Report 77-12
Stanford -- KSL

Scientific DataLink

The Contract Net: A Formalism for the Control of Distributed Problem Solving.
Reid G. Smith,
Mar 1977

HPP-77-12
March 77

THE CONTRACT NET: A FORMALISM FOR THE CONTROL OF DISTRIBUTED PROBLEM SOLVING

By

Reid G. Smith[1]

Heuristic Programming Project,
Department of Computer Science,
Stanford University,
Stanford, California, 94305.

Abstract

This paper presents a formalism in which to express the control of problem solving in a distributed processor architecture. It is proposed that effective problem solving can be achieved in such an environment by dealing with individual tasks as contracts that exist between pairs of processor nodes which take on the roles of contract manager and contractor. Control is thus distributed throughout the net. The concept of priority is generalized to allow the use of arbitrarily complex priority descriptions by contractors to choose among competing contracts, and by contract managers to choose among competing contractors.

Descriptive Terms: distributed processing, parallel processing, contract net, contract, priority description.

1 Introduction

The past several years have seen a developing interest in parallel and distributed processing, coupled with advances in LSI technology[2],

---

[1] This work was supported by the Advanced Research Projects Agency under contract DAHC 15-73-C-0435. Computer facilities were provided by the SUMEX-AIM facility at Stanford University under National Institutes of Health grant RR-00785. The author is supported by the Research and Development Branch of the Department of National Defence of Canada. E. A. Feigenbaum, B. G. Buchanan, G. Wiederhold, K. G. Knutsen, and E. J. Gilbert provided a number of useful suggestions. F. K. Buelow was a valuable source of LSI technology data.

[2] Single silicon wafers with at least 10^6 active elements are expected to be produced by 1981 [30]. These wafers could be configured, for example, as 16 bit processors with 65 kilobytes of memory. Larger, faster single chip processors with more memory are expected to soon follow.

and the emergence of networks of resource-sharing computers [21] [23]. The focus of this paper is distributed processing, which we define as processing that involves one or both of the following attributes: physical decomposition of the processor into relatively independent processor nodes, and functional decomposition of the top level task into relatively independent subtasks. Distributed processing has been proposed as a method for achieving higher speed, more reliable computation than is possible with a uniprocessor approach [16] [2] [13]. It has also been proposed as a more appropriate model of some types of human problem solving [14], [20]. In addition, some applications have a natural spatial distribution, and use of a uniprocessor approach is inappropriate because of high communications channel cost or real-time response requirements.

Distributed processing is especially germane to Artificial Intelligence (AI) research. AI programs have grown dramatically in size and complexity in recent years, and the orders of magnitude increases in computational power that seem possible for distributed processors are extremely desirable. However, there are several major problems to be overcome before the full potential of distributed processing can be realized. Some examples are the small address space of the microprocessors often proposed as components of a distributed processor, the handling of global information, the difficulty of protecting different user processes from one another, and the lack of tools for performance measurement and evaluation (see, for example [28] and [9] for a more complete listing).

We are interested in the study of problem solving in a distributed processing environment, and in the constraints placed on a distributed processor architecture by the applications for which it is intended. In this paper, we present a formalism in which to express the control of distributed problem solving.

In AI research there are many instances of processing that is potentially distributed in nature. Applications which involve heuristic

search [29] [5], graph matching [6], and scene analysis [8], [33], for example, can often be partitioned into independent subtasks.

There are in addition several applications areas in which distributed processing and AI techniques appear attractive. Examples include distributed sensor nets (involving, for example, sonobuoys or radar sites with relatively powerful local processors), distributed command and control systems, and tasks involving multiple robots with individual processors.

Some AI systems have been implemented in a distributed manner. The HEARSAY II speech understanding system [26], for example, is partitioned into a number of cooperating, functionally-defined "knowledge-sources", in order to deal with the high error rate associated with the application of any single type of processing to the speech understanding task. It has also been implemented on a closely-coupled multiprocessor, C.mmp [9] [35].

Lenat's AM system [25] [24] is another example of a distributed approach, in which a group of expert procedures is given the task of discovering interesting results in elementary mathematics. Individual "experts" are able to pose new problems to the rest of the group via a central "agenda" mechanism, and remove problems they wish to attack. Lenat argues that knowledge can be appropriately organized as a group of such interacting modules. Hewitt's ACTOR model of computation uses a similar concept [19] [20]. The fundamental mode of interaction in ACTOR-based systems, however, is the explicit passing of messages between pairs of ACTORS (where an ACTOR is a fundamental computational entity [17]).

Selfridge's PANDEMONIUM [31] paradigm for pattern recognition is an early example of the use of distributed processing. Recognition is effected by a set of independent "feature demons", each looking for a particular feature in the input pattern, and "shouting" out its name when it finds an instance of the feature. The intensity of the shout corresponds to how confident the demon is that the feature is actually

present in the input pattern. A "decision-making demon" then assigns the input pattern to one of a number of pre-determined classes on the basis of the shouts of the feature demons.

2 Attributes of a Distributed Processor Architecture

As a prelude to the discussion of control of distributed problem solving, it is appropriate to discuss the constraints that are placed on the underlying distributed processor architecture by the applications for which it is intended. In each case, we will note the application, the architectural attribute it suggests, and how such an attribute might be achieved[3].

There is great variety in the control structures used in AI programming. Examples are the goal-driven "heterarchical" structures used in machine vision systems [34], the data-driven structures used in speech understanding systems [26], and the production rule interpreters used in knowledge-based systems [32] [7]. In order to support such control structures, the distributed processor should be composed of self-contained, autonomous processor nodes. It should also allow dynamic reconfiguration of its components so that a programmer can select the configuration best suited to a particular application. The key to achieving this capability is an interconnection mechanism by which all nodes can communicate with each other in a direct manner, both for private exchanges and for broadcast.

If we are to explore applications which involve a natural distribution of information, such as distributed sensor nets or command and control systems, the architecture must be able to support spatially distributed processor nodes. Communications channel cost, delay and reliability are therefore additional factors to be considered. Spatially distributed systems may be restricted to the use of a broadcast communications medium, such as packet radio [22], and it is therefore

---

[3] See Anderson and Jensen [1] for an extensive taxonomy of computer interconnection methods.

especially important to minimize overall bandwidth requirements. A distributed processor comprised of autonomous processor nodes that execute large "kernel" tasks is suggested to minimize the possibly adverse effects of these factors. The distribution of control among the nodes will also avoid the necessity for a small number of high bandwidth "master" nodes.

The architecture should be extensible and robust. It should be relatively simple to add new nodes at time instants and locations that are not pre-determined or regular. The performance of the distributed processor should degrade gracefully in the event of individual component failure. If, for example, the individual nodes of a distributed sensor net are to be battery-operated devices like sonobuoys, then the architecture of the net must allow for ready addition of new nodes, and the transfer of responsibilities from nodes that fail. Hence a simple interconnection scheme is desirable, nodes should be interchangeable to a large degree, and there should be a minimum of centralized functions and critical resources.

The architecture should have low logical complexity, and allow for an abundance of easily duplicated components in order to minimize manufacturing cost, but should at the same time have the capability to support nodes with different characteristics in order to maximize flexibility.

Finally, the architecture should allow for high speed computation at a reasonable cost. However, our current main goal is a flexible architecture combined with control mechanisms that will lead to effective problem solving in domains where a distributed approach is viable.

The above requirements lead us to a distributed processor architecture that is based upon a network of "loosely-coupled", asynchronous processor nodes, with distributed executive control, a flexible interconnection mechanism, and a minimum of centralized, shared resources. By loosely-coupled we mean that an individual node is

autonomous, and deals with large kernel tasks. We are thus referring to rather "coarse grain" parallelism, in which individual nodes are primarily involved with computation, pausing only occasionally to communicate with other nodes.

3 The Contract Net - Overview

The **CONTRACT NET** represents a formalism in which to express the control of problem solving within a distributed processor architecture. We would like to relate problem solving in such an architecture to the human model of a group of experts working together to complete some top-level task.[4] Each expert spends most of his time working alone on various subtasks he has partitioned from the top-level task by himself, or in cooperation with other members of the group. When he encounters a subtask either too large to handle alone, or one for which he has no expertise, he makes the new problem known to the group. For tasks which require special expertise, the expert may know another expert (or several other experts) in the group with such expertise, and notify him (them) directly. If the expert does not know anyone in particular who may be able to assist him, or if the new task requires no special expertise, then he can simply describe it to the group as a whole. If some other expert chooses to process the task, then he will obtain further details from the original expert, and the two will carry on private exchanges, separate from the rest of the group for the duration of the task. The two experts will have formed their own subgroup. Such exchanges are not, of course, limited to just two members. Various dynamically-defined subgroups may form and break up during the course of the work on the top-level task.

We can now map this human model into the framework of a distributed processor architecture. The experts are the individual processor nodes, and their expertise corresponds to hardware

---

[4] Such a model has also been used as a starting point by Lenat [24] [25] and Hewitt [18] [20].

characteristics, such as list processing hardware, specialized resources, such as secondary storage media, or particular applications software.

We have need of essentially three kinds of communication: **general broadcast**, **limited broadcast**, and **point-to-point**. General broadcast is used to announce new tasks or data to the whole group (i.e., all nodes). Limited broadcast is used to exchange information among the members of one of the dynamically-defined subgroups over the course of cooperation on a particular subtask, or related class of subtasks. Point-to-point communication is used by two group members to exchange information about a particular subtask of mutual interest.

We also require, as do human groups, some method of selecting the small number of problems that can be handled at any given instant from the large number generally available (i.e., a focusing mechanism). In the case of group problem solving, there are really two kinds of selection. First, an expert must choose among tasks competing for his attention. Second, an expert that generates a new task must choose among group members that offer assistance in processing the task.

These considerations have led us to develop the CONTRACT NET formalism, which can be described in general terms as follows. Individual tasks are dealt with as **contracts**[5]. A node that generates a task broadcasts its existence to the other nodes in the net as a **contract announcement**, and acts as the **contract manager** for the duration of that contract. **Bids** are then received from potential **contractors**, which are simply other nodes in the net. An **award** is made to one node which assumes responsibility for the execution of the contract. Subcontracts may be let in turn as warranted by the size of a contract, or a requirement for special expertise or data that the contractor does not have (similar to calling in a consultant). When a contract has been

---

[5] Contracts are used by Hewitt [17] [19], with a somewhat different emphasis. In ACTOR-based systems, a contract is a statement of abstract requirement, an expression of what is supposed to be accomplished by an ACTOR. Farber [10] also uses contracts in the Distributed Computer System as a resource allocation mechanism.

executed, a report is transmitted to the appropriate contract manager. Progress reports may also be required by the manager.

Contracts may be announced via the limited broadcast or point-to-point communications mechanisms, depending on the information about relevant contractors available to the contract manager. These contracts are directed contracts. For example, they may arise naturally in distributed data base applications. If a contract manager has some knowledge about the location of particular data, then his contract announcement will be directed to the node(s) believed to possess the data, so that the complete network is not needlessly involved.

There are several reasons why the notion of contracts is quite well-suited to the community of experts model of distributed computation. The use of contracts implies loose-coupling between processor nodes, in that contracts are typically relatively large undertakings (compared to the "fine grain" parallelism that has marked a number of parallel processor designs in the past [12]). Contractors are independent entities, as are the experts in the human model.

Broadcasting newly generated problems (both general and limited broadcast) is clearly suited to the model. Many systems use a central "agenda" to effect this type of communication [24] [4]. Such a mechanism does not, however, map well into a processor which is comprised of spatially distributed nodes, because of reliability and communications problems associated with it. If the agenda is kept in one central node, then the entire distributed processor is dependent on the proper functioning of that node. On the other hand, if the agenda is copied in all nodes, then updating is costly in terms of communications channel bandwidth. In addition, an agenda is a single ordered list of tasks to be processed, and does not appear suited to a distributed processing environment in which multiple goals may exist.

Contracting effectively distributes control throughout the network, thus allowing for flexibility and reliability. Decisions about what to do next are made as a result of relatively local considerations,

between pairs of processors, although the nature of contract announcements and bidding maintain an adequate global context; that is, the decision to bid on a particular contract is made on the basis of local knowledge (the task being processed in the node contemplating a bid), and global knowledge (other current contract announcements). [6] (see Section 5 for a more in-depth discussion of priorities and scheduling).

In addition, the decision as to which node will handle which task has two parts. The first part involves an evaluation of the task by potential task processors, and the second part involves an evaluation of the potential task processors by the generator of the task. The notions of contract announcements and bidding offer just this two part decision capability.

Finally, the CONTRACT NET stresses the idea that some one processor node (or small group of nodes) is responsible for particular tasks. The contract manager is responsible for following the progress of a contract, and the contractor is responsible for its execution. The failure of a contractor node is therefore not fatal, since the contract manager can re-announce the contract and recover from the failure[7]. At the top level, of course, other steps must be taken to make recovery from failure possible (like duplication, for example). This concept of responsibility is deemed essential for reliable, computation in the event of individual component failures. It is also useful as a focusing mechanism in distributed problem solving, in that new tasks are linked to older tasks that have been deemed suitably interesting to warrant the award of further contracts.

---

[6] By way of contrast, ACTORS are described as a purely local model of computation [20] in which there is no broadcast mechanism.

[7] Such failures can be discovered in a number of ways. For example, IMPs in the ARPAnet periodically exchange "Hello" and "I heard you" packets as status checks [15].

4 Contract Management

A node in a distributed processor can be described at a number of different levels of abstraction (e.g., the circuit level, the message-passing level, the contract level, and the applications level). In this section, we consider specification at the contract level, together with the functions that must be performed to effect contract management.

A node in the CONTRACT NET is composed of a CONTRACT PROCESSOR, MANAGEMENT PROCESSOR, and communications interface, all sharing a local memory. The contract processor is responsible for the applications-related computation of the node. The management processor is responsible for network communications (in concert with the communications interface), contract management, bidding, and the management of the node itself. Local memory is required to hold the basic contract management software and data, and the specific applications software and data required for contract execution. It is also required to supply communications buffers.

A node is thus a closely-coupled cluster of processors, memory, and associated hardware[8]. Individual nodes are not designated a priori as contract managers or contractors. Any node can take on either role, and during the course of problem solving, a particular node normally takes on both roles simultaneously for different contracts.

Figure 1 is a schematic representation of a node in the CONTRACT NET. Contracts are handled by the management processor through two lists: the MANAGEMENT LIST and the WAIT LIST. The management list contains information pertinent to the current contract (the contract being executed at the present time) and contracts (that were formerly current contracts) for which subcontracts have been generated but results are still pending. The wait list contains pointers to subcontracts that have been generated but are awaiting execution.

---

[8] In keeping with the definition of distributed processing presented in Section 1 a node could in fact be composed of a cluster of processes and a local memory in a single physical processor.

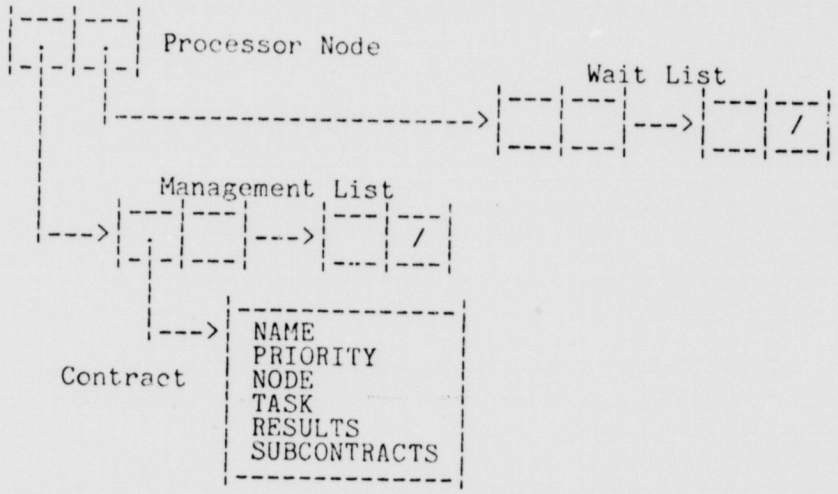

Figure 1: Processor Node Schematic Diagram

As shown in the diagram, a contract is represented as a record structure with the following fields: **NAME** - the name of the contract, **PRIORITY** - a description of the "importance" of the contract, **NODE** - the name of a processor node associated with the contract, **TASK** - a description of the task to be performed, **RESULTS** - a description of the results obtained to date, and **SUBCONTRACTS** - a pointer to the list of subcontracts that have been generated from the contract.[9]

In the following paragraphs, we describe the control cycle for a processor node. In Section 6, a simple heuristic search example is presented, and the reader may find it helpful to refer occasionally to that section while reading the following discussion.

When a node is awarded a new contract, the management processor creates a record structure for it and initializes the fields. The name of the contract, and its priority and task descriptions have already been specified by the contract manager, and are simply copied into the appropriate fields. The NODE field is filled in with the name of the contract manager. The RESULTS and SUBCONTRACTS fields are initially empty.

---

[9] To increase flexibility, the contents of the fields in our implementation are themselves arbitrary record structures.

The new contract record is placed at the head of the management list (the current contract is always found at the head of this list), and the contract processor is directed to begin execution. As execution proceeds, results are compiled in the RESULTS field. If no subcontracts are generated from the contract, then the management processor is informed when execution is completed, and it takes the appropriate action (e.g., transmission of results to the contract manager). In a heuristic search task, for example, no subcontracts would be necessary if the contract called for expansion of a tip node (in the graph to be searched) and the required applications software and data were already resident in the contractor node.

If subcontracts are generated from the contract, then the management processor is informed when each new one arises. It creates a record structure for the subcontract, and fills in the NAME, PRIORITY and TASK fields. The remaining fields are initially empty. The new record is then added to the list of subcontracts that have been generated by the current contract, and a pointer to the record is placed on the wait list in priority order (see Section 5 for details).

The management processor attempts to award subcontracts waiting for execution to other nodes in the net. When a subcontract is awarded, the name of its contractor is written in its node field. The node in which the subcontract was generated is its contract manager. For a particular contract, then, the processor node has links up one level to the contract manager (to whom results will be reported), and down one level to contractors (that are working on subcontracts generated by the contract). These links are required for effective contract management, but also provide for reliable computation as mentioned in Section 3.

When the execution of the current contract has been completed, the management processor is informed. It reports the results to the contract manager, removes the record for the contract from the management list, and attempts to acquire a new contract. Similarly, If the current contract has been fully partitioned into subcontracts (and the node

cannot continue execution of the contract without the results of these subcontracts), the management processor is again made aware of the need for a new contract.

If subcontracts are available on the wait list of the node itself, then the subcontract at the head of the list is installed as the current contract and the control cycle continues. In this case, the node acts as both contract manager and contractor for the new contract. If the wait list is empty, then the management processor attempts to obtain a new contract from another node in the net through the award mechanism[10].

When the node receives a message about a subcontract, the management processor is responsible for taking the appropriate action, such as integrating new information into the RESULTS field of the record for the contract that generated the subcontract, passing on a report to the contract manager, or sending contract termination messages to other contractors working on related subcontracts (e.g., in the case of an OR node in a graph search [29]). Reports to contract managers may, of course, cause further messages to ripple through the net.

It is apparent from the above discussion of contract management that messages in the CONTRACT NET are transmitted for essentially two purposes: action and information. A message sent for the purpose of action operates, in effect, like an interrupt to a processor node (e.g., causing it to terminate a contract). Information messages are primarily used to notify applications programs being executed at a node of a particular event of interest (e.g., to notify nodes other than a contract manager node that a particular result has been obtained).

5 Contract Descriptions

As shown in Figure 1, a contract record has a number of fields.

---

[10] Note that an idle node could make an unsolicited proposal for a new contract. At the present time, however, we feel that the potentially excessive inter-node communication that might result from such a mechanism is not justified by the benefits to be accrued from its inclusion in the formalism.

In this section, we focus on the fields that contain task and priority descriptions.

5.1 Task Descriptions

A task description may contain two types of information: the local context in which the task is to be executed, and the applications software required to execute it. Depending on the task, the required global context may be passed with the contract, or further contracts may be let to obtain it as required. As part of the announcement-bidding-award sequence, it will be determined whether the contractor that is awarded the contract actually has the software required to execute it. If this is the case, then the task description need not include it. If the contractor does not have the software, however, then it must be obtained either from the contract manager, or from some other processor node in the net. In the former case, it will be included as part of the task description. In the latter, further contracts will have to be let.

From the above discussion, it is apparent that we do not foresee pre-loading of all processor nodes in the distributed processor with the software necessary to execute all possible subtasks that might arise during the course of processing a top-level task. Instead, we advocate distributing software "on the fly", as it is required. Software distribution could, of course, be carried out by contract in parallel with top-level task processing.

5.2 Priority Descriptions

Focusing mechanisms are crucial to effective problem solving in any environment. However, the problem of focus in a distributed processing environment is compounded over that found in the uniprocessor environment due to the larger number of choices available.

We advocate the use of priority descriptions of arbitrary complexity as a solution to the dual priority assignment problem (i.e., assignment of priorities to contracts so that contractors are in a

position to make relevant bids, and assignment of priorities to contractors so that contract managers can choose among them). It is well known in AI that integer estimates of task importance result in too much data compression, and do not adequately summarize its salient features [27] [3]. We want to include more information. The priority description of a contract in the distributed processor might include specifications of the code required to execute it, the special resources or information required, and the projected "size" of the contract (in memory space and time). In the distributed sensor net example, we include the location of the processor node that generated the contract, as that may affect bids by contractors, and the selection of a contractor. The net may be "keying" on contracts with specific features, and this type of information would be useful.

The priority description of a contractor might specify its processor type (e.g., 8080 or KL-10), special resources (hardware and/or software), and location. The key point is that priority should be formulated as a description (of potentially arbitrary complexity) rather than be restricted to an integer.

Each node is able to assign a partial order to such descriptions when placing subcontracts on its wait list, selecting a contract announcement on which to bid, or selecting a contractor to which to award a contract. We feel that the "no preference" response to an ordering attempt is a valid and useful one. It may mean that the two contracts (or contractors) in question are not comparable (e.g., this might happen for contracts if a number of different top-level tasks are being simultaneously processed by the distributed processor). It is entirely possible that individual subgroups may effect dynamic changes in the method used to generate or order priority descriptions. As a result, two descriptions may not be comparable, or may have orderings which vary according to processor node.

In order to maintain reasonable efficiency, it should not be necessary to compare complete priority descriptions each time that an

ordering is required. The descriptions should be constructed so that gross mismatches are detectable with simple tests.

Priority descriptions also offer a mechanism for imposing some structure on the rather amorphous distributed processor architecture whose attributes were discussed in Section 2. For example, a central master node is easily defined by allowing its priority descriptions to carry more influence.

6 An Example - Heuristic Search

In this section we present a simple example of the operation of the CONTRACT NET. The top-level task is the exhaustive search of the simple binary tree shown in Figure 2, and the distributed processor is comprised of three processor nodes. The name that will be used for each node in the tree in the following discussion is shown beside the node.

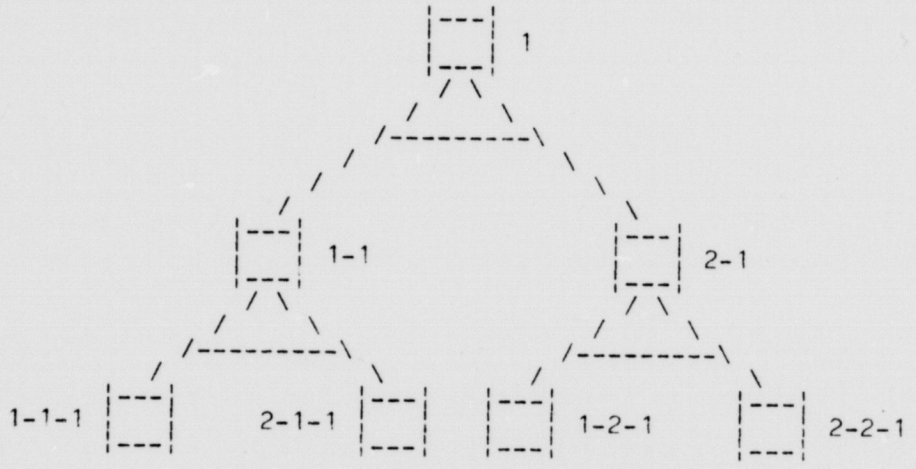

Figure 2: Binary Tree.

For this example we define a contract to be expansion of a node for interior nodes, or evaluation of a node for tip nodes. We assume that contracts, once commenced, carry on to completion without interruption by newly generated contracts. Furthermore, we assume that both generation of one successor and evaluation of one tip node require one time unit. We will not be concerned with the details of

announcements and bidding, but only with contract flow. We will further assume that these functions require much less than one time unit. A net with three processor nodes has been chosen because it can be shown that this number of nodes results in the minimum attainable search time for this tree under the above assumptions.

The flow of contracts for this example is shown schematically in Figure 3, under the assumption that the top-level task is given to Processor 1 to commence the search. Parentheses surround the names of contracts (nodes in the tree) that are commenced. When a new contract (a successor to a node in the tree) is generated, its name is shown in the flow of the processor node that generated it (shown vertically with time increasing down the page). The completion of the evaluation of a tip node is shown by "T". Contract awards are shown by "==>". The manipulation of management and wait lists, and reporting of results are not shown.

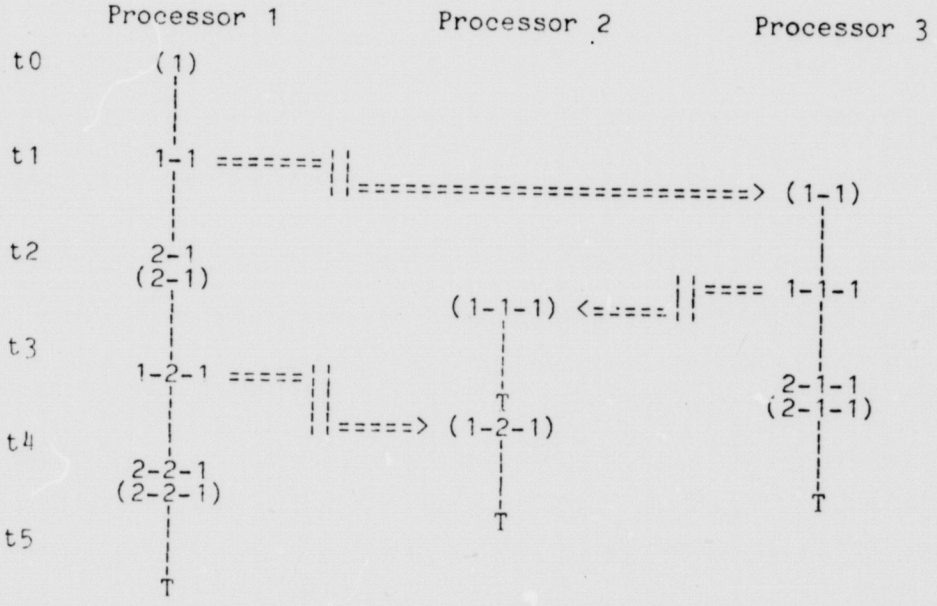

Figure 3. Distributed Tree Search - Contract Flow.

A SAIL simulation of the CONTRACT NET runs at the SUMEX-AIM dual

KI-10 facility at Stanford. It accepts simple applications programs, and maps them onto a simulated distributed processor with a variable number of nodes. The simulation has been tested with some simple heuristic search tasks, such as the n queens problem [11].

The applications programmer must supply procedures which actually perform the computation required to execute a contract (e.g., the generation of a new board in the n queens problem), generate and order priority descriptions, generate results descriptions, and so on. Priority descriptions for the n queens problem are trivial. If the number of queens placed on the board is used as its priority description, then boards are searched in a local depth-first manner. If the negative of this measure is used, then boards are searched in a local breadth-first manner.

At present, the simulation only handles the contract level (as of January, 1977), but will be extended down to the processor interconnection level during the next few months. The simulation will be our primary mechanism for debugging the CONTRACT NET formalism.

7 Summary

We have presented a formalism for the control of distributed problem solving, together with the attributes of an underlying distributed processor architecture suitable for AI applications. The formalism stresses loosely-coupled processors and distributed control for problem solving in a distributed processing environment.

The basic operations required to manage contracts have been described, and a simulation has been constructed to test the ideas.

We have advocated the use of priority descriptions, as opposed to simple integer ratings, to provide an adequate focusing mechanism.

In the future, we intend to evaluate the merits of the CONTRACT NET formalism, explore the variations in control that can be achieved in a distributed processor through the use of priority descriptions,

explore the advantages of various processor interconnection methods, and apply the ideas to a number of specific tasks with AI interest.

References

1. G. A. Anderson and E. D. Jensen, "Computer Interconnection Structures: Taxonomy, Characteristics, and Examples", Computing Surveys, Vol. 7, No. 4, December 1975, pp. 197-213.

2. B. W. Arden and A. D. Berenbaum, "A Multi-Microprocessor Computer System Architecture", Proceedings of the Fifth Symposium on Operating Systems Principles, University of Texas at Austin, November 1975, pp. 114-121.

3. H. J. Berliner, "Some Necessary Conditions For A Master Chess Program", IJCAI3 Advance Papers, Stanford, Calif., August 1973, pp. 77-85.

4. D. G. Bobrow and T. Winograd, "An Overview of KRL, A Knowledge Representation Language", STAN-CS-76-581, Stanford University, November 1976.

5. B. G. Buchanan and J. Lederberg, "The Heuristic DENDRALProgram for Explaining Empirical Data", Information Processing 71, North-Holland, Amsterdam, 1972.

6. B. G. Buchanan, "Scientific Theory Formation by Computer", Proceedings of NATO Advanced Study Institute on Computer Oriented Learning Processes, Bonas, France, 1974.

7. R. Davis and J. King, "An Overview of Production Systems", STAN-CS-75-524, Stanford University, October 1975.

8. R. O. Duda and P. E. Hart, Pattern Classification and Scene Analysis, Wiley, N. Y., 1973.

9. L. D. Erman, R. D. Fennell, V. R. Lesser, and D. R. Reddy, "System Organizations for Speech Understanding", IJCAI3 Advance Papers, Stanford, Calif., August 1973, pp. 194-199.

10. D. J. Farber, J. Feldman, F. R. Heinrich, M. D. Hopwood, K. C. Larson, D. C. Loomis, and L. A. Rowe, "The Distributed Computer System", IEEE COMPCON 73, February 1973, San Francisco, Calif., pp. 31-34.

11. R. W. Floyd, "Nondeterministic Algorithms", JACM, Vol. 14, No. 4, October 1967, pp. 636-644.

12. M. J. Flynn, "Some Computer Organizations and Their Effectiveness", IEEE Trans. on Computers, Vol. C-21, No. 9, September 1972, pp. 948-960.

13. J. D. Foster, "The Development Of A Concept For Distributive Processing", IEEE COMPCON 76, February 1976, San Francisco, Calif., February 1976, pp. 28-30.

14. W. T. Harwood and F. K. Hanna, "A Distributed Activity Processing System For A.I.", Proceedings of the AISB Summer Conference, Edinburgh, July 1976, pp. 130-136.

15. F. E. Heart, R. E. Kahn, S. M. Ornstein, W. R. Crowther and D. C. Walden, "The interface message processor for the ARPA computer network", 1970 SJCC Proceedings, pp. 551-566.

16. F. E. Heart, S. M. Ornstein, W. R. Crowther, and W. B. Barker, "A new minicomputer/multiprocessor for the ARPA network", 1973 NCC Proceedings, pp. 529-537.

17. C. Hewitt, P. Bishop, and R. Steiger, "A Universal Modular ACTOR Formalism for Artificial Intelligence", IJCAI3 Advance Papers, Stanford, Calif., August, 1973, pp. 235-245.

18. C. Hewitt, "Protection and Synchronization in ACTOR Systems", MIT AI Working Paper 83, November 1974.

19. C. Hewitt and B. Smith, "Towards a Programming Apprentice", IEEE Trans. on Software Engineering, Vol. SE-1, No. 1, March 1975, pp. 26-45.

20. C. Hewitt, "Viewing Control Structures as Patterns of Passing Messages", MIT AI Working Paper 92 (revised), August 1976.

21. R. E. Kahn, "Resource-Sharing Computer Communications Networks", Proc. IEEE, Vol. 60, No. 11, November 1972, pp. 1397-1407.

22. R. E. Kahn, "The Organization of Computer Resources into a Packet Radio Network", 1975 NCC, pp. 177-186.

23. S. R. Kimbleton and G. M. Schneider, "Computer Communications Networks: Approaches, Objectives, and Performance Considerations", Computing Surveys, Vol. 7, No. 3, September 1975, pp. 129-173.

24. D. B. Lenat, "Beings: Knowledge As Interacting Experts", IJCAI4 Proceedings, Tbilisi, USSR, September 1975, pp. 126-133.

25. D. B. Lenat, "AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search", STAN-CS-76-570, Stanford University, July 1976.

26. V. R. Lesser, R. D. Fennell, L. D. Erman, and D. R. Reddy, "Organization of the HEARSAY II Speech Understanding System", IEEE Trans. on Acoustics, Speech, and Signal Processing, Vol. ASSP-23, No. 1, February 1975, pp. 11-24.

27. M. Minsky and S. Papert, "Artificial Intelligence - Progress Report", MIT AI Memo 252, January 1972.

28. A. Newell and G. Robertson, "Some Issues in Programming Multi-Mini-Processors", AFOSR-TR-75-0544, Carnegie-Mellon University, January 1975.

29. N. J. Nilsson, Problem Solving Methods in Artificial Intelligence, McGraw-Hill, N. Y., 1971.

30. R. N. Noyce, "From Relays to MPU's", Computer, Vol. 9, No. 12 December 1976, pp. 26-29.

31. O. G. Selfridge, "Pandemonium: a paradigm for learning", in D. V. Blake and A. M. Uttley, eds., Proceedings of the Symposium on Mechanisation of Thought Processes", National Physical Laboratory, Teddington, England, London: H. M. Stationary Office, pp. 511-529.

32. E. H. Shortliffe, MYCIN: Computer-Based Medical Consultations American-Elsevier, N. Y., 1976.

33. D. L. Waltz, "Generating Semantic Descriptions From Drawings of Scenes with Shadows", MIT AI-TR-271, November 1972.

34. P. H. Winston, The Psychology of Computer Vision, McGraw-Hill, N. Y., 1975.

35. W. A. Wulf and C. G. Bell, "C.mmp - A multi-mini-processor", 1972 FJCC Proceedings, pp. 765-777.