The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver

REID G. SMITH, MEMBER, IEEE

Abstract — The contract net protocol has been developed to specify problem-solving communication and control for nodes in a distributed problem solver. Task distribution is affected by a negotiation process, a discussion carried on between nodes with tasks to be executed and nodes that may be able to execute those tasks.

We present the specification of the protocol and demonstrate its use in the solution of a problem in distributed sensing.

The utility of negotiation as an interaction mechanism is discussed. It can be used to achieve different goals, such as distributing control and data to avoid bottlenecks and enabling a finer degree of control in making resource allocation and focus decisions than is possible with traditional mechanisms.

Index **Terms—Artificial Intelligence (AI), connection, cooperation, distributed problem solving, focus, high-level protocols, negotiation, resource allocation, task-sharing.**

I. INTRODUCTION

DISTRIBUTED problem solving is the cooperative solution of problems by a decentralized and loosely coupled collection of knowledge-sources *(KS's)* (procedures, sets of production rules, etc.), located in a number of distinct processor nodes. The *KS's cooperate* in the sense that no one of them has sufficient information to solve the entire problem; mutual sharing of information is necessary 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 *KS's* spend most of their time in computation rather than communication.

Such problem solvers offer the promise of speed, reliability, extensibility, and the potential for increased tolerance to uncertain data and knowledge, as well as the ability to handle applications with a natural spatial distribution. There has been much recent interest in this type of problem solving in the Artificial Intelligence (AI) community. Its use has been considered in such applications as traffic-light control [5], distributed sensing [7], and heuristic search [10].

In this paper, we present the **contract net** protocol, a high-level protocol for communication among the nodes in a distributed problem solver. It facilitates distributed control of cooperative task execution (which we call *task-sharing* [9]) with efficient internode communication.

The role of a high-level protocol in a network such as the ARPANET has been discussed in previous papers (see, for example [11]). Traditional communication protocols form a low-level base for problem-solving communication. They enable reliable and efficient transmission of bit streams between nodes, but do not consider the semantics of the information being passed. A high-level protocol assigns interpretations to the bit streams. It offers a structure that assists the system designer in deciding *what* the nodes should say to each other, rather than *how* to say it.

We are not primarily concerned with the physical architecture of the problem solver. It is assumed to be a network of loosely coupled, asynchronous nodes. Each node contains a number of distinct *KS's*. The nodes are interconnected so that each node can communicate with every other node by sending messages. No memory is shared by the nodes. We also assume the existence of a low-level communication protocol to support reliable and efficient communication of bit streams between nodes. A functional model of a node is shown in Appendix A.

II. CONNECTION AND CONTRACT NEGOTIATION

The key issue to be resolved in task-sharing is how tasks are to be distributed among the processor nodes. There must be a means whereby nodes with tasks to be executed can find the most appropriate idle nodes to execute those tasks. We call this the *connection problem.* Solving the connection problem is crucial to high performance in a distributed problem solver. It has two aspects: 1) *resource allocation* and 2) *focus.* Effective resource allocation is achieved by balancing the computational load among the nodes. It is essential if the maximum speedup possible from applying multiple nodes to a single overall problem is to be obtained.

Focus is achieved by effective selection of tasks for allocation to nodes and by effective selection of KS's for execution of tasks. It is essential for problems that do not have well-defined

Manuscript received March 4, 1980; revised May 9, 1980. This work was supported in part by the Advanced Research Projects Agency under Contract MDA 903-77-C-0322, the National Science Foundation under Contract MCS 77-02712, and the National Institutes of Health under Grant RR-00785 on the SUMEX-AIM Computer Facility at Stanford University, Stanford, CA and by the Defence Research Establishment Atlantic of the Department of National Defence, Research and Development Branch, Dartmouth, N.S. Canada.

The author is with the Defence Research Establishment Atlantic, Dartmouth, N.S., Canada.

algorithms for their solutions (i.e., problems of the type most often considered in AI). For such problems, many tasks are typically generated during the search for solutions; and the execution of many of these tasks will not lead to a solution. In addition, the most appropriate *KS* to invoke for the execution of any given task generally cannot be identified *a priori.* The combination of many tasks and many applicable KS's can lead to *a combinatorial explosion.* A problem solver must therefore maintain focus to achieve high performance in practical applications.

The connection problem can also be viewed from the perspective of an idle node. It must find another node with an appropriate task that is available for execution. In our approach, both nodes with tasks to be executed and nodes ready to execute tasks proceed simultaneously. They engage each other in discussions that resemble contract negotiation to solve the connection problem. It is this process that is the basis for the contract net protocol.

For our purposes, negotiation has four important components: 1) it is a local process that does not involve centralized control, 2) there is a two-way exchange of information, 3) each party to the negotiation evaluates the information from its own perspective, and 4) final agreement is achieved by mutual selection.

The collection of nodes is referred to as a contract net and the execution of a task is dealt with as a contract between two nodes. Each node in the net takes on one of two roles related to the execution of an individual task: manager or contractor. A **manager** is responsible for monitoring the execution of a task and processing the results of its execution. A **contractor** is responsible for the actual execution of the task. 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. As a result, nodes are not statically tied to a control hierarchy. This also leads to more efficient utilization of nodes, as compared, for example, to schemes that do not allow nodes that have contracted out tasks to take on other tasks while they are waiting for results.

A contract is established by a process of local mutual selection based on a two-way transfer of information. In brief, available contractors evaluate task announcements made by several managers and submit bids on those for which they are suited. The managers evaluate the bids and award contracts to the nodes they determine to be most appropriate. The negotiation process may then recur. A contractor may further partition a task and award contracts to other nodes. It is then the manager for those contracts. This leads to the hierarchical control structure that is typical of task-sharing. Control is distributed because processing and communication are not focused at particular nodes, but rather every node is capable of accepting and assigning tasks.

The basic idea of contracting is not new. A rudimentary bidding scheme, for example, was used for resource allocation in the distributed computing system (DCS) [2], [3]. We will note the similarities and differences between that scheme and the contract net protocol as we proceed.

Throughout the paper, reference is made to an experimental contract net system called CNET. It is a system of INTERLISP [12] functions that enables a user to simulate the solution of problems with a distributed processor.

III. Example

This example is taken from a CNET simulation of a distributed sensing system (DSS) [7]. A DSS is a network of sensor and processor nodes spread throughout a relatively large geographic area. It attempts to construct and maintain a dynamic map of vehicle traffic in the area. Construction and maintenance of such a map requires the interpretation and integration of a large quantity of sensory information received by the collection of sensor elements.

Use of the contract net protocol in a DSS makes it possible for the sensor system to be configured dynamically, taking into account such factors as the number of sensor and processor nodes available, their locations, and the ease with which communication can be established.

We will examine the negotiation for one particular task, called the *signal* task, that arises during the initialization phase of DSS operation. The task involves gathering of sensed data and extraction of signal features. The managers for this task are nodes that do not have sensing capabilities, but do have extensive processing capabilities. They attempt to find a set of sensor nodes to provide them with signal features. The sensor nodes, on the other hand, have limited processing capabilities and attempt to find managers that can further process the signal features they extract from the raw sensed data.

Recall that we view node interaction as an agreement between a node with a task to be performed 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 participant. For example, from the perspective of the *signal* task managers, the best set of contractors has 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 best managers are those 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 protocol. 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. 1. Each message in the contract net protocol has a set of slots for the task-specific information in the message. The four slots of the task announcement are shown in the figure. The information that fills the slots is encoded in a simple language common to all nodes.

The task abstraction is the type of task and the position of the manager making the announcement. The position 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 this task are those which 1) have sensing capabilities and 2) are located in the same area as the manager that announced the task. This helps to reduce extraneous message traffic and bid processing. The bid specifi

To: * indicates a broadcast message.
From: 25
Type: TASK ANNOUNCEMENT
Contract: 22-3-1
Task Abstraction:
TASK TYPE SIGNAL
POSITION LAT 47*N* LONG 17*E*
Eligibility Specification:
MUST-HAVE SENSOR
MUST-HAVE POSITION AREA *A*
Bid Specification:
POSITION LAT LONG
EVERY SENSOR NAME TYPE
Expiration Time:
28 1730Z FEB 1979

Fig. 1. Signal task announcement.

cation indicates the information that a manager needs to select a suitable set of sensor nodes–the position of the bidder and the name and type of each of its sensors. Finally, the expiration time is a deadline for receiving bids.

Each potential contractor listens to the task announcements made by *signal* managers. It ranks each announcement relative to the others thus far received, according to the distance to the manager. Just before the deadline for the task announcement associated with the perceived nearest manager, the node submits a bid (Fig. 2). The bid message supplies the position of the bidder and a description of its sensors. A manager uses this information to select a set of bidders that covers its area of responsibility with a suitable variety of sensors, and then awards *a signal* contract on this basis (Fig. 3). The award message specifies the sensors that a contractor must use to provide signal-feature data to its manager.

IV. The Contract Net Protocol

We now describe the messages of the protocol and the encoding of information in their slots. We also describe the processing of each message. The BNF specification of the protocol is presented in Appendix B. The reader may find it helpful to refer to the specification while reading the following sections.

A. The Basic Messages

1) *Task Announcements*

A node that generates a task normally initiates contract negotiation by advertising existence of that task to the other nodes with a **task announcement** message. It then acts as the manager of the task. A task announcement can be addressed to all nodes in the net *(general broadcast),* to a subset of nodes *(limited broadcast),* or to a single node *(point-to-point).* The latter two modes of addressing, which we call *focused addressing,* reduce message processing overhead by allowing nonaddressed nodes to ignore task announcements after examining only the **addressee** slot. The saving is small, but is useful because it allows a node's communication processor alone to decide whether the rest of the message should be examined and further processed. It is also useful for reducing message traffic when the nodes of the problem solver are not interconnected with broadcast communication channels.

As shown in the example, a task announcement has four main slots. The **eligibility specification** is a list of criteria that a node must meet to be eligible to submit a bid. This slot reduces message traffic by pruning nodes whose bids would be clearly unacceptable. In a sense, it is an extension to the addressee slot. Focused addressing can be used to restrict the possible respondents only when the manager knows the *names* of appropriate nodes. The eligibility specification slot is used to further restrict the possible respondents when the manager is not certain of the names of appropriate nodes, but can write *a description* of such nodes.[1,2]

To: 25
From: 42
Type: BID
Contract: 22-3-1
Node Abstraction:
POSITION LAT 62*N* LONG 9*W*
SENSOR NAME *SI* TYPE *S*
SENSOR NAME *S2* TYPE *S*
SENSOR NAME *Tl* TYPE *T*

Fig. 2. Signal task bid.

To: 42
From: 25
Type: AWARD
Contract: 22-3-1
Task Specification:
SENSOR NAME *S1*
SENSOR NAME *S2*

Fig. 3. Signal task award.

The **task abstraction** is a brief description of the task to be executed. It enables a node to rank the task relative to other announced tasks. An abstraction is used rather than a complete description in order to reduce the length of the message.[3]

The **bid specification** is a description of the expected form of a bid. It enables the manager to specify the kind of information that it considers important about a node that wants to execute the 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 is relevant to the task, rather than a complete description. This both simplifies the task of the manager in evaluating bids and further reduces message traffic.

The **expiration time** is a deadline for receiving bids. We assume global synchronization among the nodes. However, time is not critical in the negotiation process. For example, bids received after the expiration time of a task announcement are not catastrophic: at worst, they may result in a suboptimum selection of contractors.

a) *The Common Internode Language*

It is useful to encode slot information in a single high-level language understandable to all nodes We call this a *common internode language.* Such a language, along with a high-level programming language (for transfer of procedures between nodes), forms a common basis for communicating slot information among the nodes.

[1] Note that focused addressing is typically a heuristic process, since the information upon which it is based may not be exact (e.g., it may be inferred from prior responses to task announcements).

[2] We will see that *all* messages in the protocol that can be addressed to more than one node have an addressee slot and an eligibility specification slot to accommodate addressing by name and by description.

[3] A numerical *priority* measure is not, in general, sufficient to allow potential contractors to rank announced tasks. It assumes first, that all nodes agree on what constitutes an important task, and second, that the importance of a task can be captured in a one-dimensional quantity.

While the contract net protocol offers a framework that specifies the *type* of information that is to fill a message slot, it remains the difficult task of the user to specify the actual *content* of the slot for any particular problem domain. In this. sense, the protocol is similar to AI problem-solving languages like PLANNER [4], which supply a framework for problem solving (e.g., the notions of goal specifications and theorem patterns), but leave to the user the task of specifying the content of that framework for any given problem.

CNET does, however, offer the user some additional assistance. It provides a very simple language, based on an *object, attribute, value* representation. The language includes a simple grammar, predefined for each slot, and a number of predefined domain-independent terms (e.g., TASK, TYPE, PROCEDURE, and NAME). The representation, the grammars, and the domain-independent terms are offered to the user to help him organize and specify the slot information. He must augment the language with domain-specific terms (e.g., SENSOR) as needed for the application at hand.

A message that does not have to be understood by many nodes (e.g., messages exchanged by a manager and contractor during execution of a contract) can be usefully encoded in a private language. This can reduce both the length of the messages and the overhead required to process them. In CNET, such "private" information is preceded by an "escape" character; this allows private information to be inserted in any message, even one that includes some public information encoded in the normal manner.

2) *Task Announcement Processing*

In CNET, all tasks are typed. For each type of task, a node maintains a rank-ordered list of announcements that have been received and have not yet expired. Each node checks the eligibility specifications of all task announcements that it receives. This involves ensuring that the conditions expressed in the specification are met by the node (e.g., MUST-HAVE SENSOR). If it is eligible to bid on a task, then the node ranks that task relative to others under consideration.

Ranking a task announcement is, in general, a task-specific operation. Many of the operations involved in processing other messages are similarly task-specific. CNET defines a *task template* for each type of task. This template enables a user to specify the procedures required to process that type of task. In Appendix E we describe the roles of the required procedures, together with the default actions taken by CNET when the user chooses to omit a procedure. In the following sections, whenever reference is made to task-specific actions, the reader may refer to Appendix E for further details.

3) *Bidding*

This announcement-ranking activity proceeds concurrently with task processing in a node until the task processor (see Appendix A) completes processing of its current task and becomes available for processing another task. At this point, the contract processor is enabled to submit bids on announced tasks. It checks its list of task announcements and selects a task on which to submit a bid. If there is only one type of task, the procedure is straightforward. If, on the other hand, there are a number of task types available, the node must select one of them. The current version of CNET selects the most recently received task (older tasks are more likely to have been already awarded).

An idle node can submit a bid on the most attractive task when either of the following events occur: 1) the node receives a new task announcement or 2) the expiration time is reached for any task announcement that the node has received. At each opportunity, the node makes a (task-specific) decision whether to submit a bid or wait for further task announcements. (In the *signal* task, a potential contractor waits for further announcements in an attempt to find the closest manager.)

The **node abstraction** slot of a bid is filled with a brief specification of the capabilities of the node that are relevant to the announced task. It is written in the form indicated by the bid specification of the corresponding task announcement.

The node abstraction slot can also include a number of REQUIRE statements (e.g., REQUIRE PROCEDURE NAME FFT). Statements of this form are used by a bidder to indicate that it needs additional information if it is awarded the task. REQUIRE statements can be made if two conditions are met: 1) the required objects were not preceded by MUST-HAVE terms in the eligibility specification of the task announcement and 2) the objects are *transferable;* that is, they can be transferred by message. (A procedure falls into this class, but a hardware device does not.)

The task template is helpful here. If a node receives an announcement for a type of task with which it is not familiar (i.e., does not have the template), then it can request the template as a convenient shorthand for the entire set of procedures associated with that type of task.

4) *Bid Processing*

Contracts are queued locally by the manager that generated them until they can be awarded. The manager also maintains a rank-ordered list of bids that have been received for the task. When a bid is received, the manager ranks the bid relative to others under consideration. If, as a result, any of the bids are determined to be *satisfactory,* then the contract is awarded immediately to the associated bidder. (The definition of *satisfactory* is task-specific.) Otherwise, the manager waits for further bids.

Because a manager is not forced to always wait until the expiration time before awarding a contract, the average negotiation time for contracts is reduced over that of DCS [2].

If the expiration time is reached and the contract has not yet been awarded, several actions are possible. The appropriate action is task-specific, but the possibilities include: awarding the contract to the most acceptable bidder(s); transmitting another task announcement (if no bids have been received); or waiting for a time interval before transmitting another task announcement (if no acceptable bids have been received). This is in contrast to the traditional view of task allocation where the most appropriate node available at the time would be selected.

Successful bidders are informed that they are now contractors for a task through an **announced award** message. The **task specification** slot contains a specification of the data

needed to begin execution of the task, together with any additional information requested by the bidder.

5) *Contract Processing, Reporting Results, and Termination*

Once a contract has been awarded to a node, it follows the state transition diagram shown in Appendix C. The data structures shown in Appendix D form a local context for communication between the contractor and manager (and other nodes) about the task being performed.

The **information message** is used for general communication between manager and contractor during the processing of a contract. (See Section IV-B.3 for further discussion of this message.)

The **report** is used by a contractor to inform the manager (and other report recipients, if any) that a task has been partially executed (an **interim report)** or completed (a **final report).** The **result description** slot contains the results of the execution. Final reports are the normal method of result communication. Interim reports, however, are useful when generator-style control is desired. A contractor can be set to work on a task and instructed to issue interim reports whenever the next result is ready. It then suspends the task until it is instructed by the manager to *continue* (with an information message) and produce another result.

The manager can also terminate contracts with a **termination** message. The contractor receiving such a message terminates execution of the contract indicated in the message and all of its outstanding subcontracts.

6) *Negotiation Tradeoffs*

In this section, we discuss choices made in the CNET implementation of the negotiation process. In the main, our concern has been with problem solving. We have been more interested in the types of information that must be passed between nodes than with these aspects of the negotiation process. As a result, the choices are only tentative and warrant further detailed analysis.

Because bids are binding and a node is allowed to have more than one bid outstanding at a time, a node may receive multiple awards. These are queued for processing in order of receipt. The cost is potentially slower overall system performance (the load may be less evenly balanced) than would be the case if multiple awards were prevented.

If nodes could refuse awards (as in DCS [3]), multiple awards could be prevented. However, the cost is at least one additional acknowledgment message per transaction. In some cases, it may be many additional messages (if an award is refused by several bidders).

Similarly, if a node could only have a single bid outstanding, multiple awards could be prevented. However, the cost would be significant delay. Nodes would be forced to remain idle until a task announcement had expired to find that their bids had been rejected, and have to start the process again. This could lower overall system performance.

The above delay could be reduced in some instances by explicitly informing unsuccessful bidders that their bids had been refused. The cost, however, would likely be a very large increase in message traffic, assuming that there are several bidders per task. In addition, it would only lower the delay for contracts that were awarded before the expiration times of their task announcements were reached.

We have allowed a node to bid at intervals related to the receipt of task announcements rather than at fixed intervals. It is intuitively appealing, offers a reasonable compromise between message traffic and delay in allocating tasks, and has exhibited good performance in experience to date with CNET.

We have chosen to allow a node to submit, at most, a single bid at each opportunity. This reduces message traffic and the possibility that a single node could bid on far more tasks than it could process. For the same reasons, we allow only idle nodes to submit bids. Because of these choices, contracts are not-centrally notarized as they were in DCS [2]. This further reduces message traffic and maintains the distributed nature of the negotiation process.

The current version of CNET uses a nonpreemptive scheduler in each node. It would appear to be useful, however; to allow preemptive scheduling instead of simply queueing contracts in order of receipt. This would help avoid the situation where time-critical tasks are not executed soon enough because less important tasks are queued ahead of them. The difficult question here is determining the criteria for preemption and providing a place for them in the common internode language. We are currently exploring this issue.

B. Complications and Extensions

In the following sections we consider some problems with the basic negotiation mechanism and present extensions to solve these problems. The extensions are evolving as we gain more experience with using the protocol in practical applications.

1) *Immediate Response Bids*

We have, thus far, discussed the negotiation process under the assumption that a node cannot submit a bid until it goes idle and is actually ready to process a new contract. This strategy can, however, lead to difficulty. For instance, a node that issues a task announcement may not receive any bids for one of several reasons: 1) there are no idle nodes; 2) some node is both idle and eligible, but ranks the task too low; or 3) no node is capable of working on the task even if it were idle (as may happen if the eligibility specification is too stringent or no node has the data necessary for executing the task). In the first two cases, the task announcement may be usefully reissued until a bid is obtained from an idle node; in the third case it would be pointless. Therefore, a node requires a way of determining what caused the lack of response.

A class of bids we call **immediate response** bids offers a mechanism for doing this. Three such bids have been identified, allowing a node to indicate that it is eligible but BUSY, that it is INELIGIBLE, or that it gave the task a LOW RANKING. A manager can specify that nodes are to respond in any of these cases or to respond in a subset of the cases (e.g., *respond if eligible, but busy).*

A node receiving a task announcement whose.bid specification asks for an immediate response bid does not deal with

the announcement in the usual way (ranking it immediately, but waiting until it is idle to submit a bid), but instead responds immediately with either a standard bid or the appropriate special form.

The immediate response mechanism permits a manager to take a more appropriate course of action if a task announcement elicits no bids. The normal procedure is to simply reissue the task announcement. If this continues to elicit no bids, then the manager can specify an immediate response bid. If the response is uniformly BUSY, then the manager can wait and reissue the task announcement later. If all nodes are INELIGIBLE, then the manager may loosen the eligibility specification. If all nodes gave the task a LOW RANKING, the manager can wait and reissue, in the hope that the more interesting tasks currently occupying the nodes will soon be done. The manager may also decide to execute the task itself when it becomes idle.

2) *Directed Contracts*

The normal contract negotiation process can be simplified in some instances, with a resulting enhancement in the efficiency of the protocol. If a manager knows exactly which node is appropriate for execution of a task, a **directed contract** can be awarded. No task announcement is made and no bids are submitted. Instead, a **directed award** message is sent to the selected node without negotiation.

In addition to the low-level acknowledgment used by an underlying communication protocol to ensure reliable communication, the contract net protocol uses a further high-level **acknowledgment** message for directed award messages. This allows a refusal or negative acknowledgment of the form, I *can't execute this contract because . . .*, in addition to the low-level negative acknowledgment of the form, *I didn't receive your message correctly.*

If the node that receives a directed award either does not meet the eligibility specification or does not have a template for the task, then it transmits a **refusal** to the manager. Otherwise, the node uses task-specific criteria to determine whether to transmit an **acceptance** or **refusal** to the manager. The refusal message contains a **refusal justification** slot that specifies the reasons for the refusal (e.g., INELIGIBLE or NO TEMPLATE).

The action taken by a manager upon receipt of a refusal is also task-specific, in general.

3) *Request and Information Messages*

If a simple transfer of information is required, then a request–response sequence can be used without further embellishment. The **request** message is used to encode straightforward requests for information. The **information** message is used to respond to a request, and for manager/contractor communication during the execution of a contract (Section IV-A.5). This message is also used in *result-sharing* [9], a style of distributed problem solving in which nodes cooperate by sharing partial results.

If a node that receives a request message meets the eligibility specification and has the necessary information, then it responds with an information message. No task-specific decisions need to be made.

If a node that receives an information message meets the eligibility specification, then the action it takes is task-specific. If the contract named in the message is one that the node is processing, then the appropriate action is determined by a procedure associated with that contract. Otherwise, the node takes a default action. (The latter possibility arises in result-sharing because nodes communicate partial results to other nodes without being requested to do so, and without being explicitly linked by a contract.)

4) *Node Available Messages*

The protocol has been designed to allow a reversal of the normal negotiation process. When the processing load on a net is high, most task announcements will not be answered with bids because all nodes will be busy. Hence, the protocol includes a **node available** message. A node that transmits such a message is idle and searching for a task to execute. In this case, the **eligibility specification** is a list of criteria that the node will look for in a task. The **node abstraction** is a brief specification of the capabilities of the node and the **expiration time** is a deadline for receiving an award.

When a node receives a node available message, it tries to determine if it can match the node with a task that it has announced, but not yet awarded. First, it tries to match the eligibility specification in the node available message against the task abstractions of the tasks that it has waiting. A task for which this match succeeds is of interest to the volunteering node. The manager then tries to match the eligibility specification of the task against the node abstraction of the node available message. If this match succeeds, then the node is the sort which is required for the task. Hence, even in this case we have the mutual selection aspect, as the volunteering node selects a task and the task specifies the criteria necessary in a volunteer. If the two matches are successful, the manager sends a directed award message to the volunteer. If a node has several tasks for which both matches succeed, then CNET selects the oldest task (it is least likely to be awarded in the normal way).

In CNET, a node maintains a list of unexpired node available messages. Before it announces a new task, a node first tries to find a suitable volunteer in the list. If several volunteers are suitable, the newest volunteer is selected (it is least likely to have received an award).

A node can thus acquire a contract in one of two ways: it can wait for a suitable task announcement and submit a bid; or it can transmit a node available message and wait for a directed award. The decision as to which method to use is net-load dependent. If the net is not heavily loaded, then the use of the task announcement is warranted in all cases, since the availability of a task to be executed is the event of primary importance. If the net is heavily loaded, then unlimited use of task announcements serves only to saturate the available communication channels. In this case, the availability of an idle node is the event of primary importance and the use of node available messages is preferred.

In CNET, a node selects one of the two modes based on experience with its own task announcements. The scheme is rudimentary, however, and warrants further analysis.

V. Dynamic Distribution of Information

The contract net protocol enables dynamic distribution of information (i.e., procedures and data) via three methods. First, a node can transmit a request directly to another node for the transfer of the required information. The response is the information requested (e.g., the code for a procedure). Second, a node can broadcast a task announcement in which the task is a transfer of information. A bid on the task indicates that the bidder has the information and is willing to transmit it. Finally, a node can note, in its bid on a task, that it requires particular information in order to execute the task. The manager can then send the required information in the award message if the bid is accepted.

Dynamic distribution enables effective use of available computational resources: a node that is standing idle because it lacks information required to execute a previously announced task can acquire that information as indicated above. This also means that nodes do not have to be preloaded with extensive amounts of information which may or may not prove useful. Dynamic distribution also facilitates the addition of a new node to an existing net; the node can dynamically acquire the procedures and data necessary to allow it to participate in the operation of the net. This is especially useful in the distributed sensing application.

Dynamic distribution works, first, because all nodes know the syntax of the contract net protocol, which enables them to identify the slots that contain task-dependent information (e.g., the eligibility specification of a task announcement); second, because they know how to parse the common internode language, which enables them to identify the information in the slots that they do not possess; and third, because a negotiation process is used to effect task distribution, which makes it possible for a new node to begin participating in the operation of the net by listening to the messages being exchanged (specifically, task announcements) and submitting bids. This may be contrasted with more traditional task allocation schemes that explicitly assign tasks to nodes. In such schemes, new nodes must be explicitly linked to other nodes in the network.

VI. Discussion

We have presented a high-level protocol for distributed problem solving. The protocol facilitates distributed control of cooperative task execution with efficient internode communication. It has been designed to provide a more powerful mechanism for *connection* than is available in current problem-solving systems. The message types have been selected to capture the types of interactions that arise in a task-sharing approach to distributed problem solving. The message slots have been selected to capture the types of information that must be passed between nodes to make these interactions effective.

The connection of nodes with tasks to be executed and nodes that can execute those tasks is more general than load balancing. The difference can be illustrated by comparing the information placed in the slots of the task announcement with that used in DCS. In that system, a task announcement *(request for quotation)* contains only the name of the process to be executed and the amount of memory required. A bid is a measure of the excess memory available in a node. Note, however, that this is only one of the possible dimensions along which tasks and nodes can be evaluated. In general, such evaluation is task-dependent (as we saw in Section III) and may require a different form of information for each task. Recognizing this, the contract net protocol takes a wider perspective and allows a range of descriptions to be passed between the manager and bidders. In more general terms, since our concern is with problem solving, more attention is paid to the type of information that must be transferred between nodes to solve the connection problem. The connection that is effected with the contract net protocol is an extension to the *pattern-directed invocation* used in many AI programming languages (see [8] for a more in-depth discussion).

There are several reasons for adopting a distributed approach to problem solving. These include speed, reliability, extensibility, and the ability to handle applications that have a natural spatial distribution. The design of the contract net protocol attempts to ensure that these potential benefits are indeed realized.

In order to achieve high speed we wish to avoid bottlenecks. Such bottlenecks 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, data, and *KS's*. Control is distributed through the use of negotiation to achieve connections for task distribution. Every node is capable of assigning and accepting tasks, and managers and contractors simultaneously seek each other out. Data and *KS's* are also distributed dynamically as part of the negotiation process or with a request-response mechanism.

The contract net protocol includes several elements aimed at avoiding communication channel saturation. First, the information in task announcements (like eligibility specifications) helps eliminate extraneous message traffic. Similarly, bid messages can be kept short and "to the point" through the use of the bid specification mechanism. Second, specialized interactions, like directed contracts and requests, reduce communication for transactions that do not require the complexity of negotiation. Third, the protocol enables dynamic distribution of information on an as-required basis. Finally, it provides a very general form of guidance in determining appropriate partitioning of problems: the notion of tasks executed under contracts suggests relatively large task sizes. This is important if the speedup obtained from distribution is not to be outweighed by the effort required to effect that distribution.

Distribution of control also enhances reliability and permits graceful degradation of performance in the case of individual node failures. There are no nodes whose failure can completely block the contract negotiation process. In addition, recovery from the failure of a node is aided by the presence of explicit links between managers and their contractors. The failure of any contractor, for example, can be detected by its manager;

the contract for which it was responsible can then be reannounced and awarded to another node.

The use of a negotiation process for allocation of nodes to tasks facilitates the extension of an existing net to include new nodes. Such nodes can begin to participate in the operation of the net without first explicitly informing the other nodes of their presence. New node entry is also facilitated by the common internode language and dynamic distribution of information.

The ability to handle applications with a natural spatial distribution is facilitated by the local nature of negotiation. A net is able to configure itself dynamically according to the positions of the nodes and the ease with which they can establish communication.

The protocol is best suited to problems in which it is appropriate to define a hierarchy of tasks (e.g., heuristic search) and a hierarchy of levels of data abstraction (e.g., audio or video signal interpretation). Such problems lend themselves to decomposition into a set of relatively independent subtasks with little need for global information or synchronization. Individual subtasks can be assigned to separate processor nodes; these nodes can then execute the subtasks with little need for communication with other nodes.

VII. Conclusion

The main contribution of the contract net protocol is the mechanism it offers for structuring high-level interactions between nodes for cooperative task execution. It stresses the utility of negotiation as an interaction mechanism. Negotiation can be used at different levels of complexity. At one extreme, it is a means of achieving task distribution with distributed control and shared responsibility for tasks to maintain reliability and avoid bottlenecks. At the other extreme, the two-way transfer of information and mutual selection attributes of negotiation make possible a finer degree of control in making resource allocation and focus decisions than is possible with traditional mechanisms.

Appendix A

Contract Node Architecture

We view a node as having four major functional components (Fig. 4): *a local database, a communication processor* that handles low-level message traffic with other nodes, *a task processor* that carries out the computation associated with user tasks, and *a contract processor* that processes high-level protocol messages and manages node resources. It is assumed that the functions of the three processors are carried out concurrently.

Appendix B

Contract Net Protocol Specification

The BNF specification of the contract net protocol is shown in Fig. 5. Nonterminal symbols are enclosed by "( )," terminal symbols are written without delimiters, and nonterminals whose specific expansion is not germane to the discussion

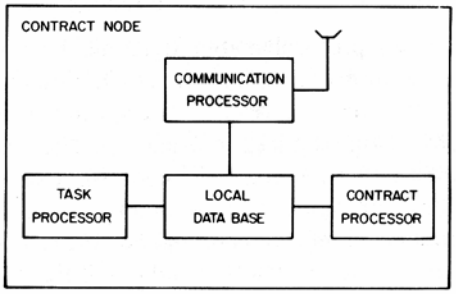

Fig. 4. Contract node architecture.

<message> ⇒ <header> <addressee> <originator> <text> <trailer>
<header> ⇒ [line-header] [identifier] [time] [acknowledge]
<trailer> ⇒ [error-control] [line-trailer]
<addressee> ⇒ [net-address] | [subnet-address] | [node-address]
<originator> ⇒ [node-address]
<text> ⇒ [cctext] | <pstext>
<pstext> ⇒ <task-announcement> | <bid> | <announced-award> |
<directed-award> | <acknowledgment> | <report> |
<termination> | <node-available-message> |
<request-message> | <information-message>
<task-announcement> ⇒ TASK-ANNOUNCEMENT [name]
{task-abstraction} {eligibility-specification}
{bid-specification} [expiration-time]
<bid> ⇒ BID [name] {node-abstraction}
<announced-award> ⇒ ANNOUNCED-AWARD [name] {task-specification}
<directed-award> ⇒ DIRECTED-AWARD [name] {task-abstraction}*
{eligibility-specification} {task-specification}
<acknowledgment> ⇒ ACCEPTANCE [name]*
⇒ REFUSAL [name] {refusal-justification}
<report> ⇒ INTERIM-REPORT [name] {result-description}
FINAL-REPORT [name] {result-description}
<termination> ⇒ TERMINATION [name]*
<node-available-message> ⇒ NODE-AVAILABLE {eligibility-specification}*
{node-abstraction} [expiration-time]
<request-message> ⇒ REQUEST [name] {eligibility-specification}*
{request-specification}
<information-message> ⇒ INFORMATION [name] {eligibility-specification}*
{information-specification}

Fig. 5. Contract net protocol specification.

are enclosed by "[ ]." Slots that are to be filled with information encoded in the common internode language are enclosed in "{ }." Message types that need not be included in a basic implementation are followed by "*."

One possible set of low-level message slots is shown for completeness. Other variations may be dictated by the specific communication architecture for which the protocol is implemented. Briefly: **line-header** delimits the beginning of a message; **identifier** is a unique identifier for the message; **time** specifies the time at which the message is transmitted; **acknowledge** is set only if acknowledgment of receipt of the message is required (this slot is included because delivery of some messages is not essential to the operation of the protocol (e.g., broadcast task announcements) and acknowledgments for such messages might greatly increase message traffic); **error-control** is used by an addressee to determine that the message has been correctly received; and **line-trailer** delimits the end of the message. Three forms of **addressee** slot are shown. These are used for general broadcast, limited broadcast, and point-to-point messages, respectively.

There are two forms of **text** slot. **cctext** (communication control text) indicates that the message is for checking net integrity, acknowledging receipt of messages, maintaining

basic net communication data, such as routing tables, and so on. Messages of the form *Hello* and *I heard you,* exchanged periodically by IMP's in the ARPANET as status checks, fall into this category.

pstext (problem-solving text) indicates that the message is for high-level problem solving (see Section IV).

APPENDIX C

CONTRACT PROCESSING STATES

Contracts are processed according to the state transition diagram shown in Fig. 6, which uses the terminology of operating systems [1]. The contract processor of a node (Appendix A) is responsible for moving a contract through the states of the diagram. The standard progression through the states is shown by solid lines. Optional progressions are shown by dashed lines.

When a contract is awarded to a node (IN), it is placed in the READY state where it waits until the task processor is available. At that time, the contract is placed in the EXECUTING state and its processing is started. If subcontracts are generated, they are placed in the ANNOUNCED state. A subcontract may either be awarded to another node (OUT) or to this node (READY).

If execution cannot continue on the contract until some event occurs (e.g., receipt of a subcontract report), then the contract is placed in the SUSPENDED state; and another contract in the READY state is selected for processing. When the awaited event occurs, the contract is transferred back to the READY state where it awaits further processing.

When processing of the contract has been completed, it is placed in the TERMINATED state; and another contract in the READY state is selected for processing. Recently completed contracts are held in the TERMINATED state in order to facilitate recovery from the failure of a manager.

If a contract in the EXECUTING state is moved to either the SUSPENDED or TERMINATED state and if there are no contracts in the READY state, then the node attempts to acquire a new contract—either by making a bid on a recent task announcement or by transmitting a node available message (Section IV).

The scheduler used by a contract processor in the current version of CNET is nonpreemptive and gives priority to a contract whose execution can continue (e.g., due to receipt of a report) over a contract whose execution has not yet been started.

APPENDIX D

CONTRACT SPECIFICATION

A node maintains a structure of the form shown in Fig. 7 for each task for which it is the contractor. Such a structure forms a local context for the execution of a task.

name is a unique identifier for the contract. **manager** is the node that generated the task. **report recipients** are the nodes to which reports for the contract are to be sent. The default report recipient is the manager. **related contractors** are the nodes that are working on related contracts (e.g., subcontracts of the same contract).

The report recipients and related contractors slots facilitate more flexible communication and cooperation. They give IN node an explicit indication of other nodes in the net with which it may be useful to communicate (e.g., to enable use of focused addressing in task announcements) and help to reduce message traffic. Messages from one related contractor to another, for example, can be addressed directly and therefore do not have to follow the communication paths of the control hierarchy.

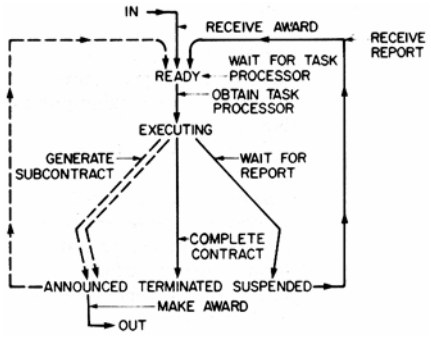

Fig. 6. Contract processing states.

<contract> ⇒ [name]
[manager]
[report-recipients]
[related-contractors]
<task>
[results]
<subcontract-list>

Fig. 7. Contract structure.

task is filled with a structure of the form shown in Fig. 8. The structure lists the type of the task, its specification, and the procedures that are required to handle the task in a contract net. These procedures are described in Appendix E.

results are the outcome of executing a task. They are transmitted to the report recipients of the contract. **subcontract-list** is the collection of subtasks generated from the initial contract. This slot is filled with a list of structures of the form shown in Fig. 9. Such structures contain the information held by a manager for a subtask.

name is a unique identifier for the subcontract. **contractor** is the name of the node responsible for its processing. **task** is the task structure (Fig. 8). **results** are the outcome of executing the subtask. **predecessors** are the names of subcontracts that are preconditions for the subcontract and **successors** are the names of subcontracts for which the subcontract is a precondition. Predecessors and successors are necessary for tasks that must be performed in a particular order, such as those that occur in planning applications [6]. A subcontract will not be announced (Section IV-A.1) until its predecessors have been completed.

APPENDIX E

TASK PROCESSING PROCEDURES

In this appendix, we discuss the task-specific procedures, the roles they play, and the default actions taken in CNET when no procedure is provided by the user. In the terms of the functional model of a node presented in Appendix A, the *execution procedure* runs in the task processor. The rest of the procedures run in the contract processor.

Announcement Procedure: Called each time a subtask is generated. Its role is to determine whether the task should be announced or awarded directly. In either case, it must determine the addressee. It must also compute the information re-

```
<task> ⇒ [name]
         [type]
         [specification]
         [announcement-procedure]
         [announcement-ranking-procedure]
         [bid-procedure]
         [bid-ranking-procedure]
         [award-procedure]
         [acknowledgment-procedure]
         [refusal-processing-procedure]
         [report-acceptance-procedure]
         [termination-procedure]
         [information-acceptance-procedure]
         [execution-procedure]
```

Fig. 8. Task structure.

```
<subcontract> ⇒ [name]
                [contractor]
                <task>
                [results]
                [predecessors]
                [successors]
```

Fig. 9. Subcontract structure.

quired to fill the slots of a task announcement or directed award and indicate the names of any predecessor subcontracts. The default is to announce the task to all nodes with a task abstraction that states the task type and a default expiration time.

Announcement Ranking Procedure: Called when a task announcement is received. Its role is to rank the new announcement relative to others under consideration. The default is to give the most recently received task announcement the highest ranking.

Bid Procedure: Called when a node has an opportunity to submit a bid. Its role is to decide whether or not to submit a bid and to add any common internode language statements to those called for in the bid specification of the associated task announcement. No default action is taken.

Bid Ranking Procedure: Called when a bid is received. Its role is to rank the new bid relative to others under consideration and determine whether the contract should be awarded to any of the current bidders. The default is to insert the new bid into the list of bids so that bids that REQUIRE the least information (Section IV-A.3) are ranked highest.

Award Procedure: Called when the expiration time for a task announcement is reached and the contract has not yet been awarded. Its role is to decide what action is to be taken. The default is to award the contract to the bidder that REQUIRE's the least additional information (or the first bidder in case of a tie) or to transmit another task announcement if no bids have been received.

Acknowledgment Procedure: Called when a node receives a directed contract for which it is eligible and has a task template. Its role is to decide whether to accept or reject the contract. The default is to accept.

Refusal Processing Procedure: Called when a refusal message is received. Its role is to determine what action is to be taken. The default is to transmit a task announcement.

Report Acceptance Procedure: Called when a report message is received. Its role is to determine what to do with the contents of the result description slot. The default is to add the contents of the slot to the results slot of the contract.

Termination Procedure: Called when a contract that has been terminated is about to be deleted from the local database. (CNET maintains a list of the recently terminated contracts [Appendix C].) Its role is to take any specialized action that is required before the contract is deleted (e.g., storing contract slot information in the local database). No default action is taken.

Information Acceptance Procedure: Called when an information message is received. Its role is to determine what to do with the contents of the information specification slot. The default is to store the information in the local database.

Execution Procedure: Called when processing is started on the contract. Its role is to perform the computation required to execute the task.

ACKNOWLEDGMENT

Some aspects of this work are being pursued in collaboration with R. Davis at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology, Cambridge. The contributions of B. Buchanan and G. Wiederhold are also gratefully acknowledged.

REFERENCES

[1] P. Brinch Hansen, *Operating System Principles*. Englewood Cliffs, NJ: Prentice-Hall, 1973.

[2] D. J. Farber and K. C. Larson, "The structure of the distributed computing system—Software," in *Proc. Symp. on Comput.-Commun. Networks and Teletraffic*, J. Fox, Ed. Brooklyn, NY: Polytechnic Press, Polytechnic Inst. of Brooklyn, Apr. 1972, pp. 539-545.

[3] D. J. Farber, J. Feldman, F. R. Heinrich, M. D. Hopwood, K. C. Larson, C. Loomis, and L. A. Rowe, "The distributed computing system," *IEEE COMPCON Spring*, 1973, pp. 31-34.

[4] C. Hewitt, "Description and theoretical analysis (using schemata) of PLANNER: A language for proving theorems and manipulating models in a robot," Mass. Inst. Technol., Cambridge, MA, AI TR 258, Apr. 1972.

[5] V. R. Lesser, "Cooperative distributed processing," Dep. Comput. Inform. Sci., Univ. of Massachusetts, Amherst, MA, COINS TR 78-7, May 1978.

[6] E.D. Sacerdoti, "A structure for plans and behavior," SRI Int., Menlo Park, CA, AIC TN 109, Aug. 1975.

[7] R. G. Smith and R. Davis, "Applications of the contract net framework: Distributed sensing," in *Proc. ARPA Distributed Sensor Net Symp.*, Pittsburgh, PA, Dec. 1978, pp. 12-20.

[8] R. G. Smith, "A framework for problem solving in a distributed processing environment," Dep. Comput. Sci., Stanford, CA, STAN-CS-78-700 (HPP-78-28), Dec. 1978.

[9] R. G. Smith and R. Davis, "Cooperation in distributed problem solving," in *Proc. /979 Int. Conf Cybern. Soc.*, Oct. 1979, pp. 366-371.

[10] R. G. Smith, "Applications of the contract net framework: Search," in *Proc. 1980 Nat. Conf. Canadian Soc. for Computational Studies of Intell.*, May 1980, pp. 232-239.

[11] R. F. Sproul and D. Cohen, "High-level protocols," *Proc. IEEE*, vol. 66, pp. 1371-1386, Nov. 1978.

[12] W. Teitelman, INTERLISP Reference Manual. Palo Alto, CA: Xerox Palo Alto Research Center, Dec. 1975.

Reid G. Smith (S'67-M'69-S'75-M'78) was born in Toronto, Ont., Canada, on October 4, 1946. He received the B.Eng. and M. Eng. degrees in electrical engineering from Carleton University, Ottawa, Ont., Canada in 1968 and 1969, respectively, and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA in 1979.

He has been associated with the Defence Research Establishment Atlantic in Dartmouth, N.S., Canada since 1969. He is currently engaged in research in artificial intelligence and distributed problem solving.