Negotiation as a Metaphor for Distributed Problem Solving

Randall Davis
Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139, U.S.A.

Reid G. Smith*
Defence Research Establishment Atlantic, Dartmouth, Nova Scotia B2Y 3Z7, Canada

Recommended by Lee Erman

ABSTRACT
We describe the concept of distributed problem solving *and define it as the cooperative solution of problems by a decentralized and loosely coupled collection of problem solvers. This approach to problem solving offers the promise of increased performance and provides a useful medium for exploring and developing new problem-solving techniques.*

We present a framework called the contract net *that specifies communication and control in a distributed problem solver. Task distribution is viewed as an interactive process, a discussion carried on between a node with a task to be executed and a group of nodes that may be able to execute the task. We describe the kinds of information that must be passed between nodes during the discussion in order to obtain effective problem-solving behavior. This discussion is the origin of the negotiation metaphor: Task distribution is viewed as a form of contract negotiation.*

We emphasize that protocols for distributed problem solving should help determine the content of the information transmitted, rather than simply provide a means of sending bits from one node to another.

The use of the contract net framework is demonstrated in the solution of a simulated problem in area surveillance, of the sort encountered in ship or air traffic control. We discuss the mode of operation of a distributed sensing system, a network of nodes extending throughout a relatively large geographic area, whose primary aim is the formation of a dynamic map of traffic in the area.

From the results of this preliminary study we abstract features of the framework applicable to problem solving in general, examining in particular transfer of control. Comparisons with PLANNER, CONNIVER, HEARSAY-II, *and* PUP6 *are used to demonstrate that negotiation—the two-way transfer of information—is a natural extension to the transfer of control mechanisms used in earlier problem-solving systems.*

* The author's current address is: Schlumberger–Doll Research, Ridgefield, CT 06877, U.S.A.

1. Introduction

Traditional work in problem solving has, for the most part, been set in the context of a single processor. Recent advances in processor fabrication techniques, however, combined with developments in communication technology, offer the chance to explore new ideas about problem solving employing multiple processors.

In this paper we describe the concept of *distributed problem solving*, characterizing it as the cooperative solution of problems by a decentralized, loosely coupled collection of problem solvers. We find it useful to view the process as occurring in four phases: problem decomposition, sub-problem distribution, sub-problem solution, and answer synthesis. We focus in this paper primarily on the second phase, exploring how negotiation can help in matching problem solvers to tasks.

We find three issues central to constructing frameworks for distributed problem solving: (i) the fundamental conflict between the complete knowledge needed to ensure coherence and the incomplete knowledge inherent in any distribution of problem solving effort, (ii) the need for a problem solving protocol, and (iii) the utility of negotiation as an organizing principle. We illustrate our approach to those issues in a framework called the contract net.

Section 2 describes our concept of distributed problem solving in more detail, contrasting it with the more widely known topic of distributed processing. Section 3 explores motivations, suggesting what we hope to gain from this work. In Section 4 we consider the three issues listed above, describing what we mean by each and documenting the importance of each to the problems at hand.

Section 5 describes how a group of human experts might cooperate in solving a problem and illustrates how this metaphor has proved useful in guiding our work. Section 6 then considers how a group of computers might cooperate to solve a problem and illustrates how this has contributed to our work.

Section 7 describes the contract net. We focus on its use as a framework for orchestrating the efforts of a number of loosely coupled problem solvers. More detailed issues of its implementation, as well the tradeoffs involved in its design, are covered elsewhere (see, e.g., [26, 27, 28]). Section 8 describes an application of the contract net. We consider a problem in distributed sensing and show how our approach permits a useful degree of self-organization.

Section 9 then takes a step back to consider the issue of transfer of control. We show how the perspective we have developed—notably the issue of negotiation—offers useful insights about the concept of control transfer. We review invocation techniques from a number of programming languages and illustrate that the whole range of them can be viewed as a progression from simple to increasingly more sophisticated information exchange. In these terms the negotiation technique used in the contract net becomes a natural next step.

Sections 10 and 11 consider the sorts of problems for which our approach is well suited and describe the limitations and open problems in our work to date.

2. Distributed Problem Solving: Overview

In our view, some of the defining characteristics of distributed problem solving are that it is a *cooperative* activity of a group of *decentralized* and *loosely coupled* knowledge-sources (KSs). The KSs cooperate in the sense that no one of them has sufficient information to solve the entire problem: information must be shared to allow the group as a whole to produce an answer. By decentralized we mean that both control and data are logically and often geographically distributed; there is neither global control nor global data storage. Loosely coupled means that individual KSs spend most of their time in computation rather than communication.

Interest in such problem solvers arises from the promise of increased speed, reliability, and extensibility, as well as ability to handle applications with a natural spatial or functional distribution, and the potential for increased tolerance to uncertainty in data and knowledge.

Distributed problem solving differs in several fundamental respects from the more widely known topic of *distributed processing*. Perhaps the most important distinction arises from examining the origin of the system and the motivations for interconnecting machines.

Distributed processing systems often have their origin in the attempt to synthesize a network of machines capable of carrying out a number of widely disparate tasks. Typically, several distinct applications are envisioned, with each application concentrated at a single node of the network, as for example in a three-node system intended to do payroll, order entry, and process control. The aim is to find a way to reconcile any conflicts and disadvantages arising from the desire to carry out disparate tasks, in order to gain the benefits of using multiple machines (sharing of data bases, graceful degradation, etc.).

Unfortunately, the conflicts that arise are often not simply technical (e.g., word sizes, database formats, etc.) but include sociological and political problems as well (see, e.g., [6]). The attempt to synthesize a number of disparate tasks thus leads to a concern with issues such as access control and protection, and results in viewing cooperation as a form of *compromise* between potentially conflicting desires.

In distributed problem solving, on the other hand, there is a single task envisioned for the system and the resources to be applied have no other predefined roles to carry out. We are building up a system de novo and can as a result choose hardware, software, etc. with one aim in mind: the selection that will lead to the most effective environment for cooperative behavior. This also means we view cooperation in terms of benevolent problem solving behavior, i.e., how can systems that are perfectly willing to accommodate one another act so as to be an effective team? Our concerns are thus with developing frameworks for *cooperative behavior between willing entities*, rather than frameworks for enforcing cooperation as a form of compromise between potentially incompatible entities.

A second important distinction arises from our focus on traditional issues of problem solving. We intend, for example, that the system itself should include as part of its basic task the partitioning and decomposition of a problem. Work in distributed processing, by comparison, has not taken problem solving as a primary focus. It has generally been assumed that a well-defined and a priori partitioned problem exists. The major concerns lie in an optimal static distribution of tasks, methods for interconnecting processor nodes, resource allocation, and prevention of deadlock. Complete knowledge of the problem has also been assumed (i.e., explicit knowledge of timing and precedence relations between tasks) and the major reason for distribution has been assumed to be load-balancing (e.g., [1, 2]). Since we do not make these assumptions, we cannot take advantage of this pre-planning of resources. As will become clear, this makes for significant differences in the issues which concern us and in the design of the system.

A final distinction results from the lack of substantial cooperation in most distributed processing systems. Typically, for instance, most of the processing is done at a central site and remote processors are limited to basic data collection (e.g., credit card verification). The word *distributed* is usually taken to mean spatial distribution of data—distribution of function or control is not generally considered.

One way to view the various research efforts is in terms of the three levels indicated in Fig. 1. At the lowest level the focus is the processor architecture. The main issues here are the design of the individual nodes and the interconnection mechanism. The components of an individual node must be selected (e.g., processors and memory), and appropriate low-level interconnection methods must be chosen (e.g., a single broadcast channel, complete interconnection, a regular lattice, etc.).

The middle level focuses on systems aspects. Among the concerns here are issues of guaranteeing message delivery, guaranteeing database consistency, and techniques for database recovery.

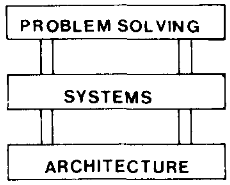

FIG. 1. A layered approach to distributed problem solving.

The focus at the top level is problem solving, where the concerns are internode control and knowledge organization; in particular how to achieve effective problem-solving behavior from a collection of asynchronous nodes. There is therefore a greater concern with the *content* of the information to be communicated between nodes than with the *form* in which the communication is effected.

All of these levels are important foci of research and each successive level depends on the ones below it for support. Our concern in this paper, however, lies primarily at the level of problem solving.

For the remainder of this paper we will assume that the hardware is a network of loosely coupled, asynchronous nodes. Each node has a local memory; no memory is shared by all nodes. Each node typically contains several distinct KSs. There is no central controller; each node makes its own choices about tasks to work on. The nodes are interconnected so that every node can communicate with every other by sending messages, perhaps over a broadcast channel. We also assume the existence of a low-level protocol to effect communication of bit streams between nodes.

3. Distributed Problem Solving: Motivation

A major motivation for this work lies in the potential it offers for making available more problem solving power, by applying a collection of processors to the solution of a single problem. It may, for example, prove much easier to coordinate the actions of twenty medium-sized machines than it is to build a single machine twenty (or even ten) times as large.

A distributed approach may also be well suited to problems that have either a spatial distribution or a large degree of functional specialization. Spatial distribution often occurs in problems involving interpretation of signal data from multiple sensors (e.g., [20]). Functional specialization may occur in problems like understanding continuous speech (e.g., [7]): information from many different knowledge-sources (e.g., signal processors, parsers, etc.) must be combined to solve the problem.

Distributed problem solving also offers a way to apply to problem solving the recent advances in both processor fabrication and communication technology. Low-cost, small-scale VLSI processors are now commonplace, with larger scale processors expected in the near future [21]. The synthesis of advanced computer and communication technology that has resulted in networks of resource-sharing computers (e.g., [13, 15]) offers a foundation for work on distributed architectures. With these two developments as foundations, work can begin focusing on techniques for effective use of networks of machines.

One reason for interest in distributed architectures in general is their capacity for reliable computation and graceful degradation. By placing problem solving in this environment, we have the chance to make it similarly reliable.

The use of an approach like the contract net, which distributes both control and data, also makes possible additional responses to component failure. In addition to the standard response of continuing to function as before (albeit more slowly), the option may exist of having the system reconfigure itself to take into account the hardware available.

Finally, and somewhat more speculatively, there is the issue of 'bounded rationality'. Some tasks appear difficult because of their size. They are 'too big' to contemplate all at once and are not easily broken into modular sub-problems (e.g., the working of the national economy, the operation of a large corporation). In such cases it may be difficult, both conceptually and practically, for a single problem solver to deal effectively with more than a small part of all of the data or knowledge required to solve the problem. Trying to scale up the hardware of a single problem solver may ease the practical problem but does not solve the conceptual difficulty. It may instead prove more effective to use multiple problem solvers, each of which handles some fraction of the total problem, and to provide techniques for dealing with the interaction between the sub-problems.

Recent work has explored a number of ideas relevant to accomplishing this goal. There is, for example, the original HEARSAY-II model of cooperating KSs ([7]), in which each KS had a sharply limited domain of expertise. It demonstrated the practicality of using a number of independent KSs to encode large amounts of knowledge about a domain. The work in [17] reports on an experiment that distributed knowledge and data, and to a limited degree, control. In Section 7 we describe an approach to distributing problem solving effort that dynamically distributes knowledge, data and control.

4. The Fundamental Issues

Our study of distributed problem solving to date has identified three issues that appear to be central to the undertaking: (i) the fundamental difficulty of ensuring global coordination of behavior when that behavior results from the aggregation of actions based on local *incomplete* knowledge, (ii) the necessity of a protocol dealing with *problem solving* rather than with *communication*, and (iii) the utility of *negotiation* as a fundamental mechanism for interaction. In this section we describe each of the issues briefly, Sections 5 and 6 then demonstrate how these issues arise from basic considerations of the task at hand.[1]

4.1. Global coherence and limited knowledge

One obvious problem that arises in employing multiple problem solvers is

[1]Other work on distributed problem solving is based on similar issues. Work described in [18], for example, also finds (i) and (ii) above to be central issues.

'coherence'. Any time we have more than one active agent in the system there is the possibility that their actions are in some fashion mutually interfering rather than mutually supportive. There are numerous ways in which this can happen. We may have conflict over resources, one agent may unknowingly undo the results of another, the same actions may be carried out redundantly, etc. In general terms, the collection of agents may somehow fail to act as a well-coordinated, purposeful team.

We believe that this problem is due to the fundamental difficulty of obtaining coordinated behavior when each agent has only a limited, local view. We could, of course, guarantee coordination if every agent 'knew everything', i.e., it had complete knowledge. If, for example, every problem solver had complete knowledge of the actions of all the others, it would be possible to avoid redundant or conflicting efforts.[2]

Yet any reasonable model of distribution appears to require incomplete, local views of the problem. Complete information is, for example, at least impractical. As we argue in Section 6, bandwidth limitations make it unreasonable to consider having every node constantly informed of all developments.

A limited local view also simplifies the problem conceptually. The problem becomes far more difficult to think about (and to program) if every problem solver has to keep track of everything. It also seems contrary to the basic notion of distribution: Part of the motivation is to allow a problem solver to focus on one part of the problem and ignore the rest.

For these reasons at least, then, any distribution of problem solving effort appears to imply incomplete, local knowledge.

And when we say "incomplete knowledge", we include in "knowledge" the information indicating "who needs to know what". That is, we do not assume that we start out with a map of subproblems and their interactions. Without such a map, there is the chance that necessary interactions are overlooked and hence we lose a guarantee of coordinated behavior.

As noted earlier, we consider problem decomposition—the creation of the map of subproblems—to be part of the system's task. Once the system creates its best guess at such a map, we can count on the locality of action and information to make distributed problem solving practical. By locality of action and information, we mean that the problems typically attacked in AI are generally decomposable into a set of subproblems in which the effects of actions and the relevance of information is local. The actions taken to solve one subproblem generally affect only a few other subproblems; the

[2]This difficulty is not limited to distributed problem solving, it is only more painfully obvious there. The standard notion of problem decomposition in centralized systems results in limited, local knowledge, and the same difficulty manifests itself as the well-known problem of interacting subgoals.

information discovered in solving one subproblem is generally relevant to only a few other subproblems.[3] As a result, each problem solver will have to interact with at most a few others, making limited bandwidth a challenging but not fatal constraint.

To summarize: the conflict arises because distribution seems by its nature to require supplying each problem solver with only a limited, local view of the problem, yet we wish to accomplish a global effect—the solution of the problem at hand. It is not obvious how we can guarantee overall coordination from aggregations of actions based on local views with incomplete information. Thus, while the locality of action and information means that distributed problem solving is feasible, the necessity of incomplete knowledge means that guaranteeing coordinated activity is difficult.

One general answer is to provide something that extends across the network of nodes, something that can be used as a foundation for cooperation and organization. As will become clear, three elements of our framework help provide that foundation: (i) the concept of negotiation as a mechanism for interaction (Section 7.1), (ii) the network of tasks that results from decomposing a problem (Section 8.2), and (iii) a common language shared by all nodes (Section 7.4). The announcement—bid—award sequence of messages (Section 7.3) also offers some support. Even though each problem solver has only a limited view of the problem, these messages offer one way for a node to find out who else has relevant information. Together, all of these mechanisms provide an initial step toward a basis for achieving coordinated behavior.

4.2. The need for a problem solving protocol

In most work on protocols for distributed computation the emphasis has been on establishing reliable and efficient communication. Some degree of success has been achieved, at levels ranging from individual packets to atomic actions (see, e.g., [29]). But these protocols are only a prerequisite for distributed problem solving. In the same sense that communication among a group of entities needs a carefully constructed communication protocol, so problem solving by a group of entities requires a problem solving protocol. Cooperation cannot be established between nodes simply by indicating how they are to communicate; we must also indicate what they should say to each other.

The issue can also be viewed in the terms suggested by Fig. 1. At each level we need to give careful consideration to the basic architecture and we need the

[3]The first half of this observation—the locality of the effects of actions—is typically used to justify informal solutions to the frame problem. We can, for instance, account for the effects of an action with a list of consequences, because that list tends to be short and predictable.

Similarly, the impact of information tends to be local. If I, as one member of a team, am working on one part of a problem, most of what is discovered about the rest of the problem is irrelevant to me. Keeping me up to date on every detail will only prove to be a distraction.

appropriate protocols. In the same sense that we pay attention to hardware and systems architecture, so we need to consider a 'problem solving architecture'; as we have protocols that organize the communication of bits and files, so we need protocols to organize the problem solving activity.

As discussed in Section 7, the contract net takes a first step in this direction by providing a set of message types indicating the kind of information that nodes should exchange in order to effect one form of cooperation.

4.3. The utility of negotiation

The central element in our approach to a problem solving protocol is the concept of negotiation. By negotiation, we mean a discussion in which the interested parties exchange information and come to an agreement. For our purposes negotiation has three important components: (a) there is a two-way exchange of information, (b) each party to the negotiation evaluates the information from its own perspective, and (c) final agreement is achieved by mutual selection.

Negotiation appears to have multiple applications. In Section 7, for example, we explore its application to the problem of matching idle problem solvers to outstanding tasks. This matching is carried out by the system itself, since, as noted, we do not assume that the problem has already been decomposed and distributed.

In Section 9.2 we explore a second application of negotiation by considering its utility as a basis for transfer of control and as a way of viewing invocation as the matching of KSs to tasks. This view leads to a more powerful mechanism for control transfer, since it permits a more informed choice from among the alternative KSs which might be invoked. The view also leads to a novel perspective on the outcome of the interaction. In most previous systems, the notion of selecting what to do next typically involves taking the best choice from among those currently available. As will become clear, in the contract net either party has the option of deciding that none of the currently available options is good enough, and can decide instead to await further developments.

5. A Cooperating Experts Metaphor

A familiar metaphor for a problem solver operating in a distributed environment is a group of human experts experienced at working together, trying to complete a large task.[4] Of primary interest to us in examining the operation of a group of human experts are: (a) the way in which they interact to solve the overall problem, (b) the manner in which the workload is distributed among them, and (c) how results are integrated for communication outside the group.

[4]This metaphor has been used as a starting point by [11], [16] and [18], but has resulted in systems that differ from ours in several ways. The different systems are compared in Section 9.

For reasons discussed above, we assume that no one expert is in total control of the others, although one expert may be ultimately responsible for communicating the solution of the top-level problem to the customer outside the group.

One possible model for the interaction involves group members cooperating in the execution of individual tasks, a mode we have called 'task-sharing' [28]. In such a situation we might see each expert spending most of his time working alone on various subtasks, pausing only occasionally to interact with other members of the group. These interactions generally involve requests for assistance on subtasks or the exchange of results.

An expert (E1) may request assistance because he encounters either a task too large to handle alone, or a task for which he has no expertise. If the task is too large, he will first attempt to partition it into manageable subtasks and then attempt to find other experts who have the appropriate skills to handle the new tasks. If the original task is beyond his expertise, he attempts right away to find another, more appropriate expert to handle it.

In either case, E1's problem is now to find experts whose skills match the tasks that he wishes to distribute. If E1 knows which other experts have the necessary expertise, he can notify them directly. If he does not know anyone in particular who may be able to assist him (or if the tasks require no special expertise), he can simply describe the tasks to the entire group.

If another, available expert (E2) believes he is capable of carrying out the task that E1 announced, he informs E1 of his availability and perhaps indicates as well any especially relevant skills he may have. E1 may wind up with several such volunteers and can choose from among them. The chosen volunteer might then request additional details from E1 and the two will engage in further direct communication for the duration of the task.

In order to distribute the workload in a group of experts, then, those with tasks to be executed must find others capable of executing those tasks. At the same time, it is the job of idle experts to find suitable tasks on which to work. Those with tasks to be executed and those capable of executing the tasks thus engage in a form of *negotiation* to distribute the workload. They become linked together by agreements or informal contracts, forming subgroups of varying sizes that are created and broken up dynamically during the course of work.[5]

6. Observations and Implications

The metaphor of a group of human experts offered several suggestions about

[5]Subgroups of this type offer two advantages. First, communication among the members does not needlessly distract the entire group. This is important, because communication itself can be a major source of distraction and difficulty in a large group (see for example [9]). Thus one of the major purposes of organization is to reduce the amount of communication that is needed. Second, the subgroup members may be able to communicate with each other in a language that is more efficient for their purpose than the language in use by the entire group (for more on this see [27]).

organizing problem solving effort. Here we consider how a group of computers might cooperate and examine what that can tell us about how to proceed. We approach this by comparing the use of multiple, distributed processors with the more traditional model of operation on a uniprocessor. We list several basic observations characterizing the fundamental differences and consider the implications that follow. While the list is not exhaustive, it deals with the differences we find most important.

Communication is slower than computation.

That is, bits can be created faster than they can be shipped over substantial distances.[6] With current technology, communication over such distances is in fact much slower than computation. Attempting to interconnect large numbers of high speed processors can easily lead to saturation of available bandwidth. Present trends indicate [23] that this imbalance in speed will not only continue, but that the disparity is likely to increase. It appears as well that the relative costs of communication and computation will follow a similar trend.

Several implications follow from this simple observation (Fig. 2). It means for example that we want problem decompositions that yield *loosely coupled* systems—systems in which processors spend the bulk of their time computing and only a small fraction of their time communicating with one another. The desire for loose coupling means in turn that we need to pay attention to the *efficiency* of the communication protocol: With a more efficient protocol, fewer bits need to be transmitted and less time is spent in communicating. It also means that we need to pay attention to both the *modularity* and *grain size* of the problems chosen. Problems should be decomposed into tasks that are both independent and large enough to be worth the overhead involved in task distribution. Non-independent tasks will require communication between processors, while for very small tasks (e.g., simple arithmetic) the effort involved in distributing them and reporting results would likely be greater than the work involved in solving the task itself.

```
Communication is slower than computation
→ loose-coupling
   → efficient protocol
   → modular problems
   → problems with large grain size
```

FIG. 2. Observations and implications.

[6]Over short distances, of course, permanent hardwired links can be very effective. Where distances are large or varying (e.g., mobile robots), bandwidth again becomes a limiting factor.

Note also that we mean communicating all the bits involved in a computation, not just the final answer. Otherwise communicating, say, one bit to indicate the primality of a 100-digit number would surely be faster than doing the computation to determine the answer.

We have argued above for loose coupling and based the argument on technological considerations. The point can be argued from two additional perspectives as well. First, the comments earlier concerning the locality of action and information suggest that, for the class of problems we wish to consider, tight coupling is unnecessary. The activities and results of any one problem solver are generally relevant to only a few others. More widespread dissemination of information will mostly likely only prove to be distracting.

A second argument, described in [18], takes a more emphatic position and argues for loose coupling even where it is known to produce temporary inconsistencies. They note that standard approaches to parallelism are typically designed to ensure that all processors always have mutually consistent views of the problem. Such complete consistency, and the tight coupling it requires, is, they claim, unnecessary. They suggest instead that distributed systems can be designed to be 'functionally accurate', i.e., the system will produce the correct answer eventually even though in an intermediate state some processors may have inconsistent views of the problem.

Thus we have arguments against tight coupling based on technological considerations (the communication/computation imbalance), pragmatic issues (the locality of action and information), and empirical results which suggest that it may be unnecessary.

Any unique node is a potential bottleneck.

Any node with unique characteristics is potentially a bottleneck that can slow down the system (Fig. 3). If those characteristics make the distinguished node useful to enough other nodes in the system, eventually those nodes may be forced to stand idle while they wait for service. This is equally true for a resource like data (for which the issue has been extensively studied) and a 'resource' like control (for which considerably less work has been done). If one node were in charge of directing the activities of all other nodes, requests for decisions about what to do next would eventually accumulate faster than they could be processed.[7]

What steps can we take to reduce the likelihood of bottlenecks due to centralized control? First, we can distribute it: Each node should have some

Any unique node is a potential bottleneck
→ distribute data
→ distribute control
  → organized behavior is hard to guarantee

FIG. 3. Further observations and implications.

[7]Such a node would also be an Achilles' heel in the system, since its failure would result in total failure of the system.

degree of autonomy in generating new tasks and in deciding which task to do next. By so dividing up and distributing the responsibility for control, we reduce the likely load on any one node. Second, we might distribute it redundantly: If more than one node is capable of making decisions about control, we further reduce the likelihood that any one node becomes saturated, and can ensure that no one node is unique. Finally, we can distribute control dynamically: We might provide a mechanism that allows dynamic redistribution in response to demands of the problem.

Organized behavior is difficult to guarantee if control is decentralized.

In a system with completely centralized control, one processor is responsible for directing the activities of all the others. It knows what all the other processors are doing at any given time, and, armed with this global view of the problem, can assign processors to tasks in a manner that assures organized behavior of the system as a whole. By 'organized', we mean that (among other things) all tasks will eventually be attended to, they will be dealt with in an order that reduces or eliminates the need for one processor to wait for results from another, processor power will be well matched to the tasks generated, etc. In more general terms, the set of processors will behave like a well-coordinated, purposeful team.

In the absence of a global overview, coordination and organization becomes much more difficult. When control is decentralized, no one node has a global view of all activities in the system; each node has a local view that includes information about only a subset of the tasks. The appropriate organization of a number of such subsets does not necessarily result in appropriate organization and behavior of the system as a whole.

In Section 4 we described the general problem of ensuring well-coordinated behavior; this is a specific instantiation of that problem with respect to control. We are trying to achieve a global effect (coherent behavior) from a collection of local decisions (nodes organizing subsets of tasks). We cannot centralize control for reasons noted above, yet it is not clear how to ensure coherent behavior when control is distributed.

7. A Framework for Distributed Problem Solving

7.1. A view of distributed problem solving

We view distributed problem solving as involving four central activities: problem decomposition, sub-problem distribution, solution of sub-problems, and synthesis of the overall solution. By decomposition we mean the standard notion of breaking a large problem into smaller, more manageable pieces; distribution involves the matching of sub-problems with problem solvers capable of handling them; the sub-problems are then solved; and finally those individual solutions may need to be synthesized into a single, overall solution.

Each of these can happen several times as a problem is decomposed into several levels of subproblems.

These four activities may occur individually and in the sequence noted, or may be combined or carried out in parallel. The point is simply that all of them can make important contributions to the problem solving process, so we need some mechanism for dealing with each.[8]

7.2. Task-sharing, negotiation and the connection problem

We have emphasized above the importance of having a protocol for organizing problem solving activity and proposed negotiation as a plausible basis for that protocol. But what shall we negotiate? Our work to date has followed the lead suggested by the cooperating experts metaphor and explored the distribution of tasks as an appropriate subject. Thus, in this paper we focus on application of the contract net to the distribution phase of distributed problem solving and show how negotiation appears to be an effective tool for accomplishing the matching of problem solvers and tasks.

To illustrate this, recall that the group of experts distributed a problem by decomposing it into ever smaller subtasks and distributing the subtasks among the group. We term this mode of operation 'task-sharing', because cooperation is based on the dynamic decomposition and distribution of subproblems.[9] But to enable distribution of the subproblems, there must be a way for experts with tasks to be executed to find idle experts capable of executing those tasks. We call this the 'connection problem'.

The contract net protocol supplies a mechanism to solve the connection problem: As we will see, nodes with tasks to be executed negotiate with idle nodes over the appropriate matching of tasks and nodes.

This approach is appropriate for a distributed problem solver because it requires neither global data storage nor global control. It also permits some degree of dynamic configuration and reconfiguration. A simple example of dynamic configuration is given in Section 8.3; reconfiguration is useful in the event of node failure or overloading. We have explored a number of simple mechanisms for detecting node failure and reconfiguring in response [26, 22], but the problem is not yet well studied.

A few words of terminology will be useful. The collection of nodes is referred to as a *contract net*. Each node in the net may take on the role of a *manager* or a *contractor*. A manager is responsible for monitoring the execu-

[8]For some problems the first or last activity may be trivial or unnecessary. Where a problem is geographically distributed, for example, the decomposition may be obvious (but see the discussion of the sensor net in Section 8). In problems of distributed control (e.g., traffic light control), there may be no need to synthesize an 'overall' answer.

[9]Task-sharing in its simplest form can be viewed as the distributed version of the traditional notion of problem decomposition. For a different approach to distribution, see [18].

tion of a task and processing the results of its execution. A contractor is responsible for the actual execution of the task.[10]

Individual nodes are not designated *a priori* as managers or contractors; these are only roles, and any node can take on either role dynamically during the course of problem solving. Typically a node will take on both roles, often simultaneously for different contracts. This has the advantage that individual nodes are not statically tied to a control hierarchy.

For the sake of exposition, we describe the protocol in successive layers of detail, describing first the *content* of the messages exchanged (Section 7.3), then their *format* (Section 7.4), and finally the details of the *language* in which they are written (Section 7.5).

7.3. Contract net protocol—message content

Message content is the heart of the issue, since it indicates what kinds of things nodes should say to one another and provides the basis for cooperation.

Negotiation is initiated by the generation of a new task. As suggested in the experts metaphor, this may occur when one problem solver decomposes a task into sub-tasks, or when it decides that it does not have the knowledge or data required to carry out the task. When this occurs, the node that generates the task advertises existence of the task with a *task announcement* message (Fig. 4). It then acts as the manager of that task for its duration. Many such announcements are made over the course of time as new tasks are generated.

Meanwhile, nodes in the net are listening to the task announcements (Fig. 5). They evaluate their own level of interest in each task with respect to their specialized resources (hardware and software), using *task evaluation procedures* specific to the problem at hand.[11]

When a task is found to be of sufficient interest, a node submits a bid (Fig. 6). A bid message indicates the capabilities of the bidder that are relevant to execution of the announced task.

A manager may receive several bids in response to a single task announcement (Fig. 7). Based on the information in the bids, it selects one or more nodes for execution of the task, using a task-specific *bid evaluation procedure*.

The selection is communicated to the successful bidders through an *award* message (Fig. 8). The selected nodes assume responsibility for execution of the task, and each is called a contractor for that task.

A contractor will typically partition a task and enter into (sub)contracts with

[10]The basic idea of contracting is not new. For example, a rudimentary bidding scheme was used for resource allocation in the Distributed Computing System (DCS) [8]. The contract net takes a wider perspective and allows a broader range of descriptions to be used during negotiation. For a detailed discussion see [27].

[11]It is in general up to the user to supply this and other task-specific procedures, but useful defaults are available (see [26]).

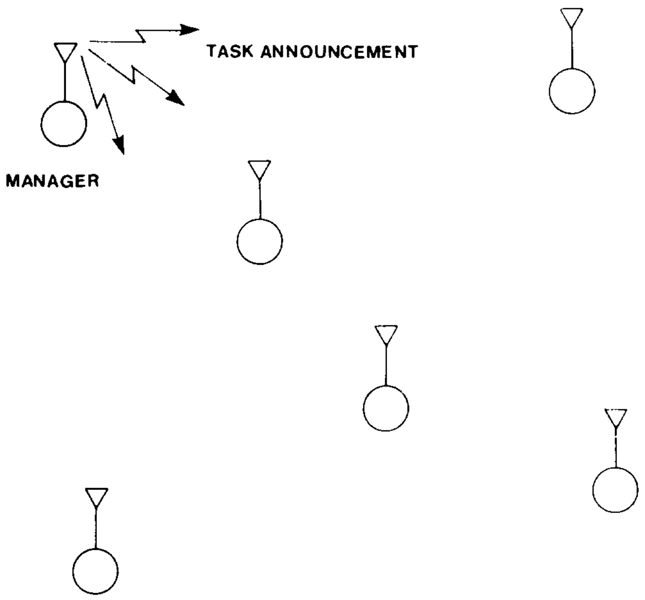

FIG. 4. Node issuing a task announcement.

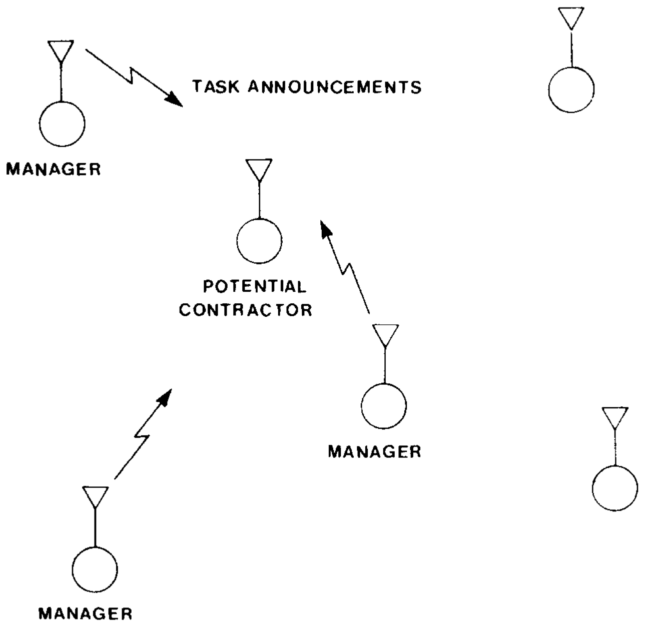

FIG. 5. Idle node listening to task announcements.

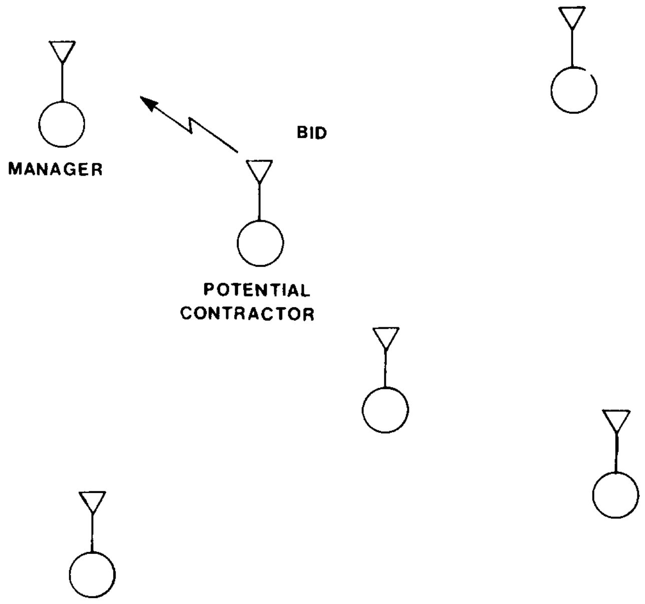

FIG. 6. Node submitting a bid.

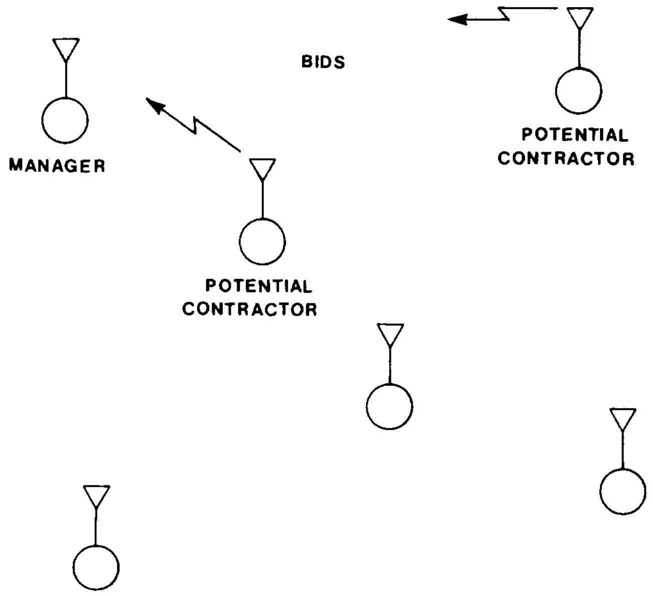

FIG. 7. Manager listening to bids coming in.

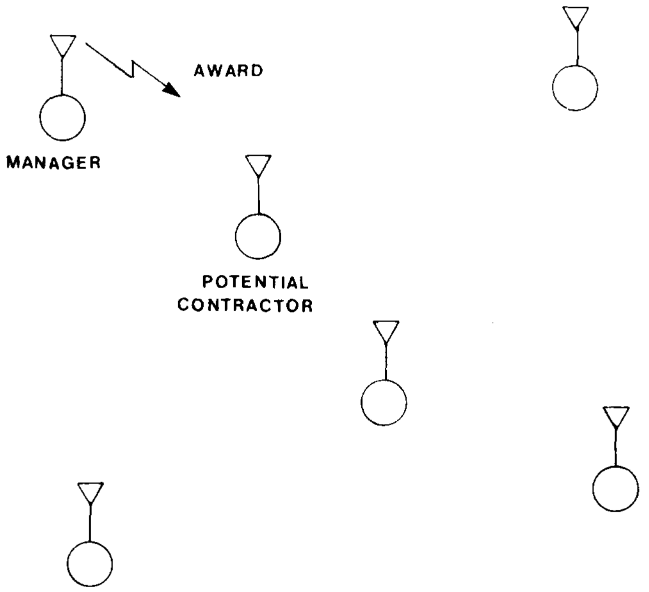

FIG. 8. Manager making an award.

other nodes. It is then the manager for those contracts. This leads to the hierarchical control structure that is typical of task-sharing.

A report is used by a contractor to inform its manager that a task has been partially executed (an *interim report*) or completed (a *final report*). The report contains a *result description* that specifies the results of the execution.[12]

The manager may terminate contracts with a *termination* message. The contractor receiving such a message terminates execution of the contract and all related outstanding subcontracts.

A contract is thus an explicit agreement between a node that generates a task (the manager) and a node that executes the task (the contractor, Fig. 9). Note that establishing a contract is a process of mutual selection. Available contractors evaluate task announcements until they find one of interest; the managers then evaluate the bids received from potential contractors and select the ones they determine to be most appropriate. Both parties to the agreement have evaluated the information supplied by the other and a mutual selection has been made.

We have dealt here with a simple example in order to focus on the issue of

[12]Interim reports are useful when generator-style control is desired. A node can be set to work on a task and instructed to issue interim reports whenever the next result is ready. It then pauses, awaiting a message that instructs it to continue and produce another result.

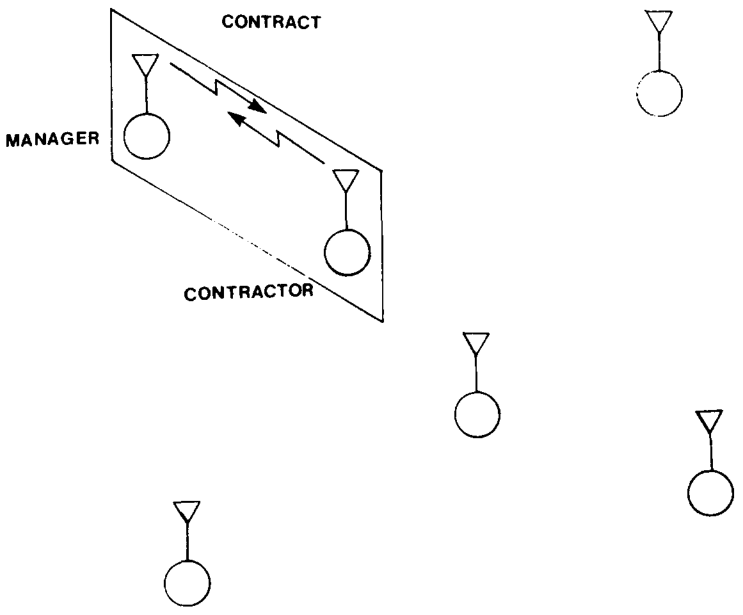

FIG. 9. A contract established.

cooperation. Additional complications which arise in implementing the protocol are discussed in detail in [26]; we note them briefly here for reference. *Focused addressing* is a more direct communication scheme used where the generality of broadcast is not required. *Directed contracts* are used when a manager knows which node is appropriate for a task. A *request-response* mechanism allows simple transfers of information without the overhead of contracting. And finally, a *node-available* message allows reversal of the normal negotiation process: When the computation load on the net is high, most task announcements will not be answered with bids because all nodes will already be busy. The node-available message allows an idle node to indicate that it is searching for a task to execute. The protocol is thus load-sensitive in response to changing demands of the task: When the load is low, the spawning of a task is the important event; when the load is high, the availability of a node is important.

7.4. Contract net protocol—message format

Each message is composed of a number of slots that specify the kind of information needed in that type of message. A task announcement message, for example, has four main slots (Fig. 10).[13] The *eligibility specification* is a list

[13]There are also slots that contain bookkeeping information.

Main Task Announcement Slots

Eligibility specification
Task abstraction
Bid specification
Expiration time

FIG. 10. Task announcement format.

of criteria that a node must meet to be eligible to submit a bid. The *task abstraction* is a brief description of the task to be executed. It enables a node to rank the announced task relative to other announced tasks. The *bid specification* is a description of the expected form of a bid. It gives a manager a chance to say, in effect, "Here's what I consider important about a node that wants to bid on this task." This provides a common basis for comparison of bids, and enables a node to include in a bid only the information about its capabilities that are relevant to the announced task. Finally, the *expiration time* is a deadline for receiving bids.

For any given application, the information that makes up the eligibility specification, etc., must be supplied by the user. Hence while the contract net protocol offers a framework specifying the types of information that are necessary, it remains the task of the user to supply the actual information appropriate to the domain at hand.

7.5. Contract net protocol—the common internode language

Finally, we need a language in which to specify the information in the slots of a message. For a number of reasons, it is useful to specify a single, relatively high level language in which all such information is expressed. We call this the *common internode language.* This language forms a common basis for communication among all the nodes.

As an example, consider a task announcement message that might be used in a system working on a signal processing task. Assume that one node attempting to analyze a signal determines that it would be useful to have a Fourier transform of that signal. Unwilling or unable to do the task itself (perhaps because of hardware limitations), it decides to announce the task in order to solicit assistance. It might issue a task announcement of the sort shown in Fig. 11.

The announcement is broadcast to all nodes within range ("To: *"), and indicates that there is a TASK of TYPE FOURIER-TRANSFORM to be done. In order to consider bidding on it a node must have an FFTBOX and a bid should specify estimated time to completion of the task.

The common internode language is currently built around a very simple *attribute, object, value* representation. There are a number of predefined

To: *
From: 25
Type: TASK ANNOUNCEMENT
Contract: 43-6

Eligibility Specification
MUST-HAVE FFTBOX

Task Abstraction
TASK TYPE FOURIER-TRANSFORM
NUMBER-POINTS 1024
NODE NAME 25
POSITION LAT 64N LONG 10W

Bid Specification
COMPLETION-TIME

Expiration Time
29 1645Z NOV 1980

FIG. 11. Task announcement example.

(domain-independent) terms (like TYPE of TASK); these are supplemented with domain-specific terms (like FFTBOX). The domain-independent terms are part of the language offered to the user and help him organize and specify the information he has to supply. The domain-specific terms have to be added by the user as needed for the application at hand.

All of this information is stated in terms of something we here called a common internode language. The two important points here are that the information in messages is viewed as statements in a language, and that the language is common to all the nodes.

It is useful to view the messages as statements in a language because this sets the appropriate perspective on the character of the interaction we are trying to achieve. Viewing the message exchange as, say, pattern matching would lead to a much more restricted form of communication: A pattern either matches or fails; if it succeeds the only information available comes from the bindings of pattern variables. Viewing the messages as statements in a language offers the chance for a more interesting exchange of information, since the nodes are examining and responding to the messages, not simply matching patterns. In particular, we find the two-way exchange of information an important capability (see Section 9).

It is useful to identify a common 'core' language shared by all the nodes. This makes it much easier to add new nodes to the net. Any new node, preloaded with only the common internode language, can use that language to isolate the information it needs to begin to participate in solving the problem at hand. It can listen to and understand task announcements and express a request for the transfer of any required information. If there were a number of distinct internode languages, then a new node entering the net could interact with only a limited subset of the nodes, those which spoke its language.[14] This would make addition of new nodes to the net less effective.

A common language also makes possible invocation schemes that are more flexible than standard procedure invocation, and this also facilitates addition of a new node to the net. For example, a common language makes it possible to use invocation based on describing tasks to be done,[15] rather than naming specific KSs (procedures) to invoke next. When this technique is used, new nodes can simply be added to the existing collection; they will find their own place in the scheme of things by listening to task announcements, issuing bids, etc. With more traditional invocation schemes (e.g., standard procedure calling), a new node would have to be linked explicitly to others in the network.

8. Example: Distributed Sensing

The protocol described above has been implemented in INTERLISP and used to solve several problems in a simulated multi-processor environment. The problems included search (e.g., the 8-queens problem) and signal interpretation (for details see [26]). In this section we describe use of the contract net on one such problem in signal interpretation: area surveillance of the sort encountered in air or ship traffic control. We explore the operation of a network of nodes, each having either sensing or processing capabilities and all spread throughout a relatively large geographic area. We refer to such a network as a distributed sensing system (DSS).

Although an operational DSS may have several functions, ranging from passive analysis to active control over vehicle courses and speeds, we focus here on the analysis function. The task involves detection, classification, and tracking of vehicles; the solution to the problem is a dynamic map of traffic in the area. Construction and maintenance of the map requires interpretation of the large quantity of sensory information received by the collection of sensor elements.

Since we want to produce a single map of the entire area, we may choose to have one processor node—which we will call the monitor node—carry out the final integration of information and transmit it to the appropriate destination. It is also useful to assign that node the responsibility for beginning the initialization of the DSS. Its total set of responsibilities therefore includes starting the initialization as the first step in net operation, integrating the overall

[14]Note that the extreme case (in which every pair of nodes communicates in its own private language) is precisely standard procedure invocation. To decode a procedure call, one must know the expected order, type, and number of arguments. This is information which is shared only by the caller and procedure involved, in effect a private language used for communication between them.

[15]As is also done in PLANNER and the other pattern-directed languages.

map as the last step in analysis, and then communicating the result to the appropriate agent. We will see that this monitor node does not, by the way, correspond to a central controller.

Since the emphasis in this work has been on organizing the problem solving activities of multiple problem solvers, work on the signal interpretation aspects did not include construction of low-level signal processing facilities. Instead it assumed the existence of appropriate signal processing modules and focused on the subsequent symbolic interpretation of that information.

8.1. Hardware

All communication in the DSS is assumed to take place over a broadcast channel (using for example packet radio techniques [14]). The nodes are assumed to be in fixed positions known to themselves but not known a priori to other nodes in the net. Each node has one of two capabilities: sensing or processing. The sensing capability includes low-level signal analysis and feature extraction. We assume that a variety of sensor types exists in the DSS, that the sensors are widely spaced, and that there is some overlap in sensor area coverage. Nodes with processing capability supply the computation power necessary to effect the high-level analysis and control in the net. They are not necessarily near the sensors whose data they process.

Fig. 12 is a schematic representation of a DSS.

In the example that follows, some assumptions about such things as node locations, what one node knows about another, etc., may seem to be carefully chosen rather than typically what one would expect to find. This is entirely true. We have combined a number of plausible but carefully chosen (and occasionally atypical) assumptions about hardware and software available in order to display a number of the capabilities of the contract net in a single, brief example.

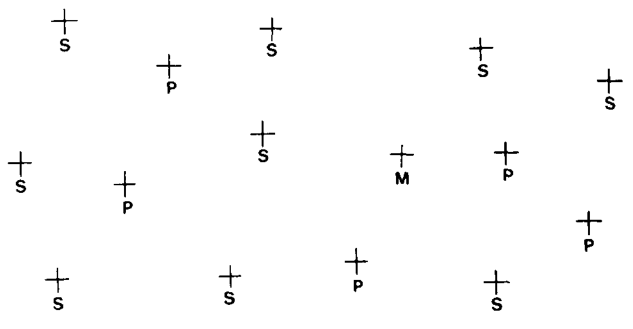

FIG. 12. A distributed sensing system. M: monitor node; P: processor node; S: sensor node.

8.2. Data and task hierarchy

The DSS must integrate a large quantity of data, reducing it and transforming it into a form meaningful to a human decision maker. We view this process as occurring in several stages, which together form a data hierarchy (Fig. 13).

As we have chosen to solve the problem for this illustration, at any given moment a particular node handles data at only one level of the data hierarchy, but may communicate with nodes at other levels. In addition, the only form of signal processing we consider is narrow band spectral analysis.[16]

At the bottom of the hierarchy we have audio *signals*, which are described in terms of several features: frequency, time of detection, strength, changes in strength, name and position of the detecting node, and name, type, and orientation of the detecting sensor.

Signals are formed into *signal groups*, collections of related signals. One common signal group is the harmonic set, a collection of signals in which the frequency of each signal is an integral multiple of the lowest frequency. In the current example, a signal group is described in terms of its fundamental frequency, time of formation, identity of the detecting node, and features of the detecting sensor.

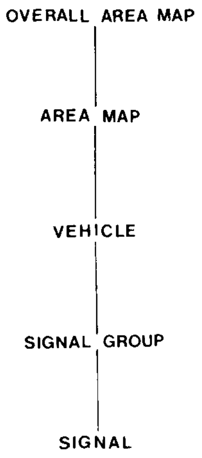

FIG. 13. Data hierarchy.

[16]Noise radiated by a vehicle typically contains narrow band signal components caused by rotating machinery. The frequencies of such signals are correlated with the type of rotating machine and its speed of rotation; hence they are indicators of the classification of the vehicle. Narrow band signals also undergo shifts in frequency due to Doppler effect or change in the speed of rotation of the associated machine; hence they also provide speed and directional information. (Unfortunately, alterations in signal strength occur both as a result of propagation conditions and variations in the distance between the vehicle and the sensor.)

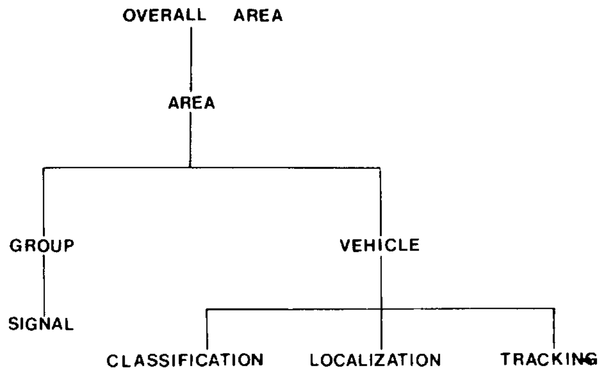

FIG. 14. Task hierarchy.

The next level of the hierarchy is the description of the *vehicle*. It has one or more signal groups associated with it and is further specified by position, speed, course, and classification. Position can be established by triangulation, using matching groups detected by several sensors with different positions and orientations. Speed and course must generally be established over time by tracking.

The *area map* forms the next level of the data hierarchy. It contains information about the vehicle traffic in a given area. There will be several such maps for the DSS—together they span the total area of coverage of the system.

The final level is the complete or *overall area map*, produced in this example by the monitor, which integrates information in the individual area maps.

The hierarchy of tasks, Fig. 14, follows directly from the data hierarchy. The monitor node manages several *area* contractors. These contractors are responsible for the formation of traffic maps in their immediate areas. Each area contractor, in turn, manages several *group* contractors that provide it with signal groups for its area. Each group contractor integrates raw signal data from *signal* contractors that have sensing capabilities.

The area contractors also manage several *vehicle* contractors that are responsible for integrating information about individual vehicles. Each of these contractors manages a *classification* contractor that determines vehicle type, a *localization* contractor that determines vehicle position, and a *tracking* contractor that tracks the vehicle.

8.3. Contract net implementation

There are two phases to this problem: initialization of the net and operation. Although there are interesting aspects to both of these phases, our concern here is primarily with initialization, since this phase most easily illustrates the transfer of control issues that form one focus of this paper. The operation phase is dealt with only briefly; for further discussion see [25].

The terminology in the discussion that follows highlights the fact that the nodes in the contract net play a dual role: They are simultaneously contractors obligated to carry out a task that they were awarded, and managers for any tasks which they in turn announce. For example, node number 2 in Fig. 15 is simultaneously (i) a *contractor* for the area task (and hence is charged with the duty of producing area maps from vehicle data), (ii) a *manager* for group formation tasks which it announces and contracts out, and (iii) a *manager* for any vehicle tasks which it contracts out. Nodes are thus simultaneously both workers and supervisors. (Compare Fig. 14 and Fig. 15.)

8.3.1. *Initialization*

The monitor node is responsible for initialization of the DSS and for formation of the overall map. It must first select nodes to be area contractors and

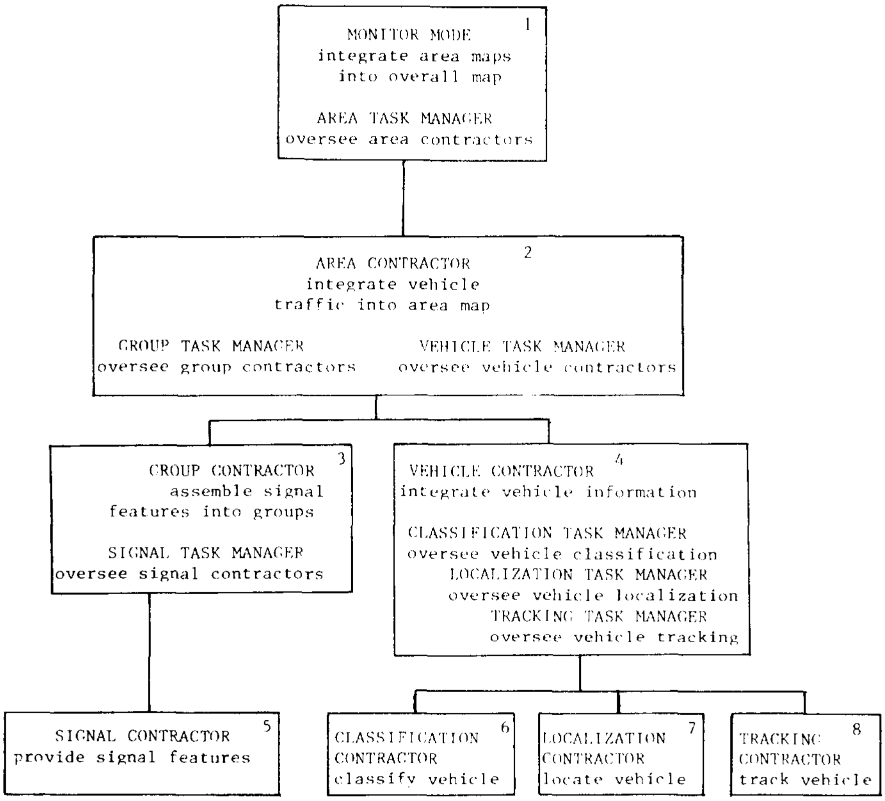
MONITOR NODE 1
integrate area maps into overall map
AREA TASK MANAGER
oversee area contractors
AREA CONTRACTOR 2
integrate vehicle traffic into area map
GROUP TASK MANAGER
oversee group contractors
VEHICLE TASK MANAGER
oversee vehicle contractors
GROUP CONTRACTOR 3
assemble signal features into groups
SIGNAL TASK MANAGER
oversee signal contractors
VEHICLE CONTRACTOR 4
integrate vehicle information
CLASSIFICATION TASK MANAGER
oversee vehicle classification
LOCALIZATION TASK MANAGER
oversee vehicle localization
TRACKING TASK MANAGER
oversee vehicle tracking
SIGNAL CONTRACTOR 5
provide signal features
CLASSIFICATION CONTRACTOR 6
classify vehicle
LOCALIZATION CONTRACTOR 7
locate vehicle
TRACKING CONTRACTOR 8
track vehicle

FIG. 15. Nodes and their roles.

partition the system's span of coverage into areas based on the positions of the nodes selected. For purposes of illustration we assume that the monitor node knows the names of the nodes that are potential area contractors, but must establish their positions in order to do the partitioning.

It begins by announcing the task of area map formation. Because it knows the names of potential contractors, it can avoid using a general broadcast and instead uses focused addressing. The components of the announcement of interest here are the task abstraction, the eligibility specification, and the bid specification. The task abstraction is simply the task type. The eligibility specification is blank, since in this case the monitor node knows which nodes are potential contractors and can address them directly. The bid specification informs a prospective area contractor to respond with its position.

Recall that the purpose of a bid specification is to inform a node of how to bid so that a manager can select from all of the bidders the most appropriate one(s) to execute the task. In this case, node position is the relevant information. Potential area contractors respond with their positions, and, given that information, the monitor node can partition the overall span of coverage into approximately equal-sized areas. It then selects a subset of the bidders to be area contractors, informing each of its area of responsibility in an award message. The negotiation sequence thus makes available to the monitor node the positions of all of the potential area contractors, making possible a partitioning of the overall area of the DSS based on these positions. This in turn enables the DSS to adjust to a change in the number or position of potential area contractors.

Area contractors integrate vehicle data into area maps. They must first establish the existence of vehicles on the basis of group data. To do this, each area contractor solicits other nodes to provide that data. In the absence of any information about which nodes are suitable, each area contractor announces the task using a general broadcast. The task abstraction in this message is the type of task. The eligibility specification is the area for which the area contractor is responsible.[17] The bid specification is again node position. Potential group contractors respond with their respective positions, and based on this information the area contractors award contracts to nodes in their areas of responsibility.

The group contractors integrate signal features into groups, and start by finding a set of contractors to provide the signal features. Recall that we view node interaction as an agreement between a node with a task to be done and a node capable of performing that task. Sometimes the perspective on the ideal character of that agreement differs depending on the point of view of the

[17]This ensures that a node is eligible to bid on this task only if it is in the same area as the announcing area contractor and helps to prevent a case in which a group contractor is so far away from its manager that reliable communication is difficult to achieve.

participant. For example, from the perspective of the signal task *managers*, the best set of contractors would have an adequate spatial distribution about the surrounding area and an adequate distribution of sensor types. From the point of view of the signal task *contractors*, on the other hand, the ideal match involves finding managers that are closest to them (in order to minimize potential communication problems).

The ability to express and deal with such disparate viewpoints is one advantage of the contract net framework. To see how the appropriate resolution is accomplished, consider the messages exchanged between the signal managers and potential signal contractors. Each signal manager announces its own signal task, using a message of the sort shown in Fig. 16. The task abstraction is the type of task, the position of the manager making the announcement, and a specification of its area of responsibility. This enables a potential contractor to determine the manager to which it should respond. The eligibility specification indicates that the only nodes that should bid on the task are those which (a) have sensing capabilities, and (b) are located in the same area as the manager that announced the task. The bid specification indicates that a bid should contain the position of the bidder and the number of each of its sensor types, information that a manager needs to select a suitable set of sensor nodes.

The potential signal contractors listen to the task announcements made by signal managers. They respond to the nearest manager with a bid (Fig. 17) that supplies their position and a description of their sensors. The managers use this

```
To:        *
From:      25
Type:      TASK ANNOUNCEMENT
Contract:  22-3-1

Eligibility Specification
           MUST-HAVE SENSOR
           MUST-HAVE POSITION AREA A

Task Abstraction:
           TASK TYPE SIGNAL
           POSITION LAT 47N LONG 17E
           AREA NAME A SPECIFICATION (. . .)

Bid Specification
           POSITION LAT LONG
           EVERY SENSOR NAME TYPE

Expiration Time
           28 1730Z FEB 1979
```

FIG. 16. Signal task announcement.

```
To:        25
From:      42
Type:      BID
Contract:  22-3-1

Node Abstraction
  LAT 62N LONG 9W
  SENSOR NAME S1 TYPE S
  SENSOR NAME S2 TYPE S
  SENSOR NAME TI TYPE T
```

FIG. 17. Signal bid.

information to select a set of bidders that covers their area of responsibility with a suitable variety of sensors, and then award signal contracts on this basis (Fig. 18).

The signal contract is a good example of the negotiation process. It involves a mutual decision based on local processing by both the managers and the potential contractors. The potential contractors base their decision on a distance metric and respond to the closest manager. The managers use the number of sensors and distribution of sensor types observed in the bids to select a set of contractors that covers each area with a variety of sensors. Thus each party to the contract evaluates the proposals made by the other using its own distinct evaluation procedure.

To review the initialization process: we have a single monitor node that manages several area contractors. Each area contractor manages several group contractors, and each group contractor manages several signal contractors. The data initially flows from the bottom to the top of this hierarchy. The signal contractors supply signal features; each group contractor integrates the features from several signal contractors to form a signal group, and these groups are passed along to the area contractors, which eventually form area maps by integrating information based on the data from several group contractors. All

```
To:        42
From:      25
Type:      AWARD
Contract:  22-3-1

Task Specification
  SENSOR NAME S1
  SENSOR NAME S2
```

FIG. 18. Signal award.

the area maps are then passed to the monitor which forms the final traffic map.[18]

The initialization process reviewed above may appear at first glance to be somewhat more elaborate than is strictly necessary. We have purposely taken a fairly general approach to the problem to emphasize two aspects of contract net performance. First, as illustrated by the signal contract, contract negotiation is an interactive process involving (i) a *two-way transfor of information* (task announcements from managers to contractors, bids from contractors to managers), (ii) *local evaluation* (each party to the negotiation has its own local evaluation procedure), and (iii) *mutual selection* (bidders select from among task announcements, managers select from among bids).

Second, the contract negotiation process offers a useful degree of flexibility, making it well suited to AI problems whose decomposition is not known a priori and well suited to problems whose configuration is likely to change over time. To illustrate this, consider that exactly the same initialization process will work across a large variation in the number of and position of nodes available (indeed the description given never mentions how many nodes there are, where they are located, or how wide the total area of coverage is). There are clearly limits to this flexibility: If the area of coverage were large enough to require several thousand area contractors, it might prove useful to introduce another level of distribution in the hierarchy (Fig. 14) between the monitor node and the area contractor. But the current approach works with a wide range of available resources and needs no modification within that range. This can be useful when available hardware resources cannot be identified a priori with certainty, or when operating environments are hostile enough to make hardware failure a significant occurrence.

8.3.2. *Operation*

We now consider the activities of the system as it begins operation. For the sake of brevity the actions are described at the level of task announcements, bids, and contracts. For additional details and examples of messages sent, see [25].

When a signal is detected or when a change occurs in the features of a known signal, the detecting signal contractor reports this fact to its manager. This node, in turn, attempts either to integrate the information into an existing signal group or to form a new signal group (recall that the manager for the signal task is also a contractor for the task of group formation, Fig. 15).

[18]As noted, in this example one area contractor manages several group contractors and each group contractor in turn manages several signal contractors. It is possible, however, that a single group contractor could supply information to several area contractors, and a single signal contractor could supply information to several group contractors. It may be useful, for instance, to have a particular group contractor near an area boundary report to the area contractors on both sides of the boundary. This is easily accommodated within our framework.

Whenever a new group is detected, the contractor reports existence of the group to its manager (an area contractor). The area contractor attempts to find a node to execute a vehicle contract, which involves classifying, localizing, and tracking the vehicle. The area contractor must first determine whether the newly detected group is attributable to a known vehicle. To do this, it uses a request-response interchange to get from all current vehicle contractors an indication of their belief that the new group can in fact be attributed to one of the known vehicles.[19] Based on the responses, the area contractor either starts up a new vehicle contractor (if the group does not seem to fit an existing vehicle) or augments the current contract of the appropriate vehicle contractor, adding to it the task of making certain that the new group corresponds to a known vehicle. This may entail such things as gathering new data via the adjustment of sensors or the creation of contracts with new sensor nodes.

The vehicle contractor then makes two task announcements: vehicle classification and vehicle localization. A classification contractor may be able to classify directly, given the signal group information or it may require more data, in which case it can communicate directly with the appropriate sensor nodes.[20] The localization task is a simple triangulation which is awarded to the first bidder.

Once the vehicle has been localized, it must be tracked. This is handled by the vehicle contractor, which issues additional localization contracts from time to time and uses the results to update its vehicle description. Alternatively, the area contractor could award separate tracking contracts. The decision as to which method to use depends on loading and communication. If, for example, the area contractor is very busy with integration of data from many group contractors, it seems more appropriate to isolate it from the additional load of tracking contracts. If, on the other hand, the area contractor is not overly busy, we can let it handle updated vehicle contracts, taking advantage of the fact that it is in the best position to integrate the results and coordinate the efforts of multiple tracking contractors. In this example, we assume that the management load would be too large for the area contractor.

A variety of other issues have to be considered in the design and operation of a real distributed sensing system. Most of them, however, are quite specific to the DSS application and hence outside the main focus of this paper.

[19]In response to the request, the vehicle contractor has two options. It can compute the answer itself, or, if it decides that that would require more processing power than it can spare, it can issue a contract and have another node compute the answer.

[20]As this example illustrates, it is possible in the contract net for two contractors to communicate directly (i.e., horizontal communication across the hierarchy) as well as via the more traditional (vertical) communication between managers and contractors. This is accomplished with request-response exchanges. If the identity of the recipient of the request is not known by name, then the request can be sent out using the focused addressing scheme mentioned in Section 7.3.

9. A Progression in Mechanisms for Transfer of Control

9.1. The basic questions and fundamental differences

The contract net appears to offer a novel perspective on the traditional concepts of invocation and transfer of control. To illustrate this, we examine a range of invocation mechanisms that have been created since the earliest techniques were developed, and compare the perspective implicit in each to the perspective used in the contract net.

In doing this comparison, we consider the process of transfer of control from the perspective of both the caller and the respondent. We focus in particular on the issue of *selection* and consider what opportunities a calling process has for selecting an appropriate respondent and what opportunities a potential respondent has for selecting the task on which to work. In each case we consider two basic questions that either the caller or the respondent might ask:

> *What is the character of the choice available*? (i.e., at runtime, does the caller know about all potential respondents and can it choose from among them; similarly does each respondent know all the potential callers for whom it might work and can it choose from among them?)
>
> *On what kind of information is that choice based*? (e.g., are potential respondents given, say, a pattern to match, or some more complex form of information? What information is the caller given about the potential respondents?)

The answers to these questions will demonstrate how our view of control transfer differs from that of the earlier formalisms with respect to:

> *Information transfer*: The announcement-bid-award sequence means that there is the potential for more information, and more complex information, transferred in both directions (between caller and respondent) during the invocation process.
>
> *Local evaluation*: The computation devoted to the selection process, based on the information transfer noted above, is more extensive and more complex that that used in traditional approaches. It is *local* in the sense that information is evaluated in a context associated with, and specific to, an individual KS (rather than embodied in a global evaluation function).
>
> *Mutual selection*: The local selection process is symmetric, in the sense that the caller evaluates potential respondents from its perspective (via the bid evaluation procedure) and the respondents evaluate the available tasks from their perspective (via the task evaluation procedures).

To put it another way, in the contract net the issue of *transfer of control* is more broadly viewed as a problem of connecting managers (and their tasks) with contractors (and their KSs). This view is inherently *symmetric* in that both the caller (manager) and respondents (bidders) have a selection to make. This symmetry in turn leads to the concept of establishing connection via *negotiation* between the interested parties. Then, if we are to have a fruitful discussion, the participants need to be able to 'say' interesting things to one another (i.e., they need the ability to transfer *complex information*). As the discussion below should made clear, previous models of invocation do not share these qualities. They view transfer of control as an essentially unidirectional process (from caller to respondent), offer minimal opportunity for selection at runtime, and provide restricted channels of communication between caller and respondent.

9.2. The comparison

In discussing the various approaches to invocation we often refer to 'standard' or 'traditional' forms of these approaches. Each of them could conceivably be modified in ways that would render our comments less relevant, but our point here is to examine the techniques as conceived and as typically used.

Standard subroutine (procedure) invocation represents, by our standard, a degenerate case. All the selection of routines to be invoked is done beforehand by the programmer and is hardwired into the code. The possible respondents are thus named explicitly in the source code, leaving no opportunity for choice or nondeterminism at runtime.

In traditional production rule systems, a degree of choice for the caller (in this case the interpreter) is available, since a number of rules may be retrieved at once. A range of selection criteria have been used (called *conflict resolution* schemes—see [5]), but these have typically been implemented with a single syntactic criterion hardwired into the interpreter. One standard scheme, for instance, is to assign a fixed priority to each rule and then from among those retrieved for possible invocation, simply select the rule with the highest priority. Selection is thus determined by a single procedure applied uniformly to every set of rules.

In this approach to invocation there is some choice available in selecting a KS to be invoked (since more than one rule may be retrieved), but the mechanism provided for making that choice allows for only a single, preselected procedure that is to be applied in all cases. In addition, all of the selection is done by the 'caller'; there is no mechanism that offers the rules any ability to select how they are to be invoked (e.g., if a rule can match the database in several ways, which of the possible matches will actually be used?). Finally, only minimal information is transferred from potential respondents back to the caller (at most a specification of what items in the database have been matched, and how).

PLANNER's [11] pattern-directed invocation provides a facility at the programming language level for nondeterministic KS retrieval, by matching goal specifications (patterns) against theorem patterns. In the simplest case, theorems are retrieved one by one and matched against the goal specification until a match is found. The order in which the theorems are tried is not defined by the language and is dependent on implementation decisions.

PLANNER does offer, in the *recommendation list*, a mechanism designed to allow the user to encode selection information. The *use* construct provides a way of specifying (by name) which theorems to try in which order. The *theorem base filter* construct offers a way of invoking a predicate function which takes one argument (the name of the next theorem whose pattern has matched the goal) and which can veto the use of that theorem.

Note that there is a degree of selection possible here, since the theorem base filter offers a way of choosing among the theorems that might possibly be used. The selection may involve a considerable amount of computation by the theorem base filter, and is local, in the sense that filters may be specific to a particular goal pattern. However, the selection is also limited in several ways. First, in the standard PLANNER invocation mechanism, the information available to the caller is at best the name of the next potential respondent. The caller does not receive any additional information (such as, for instance, exactly how the theorem matched the pattern), nor is there any easy way to provide for information transfer in that direction. Second, the choice is, as noted, a simple veto based on just that single KS. That is, since final judgment is passed on each potential KS in turn, it is not possible to make comparisons between potential KSs or to pass judgment on the whole group and choose the one that looks by some measure the best. Both of these shortcomings could be overcome if we were willing to create a superstructure on top of the existing invocation mechanism, but this would be functionally identical to the announcement-bid-award mechanism described above. The point is simply that the standard PLANNER invocation mechanism has no such facility, and the built-in depth-first search with backtracking makes it expensive to implement.

CONNIVER [19] represents a useful advance in nondeterministic invocation, since the result of a pattern-directed call is a 'possibilities list' containing *all* the KSs that match the pattern. While there is no explicit mechanism parallel to PLANNER's recommendation list, the possibilities list is accessible as a data structure and can be modified to reflect any judgments the caller might make concerning the relative utility of the KSs retrieved. Also, paired with each KS on the possibilities list is an association-list of pattern variables and bindings, which makes possible a determination of how the calling pattern was matched by each KS. This mechanism offers the caller some information about each respondent that can be useful in making the judgments noted above. CONNIVER does not, however, offer the respondent any opportunity to perform local processing to select from among callers.

The HEARSAY-II [7] system illustrates a number of similar facilities in a data-directed system. In particular, the focus of attention mechanism has a pointer to all the KSs that are ready to be invoked (i.e., those whose *stimulus frames* have been matched), as well as information (in the *response frame*) for estimating the potential contribution of each of the KSs. The system can effect some degree of selection regarding the KSs ready for invocation and has available to it a body of knowledge about each KS on which to base its selection. The response frame thus provides information transfer from respondent to caller that, while fixed in format, is more extensive than previous mechanisms. Considerable computation is also devoted to the selection process. Note, however, that the selection is not local, since there is a single, global strategy used for every selection.

The concept of meta-rules [3] offers a further advance in mechanisms to support more elaborate control schemes. It suggests that KS selection can be viewed as problem solving and can be effected using the same mechanism employed to solve problems in the task domain. It views selection as a process of pruning and reordering the applicable KSs and provides local selection by allowing meta-rules to be associated with specific goals.[21]

There are several things to note about the systems reviewed thus far. First, we see an increase in the amount and variety of information that is transferred from caller to respondent (e.g., from explicit naming in subroutines, to patterns in PLANNER) and from respondent to caller (e.g., from no response in subroutines to the response frames of HEARSAY-II). Note, however, that in no case do we have available a general information transmission mechanism. In all cases, the mechanisms have been designed to carry one particular sort of information and are not easily modified.

Second, we see a progression from the retrieval of a single KS to the retrieval of the entire set of potentially useful KSs, providing the opportunity for more complex varieties of selection.

Finally, note that all the selection so far is from one perspective; the selection of respondents by the caller. In none of these systems do the respondents have any choice in the matter.

To illustrate this last point, consider turning HEARSAY-II around and creating a system where respondents performed the selection: a 'task blackboard' system. The simplest form of such a system would have a central task blackboard that contains an unordered list of tasks that need to be performed. As a KS works on its current task, it may discover new (sub)tasks that require execution and add them to the blackboard. When a KS finishes its current task, it looks at the blackboard, evaluates the lists of tasks there, and decides which one it wants to execute.

[21]The concept of negotiation in the contract net grew, in part, from generalizing this perspective to make it 'bi-directional': Both managers and potential contractors can devote computational effort to selecting from the alternatives available to them.

Note that in this system the respondents would have all the selection capability. Rather than having a caller announce a task and evaluate the set of KSs that respond, we have the KSs examining the list of tasks and selecting the one they wish to work on. It is thus plausible to invert the standard situation, but we still have unidirectional selection—in this case, on the part of the respondent rather than the caller.

PUP6 [16], on the other hand, was the first system to suggest that transfer of control could be viewed as a *discussion* between the caller and potential respondents. In that system, if a KS receives more than one offer to execute a task, a special 'chooser' KS momentarily takes control and asks 'questions' of the respondents to determine which of them ought to be used. This is accomplished by querying the *parts* of the KS. Each KS is composed of a standard set of *parts*, each *part* designed to deal with a particular question about that KS. For example, the procedures in the WHEN and COMPLEXITY parts of a KS answer the questions "When should you take control?" and "How costly are you?" This interchange is highly stylized and not very flexible, but does represent an attempt to implement explicit two-way communication.

The contract net differs from these approaches in several ways. First, from the point of view of the caller (the manager), the standard task broadcast and response interchange has been improved by making possible a *more informative* response. That is, instead of the traditional tools that allow the caller to receive only a list of potential respondents, the contract net has available a mechanism that makes it possible for the caller to receive a description of potential utility from each respondent (the bidders). The caller also has available (as in other approaches) a list of respondents rather than a sequence of names presented one at a time.[22] Both of these make it possible to be more selective in making decisions about invocation.

Second, the contract net emphasizes *local evaluation*. An explicit place in the framework has been provided for mechanisms in which the caller can invest computational effort in selecting KSs for invocation (using its bid evaluation procedure) and the respondents can similarly invest effort in selecting tasks to work on (using their task evaluation procedures). These selection procedures are also local in the sense that they are associated with and written from the perspective of the individual KS (as opposed to, say, HEARSAY-II's global focus of attention procedure).

Third, while we have labeled this process *selection*, it might more appropriately be labeled *deliberation*. This would emphasize that its purpose for the caller is to decide in general *what to do* with the bids received and not merely *which of them to accept*. Note that one possible decision is that *none* of the bids is adequate and thus none of the potential respondents would be

[22]More precisely, the caller has available a list of all those that have responded by the expiration time of the contract.

invoked (instead, the task may be re-announced later).[23] This choice is not typically available in other problem solving systems and emphasizes the wider perspective taken by the contract net on the transfer of control issue.

Finally, there appears to be a novel symmetry in the transfer of control process. Recall that PLANNER, CONNIVER, and HEARSAY-II all offer the caller some ability to select from among the respondents, while a task blackboard system allows the respondents to select from among the tasks. The contract net (and PUP6), however, use an interactive, *mutual selection* process where task distribution is the result of a discussion between processors. As a result of the information exchanged in this discussion, the caller can select from among potential respondents while the KSs can select from among potential tasks.

10. Suitable Applications

In this section we consider the sorts of problems for which the contract net is well suited.

The framework has, for instance, been designed to provide a more powerful mechanism for transfer of control than is available in current problem-solving systems. This mechanism will be useful when we do not know in advance which KS should be invoked or do not know which node should be given the task in question. In the first of these situations—not knowing in advance which KS to invoke—we require some machinery for making the decision. The contract net's negotiation and deliberation process is one such mechanism. It will prove most useful for problems in which especially careful selection of KSs is important (i.e., problems for which we prefer the 'knowledge' end of the knowledge vs. search tradeoff).

The second situation—matching nodes and tasks—is inherent in a distributed architecture, since no one node has complete knowledge of either the capabilities of or the busy/idle state of every node in the network. We have labeled this the connection problem and have explored the negotiation and deliberation process as a way of solving it as well.

The framework is well-matched to problems that can be viewed in terms of a hierarchy of tasks (e.g., heuristic search), or levels of data abstraction (e.g.,

[23]Similarly the potential bidders deliberate over task announcements received and may decide that none is worth submitting a bid. Note also that receiving bids but deciding that none is good enough is distinctly different from receiving no bids at all. In a system using pattern-directed inference, receiving no bids is analogous to finding no KSs with matching patterns; receiving bids but turning down all of them after due consideration has no precise analogy in existing languages.

Agenda-based systems come close, in that KSs put on the agenda may have such a low ranking that they are effectively ignored. But this is not the same, for two reasons. First, if the queue ever does get sufficiently depleted, those KSs will in fact be run. Second, and more important, there is no explicit decision to ignore those KSs, simply an accident of the ordering, or perhaps the KS's own estimation of its individual utility. The contract net offers a mechanism for making the decision explicitly and based on an evaluation of all the candidates.

applications that deal with audio or video signals). Such problems lend themselves to decomposition into a set of relatively independent tasks with little need for global information or synchronization. Individual tasks can be assigned to separate processor nodes; these nodes can then execute the tasks with little need for communication with other nodes.

The manager-contractor structure provides a natural way to effect hierarchical control (in the distributed case, it's actually concurrent hierarchical control), and the managers at each level in the hierarchy are an appropriate place for data integration and abstraction.

Note, by the way, that these control hierarchies are not simple vertical hierarchies but are more complex generalized hierarchies. This is illustrated by the existence of communication links other than those between managers and contractors. Nodes are able to communicate horizontally with related contractors or with any other nodes in the net, as we saw in the DSS example, where classification contractors communicated directly with signal contractors using the request-response style of interaction.

The framework is also primarily applicable to domains where the subtasks are large and where it is worthwhile to expend a potentially nontrivial amount of computation and communication to invoke the best KSs for each subtask. It would, for instance, make little sense to go through an extended mutual selection process to get some simple arithmetic done or to do a simple database access. While our approach can be abbreviated to an appropriately terse degree of interchange for simple problems (e.g., directed contacts and the request-response mechanism), other systems are already capable of supporting this variety of behavior. The primary contribution of our framework lies in applications to problems where the more complex interchange provides an efficient and effective basis for problem solving.

Finally, the contract net is also useful in problems where the primary concerns are in distributing control, achieving reliability, and avoiding bottlenecks, even if, in these problems, the more complex variety of information exchange described above is unnecessary. The contract net's negotiation mechanism offers a means for distributing control; sharing responsibility for tasks between managers and contractors offers a degree of reliability; and the careful design of the message types in the protocol helps avoid saturating the communication channel and causing bottlenecks.

11. Limitations, Extensions, Open Problems

11.1. The other stages

Earlier we noted that this paper focuses on application of the contract net to the distribution stage of distributed problem solving. The other stages—decomposition, sub-problem solution, and answer synthesis—are important foci for additional work. Problem decomposition, for example, is not a wellunderstood process. It is easy to recognize when it is done well or badly, but there are relatively few principles that can be used prospectively to produce good decompositions. We address below the issue of sub-problem solution, noting that a more cooperative approach—one in which individual nodes share partial solutions as they work—can be useful in a variety of problems. Finally, as we have explored elsewhere [4], there are a number of approaches to synthesizing individual sub-problem results, each addressing a different anticipated level of problem interaction. In future work we intend to explore applications of the contract net and the negotiation metaphor to each of these topics.

11.2. Instantiating the framework

The framework we have proposed—the task announcement, bid, award sequence, the common internode language, etc.—offers some ideas about what kinds of information are useful for distributed problem solving and how that information can be organized. There is still a considerable problem involved in instantiating the framework in the context of a specific task domain. Our protocol provides a site for embedding particular types of information (e.g. an eligibility specification), but does not specify exactly what that information is for any specific problem.

In this sense the contract net protocol is similar to AI languages like PLANNER, CONNIVER, QLISP [24], etc., which supply a framework for problem solving (e.g., the notions of goal specifications, theorem patterns, etc.), but leave to the user the task of specifying the content of that framework for any given problem. We expect that further experience with our framework will lead to additional structure to help guide its use.

11.3. Alternate models of cooperation

We have emphasized task-sharing as a means of internode cooperation and have attempted to provide some mechanisms for the communication required to effect this mode of cooperation. We have not as yet, however, adequately studied *result-sharing* [28] as a means of cooperation. In what approach, nodes assist each other through sharing of partial results. This type of cooperation appears to be of use in dealing with several sorts of problems. For problems where erroneous data or knowledge lead to conflicting views at individual nodes, sharing results can help to resolve those inconsistencies (as for example in [17]). For some tasks, any individual subproblem is inherently ambiguous even when data and knowledge are complete and exact (e.g., the blocks world scene identification in [31]); here the sharing of intermediate results can be an effective means of reducing or removing ambiguity. It is our intention to examine the structure of communication for this mode of cooperation with a view to extending the contract net framework to incorporate it.

It would also be useful to develop a more advanced form of task-sharing. In our current formulation, task distribution results in the traditional form of "hand out a subtask and get back a result" interaction. We are currently exploring the possibility of expanding this to a more cooperative form of interaction in which "what is to be done" is negotiated as well as "who is to do it".

We are also exploring further development of the dynamic configuration capability which the contract net makes possible. As noted in Section 8.3, initialization of the DSS can take into account the resources available (number of sensors, etc.). We intend to extend this to dynamic reconfiguration: the negotiation technique should provide a mechanism that allows nodes which have become overloaded to shed some of their workload by distributing tasks to other available nodes.

11.4. Optimality of the negotiation process

As noted, a major goal of the contract net framework is to provide a mechanism for solving the connection problem—achieving an appropriate matching of tasks to processor nodes. Yet it is easily seen that the negotiation process described above does not guarantee an *optimal* matching of tasks and nodes.

There are two reasons why this may occur. First, there is the problem of timing. A node that becomes idle chooses a task to bid on from among the task announcements it has heard up to that time. Similarly, a manager chooses what to do on the basis of the bids it has received by the expiration time for its task announcement. But since the net operates asynchronously, new task announcements and new bids are made at unpredictable times. A better matching of nodes to tasks might be achieved if there were some way to know that it was appropriate for a node to wait just a little longer before bidding, or for a manager to wait a little longer before awarding a task.

Second, at any given instant in time, the complete matching of nodes and tasks results from a number of local decisions. Each idle node chooses the most interesting task to bid on, without reference to what other idle nodes may be choosing; each manager chooses the best bid(s) it has received without reference to what any other manager may be doing. The best global assignment does not necessarily result from the simple concatenation of all of the best local assignments.[24]

Consider for example a situation in which two managers (A and B) have both announced tasks, and two potential contractors (X and Y) have each responded by bidding on both tasks. Imagine further that from A's perspective, X's bid is rated 0.9 (on a 0 to 1 scale), while Y's is rated 0.8 (Fig. 19). Conversely, from B's perspective, X is rated 0.8 and Y is rated 0.2.

[24]This appears to be a variety of the 'prisoner's dilemma' problem (see e.g., [10, 30]).

A		B	
X	0.9	X	0.8
Y	0.8	Y	0.2

FIG. 19. Managers rating bids from prospective contractors.

From a purely local perspective, both of the managers want X as their contractor; from a more global perspective it may make more sense to have A 'settle' for Y, and give X to B. Yet we cannot in general create the more global perspective without exchanging what may turn out to be extensive amounts of information.

The first of the two problems (timing) appears unavoidable given that we have chosen to deal with the kinds of problems typically attacked in AI, problems whose total decomposition is not known a priori. In a speech understanding problem, for instance, we cannot set up a fixed sequence of KS invocations beforehand because the utility of any given KS is not predictable in advance. Similarly, in a DSS, we have the same inability to predict KS utility, plus the added difficulty of new signals arriving at unpredictable moments.

If we do not know in advance which subtasks will arise and when, or exactly which KSs will be useful at each point, then we clearly cannot plan the *optimal* assignment of nodes to tasks for the entire duration of the problem. Some planning may be possible, however, even if we lack complete knowledge of a problem's eventual decomposition. We are currently studying ways to make use of partial information about tasks yet to be encountered or nodes that are soon going to be idle.

The second problem (local decisions) appears inherent in any decentralization of control and decision making. As noted earlier, we want to distribute control (for reasons of speed, problem-solving power, reliability, etc.). Given distributed control, however, globally *optimal* control decisions are possible only at the cost of transmitting extensive amounts of information between managers every time an award is about to be made. With that approach, inefficiencies due to suboptimal control decisions are traded for inefficiencies arising from transmission delays and channel saturation. We are currently studying this tradeoff and exploring ways of minimizing the difficulties that arise from this problem.

It appears then, that as a result of the unpredictability of the timing of subtasks and the necessity of making local decisions, precisely optimal matching of nodes to tasks is not possible. Note, however, that our stated goal is an *appropriate* assignment of nodes to tasks. Operation of the contract net is not predicated on optimal matching. In addition, the small set of experiments we have done so far (see [27]) indicate that overall performance is not seriously degraded by suboptimal matching.

11.5. Coherent behavior

We do not yet fully understand the more general problem of achieving globally coherent behavior in a system with distributed control. The fundamental difficulty was described earlier: We require distributed control in order to effect loose coupling, yet coherent behavior usually requires a global perspective.

Some aspects of the contract net protocol were motivated by attempts to overcome this problem. First, the task abstraction supplies information which enables a node to compare announcements and select the most appropriate. In a similar fashion, information in bids (the node abstraction) enables managers to compare bids from different nodes and select the most appropriate. Second, each node in a contract net maintains a list of the best recent task announcements it has seen—a kind of window on the tasks at hand for the net as a whole. This window enables the nodes to compare announcements over time, helping to avoid mistakes associated with too brief a view of the problem at hand.

We still have the problem that good local decisions do not necessarily add up to good global behavior, as the example in the previous section showed. However, the steps noted at least contribute to local decisions that are made on the basis of an extended (not snapshot) view of system performance and decisions that are based on extensive information about tasks and bids.

In the most general terms we see our efforts aimed at developing a problem solving protocol. The protocol should contain primitives appropriate for talking about and doing problem solving, and should structure the interaction between problem solvers in ways that contribute to coordinated behavior of the group. We have thus far taken an initial step in this direction with the development of the task announcement, bid, and award sequence.

12. Summary

The preceding discussion considered the contract net in a number of different contexts. In the most specific view, it was considered a mechanism for building a distributed sensing system. More generally, it offered an approach to distributed problem solving and a view of distributed processing. In the most general view, it was considered in the context of AI problem solving techniques. In the sections that follow we consider the advantages offered by the contract net in each of these contexts, reviewing in the process the central themes of the paper.

12.1. Contributions to distributed processing

A distributed processing approach to computation offers the potential for a number of benefits, including speed and the ability to handle applications that have a natural spatial distributon. The design of the contract net framework attempts to ensure that these potential benefits are indeed realized.

In order to realize speed in distributed systems, we need to avoid bottlenecks. They can arise in two primary ways: by concentrating disproportionate amounts of computation or communication at central resources, and by saturating available communication channels so that nodes must remain idle while messages are transmitted.

To avoid bottlenecks we distribute control and data. In the DSS example, data is distributed dynamically as a result of the division of the net into areas during the initialization phase. Control is distributed dynamically through the use of a negotiation process to effect the connection of tasks with idle processors.

The contract net design also tries to avoid communication channel saturation by reducing the number and length of messages. The information in task announcements (like eligibility specifications), for instance, helps eliminate extra message traffic, thereby helping to minimize the amount of channel capacity consumed by communication overhead. Similarly, bid messages can be kept short and 'to the point' through the use of the bid specification mechanism.

Finally, the ability to handle applications with a natural spatial or functional distribution is facilitated by viewing task distribution as a connection problem and by having the processors themselves negotiate to solve the problem. This makes it possible for the collection of available processors to 'spread themselves' over the set of tasks to be done, distributing the workload dynamically.

12.2. Contributions to distributed problem solving

As we noted earlier (Section 6), a central issue in distributed problem solving is organization: How can we distribute control and yet maintain coherent behavior?

One way to accomplish this is by what we have called task-sharing, the distribution around the net of tasks relevant to solving the overall problem. As we have seen, the contract net views task-sharing in terms of connecting idle nodes with tasks yet to be done. It effects this matching by structuring interaction around negotiation as an organizing principle.

Negotiation in turn is implemented by focusing on what it is that processors should say to one another. The motivation for our protocol is thus to supply one idea on *what to say* rather than *how to communicate.*

As the example in Section 8 showed, use of the contract net makes it possible for the system to be configured dynamically, taking into account (in that example) such factors as the number of sensor and processor nodes available, their location, and the ease with which communication can be extablished. Such a configuration offers a number of improvements over a static, a priori configuration. It provides, for instance, a degree of simplicity: The same software is capable of initializing and running networks with a wide variation of available hardware. If the configuration were static, each new configuration would presumably require human intervention for its basic design (e.g., assigning nodes to tasks) and might require modifications to software as well.

Dynamic configuration also means that most nodes that must cooperate are able to communicate with one another directly. This reduces the amount of communication needed, since it reduces the need for either indirect routing of messages or the use of powerful transmitters.

The contract net also offers advantages in terms of increased reliability. By distributing both control and data, for instance, we ensure that there is no one node or even a subset of nodes whose loss would totally cripple the system. In addition, recovery from failure of a node is aided by the presence of explicit links between managers and their contractors. The failure of any contractor can be detected by its manager; the contract for which it was responsible can then be re-announced and awarded to another node. There is, in addition, the possibility of reliability arising from "load-sensitive redundancy". When load on the net is low, we might take advantage of idle processors by making redundant awards of the same contract. The system thus offers the opportunity to make resource allocation decisions opportunistically, taking advantage of idle resources to provide additional reliability.

The framework also makes it reasonably easy to add new nodes to the net at any time. This is useful for replacing nodes that have failed or adding new nodes in response to increased computational load on the net. Two elements of the framework provide the foundation for this capability. First, the contract negotiation process uses a form of "anonymous invocation": the KSs to be invoked are *described* rather than *named.* Second, there is a single language "spoken" by all the nodes.

The concept of describing rather than naming KSs has its roots in the goal-directed invocation of various AI languages and the notion of pattern-directed invocation generally (see, e.g. [32]), where it was motivated by the desire for more sophisticated forms of KS retrieval. It also however, turns out to offer an interesting and useful form of "substitutability", simply because where names are unique, descriptions are not, and a wide range of KSs may satisfy a single description. As a result, in a system with invocation by name, the addition of a new KS requires modification of the existing code to ensure that the new KS is indeed invoked. When invocation is by description, adding a new KS involves simply making it available to the existing collection of KSs; it will be invoked whenever its description is matched (in our case, whenever it chooses to bid on a task announcement). The contract net thus shares with other systems using anonymous invocation the ability to add new KSs by simply "throwing them into the pot".

Second, the use of a single language 'spoken' by all the nodes simplifies communication. If we are to add a new node, it must have some way of communicating with other nodes in the net. The contract net simplifies this issue by providing a very compact language: The basic protocol (task announcement, bid, award) provides the elementary 'syntax' for communication, while the common internode language provides the vocabulary used to express message content.

Thus, anonymous invocation means that it is possible for a new node to begin participating in the operation of the net by listening to the messages being exchanged. (If invocation were by name, listening to message traffic would do no good.) The use of a single language means that the node will understand the messages, and the use of a very simple language means that the task of initializing a node is easier.

12.3. Contributions to artificial intelligence

The contract net offers a novel view on the nature of the invocation process. As we have seen, it views task distribution as a problem of connecting tasks to KSs capable of executing those tasks, and it effects this connection via negotiation.

In Section 9 we used this perspective to examine existing models of invocation and evaluate them along several dimensions. This discussion showed, first, that in previous models connection is typically effected with a transfer of information that is unidirectional; hence the connection process is asymmetric. Control resides either with the tasks (goal-driven invocation) or with the KSs (data-driven invocation). In the contract net view, by contrast, the transfer is two-way, as each participant in the negotiation offers information about itself. This in turn means that control can be shared by both; the problem becomes one of mutual selection.

We then showed that the information transferred is typically limited in content. In the contract net, on the other hand, the information is not limited to a name or pattern, but is instead expanded to include statements expressible in the common internode language.

Third, the discussion showed that information about a more complete collection of candidate KSs is available before final selection is made. This makes possible a wider range of KS and task selection strategies than are possible if KSs and tasks must be selected or rejected as they are encountered.

Finally, we noted that this expanded view of invocation effects a true deliberation process, since one possible outcome of the negotiation is that none of the bids received is judged good enough, and hence none of the potential contractors will be selected. This appears to be a useful advance that has no precise analogy in previous programming languages and applications.

12.4. Conclusion: the major themes revisited

Two of the major themes of this paper are the notion of protocols aimed at problem solving rather than communication and the concept of negotiation as a basic mechanism for interaction. The first was illustrated by the use of message types like task announcement, bid, and award. This focused the contract net protocol at the level of problem solving and provided a step toward indicating what kinds of information should be transferred between nodes.

The utility of negotiation as an interaction mechanism was demonstrated in two settings. First, our basic approach to cooperation relies on task-sharing, and negotiation is used to distribute tasks among the nodes of the net. This makes possible distribution based on mutual selection, yielding a good match of nodes and tasks. Second, negotiation was used to effect transfer of control. In that setting it offered a framework in which the matching of KSs to tasks was based on more information than is usually available (due to the transfer of information in both directions, and the transfer of more complex information). As a result, negotiation makes it possible to effect a finer degree of control and to be more selective in making decisions about invocation than is the case with previous mechanisms.

ACKNOWLEDGMENT

This work describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology and at the Defence Research Establishment Atlantic of the Department of National Defence, Research and Development Branch, Canada. Support for the Artificial Intelligence Lab is provided in part the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research Contract N00014-80-C-505.

The assistance of Bruce Buchanan and Ed Feigenbaum in the original development of these ideas is greatefully acknowledged. Carmen Bright, Joe Maksym, Lee Erman, Carl Hewitt, Patrick Winston, and Judy Zinnikas provided useful comments on earlier drafts of this paper.

REFERENCES

1. Baer, J.-L., A survey of some theoretical aspects of multiprocessing, *Comput. Surveys* **5** (1) (1973) 31–80.
2. Bowdon, E.K., Sr. and Barr, W.J., Cost effective priority assignment in network computers, in: *FJCC Proceedings* **41** (AFIPS, Montvale, NJ, 1972) 755–763.
3. Davis, R., Meta-rules: reasoning about control, *Artificial Intelligence* **15** (1980) 179–222.
4. Davis, R., Models of problem solving: Why cooperate?, *SIGART Newsletter* **70** (1980) 50–51.
5. Davis, R. and King, J., An overview of production systems, in: E.W. Elcock and D. Michie (Eds.), *Machine Intelligence* **8** (Wiley, New York, 1977) 300–332.
6. D'Olivera, C.R., An analysis of computer decentralization, Rept. LCS, TM90, MIT, Cambridge, MA, 1977.
7. Erman, L.D., Hayes-Roth, F., Lesser, V.R. and Reddy, D.R., The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty, *Comput. Surveys* **12** (1980) 213–253.
8. Farber, D.J. and Larson, K.C., The structure of the distributed computing system—Software, in: J. Fox (Ed.), *Proceedings of the Symposium on Computer-Communications Networks And Teletraffic* (Polytechnic Press, Brooklyn, NY, 1972) 539–545.
9. Galbraith, J.R., Organizational design—an information processing view, in: Kolb (Ed.), *Organizational Psychology* (Prentice Hall, Englewood Cliffs, NJ, 2nd. ed., 1974) 313–322.
10. Hamburger, H., *N*-person prisoner's dilemma, *J. Math. Sociology* **3** (1973) 27–48.
11. Hewitt, C., Description and theoretical analysis (using schemata) of PLANNER: A language for proving theorems and manipulating models in a robot, MIT AI TR 258, MIT, Cambridge, MA, 1972.

12. Hewitt, C., Viewing control structures as patterns of passing messages, *Artificial Intelligence* **8** (1977) 323–364.
13. Kahn, R.E., Resource-sharing computer communications networks, *Proc. IEEE* **60** (11) (1972) 1397–1407.
14. Kahn, R.E., The organization of computer resources into a packet radio network, in: *NCC Proceedings* **44** (AFIPS, Montvale, NJ, 1975) 177–186.
15. Kimbleton, S.R. and Schneider, G.M., Computer communications networks: approaches, objectives, and performance considerations, *Comput. Surveys* **7** (3) (1975) 129–173.
16. Lenat, D.B., Beings: knowledge as interacting experts, *IJCA* **4** (1975) 126–133.
17. Lesser, V.R. and Erman, L.D., Distributed interpretation: a model and experiment, *IEEE Trans. Comput.* **29** (1980) 1144–1163.
18. Lesser, V.R. and Corkill, D.D., Functionally accurate cooperative distributed systems, *IEEE Trans. Systems Man Cybernet.* **11** (1) (1981) 81–96.
19. McDermott, D.V. and Sussman, G.J., The CONNIVER Reference Manual, AI Memo 259a, MIT, Cambridge, MA, 1974.
20. Nii, H.P. and Feigenbaum, E.A., Rule-based understanding of signals, in: D.A. Waterman and F. Hayes-Roth (Eds.), *Pattern-Directed Inference Systems* (Academic Press, New York, 1978) 483–501.
21. Noyce, R.N., From relays to MPU's, *Comput.* **9** (12) (1976) 26–29.
22. Prince, P.S., Recovery from failure in a contract net, B.S. Thesis, EECS Department, MIT, Cambridge, MA, 1980.
23. Roberts, L.G., Data by the packet, *IEEE Spectrum* **11** (2) (1974) 46–51.
24. Sacerdoti et al., QLISP—A language for the interactive development of complex systems, *Proc. NCC* **45** (1976) 349–356.
25. Smith, R.G. and Davis, R., Applications of the contract net framework: distributed sensing, *Proc. ARPA Distributed Sensor Net Symp.*, Pittsburgh, PA (1978) 12–20.
26. Smith, R.G., *A framework for distributed problem solving* (VMI Research Press, 1981); also: Stanford Memo STAN-CS-78-700, Stanford University Stanford, CA, 1978.
27. Smith, R.G., The contract net protocol: high level communication and control in a distributed problem solver, *IEEE Trans. Comput.* **29** (1980) 1104–1113.
28. Smith, R.G. and Davis, R., Frameworks for cooperation in a distributed problem solver, *IEEE Trans Systems Man Cybernet.* **11** (1981) 61–70.
29. Svodobova, L., Liskov, B. and Clark, D., Distributed computer systems: structure and semantics, MIT-LCS-TR-215, MIT, Cambridge, MA, 1979.
30. Tucker, A.W., A two-person dilemma, Mimeo, Stanford University, Stanford, CA, 1950.
31. Waltz, D., Understanding line drawings of scenes with shadows, in: Winston (Ed.), *The Psychology of Computer Vision* (McGraw-Hill, New York, 1975).
32. Waterman, D.A. and Hayes-Roth, F. (Eds.), *Pattern-Directed Inference Systems* (Academic Press, New York, 1978).

Received April 1981