variations required for pattern level KSs are described at the end of this section.

SLL:SL:GL

This KS combines signal location hypotheses into group location hypotheses. The stimulus is a signal location hypothesis. The precondition part finds all the groups that the stimulus signal belongs to (by consulting the grammar), and creates one element of the input set for each. Since every group has a number of component signals, the input set contains the hypotheses corresponding to each of them (having the same signal frequency within the signal frequency deviation tolerance). The location and time of hypotheses in the input set are the same as for the stimulus (within the location deviation tolerance).

The action part creates a group location hypothesis corresponding to each element of the input set. The classes of the supporting hypotheses are some component signals of the output group (those that can be found on the blackboard).

FT:SL:ST

This KS forms a signal track hypothesis from two signal location hypotheses in adjacent time frames. The stimulus is a signal location hypothesis. The precondition part of this KS forms the input set from the stimulus and the hypotheses of the same signal frequency (within the signal frequency deviation tolerance) found on the blackboard. The input set contains "previous hypotheses", "current hypotheses" and "next hypotheses". The time and location of current hypotheses is the same as that of the stimulus (within the location deviation tolerance). The time of the previous and next hypotheses is one unit smaller and one unit greater than that of the stimulus, respectively, and their locations must be close enough to that of the stimulus so that the distance can be traversed in one time frame. The action part forms all "backward tracks" (supported by current hypotheses and previous hypotheses), and all "forward tracks" (supported by current hypotheses and next hypotheses).

STT:ST:GT

This is a KS that combines signal tracks into a group track. The stimulus is a signal-track hypothesis. The precondition part finds all the groups that the stimulus signal can belong to, and then finds the signal-track hypotheses whose signal frequencies are components of these groups (much like SLL:SL:GL does, though on different levels). These tracks must overlap a certain portion of the stimulus track (defined by the track-overlap deviation tolerance), and overlaping locations must be the same as stimulus' location (within the location deviation tolerance).

The action part creates all group track hypotheses with the support from the input set. The belief contribution of each supporting hypothesis is proportional to the amount of overlap with stimulus track.

ET:ST:ST

This KS attempts to extend a signal-track hypothesis with a signal-location hypothesis, either in a forward or a backward direction. The description will be given for forward extension (backward extension is equivalent). The stimulus is either a track hypothesis or a location hypothesis. The input set contains track hypotheses and location hypotheses that are "extension-compatible". A track hypothesis and a location hypothesis are extension compatible if 1) they have the same signal frequency (within the signal frequency deviation tolerance) 2) the location hypothesis is in the next time frame from the end time of the track and 3) the location of the location hypothesis is within the range determined from the velocity of the track and the maximum velocity change possible.

If the stimulus is the track hypothesis, location hypotheses in the input set are all that are extension-compatible with the stimulus hypothesis. The track hypotheses are all that are extension-compatible with the location hypotheses. If the stimulus is a location hypothesis, track hypotheses in the input set are all tracks that are extension-compatible with the stimulus hypothesis. The location hypotheses are all that are extension-compatible with the track hypotheses. The signal frequencies of the hypotheses in the input set are the same as for the stimulus (within the signal frequency deviation tolerance).

The action part of ET:ST:ST creates extended signal track hypotheses. An output hypothesis is a signal track hypothesis from the input set, extended with a location hypothesis from the input set and supported by the two of them. The new velocity is obtained from the end location of the supporting track hypothesis and the location of the supporting location hypothesis. The belief contribution of the location hypothesis is proportional to the similarity between the new velocity

and the velocity of the supporting track.

MT:ST:ST

This KS attempts to merge tracks on the signal track level that overlap in both time and location. Here we will describe forward merging (backward merging is equivalent).

The precondition part forms the input set from the stimulus, its competitors and all "mergeable" track hypotheses. The mergeable tracks are those which overlap the stimulus track in at least its last location (within the location deviation tolerance) and have the same signal frequency as the stimulus (within the signal frequency tolerance). The competitors of the stimulus are all those signal tracks that are overlapped by a mergeable track in at least their last location (within tolerance) and have the same signal frequency as the stimulus (within tolerance). We shall call the stimulus and its competitors the "base hypotheses".

The action part of MT:ST:ST forms a track for each base hypothesis, supported by the base hypothesis, and a mergeable hypothesis. The time-location-list of an output hypothesis is that of the base hypothesis extended by the non-overlapping part of the merged hypothesis' track. The belief contribution of the merged hypothesis is proportional to the amount of its overlap with the base hypothesis.

Backward merging of tracks is the same as forward merging but the base hypotheses are merged with tracks that overlap at least their first location.

Pattern Level KSs

Two pattern KSs (SLL:VL:PL and STT:VT:PT) differ slightly from the models described (SLL:SL:GL and STT:ST:GT) because the vehicles in the pattern have different locations (as opposed to signals in a group or groups in a vehicle). The locations of the vehicles in a pattern are specified relative to the center of the pattern, and can be obtained from pattern specifications known to the system. So, when the input set of one of these KSs is formed, it will consist of vehicles properly displaced from the stimulus vehicle location (as defined in pattern specification of the pattern that the stimulus can be part of) within the location deviation tolerance. The form of the input set and processing in the action part of a KS is otherwise the same as in SLL:SL:GL or STT:ST:GT.

HS:ST:ST

This ks sends a hypothesis created locally on signal track level, to a number of nodes that are potentially interested in its attributes. When a hypothesis is created, the precondition part is invoked. Simple models of other nodes are available to hypothesis-send precondition. A model of a node from the local node perspective, is a list of ranges of relevant hypothesis attributes: event class, time, and location. If the attributes of a created hypothesis are within ranges for the target node, and belief of the hypothesis is above a specified threshold, then the pair (target-node hypothesis) is added to the output-set, provided that the hypothesis has not already been seen by that node. (this test is to prevent cycles.) The model of other nodes for the purpose of sending contains also a weight which reflects the importance of sending hypotheses in the ranges defined. This weight multiplies the belief of sent hypothesis. In the action part, the hypothesis is sent to as many nodes as there are in the output set.

HR:ST:ST

Hyp-receive KS puts on the blackboard a subset of hypotheses sent by other nodes which the local node is interested in. The precondition part is invoked when the hypotheses sent from ST level reach the node. Filtering on reception side is accomplished by checking the received hypotheses attributes against the desired range of event class, time and location. If all the attributes are within the range, the hypothesis is added to the output set, after belief modification which reflects the importance of the ranges (like for hypothesis-send). The action part of send puts each hypothesis from the output set on the local blackboard. The belief of the hypothesis can be decreased if the local node does not consider the sender "credible": the simple model of credibility contains only the percentage of belief decrease.

Appendix E. Front-End Description and Sample Input

Front-End Description

The FRONTEND is the testbed subsystem which initializes the testbed with a particular configuration of nodes and sensors and a particular scenario of patterns and vehicles in the environment. This initialization process involves reading an input file to set up local and global data structures, making the consistency blackboard, and making a data blackboard for each node.

Each configuration-scenario combination is stored compactly in an input file. A sample input file follows this section. Input files are commented to explain information content and form in detail. An interactive graphics program for defining configuration-scenarios is being developed.

The input file contains all the input data for the testbed, and consists of system, structural and environmental data. System data denotes some basic parameters of the simulated vehicle monitoring system: a seed for random number generation, location and time ranges, and numbers of classes, nodes, and sensors. Structural data denotes the spatial relationships and set membership relationships between the classes of information in the simulated system: the pattern-vehicle-group-signal "grammar" and network specifications. Environmental data denotes the true and perceived environments for the vehicle monitoring system: true locations of patterns and vehicles at various time frames, and perceived locations of patterns, vehicles, groups, and signals at various time frames (including false patterns, vehicles, groups , and signals, and excluding missing patterns, vehicles, groups, and signals).

The frontend begins by reading an input file and setting up data structures. System data is placed in global data structures for use by the testbed kernel. Structural data is also placed in global data structures so it may be accessed by knowledge sources. Environmental data is used in conjuction with the structural data to create the consistency and data blackboards. The last task for the frontend is the computation of competitor sets.

The consistency blackboard stores location and track hypotheses for all true patterns, vehicles, groups, and signals in the environment. There is only one consistency blackboard, and it is used by the oracle to simulate KSs of any power.

The frontend must make a data blackboard for each node. A node's data blackboard stores all signal location hypotheses which are sensed by sensors that report to the node. Because sensing is imperfect, noise

is added to the location and signal class of each true signal according to the sensor class and where the signal is within each sensor's range. A provision has also been made to add correlated errors by explicitly excluding missing patterns, vehicles, groups, and signals, and including false patterns, vehicles, groups, and signals.

A competitor set is a set of competing hypotheses and is used in the performance measures. The frontend computes all competitor sets.

Template for Input to the System

Below is the template for input to the system:

```
;
;                              TEMPLATE FILE
;
;     This template file, last modified apr-26-1982, defines the format
; of environment files.  An environment file begins and ends with the
; string '$$ENVIRONMENT-FILE$$' and contains all the input data to the
; system.  It contains SYSTEM DATA, GRAMMAR DEFINITION, NODE
; DEFINITIONS, SENSOR DEFINITIONS, COMMUNICATION RELIABILITY DATA,
; CONSISTENCY DATA and ENVIRONMENT DATA.
;
;     SYSTEM DATA denotes some basic parameters of the system: random
; seed, maximum velocity and maximum acceleration and some global tuning
; parameters.
;
;     GRAMMAR DEFINITION contains information concerning the grammar: a
; grammar-id, the pattern-class to vehicle-class relation, the
; vehicle-class to group-class relation, and the group-class to
; signal-class relation.
;
;     NODE DEFINITIONS contains information concerning nodes:
; interest-area definitions, ks-set definitions, topological-role
; definitions, node-class definitions, and node data.
;
;     SENSOR DEFINITIONS contains information concerning sensors:
; sensor-class definitions and sensor data.
;
;     COMMUNICATION RELIABILITY DATA contains information concerning
; the likelihood of communication errors: hyp communication errors, and
; goal communication errors.
;
;     CONSISTENCY DATA contains the information which determines hyps
; to be put on the cbb (consistency blackboard).
;
;     ENVIRONMENT DATA contains information that determines what is
; sensed in the environment.
;
;
;                              SEARCH KEYS
;
; The following strings may be used to find sections quickly:
;
;          ** ENVIRONMENT-ID AND COMMENTS **
;          ** SYSTEM DATA **
;               ** RANDOM SEED **
```

```
;              ** MAX VELOCITY AND ACCELERATION**
;              ** GLOBAL TUNING PARAMETERS **
;          ** GRAMMAR DEFINITION **
;              ** GRAMMAR-ID **
;              ** PATTERN-CLASS TO VEHICLE-CLASS RELATION **
;              ** VEHICLE-CLASS TO GROUP-CLASS RELATION **
;              ** GROUP-CLASS TO SIGNAL-CLASS RELATION **
;          ** NODE DEFINITIONS **
;              ** INTEREST-AREA DEFINITIONS **
;              ** KS-SET DEFINITIONS **
;              ** TOPOLOGICAL-ROLE DEFINITIONS **
;              ** NODE-CLASS DEFINITIONS **
;              ** NODE DATA **
;          ** SENSOR DEFINITIONS **
;              ** SENSOR-CLASS DEFINITIONS **
;              ** SENSOR DATA **
;          ** COMMUNICATION RELIABLILITY DATA **
;              ** HYP COMMUNICATION ERRORS **
;              ** GOAL COMMUNICATION ERRORS **
;          ** CONSISTENCY DATA **
;              ** CBB PT **
;              ** CBB PL **
;              ** CBB VT **
;              ** CBB VL **
;              ** CBB GT **
;              ** CBB GL **
;              ** CBB ST **
;              ** CBB SL **
;          ** ENVIRONMENT DATA **
;              ** PATTERN DATA **
;              ** VEHICLE DATA **
;              ** GROUP DATA **
;              ** SIGNAL DATA **
;
;
;                              INTEGER CONSTANTS
;
;     #-event-classes                        begin-coordinate
;     #-nodes                                end-coordinate
;     #-node-classes                         begin-time
;     #-interest-areas                       end-time
;     #-ks-sets                              max-velocity
;     #-topological-roles                    max-acceleration
;     #-sensors                              transmission-latency
;     #-sensor-classes                       transmissions-per-period
;                                              reception-latency
;                                              receptions-per-period
;
; NOTE:  constants beginning with  #-  are not explicitly included  in an
;        environment file to make the addition of new things easy;  thus
```

```
;       if a new node is added, #-nodes need not be found and changed.
;
;                              BASE DATA TYPES
;
;   id          = a string of characters
;   time        = an integer in the range [begin-time...end-time]
;   coordinate  = an integer in the range
;                       [begin-coordinate...end-coordinate]
;   location    = a pair of coordinates, (x-coordinate y-coordinate)
;   time-loc    = a pair consisting of a time and a location,
;                       (time (x-loc y-loc))
;   region      = a quadruple of coordinates, (x-min y-min x-max y-max)
;   level       = an element of the set {sl st gl gt vl vt pl pt}
;
;   seed        = an integer in the range [0..131072]
;   power       = an integer in the range [-10000..10000]
;   threshold   = an integer in the range [-10000..10000]
;   weight      = an integer in the range [0..10000]
;   probability = an integer in the range [0..10000]
;   belief      = an integer in the range [0..10000]
;   consistency = an integer in the range [0..10000]
;   credibility = an integer in the range [-10000..10000]
;
;   event-class      = an integer in the range [1..#-event-classes]
;   node             = an integer in the range [1..#-nodes]
;   node-class       = an integer in the range [1..#-node-classes]
;   interest-area    = an integer in the range [1..#-interest-areas]
;   ks-set           = an integer in the range [1..#-ks-sets]
;   topological-role = an integer in the range [1..#-topological-roles]
;   sensor           = an integer in the range [1..#-sensors]
;   sensor-class     = an integer in the range [1..#-sensor-classes]
;   size             = an integer in the range [1..system-loc-range]
;
;   pattern-class    = an event-class
;   vehicle-class    = an event-class
;   group-class      = an event-class
;   signal-class     = an event-class
;
;   level-list             = a list of levels,              or *all
;   event-class-list       = a list of event-classes,       or *all
;   time-list              = a list of times,               or *all
;   time-loc-list          = a list of time-locs
;   region-list            = a list of regions,             or *all
;   node-list              = a list of nodes,               or *all
;   sensor-list            = a list of sensors,             or *all
;   size-weight-list       = a list of pairs (size weight) or nil
;   level-size-weight-list = level and a size-weight-list
;
;
;
```

```
;
;          ** ENVIRONMENT-ID AND COMMENTS **

[  "environment-id"
   "first line of comments"
   ...
   "last line of comments"  ]

; comments are printed out at the beginning of the output trace
;
;
;
;
;          ** SYSTEM DATA **
;
;
;               ** RANDOM SEED **

[  seed  ]

;
;
;               ** MAX VELOCITY AND ACCELERATION **

[  max-velocity max-acceleration  ]

;
;
;               ** GLOBAL TUNING PARAMETERS **

[  scheduler-power
   scheduler-belief-weight
   reflection-threshold
   scheduler-threshold
   modification-threshold
   ksi-threshold
   goal-weight
   internal-subgoal-threshold
   received-hyp-subgoal-threshold
   received-goal-subgoal-threshold
   insertion-threshold
   goal-creation-threshold
   transmission-latency
   transmissions-per-cycle
   reception-latency
   receptions-per-cycle  ]

;
;
;
```

```
;
;          ** GRAMMAR DEFINITION **
;
;
;               ** GRAMMAR-ID **

[  "grammar-id"  ]

;
;               ** PATTERN-CLASS TO VEHICLE-CLASS RELATION **

[  (pattern-class ((vehicle-class location)
                   (vehicle-class location)
                   ...
                   (vehicle-class location)))
   (pattern-class ((vehicle-class location)
                   (vehicle-class location)
                   ...
                   (vehicle-class location)))
   ...
   (pattern-class ((vehicle-class location)
                   (vehicle-class location)
                   ...
                   (vehicle-class location)))  ]

; the location of a vehicle in a pattern is specified relative
;      to the center of the pattern
;
;
;               ** VEHICLE-CLASS TO GROUP-CLASS RELATION **

[  (vehicle-class (group-class group-class ... group-class))
   (vehicle-class (group-class group-class ... group-class))
   ...
   (vehicle-class (group-class group-class ... group-class))  ]

;
;
;               ** GROUP-CLASS TO SIGNAL-CLASS RELATION **

[  (group-class (signal-class signal-class ... signal-class))
   (group-class (signal-class signal-class ... signal-class))
   ...
   (group-class (signal-class signal-class ... signal-class))  ]

;
;
;
;
;          ** NODE DEFINITIONS **
```

```
;
;
;               ** INTEREST-AREA DEFINITIONS **

[ (interest-area (level-list event-class-list time-and-region-lists)
                 (level-list event-class-list time-and-region-lists)
                 ...
                 (level-list event-class-list time-and-region-lists))
  (interest-area (level-list event-class-list time-and-region-lists)
                 (level-list event-class-list time-and-region-lists)
                 ...
                 (level-list event-class-list time-and-region-lists))
  ...
  (interest-area (level-list event-class-list time-and-region-lists)
                 (level-list event-class-list time-and-region-lists)
                 ...
                 (level-list event-class-list time-and-region-lists))  ]

; and time-and-region-lists is a list of pairs,
;
;          ((time-list region-list)
;           (time-list region-list)
;           ...
;           (time-list region-list))
;
; Note: a region is specified relative to a node's center as a list
;
;
;               ** KS-SET DEFINITIONS **

[  (ks-set (ks-name goodness resolving-power runtime)
           (ks-name goodness resolving-power runtime)
           ...
           (ks-name goodness resolving-power runtime))
   (ks-set (ks-name goodness resolving-power runtime)
           (ks-name goodness resolving-power runtime)
           ...
           (ks-name goodness resolving-power runtime))
   ...
   (ks-set (ks-name goodness resolving-power runtime)
           (ks-name goodness resolving-power runtime)
           ...
           (ks-name goodness resolving-power runtime))  ]

; ks's in a ks-set are specified in order of decreasing priority
; goodness is an integer in the range [0..10000]
; runtime is a list of two constants (const-1 const-2) which are used
; to determine the runtime of a ks as:
; (const-1) * (#-stimulus-hyps) + (const-2).
;
```

```
;
;              ** TOPOLOGICAL-ROLE DEFINITIONS **

[  (topological-role node-list up-data same-level-data down-data)
   (topological-role node-list up-data same-level-data down-data)
   ...
   (topological-role node-list up-data same-level-data down-data)  ]

; where up-data, same-level-data, and down-data are lists of the form:
;
;       ((level-list ave-number-paths)...(level-list ave-number-paths))
;
; and ave-number-paths is a real number
;


;
;              ** NODE-CLASS DEFINITIONS **

[  (node-class ks-set cset-region interest-area-weight-list
                  topological-role subgoal-data)
   (node-class ks-set cset-region interest-area-weight-list
                  topological-role subgoal-data)
   ...
   (node-class ks-set cset-region interest-area-weight-list
                  topological-role subgoal-data)  ]

; where cset-region is a region specified relative to a node's location
; and interest-area-weight-list is a list
;
;      ((interest-area weight)...(interest-area weight))
;
;      of pairs of an interest-area and associated scheduling weight
; and subgoal-data is a list of pairs:
;        (subgoal-level-list (list of level-size-weight-list))
;     or (subgoal-level-list nil)
;  subgoal-level-list is a list of levels from which subgoals will be
;       created
;  level-size-weight-list is a pair:
;        (level-list (list of size-weights))
;  level-list is a list of levels at which the subgoals will be created
;       for each of the top-level goals.
;  size-weight is a pair of size-shrink-factor and a weight.
;
;  Note: The size-shrink-factor determines by how much a region will be
;        reduced for the subgoal at that level and the weight mulitplies
;        the original top-goal rating to get the new subgoal rating.  If
;        the level-size-weight-list is nil then all the subgoals will be
;        created at all the levels below the top-goal-level which are in
;        the interest area with the same region and rating as the top
;        goal.
```

```
;
[ ( (pt pl) ( ((pt vt gt st) ((1 500) (2 300)))
              ((pl vl)       ((3 300) (4 100))) ) )
  ( (gl st) ( ((gl st sl)    ((1 100) (2 500))) ) ) ]
;
;
;               ** NODE DATA **

[ (node node-class location sensor-list hyp-send-list hyp-receive-list
    goal-send-list goal-help-list goal-receive-list sensed-times-list)
  (node node-class location sensor-list hyp-send-list hyp-receive-list
    goal-send-list goal-help-list goal-receive-list sensed-times-list)
   ...
  (node node-class location sensor-list hyp-send-list hyp-receive-list
    goal-send-list goal-help-list goal-receive-list sensed-times-list) ]

; where hyp-send-list, goal-send-list and goal-help-list are lists,
;
;      ((node-list (interest-area threshold weight)
;                  ...
;                  (interest-area threshold weight))
;       (node-list (interest-area threshold weight)
;                  ...
;                  (interest-area threshold weight))
;       ...
;       (node-list (interest-area threshold weight)
;                  ...
;                  (interest-area threshold weight)))
;
; and hyp-receive-list and goal-receive-list are lists,
;
;      ((node-list (interest-area threshold weight credibility)
;                  ...
;                  (interest-area threshold weight credibility))
;       (node-list (interest-area threshold weight credibility)
;                  ...
;                  (interest-area threshold weight credibility))
;       ...
;       (node-list (interest-area threshold weight credibility)
;                  ...
;                  (interest-area threshold weight credibility)))
;
; and sensed-times-list is a list,
;
;       ((system-cycle-number time-frame-list) ...)
;
;
;
;
```

```
;          ** SENSOR DEFINITIONS **
;
;
;               ** SENSOR-CLASS DEFINITIONS **

[  (sensor-class tolerance location-mask signal-mask)
   (sensor-class tolerance location-mask signal-mask)
   ...
   (sensor-class tolerance location-mask signal-mask)  ]

; where tolerance is an integer specifying the spatial range of a sensor
; and location-mask is a list,
;
;       ((lm11 lm12 ... lm1L)
;        (lm21 lm22 ... lm2L)
;        ...
;        (lmL1 lmL2 ... lmLL))
;
;     of location-mask-values in the range 0..8; L = 2*tolerance + 1
; and signal-mask is a list,
;
;       (sm1 sm2 ... smS)
;
;     of signal-mask-values in the range 0..2; S = #-event-classes
;
;
;               ** SENSOR DATA **

[  (sensor sensor-class location
           prob-true-location prob-true-signal sensor-weight)
   (sensor sensor-class location
           prob-true-location prob-true-signal sensor-weight)
   ...
   (sensor sensor-class location
           prob-true-location prob-true-signal sensor-weight)  ]


;
;
;
;
;          ** COMMUNICATION RELIABLILITY DATA **
;
;
;
;               ** HYP COMMUNICATION ERRORS **

[  (from-node-list to-node-list probability-of-error)
   (from-node-list to-node-list probability-of-error)
   ...
   (from-node-list to-node-list probability-of-error)  ]
```

```
; communication links not mentioned have probability-of-error = 0
;
;
;               ** GOAL COMMUNICATION ERRORS **

[  (from-node-list to-node-list probability-of-error)
   (from-node-list to-node-list probability-of-error)
   ...
   (from-node-list to-node-list probability-of-error)  ]

; communication links not mentioned have probability-of-error = 0
;
;
;
;
;          ** CONSISTENCY DATA **
;
;
;               ** CBB PT **
;
;

[  (event-class time-loc-list true?)
   (event-class time-loc-list true?)
   ...
   (event-class time-loc-list true?)  ]

; true? has a value of t if data is true, otherwise nil.
;
;
;               ** CBB PL **
;
;

[  (event-class time-loc-list)
   (event-class time-loc-list)
   ...
   (event-class time-loc-list)  ]

;
;
;               ** CBB VT**
;
;

[  (event-class time-loc-list)
   (event-class time-loc-list)
   ...
   (event-class time-loc-list)  ]
```

```
;
;
;               ** CBB VL **
;
;
;

[  (event-class time-loc-list)
   (event-class time-loc-list)
   ...
   (event-class time-loc-list)  ]

;
;
;               ** CBB GT**
;
;
;

[  (event-class time-loc-list)
   (event-class time-loc-list)
   ...
   (event-class time-loc-list)  ]

;
;
;               ** CBB GL **
;
;
;

[  (event-class time-loc-list)
   (event-class time-loc-list)
   ...
   (event-class time-loc-list)  ]

;
;
;               ** CBB ST**
;
;
;

[  (event-class time-loc-list)
   (event-class time-loc-list)
   ...
   (event-class time-loc-list)  ]

;
;
;               ** CBB SL **
;
```

```
;
[  (event-class time-loc-list)
   (event-class time-loc-list)
   ...
   (event-class time-loc-list)  ]

;
;
;          ** ENVIRONMENT DATA **
;
;
;               ** PATTERN DATA **
;
;
;

[  (event-class ( (time-loc-list belief sensor-list)
                  (time-loc-list belief sensor-list)
                   ...
                  (time-loc-list belief sensor-list)))
   (event-class ( (time-loc-list belief sensor-list)
                  (time-loc-list belief sensor-list)
                  ...
                  (time-loc-list belief sensor-list)))
   ...
   (event-class ( (time-loc-list belief sensor-list)
                  (time-loc-list belief sensor-list)
                  ...
                  (time-loc-list belief sensor-list)))  ]
;
;
;               ** VEHICLE DATA **
;
;
;

[  (event-class (time-loc-list belief sensor-list)
                (time-loc-list belief sensor-list)
                ...
                (time-loc-list belief sensor-list) )
   (event-class (time-loc-list belief sensor-list)
                (time-loc-list belief sensor-list)
                ...
                (time-loc-list belief sensor-list) )
   ...
   (event-class (time-loc-list belief sensor-list)
                (time-loc-list belief sensor-list)
                ...
                (time-loc-list belief sensor-list) )  ]

;
```

```
;
;               ** GROUP DATA **
;
;
;

[  (event-class (time-loc-list belief sensor-list)
                (time-loc-list belief sensor-list)
                ...
                (time-loc-list belief sensor-list) )
   (event-class (time-loc-list belief sensor-list)
                (time-loc-list belief sensor-list)
                ...
                (time-loc-list belief sensor-list) )
   ...
   (event-class (time-loc-list belief sensor-list)
                (time-loc-list belief sensor-list)
                ...
                (time-loc-list belief sensor-list) )  ]

;
;
;               ** SIGNAL DATA **
;
;
;

[  (event-class (time-loc-list belief sensor-list)
                (time-loc-list belief sensor-list)
                ...
                (time-loc-list belief sensor-list) )
   (event-class (time-loc-list belief sensor-list)
                (time-loc-list belief sensor-list)
                ...
                (time-loc-list belief sensor-list) )
   ...
   (event-class (time-loc-list belief sensor-list)
                (time-loc-list belief sensor-list)
                ...
                (time-loc-list belief sensor-list) )  ]

;
;
```

Sample Input

Below is a sample input file for a configuration of four nodes.

```
                        $$ENVIRONMENT-FILE$$
;
;
;          ** ENVIRONMENT-ID AND COMMENTS **

[  "b2224nj"
   "env file last modified 17-may-1982"
   "complete grammar, no noise, false and missing signals"
   "single vehicle pattern"
   "four node, sensor mask=1, power=0, total belief=500"
   "goal weight=8000, goal send/hyp reply/ no subgoaling" ]

;          ** SYSTEM DATA **
;
;               ** RANDOM SEED **

[  300  ]

;               ** MAX VELOCITY AND ACCELERATION **

[  4 2  ]

;               ** GLOBAL TUNING PARAMETERS **

[ 10000 10000 100 -9999 500 -9999 8000 10000 10000 10000 0 0 2 20 2 20 ]

;          ** GRAMMAR DEFINITION **
;
;               ** GRAMMAR-ID **

[  "grammar-1x"  ]

;               ** PATTERN-CLASS TO VEHICLE-CLASS RELATION **

[ ( 1 (( 1 (  0  0 )))) ]

;               ** VEHICLE-CLASS TO GROUP-CLASS RELATION **

[ ( 1 (  2 )) ]
```

```
;                ** GROUP-CLASS TO SIGNAL-CLASS RELATION **

[ (  2 (  2  )) ]

;            ** NODE DEFINITIONS **
;
;                ** INTEREST-AREA DEFINITIONS **

[  ( 1 ( (sl)              *all ((*all (( -4  -4   4   4)))) ))
   ( 2 ( (gl)              *all ((*all (( -4  -4   4   4)))) ))
   ( 3 ( (vl)              *all ((*all (( -4  -4   4   4)))) ))
   ( 4 ( (vt)              *all ((*all *all                 )) ))
   ( 5 ( (pt)              *all ((*all *all                 )) )) ]

;                ** KS-SET DEFINITIONS **

[  ( 1 (jb:vl:vt         10000 0 (0  0) )
       (jf:vl:vt         10000 0 (0  0) )
       (eb:vt:vt         10000 0 (0  0) )
       (ef:vt:vt         10000 0 (0  0) )
       (mb:vt            10000 0 (0  0) )
       (mf:vt            10000 0 (0  0) )
       (s:gl:vl          10000 0 (0  0) )
       (s:sl:gl          10000 0 (0  0) )
       (s:vl:vt          10000 0 (0  0) )
       (s:vt:pt          10000 0 (0  0) )
       (goal-send:vt     10000 0 (0  0) )
       (goal-receive:vt  10000 0 (0  0) )
       (hyp-reply:vt     10000 0 (0  0) )
       (hyp-receive:vt   10000 0 (0  0) )
       (sensors          10000 0 (0  0) ) ) ]

;                ** TOPOLOGICAL-ROLE DEFINITIONS **

[  ( 1 (1 2 3 4) ( ) (vt 3.0) ( ) )   ]

;                ** NODE-CLASS DEFINITIONS **

[  ( 1 1 ( -4  -4   4   4) ((1 4000)(2 4000)(3 8000)(4 9000)(5 10000))
         1 ( ((vt) (((vt vl gt gl st sl) ((1 10000)))))) ) ]

;                ** NODE DATA **

[  ( 1 1 ( 5  5) (1) (((2 3 4) (4 0 10000))) ((*all (4 0 10000 0)))
        (((2 3 4) (4 0 10000)))()(((2 3 4) (4 0 10000 0)))((1 *all)))
   ( 2 1 ( 5 13) (2) (((1 3 4) (4 0 10000))) ((*all (4 0 10000 0)))
        (((1 3 4) (4 0 10000)))()(((1 3 4) (4 0 10000 0)))((1 *all)))
   ( 3 1 (13  5) (3) (((1 2 4) (4 0 10000))) ((*all (4 0 10000 0)))
        (((1 2 4) (4 0 10000)))()(((1 2 4) (4 0 10000 0)))((1 *all)))
   ( 4 1 (13 13) (4) (((1 2 3) (4 0 10000))) ((*all (4 0 10000 0)))
```

```
        (((1 2 3) (4 0 10000)))()(((1 2 3) (4 0 10000 0)))((1 *all))) ]

;          ** SENSOR DEFINITIONS **
;
;               ** SENSOR-CLASS DEFINITIONS **

[ (1 4 ((1 1 1 1 1 1 1 1 1)
        (1 1 1 1 1 1 1 1 1)
        (1 1 1 1 1 1 1 1 1)
        (1 1 1 1 1 1 1 1 1)
        (1 1 1 1 1 1 1 1 1)
        (1 1 1 1 1 1 1 1 1)
        (1 1 1 1 1 1 1 1 1)
        (1 1 1 1 1 1 1 1 1)
        (1 1 1 1 1 1 1 1 1)) (0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
                                1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0)) ]

;               ** SENSOR DATA **

[  ( 1 1 ( 5 5 ) 10000 10000 10000)
   ( 2 1 ( 5 13) 10000 10000 10000)
   ( 3 1 ( 13 5) 10000 10000 10000)
   ( 4 1 (13 13) 10000 10000 10000)  ]

;          ** COMMUNICATION RELIABLILITY DATA **
;
;               ** HYP COMMUNICATION ERRORS **

[  ]

;               ** GOAL COMMUNICATION ERRORS **

[  ]


;          ** CONSISTENCY DATA **
;
;               ** CBB PT **

[  ( 1   ((1 (3 1))(2 (5 3))(3 (7 5))(4 (9 7))(5 (11 9))
         (6 (13 11))(7 (15 13))(8 (17 15))) t) ]

;
;               ** CBB PL **

[  ]

;               ** CBB VT **

[  ]
```

```
;                ** CBB VL **

[  ]
;                ** CBB GT **

[  ]
;                ** CBB GL **

[  ]
;                ** CBB ST **

[  ]
;                ** CBB SL **

[  ]

;              ** ENVIRONMENT DATA **

;                ** PATTERN DATA **

[ (1 ( (((1 ( 3  1))
          (2 ( 5  3))
          (5 (11  9))
          (6 (13 11))
          (7 (15 13))
          (8 (17 15))) 5000 (1 2 3 4))
        (((3 ( 7  5))
          (4 ( 9  7))) 2000 (1 2 3 4))
        (((1 ( 1  3))
          (2 ( 3  5))
          (3 ( 5  7))
          (4 ( 7  9))
          (5 ( 9 11))
          (6 (11 13))
          (7 (13 15))
          (8 (15 17))) 4100 (1 2 3 4)) ) ) ]

;                ** VEHICLE DATA **

[  ]

;                ** GROUP DATA **

[  ]

;                ** SIGNAL DATA **

[  ]
```