NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, tests publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

ISBN 0-315-46700-2

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

TABLE OF CONTENTS

ABSTRACT

This thesis describes the architecture and implementation of a Distributed Problem Solving system, focusing on the control and communication aspects. The objective of this system, called Expertnet, is to allow for sharing of resources on a network of personal computers. A negotiation mechanism is used in the Expertnet for the purpose of effective resource sharing. This mechanism enables an Host Machine to request a solution of a problem on a Remote Machine which has resident expertise to solve the requested problem.

Some of the major components in the node arhcitecture are a database of Meta-Knowledge about expertise of a node's own problem solving capabilities and those of other processing nodes. This meta-knowledge is gradually accumulated during problem solving. Each node has a Planner which guides in the problem solving process. The Internode communication is carried out with the use of a question-and-answer mechanism.

CHAPTER 1

INTRODUCTION

Issues about distributed computing, as about any other aspect of computing, can be formulated at various levels of abstraction. Each level has a different conceptual content, and raises a correspondingly different set of issues. In distributed computing most of the emphasis has been at a level that is closely related to physical connection of different process, secure transmission of data among them and the corresponding operating system problems of scheduling different processors. These issues have dominated discussion so much that the term distributed processing has come to mean almost exclusively that set of issues. In this thesis, the discussion about distributed processing is at a different, "higher" level of abstraction. The questions of interest at this level concern the strategies by which the decomposition and coordination of computation in a distributed system are matched to the structural demands of the task domain. Distributed Problem Solving (DPS) is an appropriate term for the phenomenon at this level of abstraction.

1.1 NATURAL WORLD METAPHORS

The information processing phenomena that occur in the natural world are a source of number of useful metaphors (CHANDS1) for distributed processing in general and distributed problem solving in particular. It is clear that distribution of processing or computation is an intrinsic characteristic of most natural phenomenon. Social organizations from honeybee colonies to a modern corporation, from bureaucracies to medical communities, from committees to representative democracies are living examples of distributed information processing embodying a variety of strategies of decomposition and co-ordination.

Computation in biological brains (GEVINS83,GESCH80), especially in their sensory processors such as vision systems, displays a high degree of distribution. Control of movement in biological systems is also accomplished by distributed computation. Volitional movement can be viewed as being generated by low-level programs coordinated and regulated by high level controllers. The task of generating all the impulses for all muscle fibres for each movement is surely beyond the resources of any centralized biological movement processor.

In all these examples from social organization to brains and motor systems the overall computation task is distributed among a collection of separate processors. These separate processors coordinate their computations by means of exchanging appropriate

symbolic information. This leads one to believe that the co-operative problem solving techniques that have been so effectively and efficiently used in naturally occuring complex system, can be extended to the domain of complex computer systems.

1.2 DISTRIBUTED PROBLEM SOLVING MODELS

Distributed Problem Solving or Distributed Expert System can be categorized as consisting of multiple physically separated processing nodes. Each processing node has at least one expert system. Each processing node sees only a small part of the problem environment. In other words each node has a partial view of the problem environment. Due to the incompleteness of problem domain knowledge or partial view each node has of the problem environment, sharing task loads and exchanging tentative results among processing nodes are necessary during the problem solving process.

Definition of a distributed problem solving system is examined in more detail in Chapter 2.In addition this chapter provides the major differences between distributed processing and distributed problem solving. Also, the motivation for using distributed problem solving is examined in this chapter.

HEARSAY-II System (HAYES-ROTH, ERMAN80) and CONTRACT NET (SMITH80, DAVIS83) are two major models in the area of distributed problem solving. These two models are described

in detail in Chapters 3 and 4, respectively.

1.2.1 HEARSAY-II MODEL

In the Hearsay-II system the distributed system is structured so that each node can perform useful processing using incomplete data while simultaneously exchanging the intermediate results of its processing with other nodes to construct cooperatively a complete solution. A node may not only have to perform useful processing with incomplete input data, but also with the possibly incomplete, incorrect, and inconsistent tentative results received from other nodes. A node resolves these data uncertainties by:

(1) detecting inconsistencies between its tentative partial results and those received from other nodes;

(2) integrating into its local database those portions of other nodes' results which are consistent with its results;

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

1.2.2 CONTRACT NET MODEL

In Contract Net model, distributed problem solving consists of four phases; problem decomposition, sub-problem distribution, solution of sub-problem, and synthesis (integrating the results of subproblems to generate a solution to the overall problem) of

the overall solution. Decomposition involves breaking a large problem into smaller, more manageable pieces; distribution involves the matching of sub-problems with problem solvers capable of handling them; the sub-problems are then solved; and finally those individual solutions are synthesized into a single, overall solution.

Each processing node may take on the role of a manager or a 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.

1.3 OFFICE MODEL

In this thesis a technique, called Expertnet, for resource sharing on a network of workstations in an office environment is proposed. In order to develop this technique, the behavior and procedures of an office were considered.

An office can be defined as information centre of an organization (HO86). Through processing, storage and dissemination of relevant information, an office can facilitate effective execution of functions of many other components of an organization, and serve as a "glue" to make the whole

organization work as a coherent entity to achieve the organizational objectives. Whether an office is geographically centralised or dispersed, it consists of people and associated facilities that work together under the enforcement of regulations. People are assigned specific roles in the organization structure to perform functions and services generated by requests from both inside and outside of the organization.

Office communication systems provide the mechanism through which agents can co-operatively carry out office tasks. The word 'agent' here can mean an intelligent office worker capable of using reason based upon knowledge to solve a problem, or an intelligent office system. The main purpose of communication is knowledge exchange. For example, an office agent may try to gain more knowledge from other agents to help that agent solve a task. Based on how the office agent is working on the task, three modes of knowledge exchange can be identified:

(1) Self Mode: In the simplest case when an agent has sufficient relevant knowledge to solve a task, he works alone on the task.

(2) Co-operative Mode: If the agent does not have sufficient knowledge to solve the task, then he will normaly ask another agent to either explain how to continue the task or to complete the task. In this mode the two agents co-operatively solve the task.

(3) Conference Mode: If the second agent fails to complete the task or does not have the knowledge to solve the task, then the first agent will try to find help from some other agents. Hence, in this mode more than two agents can be involved in solving the the task.

Depending on how hard a task is, an agent may attempt to solve the task through Self Mode, Co-operative Mode or Conference Mode. Hence an office communication system must provide mechanisms, or knowledge exchange protocols, that allow agents to perform the three modes in order to exchange knowledge co-operatively for the achievement of office goals. The above discussion reveals certain important characteristics of office behavior - that it is stimulus driven, knowledge based and goal directed.

1.4 PROPOSED TECHNIQUE

In developing the technique proposed in this thesis, we considered the characteristics of the office behavior discussed above, as well as the Contract Net Model. The Chapter 5.0 describes the architecture and operation of each of the processing nodes.

1.4.1 ARCHITECTURE

A processing node can represent an intelligent office agent capable of using reason based upon knowledge to solve a problem.

The major components of a Processing Node are :

(i) Meta Level Knowledge Database :' which provides the knowledge about the problem solving capabilities of the host node and those of other nodes on the network. The motivation for using the meta level knowledge is to allow a high level reasoning process to be implemented in the internode communication mechanism.

(ii) Planner : to plan and organise (just as an intelligent office worker would normally do) the manner in which a given problem is to be solved. The problem can be given to the host node by the user of the host node or it can be a request received from other nodes. Depending on the source of the problem and the the level of local resident expertise, there are three modes of problem solving similar to the ones described in the Office Model.

If the host node has the expertise to solve the problem, then it solves it using the functions/procedures resident locally. This is similar to the self mode of the office model. On the other hand if the host node does not have the expertise to solve the problem then it can request for service from other nodes in either of the two ways:

(a)directly (from a node) if it knows that a particular node can solve the problem. This corresponds to the co-operative mode of the office model.

(b) or by broadcasting to other nodes (belonging to the same group as the host node) if it does not know which node can solve the problem. This corresponds to the conference mode of the office model.

(ii) Message Server provides appropriate services to messages received from other nodes. For example, if a message is a response (from a servicing node) to a request from the host node, then it is given to Response Handler for appropriate actions to be taken. On the other hand if the message is a request for service from other nodes then it is given to the Request Handler for the necessary actions.

(iii) Receive, Transmit, Broadcast are primitives used for receiving messages from other nodes and for sending messages.

(iv) Knowledge Source consisting of inference engine and knowledge base. The knowledge base contains some domain specific knowledge. The inference engine is related to the knowledge representation method used in the knowledge base. It is responsible for processing something based on the knowledge drawn from the knowledge base.

KS-ITI and KS-X25 are examples of two knowledge sources used in an application where Expertnet was used. These knowledge sources contain domain specific knowledge. They contain the knowledge required to configure ITI lines and X.25 lines on a

packet switched network.

(v) Process Server, whose function is to invoke the proper knowledge sources required for solution of a problem.

1.4.2 RESOLUTION PROOF

Planner described in the previous section, performs the planning process for the solution of a given problem. In order to do so it uses the information contained in meta knowledge base and certain rules. In Chapter 6, resolution procedure was used to prove the rules used in this proposed technique.

The various rules used were first represented in Predicate Logic form. These were then converted into Clause Form. Resolution procedure was then applied to these clauses. Two examples were chosen to carry out this resolution procedure.

Example 1: Node 'A' is given a problem 'p4' to solve. This node does not have the expertise to solve the problem. But it knows from the information contained in the meta knowledge base, that node 'C' can solve the problem. Node 'A' transmits a request to node 'C', and receives service from it.

Example 2: Node 'A' is given a problem 'p3' to be solved. This node does not have the expertise to solve the problem p3. Furthermore, it does not know if any other node can solve the problem. This results in a request being transmitted to all the

nodes which belong to the same group as node 'A'. One of the receiving nodes, node 'B', has the expertise to solve 'p3'. It also ascertains from its meta knowledge base that 'A' is not restricted from receiving services for problem 'p3'. It now solves the problem and sends the response back to node 'A'. On receiving the response, node 'A' updates its meta knowledge base, so that next time it is given problem 'p3' it knows it can get services for 'p3' from node 'B'.

1.4.3 AN APPLICATION

A function performed by network engineering department of an organization is chosen as an application for Expertnet. Let us say a network engineer who uses Node 'A' as his workstation has a task to configure a packet switch based on user provided data, and he is also required to cost the switches configured.

Let's assume that Node 'A' has the expertise to solve Switch Configuration, but does not have the expertise to provide cost information. Let us also assume that node 'B' has the expertise to provide cost information. A node is said to have expertise to solve a problem if the functions or knowledge sources required to solve the problem are resident in the processing node. If they (knowledge sources) are resident in the node, then the meta knowledge base of the node contains this information in a meta frame associated to a particular problem.

Following steps occur when Expertnet is used for performing the task identified for the network engineer.

(1) Problem SWITCH-CONFIG is given to node 'A' by the network engineer

(2) Planner in node 'A' checks in its meta knowledge base to see if the given problem exists.

(3) Since problem SWITCH-CONFIG exists in the Meta KB (knowledge base) the next slot is checked for process/function name required to solve this problem. Since node 'A' has the expertise to solve the problem, this slot will contain the process name KS-CONFIG. This implies that node 'A' can solve the problem locally. This corresponds to the Self Mode of the Office Model. The process name is given to the PROCESS SERVER which now invokes KS-CONFIG and passes to it the input data as the argument.

(4) The problem is decomposed by KS-CONFIG into sub-problems to configure X.25 lines and ITI lines. The knowledge sources KS-X25 and KS-ITI are used for this purpose. These knowledge sources have domain specific knowledge and inference mechanism. This step corresponds to the problem decomposition and sub problem distribution phases of the Contract Net Model.

(5) The result produced by the knowledge sources is an equipment list . The KS-CONFIG which acts as the task manager prepares

a combined equipment list and gives it to the user of the node 'A'. This corresponds to the synthesis of overall solution phase of the Contract Net Model.

(6) Next the network engineer wishes to obtain the cost information. So he submits the problem name COST-INFO to his machine.

(7) As before the Planner checks for problem COST-INFO in the meta KB.

(8) If it exists then the next slot for process name (name of the function which can solve the problem) is checked in the Meta Frame. In our example, this slot has a nil value which means that node 'A' does not have the expertise to solve the problem. This implies that 'A' has to request service from some other node.

(9) The next slot of Meta Frame is checked to see if a NODE ID is specified. This slot has a value 'NODE B' which means node 'B'can solve the COST-INFO problem.

(10) A Request is sent out with the help of MESSAGE SERVER to node 'B' to provide cost information. The data consisting of equipment list is also sent in the request message. This corresponds to the Co-operative Mode of the Office Model.

(11) On receiving the request, Node 'B' checks in its meta knowledge base to see if it has the expertise to solve the problem

specified in the request. Since in this example it has the expertise, the node 'B' invokes the appropriate knowledge source to solve the problem. The result is sent back to node 'A' in a Response message.

This basically provides an overview of the problem solving environment in Expertnet.

Finally Chapter 8 provides concluding remarks , shortcomings of Expertnet and some future research directions.

CHAPTER 2

DISTRIBUTED EXPERT SYSTEM OR DISTRIBUTED PROBLEM SOLVING

Distributed problem solving is the cooperative solution of problems by a decentralized and loosely coupled collection of knowledge sources (which consist of procedures, sets of rules, etc.), located in a number of distinct processor nodes. Since none of the KS's have sufficient information to solve the entire problem, they cooperate by mutual sharing of information so that the group as a whole can produce an answer. It is essential to identify clearly the meanings of decentralized and loosely coupled. By decentralized we mean that both control and data are logically and often geographically distributed; there is neither global control nor global data storage. When control is decentralised, no one node has a global view of all activities in the system; each node has local view that includes information about only a subset of the tasks. In a system with completely centralised control, one processor is responsible for directing the activities of all the others. It knows what all the other processors are doing at any given time. Loosely coupled means that individual KSs spend most of their time in computation rather than communication.

Such problem solvers offer the benefits of speed, reliability, extensibility and the potential for increased tolerance to handle applications with a natural spatial distribution. Recently there has been much interest in this type of problem solving in the Artificial Intelligence (AI) community. It has been put to use in various applications such as speech interpretation (HAYES-ROTH80, ERMAN80), image understanding (HANSON78), distributed sensing (FEIGENBAUM78, SMITHA80), Robotics (TALUKDAR83), vehicle fleet control (NARAIN84), air traffic control (McArthur82) Fault Detection and Diagnosis (LESSER84), Office Information Systems (MALONE87, HO86, HIBBARD85, LOCHOVSKY86).

2.1 DIFFERENCE BETWEEN DISTRIBUTED PROCESSING AND DISTRIBUTED PROBLEM SOLVING.

Distributed problem solving differs in several fundamental respects from distributed processing. Perhaps the most important distinction arises from examining the origin of the system and the motivation for inter connecting machine.

The Distributed Processing systems came into existence with the intention to synthesize a network of machines capable of carrying out a number of widely disparate tasks. Typically, several distinct applications are envisioned, with each application concentrated at a single node of the network, as for example in a three-node system intended to do payroll, order entry and process control. The aim is to find a way to reconcile any conflicts and

disadvantages arising from the desire to carry out disparate tasks, in order to gain the benefits of using multiple machines.

In distributed problem solving, on the other hand, there is a single task envisioned for the system and the resources to be applied have no other predefined roles to carry out.

In distributed processing it is assumed that a well defined and a priori partitioned problem exists. The major concerns lie in an optimal static distribution of tasks, methods for interconnecting processor nodes, resource allocation and prevention of deadlock. Complete knowledge of the problem has been assumed (i.e. explicit knowledge of timing and precedence relations between tasks) and the major reason for distribution has been assumed to be load balancing (BAER73). However, in distributed problem solving no such assumptions are made and there is no preplanning of resources. The system resources have no predefined role to carry out. Each processing node has either a partial view of the problem environment, or incomplete domain knowledge , or both. Moreover, both data and domain knowledge could be incomplete and inexact, respectively. Usually a number of plausible hypotheses may be generated in this kind of situation. Some of the hypotheses could lead the problem solving in a wrong direction and waste system resources. Thus, how to maintain the global coherence to integrate the efforts of all individual processing nodes toward a consistent direction becomes

an important issue in distributed problem solving.

A final distinction results from the lack of substantial co-operation in most distributed processing system. Typically, for instance, most of the processing is done at a central site and remote processor are limited to basic data collection (for example credit card verification).The word distributed is usually taken to mean spatial distribution of data, but distribution of function or control is not generally considered. As mentioned earlier , in distributed problem solving each of the processing nodes has limited view of the problem environment. As a result, there is a need for these nodes to co-operate with each other to solve the entire problem. Mutual sharing of information, including task loads and partial results, is neccessary to allow the system as a whole to produce a consistent answer.

2.2 MOTIVATION FOR DISTRIBUTED PROBLEM SOLVING

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

A distributed approach may also be well suited to problems that have either a spatial distribution or a large degree of

functional specialization. Spatial distribution often occurs in problems involving interpretation of signal data from multiple sensors (NARAIN84) or problem solving using network of workstations (BONNELL83). Functional specialization may occur in problems like understanding continuous speech (REDDY80). Information from different knowledge-sources (signal processor parsers, workstation etc.) must be combined to solve the problem. Distributed problem solving also offers a way to apply to problem solving the recent advances in both processor fabrication and communication technology. Low cost, small VLSI processors are now common place and large scale processors are also readily available. The synthesis of advanced computer and communication technology that has resulted in network of resource sharing computers offers a foundation for work on distributed architectures. With these two developments as a foundation one can focus on techniques for the effective use of network of machines. One reason for interest in distributed architecture in general is its capacity for reliable computation and graceful degradation. By placing problem solving in this environment, we have the chance to make it similarly reliable. The distribution of data and control in distributed problem solving systems, also makes possible additional responses to components to function as before (albeit more slowly), the option may exist of having the system reconfigure itself to take into account the hardware available.

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

2.3 FUNDAMENTAL ISSUES TO BE CONSIDERED IN DISTRIBUTED PROBLEM SYSTEMS

Co-operative distributed problem system as mentioned earlier, are distributed networks of semi autonomous processing nodes that work together to solve a single problem. Each of these nodes has insufficient local information to make completely accurate processing and control decisions. As a result of this there is a need for the nodes to cooperate, in an iterative manner, by exchanging inconsistent and incomplete view of the information used in their computations. In order for this cooperation to be effective among the nodes (SMITH81) there is a need for

sufficient global coherence in cooperatative distributed problem solving networks. If this coherence is not achieved, then the performance (speed and accuracy) of the node can be significantly diminished as a result of

(i) lost processing as nodes wait for something to do.

(ii) wasted processing as nodes work at cross-purposes with one another.

(iii) redundantly applied processing as nodes duplicate efforts.

(iv) misallocation of activities so that important portions of the problems are either inaccurately solved or not solved in timely fashion.

The above problems have been observed (ERMAN80, LESSER81) in a network of few nodes . One would expect that these problems will become even more significant for networks containing larger number of nodes.

CHAPTER 3

HEARSAY II SPEECH UNDERSTANDING SYSTEM

3.1 HEARSAY II PROBLEM MODEL

Hearsay II Speech Understanding System was developed at Carnegie Mellon University (ERMAN80, HAYES-ROTH80). Its basic methodology involves the application of symbolic reasoning as an aid to signal processing. It uses the problem solving paradigm of searching for an overall solution by the incremental aggregation of partial solutions. In this paradigm, errors amd uncertainity from input data and incomplete or incorrect knowledge are handled as an integral part of the interpretation process. This is in contrast to more conventional problem solving techniques, in which errors are fatal or are handled as exceptional conditions, requiring additional processing outside the normal problem solving strategy.

To solve a problem as difficult as speech understanding, the intrepretation is constructed by combining partial interpretations derived from diverse knowledge. Each area of knowledge is represented by an independent module called a "knowledge source" (KS). Each KS is schematized as a condition action pair. The condition component prescribes the situations in which the KS may contribute to the problem solving activity, and the action component specifies what that contribution is and how to

integrate into the current situation.

These KSs cover such knowledge areas as acoustics, phonetics, syntax and semantics. The KSs interact with each other in co-operative manner to resolve the uncertainty caused by noise and incompleteness in the input data and inaccurate processing by KSs.

The interaction of KSs is based on an iterative data directed form of the hypothesize and test paradigm. As iteration involves the creation of an hypothesis (which is one possible interpretation of some part of the solution), followed by test(s) of its plausibility. When performing these actions, KSs use apriori knowledge about the problem, as well as previously generated hypothese. When a KS creates an hypothesis from previously created hypotheses, the KS strengthens the existing (partial) hypotheses with more information, as a result reducing the uncertainty of the interpretation. The processing is terminated when a consistent hypothesis is generated that satisfies the requirements of a complete solution.

A KS often generates incorrect hypothesis because its knowledge or its input data including previously generated hypotheses, contains errors or is incomplete. If KS generates only a single hypothesis for each specific part of the problem, the problem-solving process would often terminate with an inaccurate interpretation or with a partial interpretation that could not

be extended because of its inconsistency. In order to avoid this problem, KSs in general, create several alternate hypotheses for each part of the problem. The KS associates with each hypothesis a credibility rating, which is its estimate of the likelihood that the hypothesis is correct.The set of all possible partial interpretations defines the problem solving search space.

The more alternative hypotheses generated, the larger the fraction of the space actually searched. This raises the possibility of combinatoric explosion since each partial interpretation can give rise to multiple extensions.In order to overcome this problem,the Hearsay II problem solving system uses a method called opportunistic and asynchronous style of problem solving. According to this method, at each step in the search a subset of the existing partial interpretation is selected for extension; the resulting extended partial interpretations then compete for selection with those previously generated. The selection of the subset of hypotheses to extend is called the focus of control or focus of attention problem (HAYES-ROTH77).

An integral part of effective focus of control is the problem solving systems ability to focus quickly on information that constrains the search, in order to contain combinatoric explosions.

3.2 HEARSAY II ARCHITECTURE

Schematic representation of a centralized HearsayII architecture is shown in FIGURE 3-1 . The major data structures are the shared global database called the blackboard, focus of control database, and scheduling queues.

BLACKBOARD

The major units on the blackboard are the hypotheses. Relationship among hypotheses at different levels are represented by the graph structure. The sequences of levels on the blackboard forms a loose hierachical structure in which the elements at each level can be described approximately as abstraction of elements at the next lower level. For example, in speech understanding an utterance is represented as a signal or as sequences of phonemes, syllables, words, phrases or concepts.

The set of possible hypotheses at a level forms a problem space for KSs operating at that level. A partial interpretation (i.e. a group of hypotheses) at one level is used to constrain the search at another level. For example, a KS can create a phrase hypothesis as an abstraction of a sequence of word hypothesis. Similarly, another KS can use the phrase hypothesis to predict (i.e. constrain) the set of possible word hypotheses that might follow the phrase. In order to implement the data directed activation of KSs each KS has two components: a pattern and an

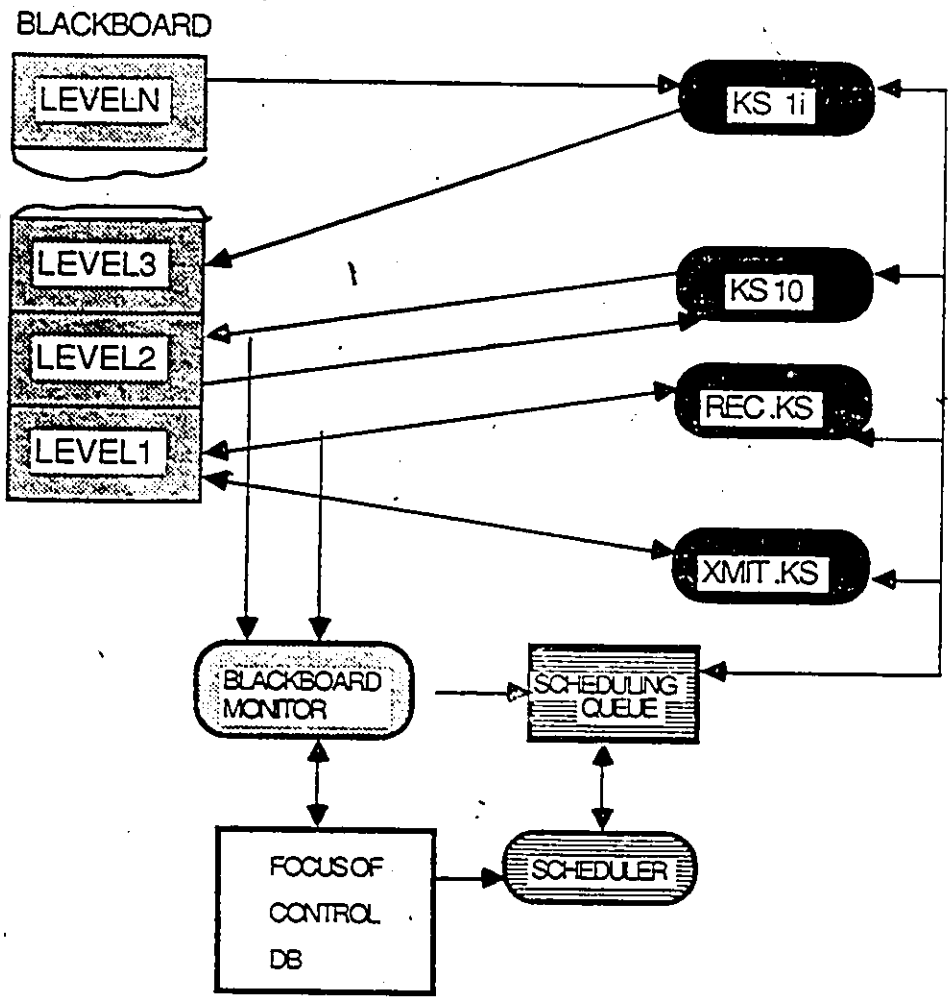


FIGURE 3-1

CENTRALISED HEARSAY II SYSTEM

action. Whenever the pattern is matched by some hypothesis structure on the blackboard, an activation of the KS is created. If the KS activation is selected eventually by the scheduler, its action is executed in the context of the matched structure. For example, the pattern of a KS might be the creation of a new syllable hypothesis and its action might be to use that syllable hypothesis and, possibly, other adjacent syllable hypotheses to create new word hypotheses.

KS activity and the search process is managed by the scheduler using the focus of control database and the scheduling queues. At any point, the scheduling queues contain the pending KS activations. The scheduler calculates a priority for each waiting activation and selects for execution the one with the highest priority. The priority calculation attempts to estimate the impact of the information to be generated by an activation on the current state of the problem solving. From the problem solving viewpoint, the impact of some information is a measure of the degree to which it reduces the number of competing interpretations (for more information refer to LESSER80). This measure changes as the problem solving progresses; thus the timelines of the creation of the information affects its impact. For example, if two pieces of information can lead to the same hypothesis, the creation of the first of them may have high impact, but the creation of second will have little other than adding confirmation to the hypothesis.

Some of the dimensions that are used to estimate the impact of information are identified below:

(i) The credibility of some information is a measure of the system's confidence in the information; the more credible the information, the higher its expected impact.

(ii) The scope of some information is a measure of the amount of total problem solution that it describes. Scope is related to level of abstraction (for example, a word has larger scope than syllable) and to the size (for example, a two-second phrase has larger scope than a one second phrase). The larger the scope, the greater the impact because a larger portion of the complete interpretation, and hence, more constraint is specified.

(iii) The diagnosticity of some information is a measure of how much competing information can be solved by the information. For example, if one part of the current partial solution has high credibility while another has only low credibility , a moderately credible piece of information in the former area will have low diagnosticity , but a moderately credible piece in the latter area will have high diagnosticity and, hence, greater impact.

FOCUS-OF-CONTROL DATABASE

The focus-of-control database contains meta information about the state of system's problem-solving activity. The meta information is used to estimate the impact of information, based on its credibility, scope, and diagnosticity. Meta information includes such things as the current best hypotheses on the blackboard and how much time has elapsed since these hypotheses were generated or combined with others. This latter kind of information allows the system to recognize a state of stagnation in part of the problem solving, and then to cause the reappraisal of the impact of the current best hypotheses. The focus-of-control database is updated by the blackboard monitor based on the generation and modification of hypotheses on the blackboard by KSs.

SCHEDULING QUEUES AND BLACKBOARD MONITOR

The Blackboard Monitor keeps track of changes to the blackboard (called Blackboard Event) and knows which events at which level of (of the blackboard) interest each KS. The occurence of a blackboard event does not guarantee that there is, in fact, sufficient information on the blackboard for a KS to be executed. The blackboard monitor executes a precondition procedure for each interested KS to make more detailed examination and, if sufficient information is found , a KS instantiation (KSI) is created and placed onto the Scheduling Queue , selecting for

execution the one with the highest rating. Execution of the KSI causes changes to the blackboard which trigger additional blackboard events and the process continues.

3.3 NETWORK OF HEARSAY-II SYSTEMS

In a Network of Hearsay II System, each node in network contains KSs, a scheduler and focus of control database for selecting the next KS activation to execute at each step, a blackboard for KS communication, and a blackboard monitor for KS activation. Therefore, each node is an architecturally complete Hearsay II system. The distribution of the dynamic information (i.e. partial interpretations and meta-information) in the network can be viewed as the distributed system synthesized from systems operating at each node. The major considerations are the selection and focusing of knowledge source at each node and the choice of mechanisms and policies for internode communication to permit effective co-operative problem solving. The primary goal of decomposition design is to minimize internode communication relative to internode processing. How this goal is achieved is explained in the following sections.

3.3.1 INTRANODE PROCESSING CONSIDERATIONS

Because each KS is an independent condition action module, KSs communicate through the blackboard. The blackboard records the hypotheses generated by KSs. Any KS can generate a hypothesis

(record it on the blackboard) or modify an existing one. These actions in turn may produce structures that satisfy the applicability conditions of other KSs. In this framework the blackboard serves two roles. It represents intermediate states of problem-solving activity and it communicates message (hypotheses) from one KS that activate other KSs.

The blackboard is divided into a set of information levels corresponding to the intermediate representation levels of the decoding processes (phrase, word, syllable, etc.). A KS typically works with a small number of information level by noticing one or more hypotheses (called the "Stimulus") at one or two levels and by creating new hypotheses or modifying existing ones (the KSs "Response") at one to two levels. For a collection of KSs to be connected across levels, then it must be that any level used by some KS as its stimulus is used by some KS as its response.

In addition to the information level, there is an orthogonal dimension (or set of dimensions) for locating hypotheses in the blackboard- this is the location of the event which the hypothesis describes. In speech understanding systems, most hypothesis (phonemes, syllables, words, phrases, etc.) are located as segments on the dimension of time within the utterance. Hypothesis closer in the location dimension are more likely to be relevant to each other and to be needed jointly for further KS activity. For example, a

word is likely to be created from adjacent syllable hypotheses. All levels in the system taken together with the full extent of the location dimension(s) define a nodes largest possible scope which is also known as "area of interest". In order to understand the notion of area of interest, let us consider a node (Figure 3-2). which has three information levels, labelled L1, L2, and L3 and two knowledge sources KS1 and KS2. The sensor associated with a node produces a single hypothesis on L1 (In general , multiple, alternative hypotheses could be produced) called H1. The knowledge scorce KS1 in the node can take three contiguous hypothesis on L1 call them H2, H1 and H3 - and produce H4 as an abstraction of them on L2. Likewise, Knowledge source kS2 produces hypotheses on L3 from triples of hypotheses on L2.

In order for KS1 to operate , the node must receive hypotheses H2 and H3 as messages from other nodes because its local sensor can generate only H1. Likewise,for KS2 to operate, the H5 and H6 hypothese must be received on L2. The scope required to be representable on L2 is larger than on L1. If processing were to continue similarly above L3 , L3's scope would have to be larger still.Thus,the location dimension of area-of-interest expands at higher level.The lateral communication (e.g. H2 and H3 , and H5 and H6) forms the context for processing and provides a connectivety in the location dimensions.

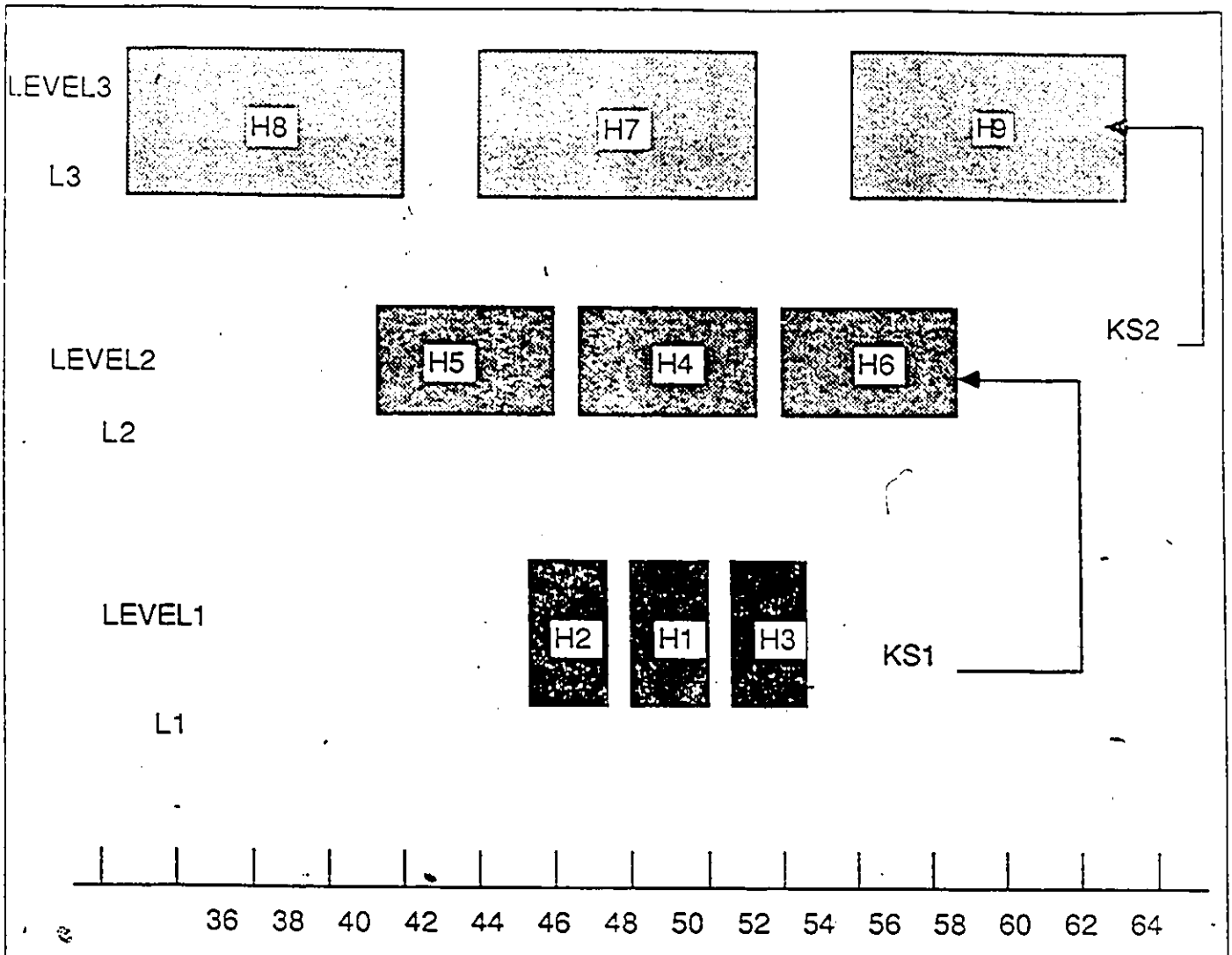


FIGURE 3-2

EXAMPLE OF AREA-OF-INTEREST

3.3.2 INTERNODE COMMUNICATION MECHANISM

In a Hearsay-II System, all inter-KS communication is handled indirectly via the creation, modification,and inspection of hypotheses on the blackboard. The same mechanism is used for internode communication. A transducer KS with access to communication medium (e.g. packet radio) for receiving messages from other nodes, is added to the node; the RECEIVE KS modifies its node's blackboard to reflect those messages, other KS's in the node can use this information. Likewise a TRANSMIT KS can select hypotheses on the blackboard and transmit them for reception by other nodes. Figure 3-3 shows a network of such system.

A KS is triggered by and uses information on the blackboard independent of what other KS created it; thus, information placed on the blackboard by the RECEIVE KS is automatically usable by the other KSs, indistinguishably from locally generated information. Likewise, each KS posts its results on the blackboard without concern for what other KSs might use it; thus, the information to be transmitted by the TRANSMIT KS is already available on the blackboard.

3.3.3. INTERNODE COMMUNICATION POLICY

The ability to run asynchronously eliminates the need for communication costs of synchronization and simplifies the

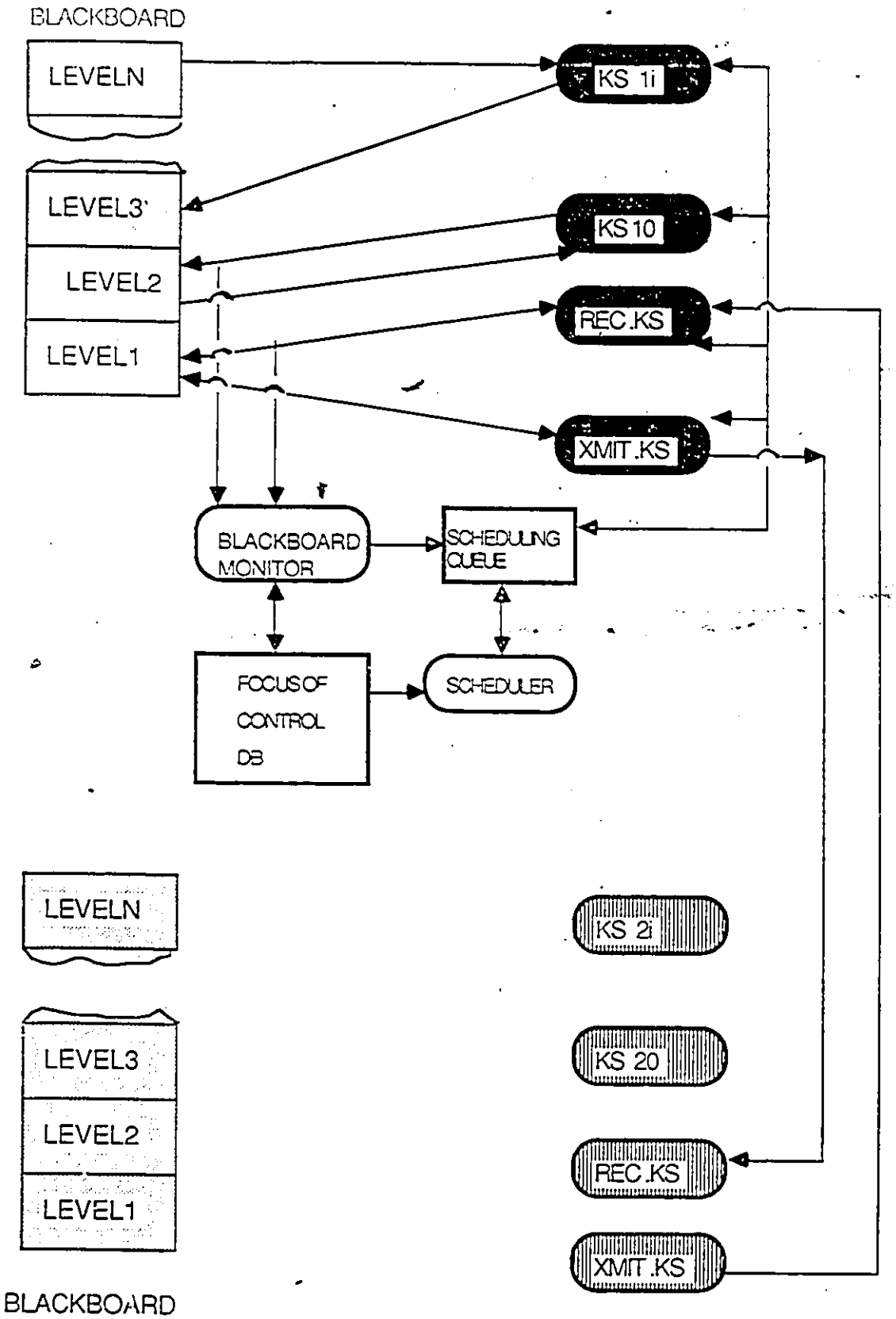


FIGURE 3-3
NETWORK OF HEARSAY II SYSTEM

interaction mechanism. In order to provide each node with information needed from other nodes there is a need for communication. This internode communication is reduced by limiting the amount of information transmitted, the set of nodes to which any particular message is transmitted and the distance the message is transmitted.

The transmission of information is achieved either through global broadcast (which would require high bandwidth) or by a method where the transmitting node knows which other nodes might be interested in the information and, thereby, direct the communication explicitly. There is a problem with latter scheme, because the cost of such a complete distributed knowledge of what is relevant to each node would be high (especially since the information changes as problem progresses). To overcome this problems, Hearsay II uses a scheme where local transmission is based on local knowledge of relevance . Each message is transmitted to a few neighbouring nodes. When a node receives information relevant to it, it incorporates the information into its problem solving state. This action may, in turn , trigger the node to transmit the information (perhaps modified by its knowledge) on the basis of its local knowledge of relevance.

The transmission of a limited subset of a nodes information to a limited subset of other nodes leads to an incremental transmission of information with problem solving at each step.

Local knowledge based processing at each step of the transmission serves to correct errors in the information. Since communication is incremental this error correction capability can serve to limit the propagation of errors.

This basically describes the architecture of Hearsay II System Model and the mechanism of problem solving in a network Hearsay II System.

3.4 ANALYSIS OF HEARSAY-II SYSTEM

In this section we will identify the advantages and the shortcomings of the Hearsay-II approach

Advantages of Hearsay-II System

MECHANISM	IMPACT
(1) ASYNCHRONOUS NATURE OF INFORMATION GATHERING	REDUCED NEED FOR SYNCHRONIZATION
Problem-solving is viewed as incremental, opportunistic and asynchronous process in which decisions if they look promising can be made with incomplete information and later	-Because of this style of problem solving ,a node does not have an apriori order for processing information and can exploit incomplete local information.Thus the processing order within nodes and transmission

re-evaluated in the light of new information.

of information among nodes does need to be synchronized.

(2) USE OF ABSTRACT INFORMATION

-Because the problem solving database is structed as a loose hierarchy of increasingly more abstract problem representations, an abstract representation of one aspect of the solution can be used to constrain analysis of other aspects of the problem.

REDUCED INTERNODE COMMUNICATION BANDWIDTH.

-The ability to use abstract information permits nodes to co-operate by using messages with high information content; thus, the communication bandwidth needed for effective co-operation is reduced.

(3) INCREMENTAL AGGREGATION

-A solution is constructed through the incremental piecing together of mutually constraining and consistent information; incorrect partial solutions naturally die out as a result of this process.

AUTOMATIC ERROR DETECTION

-This method of problem-solving allows a distributed system to detect and reduce the impact of incorrect decisions caused by incomplete and inconsistent local databases and communication losses.

(4) PROBLEM SOLVING AS A SEARCH PROCESS

-Because of uncertainty in data and KS processing, many alternate partial solutions need to be examined in the process of constructing a complete and consistent solution; in this search process, the more uncertainty there exists, the larger the number of alternatives that, in general have to be explored.

INTERNODE PARALLELISM

-The requirement that many alternative partial solutions need to be examined generates the possibility that this search can be carried out in parallel by different nodes. The asynchronous nature of of information gathering introduces the possibility for additional parallelism, since different aspects of the problem and different information levels can be worked on independently. Further, the introduction of additional uncertainty through incomplete and inconsistent local database can be traded off against more search - to the degree that this extra search can be done in parallel and does not itself generate propor-

tionately more internode communiction, internode bandwidth can be lowered without significant degradation in system response time.

(5) FUNCTIONALLY ACCURATE DEFINITION OF SOLUTION

-Due to the oppurtunistic nature of processing and over lapping KS's, the correct solution may be derivable in different ways, i.e., using different ordering sequences for incrementally constructing the solution components or using different components. Because a solution is based on a set of mutually constraining pieces of information, it is also possible for a correct solution to incorporate information that is correct but not considered very likely, or to use incorrect information that

SELF-CORRECTING

-Since there are multiple paths from which a solution can be derived, it is possible to correct for what would be considered fatal errors in a conventional distributed problem-solving system. Additionally, system reliability can be varied without modifying the basic problem-solving structure through the appropriate selection and focussing of local node processing. For example, it is possible to improve reliability by enlarging the overlap among nodes area of interest, thus increasing the

is considered very likely.

likelihood of generating redundant information. This increases the number of alternate ways that a solution can be derived.

CHAPTER 4

CONTRACT NET APPROACH FOR CO-OPERATIVE PROBLEM SOLVING

4.1 A CO-OPERATING EXPERTS METAPHOR

Davis and Smith (SMITH80, SMITH81, DAVIS83) have proposed Contract Net as a method for distributed problem solving. It is based on the paradigm, where a single agent constructs a plan to be carried out by a group of agents and then hands out the pieces of the plan to the relevant individuals.

The Contract Net approach can be best illustrated by considering the example of a group of human experts working together to complete a large task. (This metaphor of a group of human experts working together to solve a problem for a problem solver operating in a distributed processor has been used in various AI systems - for example in HEWITT77). It is assumed that no one expert is in total control of the others, although one expert may be ultimately responsible for communicating the solution of the top level problem to the client outside the group. In such a situation each expert may spend most of his time working alone on various subtasks that have been partitioned from the main task, pausing occassionally to interact with other members of the group. These interactions involve requests on subtasks or the

exchange of results.

Individual experts can assist each other in at least two ways . First, they can divide the workload among themselves, and each node can independently solve some subproblems of the overall problem. This is called 'Task Sharing '. In this mode of co-operation the primary concern is the way in which the experts decide who will perform which task. The Contract Net approach postulates negotiation as a means for effecting this agreement.

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

If another expert (E2) belives he is capable of carrying out the task that E1 described,he informs E1 of his availability. E1 may discover several such volunteers and can choose from among them. The chosen volunteer then requests additional details from E1 and the two engage in further direct communication for duration of the task.

When subproblems cannot be solved by independent experts working alone,a second form of co-operation is appropriately

used. In this form, the experts periodically report to each other the partial results they have obtained during execution of individual tasks. This is called 'Result Sharing'. It is assumed in this mode of co-operation that problem partioning has been effected apriori and that individual experts work on subproblems that have commonality.

An expert (E1) reports his partial result for his subproblem to his neighbours (E2 and E3) when that result may have some bearing on the processing done by them. E2 and E3 attempt

(1) to use E1's result to confirm or deny competing results for their subproblems, or

(2) to aggregate partial results of their own with E1's result to produce a result that is relevant to E1's subproblem as well as their own, or

(3) to use E1's result to indicate alternative lines of action that they might take to solve their own sub problems.

4.2 DISTRIBUTED PROBLEM SOLVING MODEL

4.2.1 PHASES OF DISTRIBUTED PROBLEM SOLVING

Distributed Problem Solving, using Contract Net approach, is comprised of three phases: Problem Decomposition, Subproblem Solution and Answer Synthesis, as shown in FIGURE 4-1 .

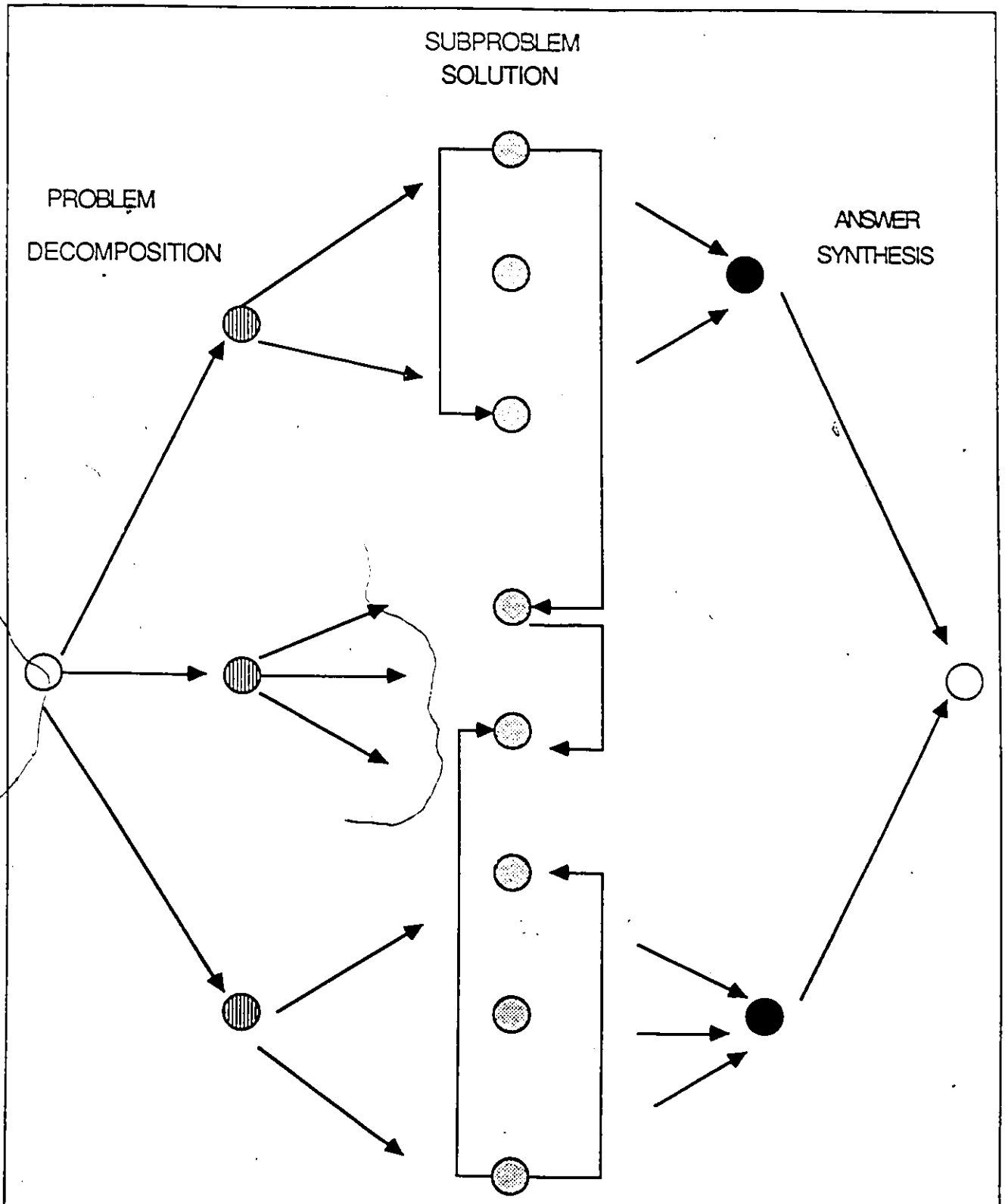


FIGURE 4-1

PHASES OF DISTRIBUTED PROBLEM SOLVING

PROBLEM DECOMPOSITION PHASE

In the first phase the problem is decomposed into subproblems. The decomposition process may involve a hierarchy of partitionings. Decomposition proceeds until kernel (non decomposable) subproblems are generated. Let us consider a simple distributed sensing system (DSS). In the problem decomposition phase, the subproblems of detecting objects in specific portions of overall area of interest are defined and distributed among the available sensors.

SUB-PROBLEM SOLUTION

The second phase involves solution of the kernel subproblems.As As shown in the Figure 4-1,this may necessitate communication and co-operation among the nodes attempting to solve the individual subproblems. In the DSS example,communication is required in the subproblem solution phase (1) if objects can move from one area to another so that it is helpful for sensors to inform their neighbours of the movement of objects they have detected , or (2) if it is difficult for a single sensor to reliably detect objects without assistance from other sensors.

ANSWER SYNTHESIS

Answer synthesis is performed in the third phase ; that is integration (i.e. integrating the results of subproblems to generate a solution to the overall problem) of subproblem results

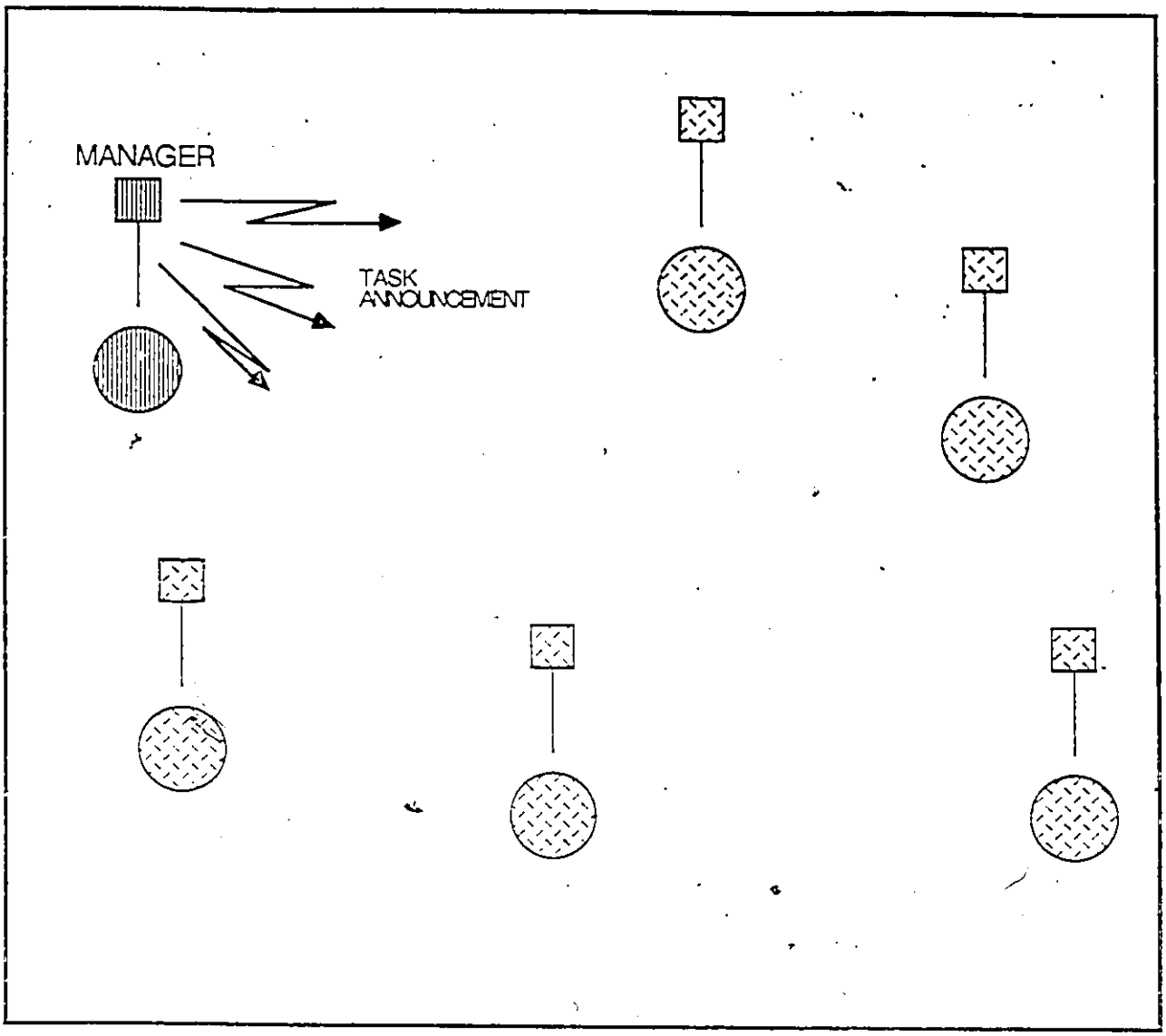


FIGURE 4-2 : SENDING A TASK ANNOUNCEMENT

to achieve a solution to the overall problem.

The central element in this approach to problem solving is the concept of negotiation.By negotiation, the interested parties exchange information and come to an agreement.Negotiations can be considered to consist of three important components.

(1) there is a two way exchange of information

(2) each party to the negotiation evaluates the information from its own perspective, and

(3) final agreement is achieved by mutual selection

The process of negotiation appears to have two major benefits:

(i) it enables the matching of idle problem solvers to outstanding tasks.

(ii) allows matching of KSs to tasks.

Next, we will see how this process of negotiations is used in a form of co-operation called the TASK SHARING.

4.2.2 TASK SHARING MECHANISM

Task Sharing is a form of co-operation in which individual nodes assist each other by sharing the computational load for the execution of the subtasks of the overall problem. 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 wherby nodes with tasks to be executed can find the most appropriate idle nodes to execute those tasks. This is known as the connection problem.

FIGURE 4-3 : RECEIVING TASK ANNOUNCEMENTS

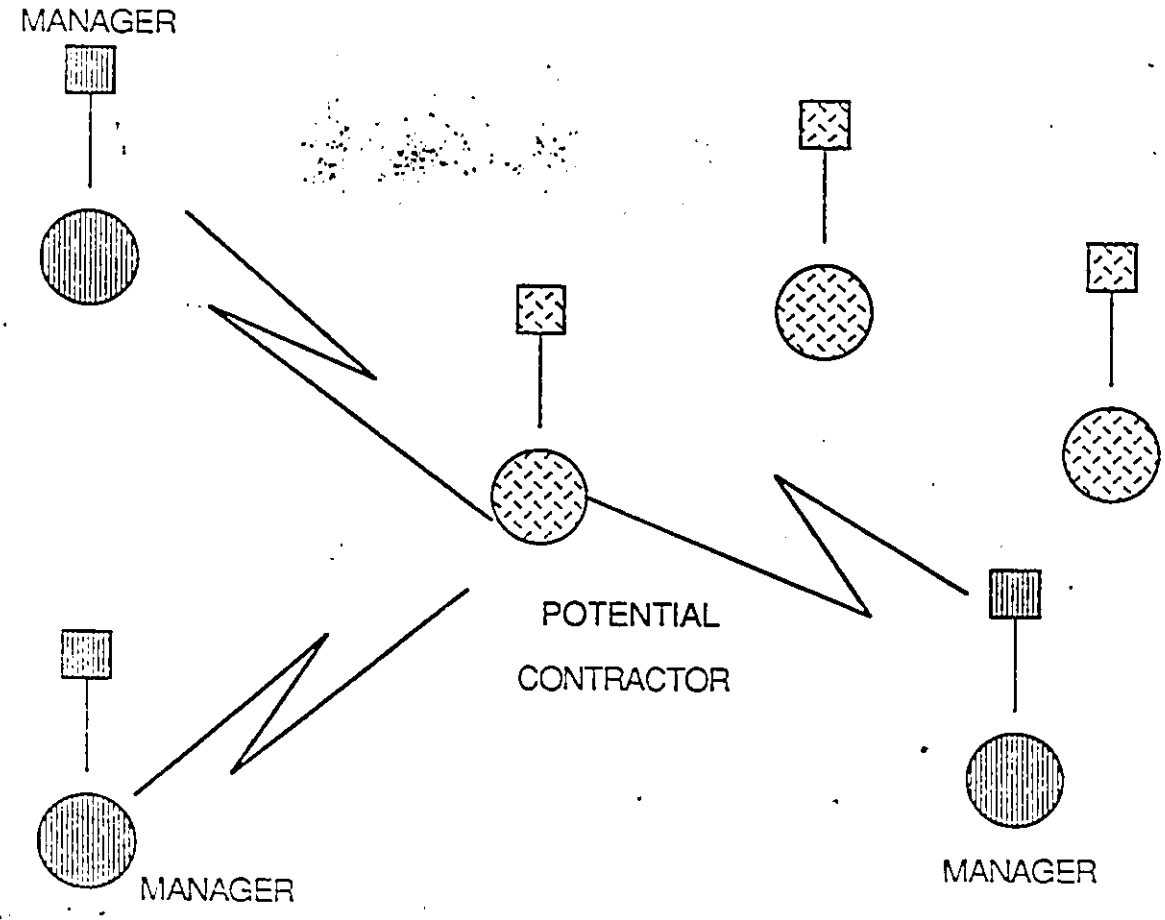
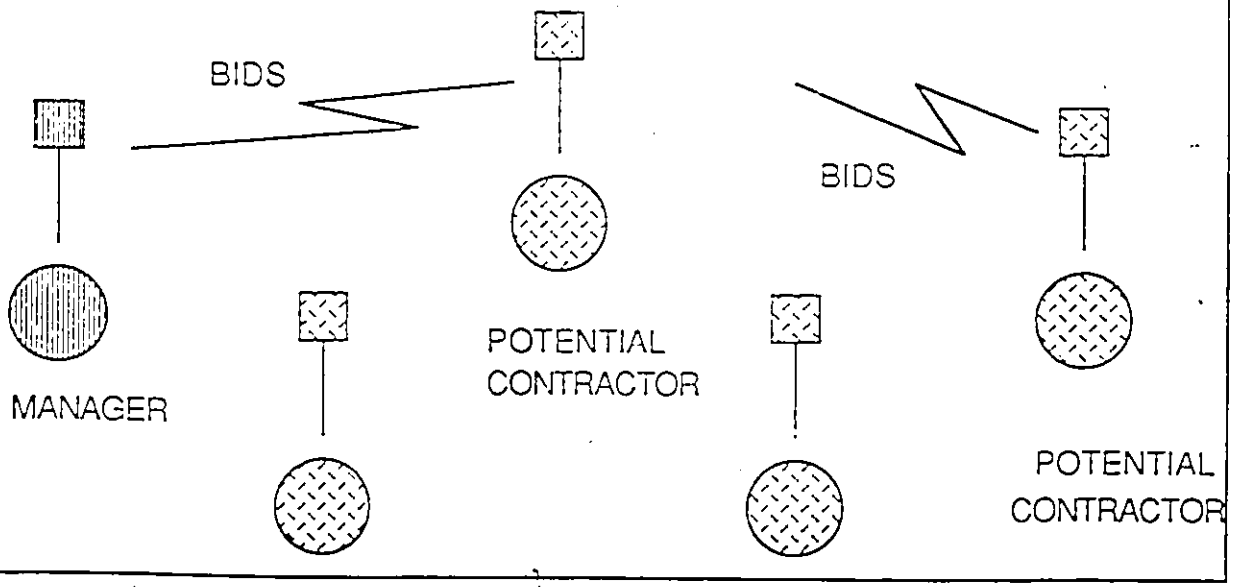


FIGURE 4-4 : BIDDING



Negotiation is used to structure node interactions and solve the connection problem in task shared systems.

In the Contract Net Model, collection of nodes is known as Contract Net. And a contract (SMITH80) is an explicit agreement between a node that generates a task (the manager) and a node willing to execute the task (the contractor). The manager is responsible for the actual execution of the task. Individual nodes are not designated apriori as manager or contractor; these are only roles and and any node can take on either role dynamically during the course of problem solving.

The manager for a task advertises the existence of the task to other nodes with a task announcement message (FIGURE 4-2). Available nodes (potential contractors) evaluate task announcements made by several managers(FIGURE 4-3)and submit bids for which they are suited (FIGURE 4-4). An individual manager evaluates the bids and awards contracts for execution of the task to the node it determines to be most appropriate (FIGURE 4-5). Manager and contractor are thus linked by a contract (FIGURE 4-6) and communicate privately while the contract is being executed. 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.

FIGURE 4-5 : MANAGER MAKING AN AWARD

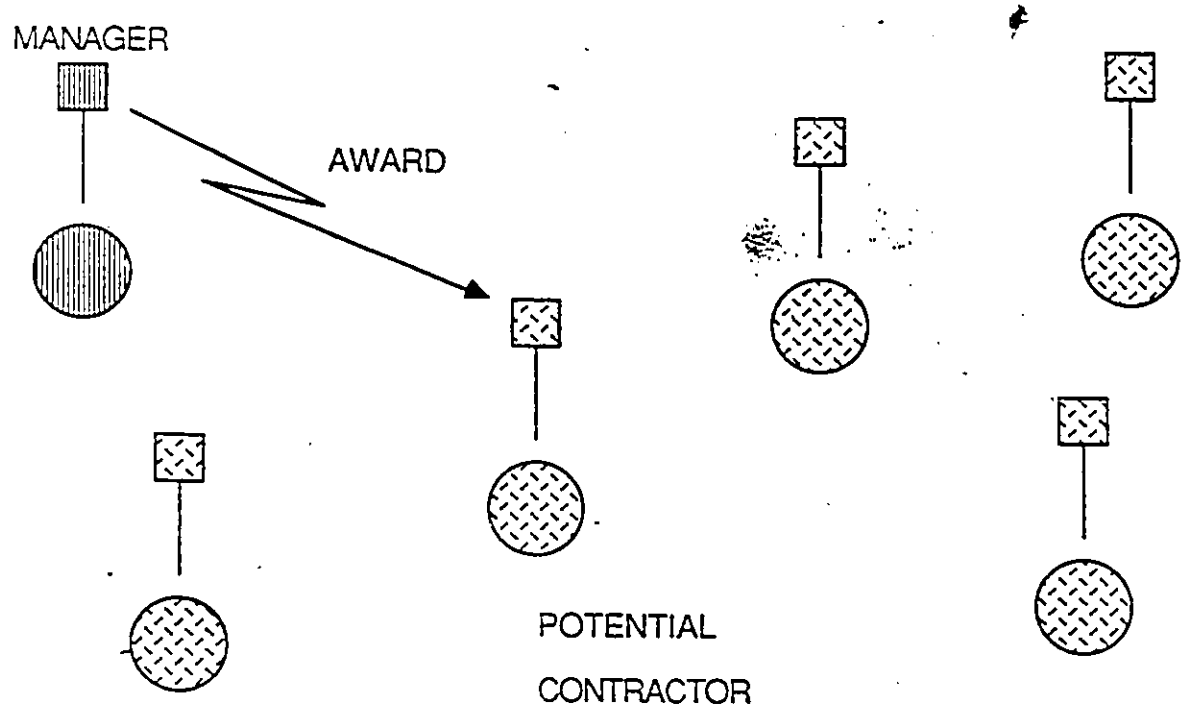
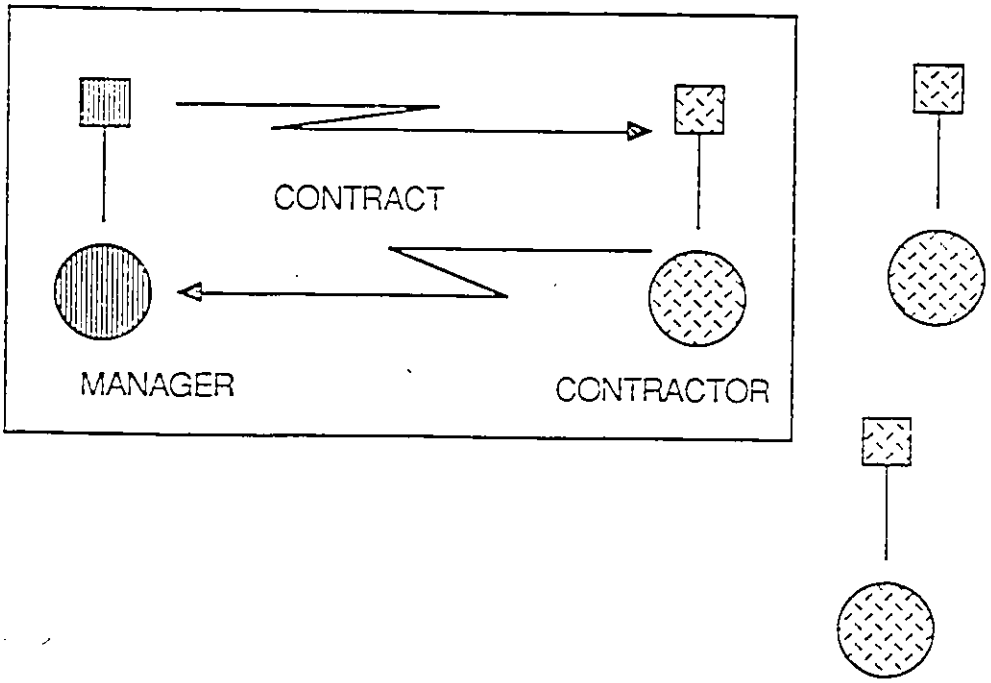


FIGURE 4-6 : CONTRACT ESTABLISHED



A report is used by a contractor to inform its manager that a task has been partially executed (an interim report) or completed (a final report).The report contains a result description that specifies the the results of the execution.The example discussed here is simple to focus on the issue of co-operation in Contract Nets. Additional complications could arise and how these issues can be addressed are explained in detail in (SMITH80).

4.3 CONTRACT NET APPLICATION: DISTRIBUTED SENSING SYSTEM

In this application, Contract Net is used for the purpose of area surveillance of the sort encountered in air or ship traffic control. The distributed sensing system (DSS) consists of a network of nodes, each having either sensing or processing capabilities and all spread throughout a large geographical area.

The solution to the problem is a dynamic map of traffic in the area. The task involves detection, classification, and tracking of vehicles. Construction and maintenance of the map require interpretation of the large quantity of sensory information received by the collection of sensor elements.

Since the objective is to produce a single map of the entire area, a processor node - called the monitor node - is chosen to carry out the final integration of information and transmit it to the appropriate destination. The node is also assigned the responsibility for begining the initialization of the DSS. Its

total set of responsibilities includes starting the initialization as the first step in net operation, integrating the overall map as the last step in analysis, and then communicating the result to the appropriate agent.

Hardware

All communication in the DSS is assumed to take place over a broadcast channel (using for example packet radio techniques). Each node has one of two capabilities: sensing or processing. The sensing capability includes signal analysis. Nodes with processing capability supply the computation power necessary to effect the high-level analysis and control in the net.

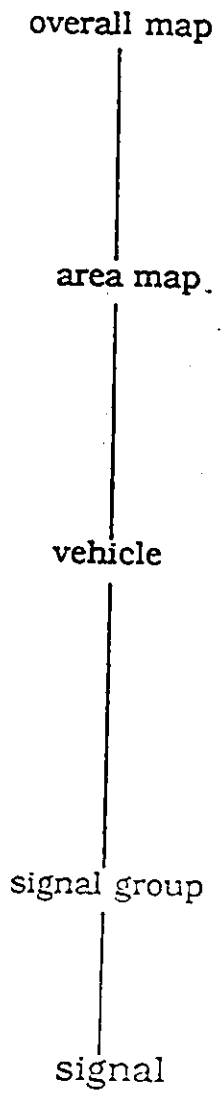
Data and Task Hierarchy

The DSS must integrate a large quantity of data, reducing it and transforming it into a form meaningful to a human decision maker. This process can be viewed as occuring in several stages, which together form a data hierarchy (Figure 4-7).

In this application, at any given moment a particular node handles data at any one level of the data hierarchy, but may communicate with nodes at other levels.

At the bottom of the hierarchy are the audio signals, which are described in terms of features: frequency, time of detection,

FIGURE 4-7



DATA HIERARCHY

strength, changes in strength, name and position of the detecting node, and name, type, and orientation of the detecting sensor. Signals are formed into signal groups, collection of related signals.

The next level of the hierarchy is the description of the vehicle. It has one or more signal groups associated with it and is further specified by position, speed, course and classification. The area map forms the next level of the data hierarchy. It contains information about the vehicles in a given area. There will be several such maps for the DSS. Together they span the total coverage of the system.

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

The hierarchy of tasks, Figure 4-8, follows directly from the data hierarchy. The monitor node manages several area contractors. These contractors are responsible for the formation of traffic maps in their immediate areas. Each area contractor in turn, manages several group contractors that provide it with signal groups for its area. Each group contractor integrates raw signal data from signal contractors that have sensing capabilities.

The area contracors also manage several vehicle contractors that are responsible for integrating information about individual

TASK HIERARCHY

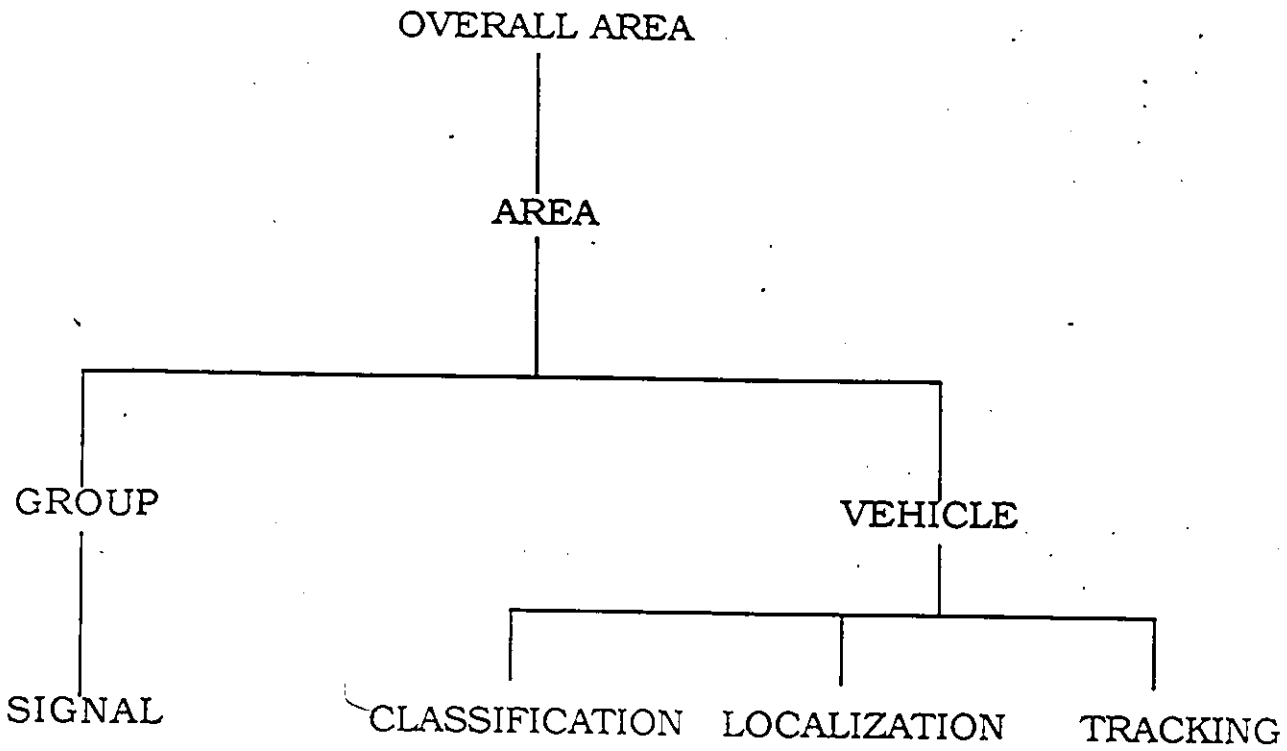


FIGURE 4-8

vehicles. Each of these contractors manages a classification contractor that determines vehicle type, a localization contractor that determines vehicle position, and a tracking contractor that tracks the vehicle.

CONTRACT NET IMPLEMENTATION

There are two phases to this problem: initialization of the net and operation. The nodes in the contract net play dual role: They are simultaneously contractors obligated to carry out a task they were awarded, and managers for any tasks which they in turn announce. For example, node number 2 in Figure 4-9 is simultaneously

(i) a contractor for the area task (and hence is charged with the duty of producing area maps from vehicle data),

(ii) a manager for group formation tasks it announces and contracts out, and

(iii) a manager for any vehicle tasks which it contracts out.

Nodes are thus simultaneously workers and supervisors (Compare Figure 4-8 and Figure 4-9).

INITIALIZATION

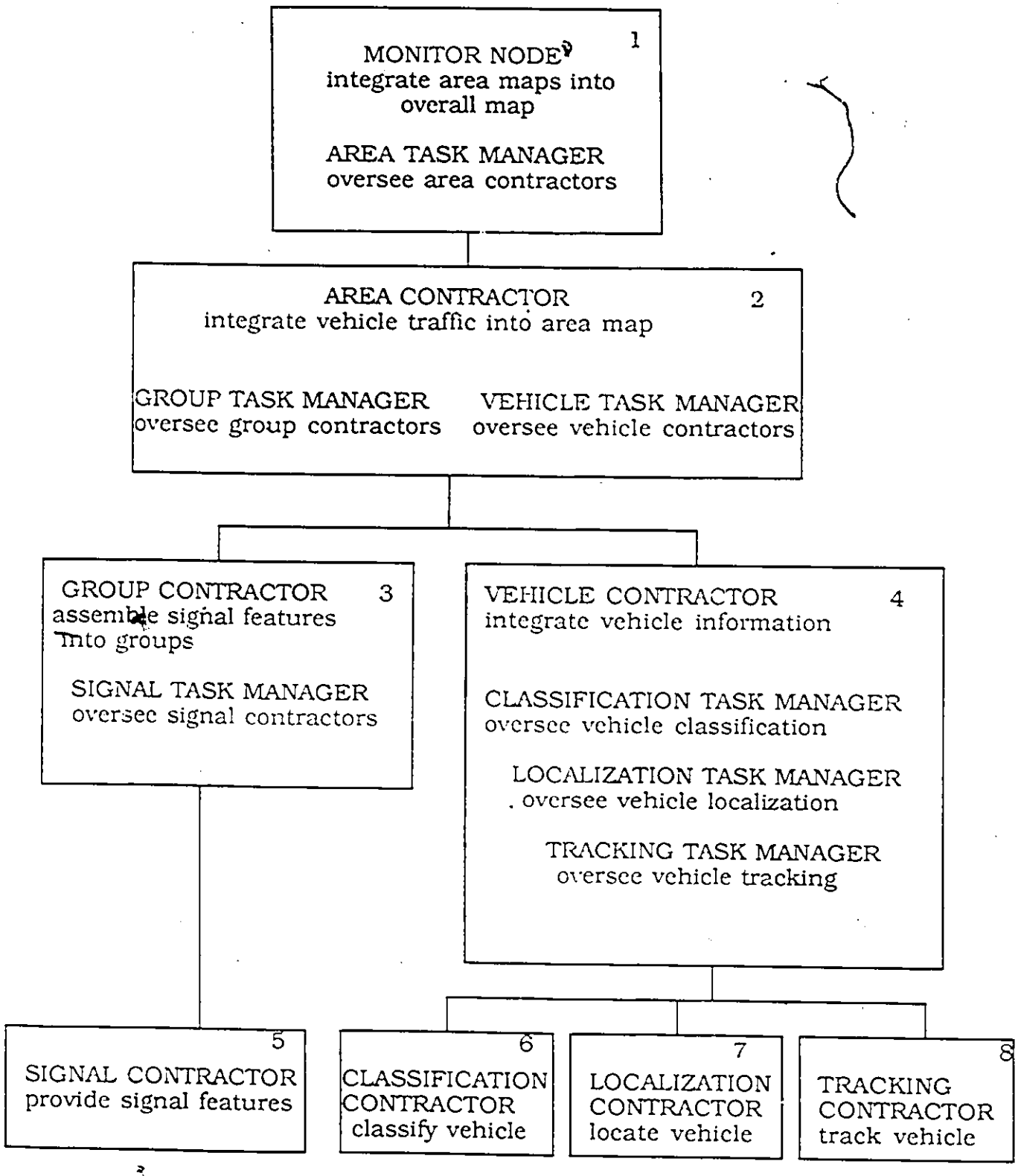
The monitor node is responsible for formation of the overall map. It must first select nodes to be area contractors and partition the system's span of coverage into areas based on the positions of the nodes selected. For purposes of illustration

here, it is assumed that the monitor node knows the names of the nodes that are potential area contractors, but must establish their positions in order to do the partitioning.

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

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

FIGURE 4-9



NODES AND THEIR ROLES

of potential area contractors.

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

The group contractors integrate signal features into groups, and start by finding a set of contractors to provide the signal features. The node interaction is an agreement between a node with a task to be done and a node capable of performing that task.

Let us now see how the messages are 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 Figure 4-10. The task abstraction is the type of task, the position of the manager making the announcement, and a specification of its area of responsibility. This enables a

SIGNAL TASK ANNOUNCEMENT

To :

From: 25

Type: TASK ANNOUNCEMENT

Contract: 22-3-1

Eligibility Specification

MUST-HAVE SENSOR
MUST-HAVE POSITION AREA A

Task Abstraction

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

Bid Specification

POSITION LAT LONG
EVERY SENSOR NAME TYPE

Expiration Time

30 1730Z OCT 1987

FIGURE 4-10

potential contractor to determine the manager to which it should respond. The eligibility specification indicates that the only nodes that should bid on the task are those which

(a) have sensing capabilities, and

(b) are located in the same area as the manager that announced the task.

The bid specification indicates that a bid should contain the position of the bidder and the number of each of its sensor types, information that a manager needs to select a suitable set of sensor nodes.

The potential signal contractors listen to the task announcement made by the signal managers. They respond to the nearest manager with a bid (Figure 4-11) that supplies their position and a description of their sensors. The managers use this information to select a set of bidders that covers their area of responsibility with a suitable variety of sensors, and then award signal contracts on this basis (figure 4-12).

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

FIGURE 4-11 Signal Bid

To: 25

From : 42

Type: BID

Contract: 22-3-1

Node Abstraction

LAT 62N LONG 9W
SENSOR NAME S1 TYPE S
SENSOR NAME S2 TYPE S
SENSOR NAME T1 TYPE T

To: 42

From : 25

Type: AWARD

Contract: 22-3-1

Task Specification

SENSOR NAME S1
SENSOR NAME S2

FIGURE 4-12 Signal Award

other using its own distinct evaluation procedure.

COOPERATION

The activities of the system as it begins operation is examined here. For the sake of brevity the actions are described at the level of task announcements, bids, and contracts.

When a signal is detected or when a change occurs in the features of a known signal, the detecting signal contractor reports this fact to its manager. This node in turn, attempts either to integrate the information into an existing signal group or to form a new signal group.

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

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

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

CHAPTER 5

EXPERTNET: SYSTEM ARCHITECTURE AND IMPLEMENTATION

In the previous chapters the two major models for Distributed Problem solving namely - Hearsay-II System and Contract Net were described. The goal of these models is to develop a theoretical framework for distributed problem solving. The major issues involved in the development of this goal include:

(1) How to organise the processing nodes
(2) How to distribute subproblems among the processing nodes
(3) What control mechanism is needed to maintain global coherence during problem solving, to effectively utilize knowledge sources, and to achieve a better performance.

In this thesis, a technique is developed to allow for resource sharing on a network of workstations or personal computers. The network using this technique has been named Expertnet. The objective of this technique is to:

(1) allow workstations to request services, not available locally, from other workstations on the network
(2) utilize resources of idle workstations on the network

(3) update local workstation knowledge base ,on an ongoing basis, about the expertise and capabilties of other workstations

For the purpose of implementing Expertnet, the Hearsay-II and Contract Net models were considered. In addition the Office Model described in Chapter 1 was considered. The Contract Net approach and the Office Model were chosen for the development. A simple question-and-answer inter-node communication mechanism is used instead of the negotiation mechanism used in the Contract Net. In this mechanism two types of messages,question-type and answer-type are used. All the requests are treated as question-type messages. And all the responses are considered as answer-type messages.

At a recent study (HIBBARD85) at Carnegie-Mellon University, the negotiation mechanism described in Contract Net was used for resource sharing in a Spice personal computing environment. The major issues, in a process called Butler Process for resource sharing, are security and protection. However, in the Expertnet the major issue is effective usage of workstations on a network, both in terms of hardware and applications software. Security issues have been taken into consideration by enabling the host machine (the workstation providing services to requesting workstation) to provide restricted access to some of its applications/resources.

5.1 EXPERTNET NODE ORGANIZATION

Figure 5-1 represents the architecture of a processing node (i.e. workstation). Each node consists of a Receive, Broadcast, Transmit, Planner, Message Server, Process Server, Response Handler, Request Handler, Meta Knowledge Base, and Knowledge Sources (set of functions or procedures).

The Meta Knowledge Base at the processing node provides the necessary information to allow a high level reasoning process to be implemented in the internode communication and planning process. The Planner generates a problem solving plan for messages received from other nodes or if the problem is given to the host machine by its user. A detailed description of this mechanism is given in this chapter later.

The Receiver receives messages from other nodes and gives them to the Message Server for necessary actions to be taken. If the message is a response to the request from the host node, then the message is given to the Response Handler. On the other hand if the message is a request for service from other nodes, then the message is given to Request Handler for appropriate action to be taken.

The Process Server is responsible for providing the Process/Function, on the Host Node, to solve the the requested problem. The result is given to the Message Server to be transmitted to the requesting node with the help of Sender.

EXPERTNET ARCHITECTURE

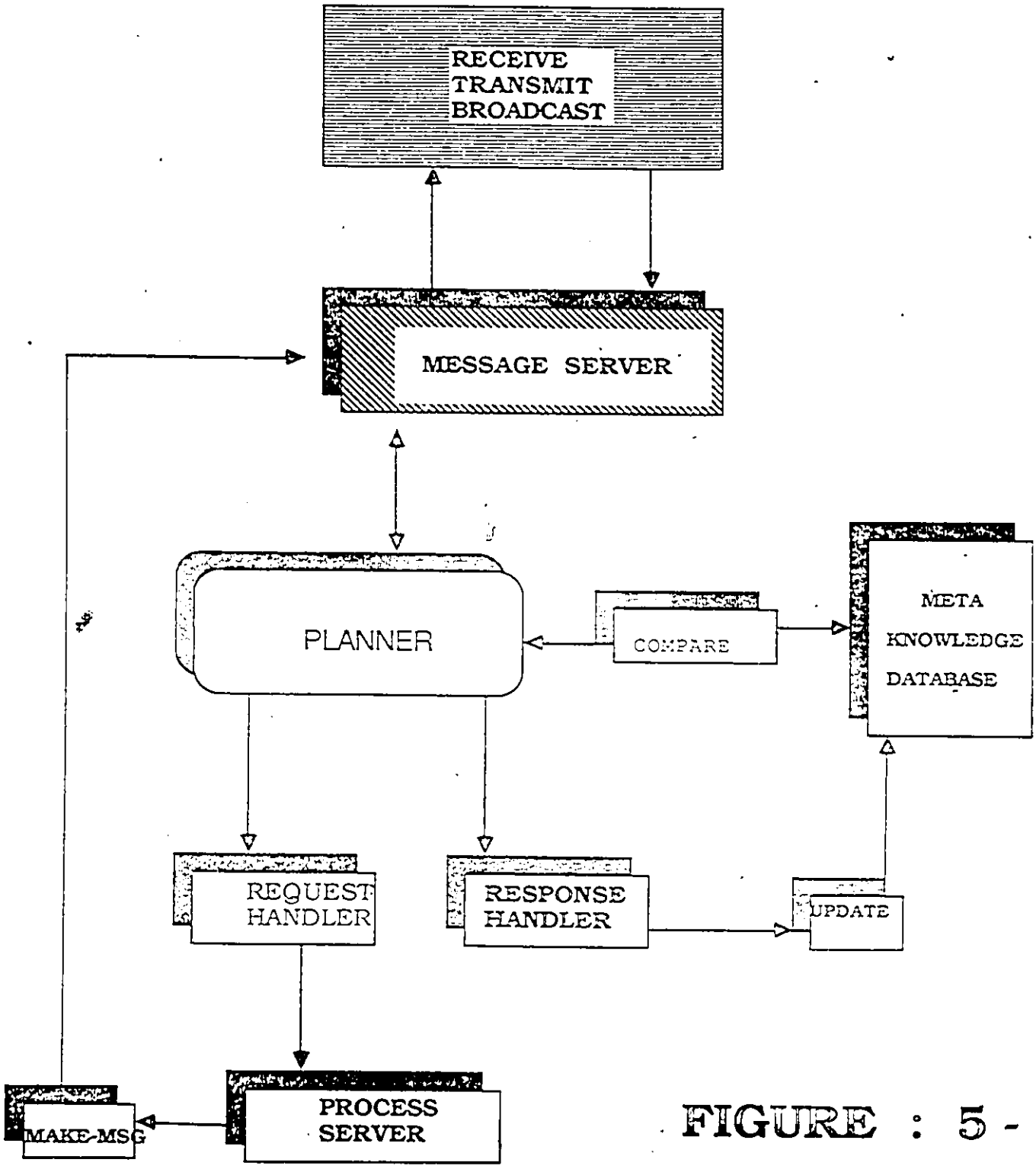


FIGURE : 5 - 1

5.2 META KNOWLEDGE BASE

Two types of meta knowledge are available to the processing node. One type stores the names of problems which can be solved by the Host Node,as well as the process/function names which are required to solve these problems. The other type contains the list of processing nodes which can solve the problems for which the expertise is not resident in the Host Node. Inaddition, the list of processing nodes which are restricted from receiving services for a particular problem is also stored in the meta knowledge base. The motivation for using meta-level knowledge is to allow a high level reasoning process to be implemented in the internode communication and planning process.

5.2.1 KNOWLEDGE REPRESENTATION

Each type of Meta-Level Knowledge is organised according to frame like structure. The format of this structure is shown in Figure 5-2 (a). Each piece of meta-level knowledge is represented by a frame. There are five slots in each frame. The first slot is a brief description of the piece of meta-level knowledge which is usually a problem either within or outside the area of its own expertise.

For a solvable problem, the next slot contains the name of the process or function which can solve the problem. A solvable problem is one which can be solved by the host node. In other

FIGURE 5-2 : Examples of Meta-Level Knowledge And Their Representation

PROB	:	A Brief Description of a Problem
PROC	:	Name of a Process or Function which can Solve the Problem
CANDIDATE	:	Processing Node/Nodes which can Solve the Problem
RESTR-NODE	:	Node/Nodes Restricted from Receiving Service for a Certain Problem
GROUP	:	Nodes belonging to the same group as Host node

(A) The Format of Meta Knowledge Base Frame

PROB	:	CALCULATE-TOTAL
PROC	:	1-2-3
CANDIDATE	:	NIL
RESTR-NODE	:	NIL
GROUP	:	NODE 10 NODE 20

(B) A Frame of Meta Knowledge Base Within the Area of Expertise of a Processing Node

PROB	:	FED-TAX
PROC	:	NIL
CANDIDATE	:	NODE1
RESTR-NODE	:	NIL
GROUP	:	NODE 10 NODE 20

(C) A Frame of Meta Knowledge Base outside the Area of Expertise of a Processing Node

words the host node has the expertise locally to solve the problem. For a problem which cannot be solved by the Host Node, the third slot stores a list of identifications of processing nodes which are believed to have the expertise to solve the problem. Initially this slot contains nil value, but gradually accumulates this information during problem solving.

The fourth slot stores the list of identifications of the processing nodes which are restricted from receiving services, of Host Node, for solution of a certain problem. In other words, if the host node does not wish to offer services to a particular node for certain problem, then this restricted node will be included in this slot of the host node. If there is no restriction imposed by the Host Node, then the value of this slot is nil.

The fifth and final slot contains the list of nodes which belong to the same group as the host node. This list is used when the host node has to broadcast for services to other nodes. Instead of broadcasting to all the nodes in the network, the host node now only has to broadcast to only those nodes which belong to the group. The assumption is that in an office environment normally office workers belonging to the same group communicate frequently for solving a problem. Very rarely workers communicate with other workers outside the group for solution of a task. Normally a node broadcasts for service if it cannot solve a problem and it does not know of any other node which can solve the problem.

Two examples are given in Figure 5-2 (b) and (c) respectively. One is a frame containing a piece of meta-level knowledge within the area of the Host Node's expertise. The other example is for a piece meta-level knowledge outside the area of Host Node's expertise . The value nil is used to indicate that there is no information in that slot.

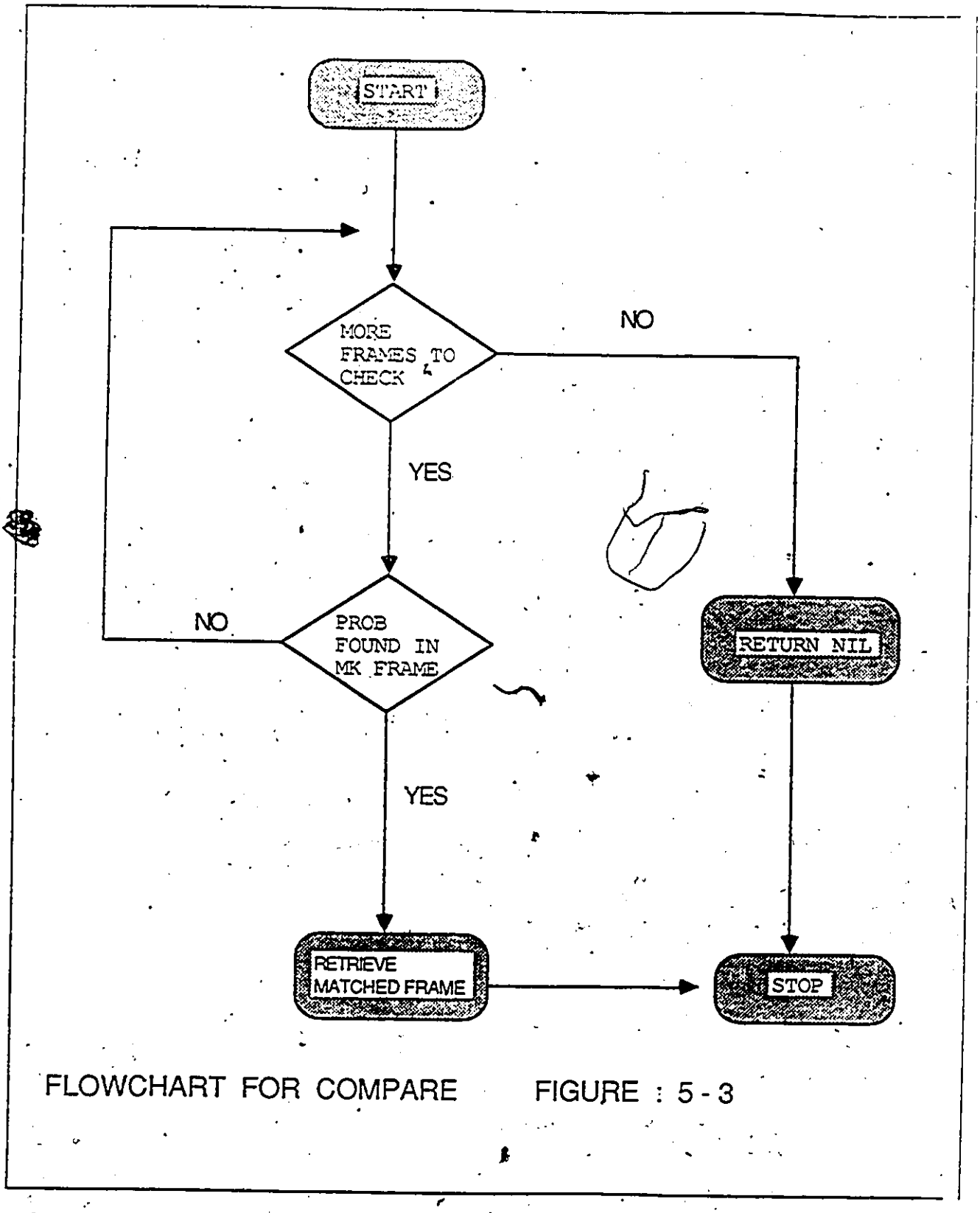
5.2.2 ASSOCIATED FACILITIES

Four facilities are available for manipulating the contents of the Meta Knowledge Base: COMPARE , ADD-MK-FRAME, UPDATE-CAND-LIST and UPDATE-PROC-NAME.

The COMPARE is used to get a frame out of the meta knowledge base. The ADD-MK-FRAME adds a new frame to the meta knowledge base. The UPDATE-CAND-LIST and UPDATE-PROC are used for updating the meta knowledge base frame's attribute values .

COMPARE

Compare is designed to get from the meta knowledge base,a frame whose problem slot attribute value matches the problem name in the message received from other nodes or the problem given by the user of the host machine. A flowchart of the operation of the COMPARE is shown in Figure 5-3. In this architecture , the meta knowledge base is assumed to have no hierarchies.



FLOWCHART FOR COMPARE FIGURE : 5 - 3

The Compare searches through the meta knowledge base to find a frame whose problem slot value matches the attribute value provided either by the user or the message received from other nodes. If the match is successful the matched frame is returned . Otherwise, the Compare simply takes another frame and continues searching. The process continues until the meta knowledge base is completely searched. If no successful match is found ,a nil is returned to indicate a search failure. Otherwise, the retrieved matched frame is returned. The Compare uses a primitive called GET-PROBLEM-NAME to retrieve the problem slot attribute value from a frame in the meta knowledge base. The retrieved attribute value is compared to the attribute value provided by the user or the received message.

ADD-MK-FRAME

This function is used for the purpose of adding a new frame to the Meta-Knowledge Base.

UPDATE-CAND-LIST

This function is used to update the candidate list of a Meta-Knowledge frame in the Knowledge Base. As discussed earlier , a Host Node broadcasts Messages to other workstations

- if it does not have the expertise to solve a particular problem

- and it has no knowledge about any other workstation, on the

network, which has the expertise to solve the particular problem.

A Guest Node on receiving the Broadcast Message will solve the problem and return the result in Response Type Message to the Requesting Node. The latter will on receiving the Response Message update the Meta Frame in the Knowledge Base ,by including in the candidate list (of the Meta Frame) the Guest Node Identification. Shown below are two meta frames in the knowledge base of the host node:

(1) Meta Frame with NIL value for attribute PROC (indicating non availability of expertise to solve the problem ' YEAR-END-EARNINGS ' by the host node) and NIL value for the slot Cand-List (indicating that the host node has no knowledge about the other nodes which can solve the problem).

((PROB YEAR-END-EARNINGS (PROC NIL) (CAND-LIST NIL)
(RESTR-NODE NIL))

(2) Meta Frame with CAND-LIST attribute value NODE20 (indicating that the NODE20 has the expertise to solve the problem YEAR-END-EARNINGS). The UPDATE-CAND-LIST function is used for updating the Meta Frame.

((PROB YEAR-END-EARNINGS) (PROC NIL) (CAND-LIST NODE20)

(RESTR-NODE NIL))

5.2.3 THE PLANNER

The purpose of the Planner is to plan and organise the manner in which a given problem is to be solved.The problem can be given to the Host Node either by the user of the host node or it can be received from other Nodes (Guest Node or Requesting Node) on the the network. Depending on the source of the given problem the Planner organises the problem solving process in the following manner:

A. Planning for a Problem provided by the User to the Host Node

The following steps are followed in this planning process:

(1) The meta knowledge base is checked to see if the given problem is present in the knowledge base. A function called COMPARE is used for this purpose. The function returns the entire Meta Frame if the problem name matches the attribute value of the problem slot.

(2) Next the process or function which can solve this problem is retrieved from the Meta Frame using a primitive GET-PROC-NAME.

(3) The retrieved process name is given to the function PROCESS-SERVER which solves the problem using the specified process and returns the result to the user.

(4) If the Meta Frame does not contain the process name to solve

the problem, then the next slot of the Meta Frame is checked to see if there is any node or the network which can solve this problem. This is done using the primitive GET-CANDIDATE. If there exists a node that can solve this problem,then the problem and the data are given to the MESSAGE-SERVER to be transmitted to the other node.

(5) If the Meta Frame does not contain ID of any other node on the network which can solve this problem, then the problem and the data are given to the MESSAGE-SERVER to be broadcast to other nodes.

B. Planning for the Problem Received from Other Nodes.

The following steps are followed in this planning process:

(1) Destination address is checked to see if the received message is meant for the Host Node or if it is a Broadcasted Message.

(2) If the received message is of Response Type (meaning it contains solution to the problem requested by the Host Node) then the message is given to the RESPONSE-HANDLER for appropriate action to be taken.

(3) If the received message is of Request Type (meaning it is a Request, from a node on the network, for services from the Host Node) then the message is given to the REQUEST-HANDLER for appropriate action.The REQUEST-HANDLER performs the

following actions:

- Checks if the problem specified in the received message can be solved by the host node. If it can then the Meta Frame is retrieved from the meta knowledge base.

- The Meta Frame is checked to see if the node requesting service is in the restricted candidate list of the host node. If it is in the restricted list, then the requesting node is not provided any service and message is discarded.

- If the Requesting Node is not a member of the restricted List of the host node, then process name to solve the problem is retrieved from the Meta Frame.

- The retrieved process name is given to the PROCESS-SERVER to solve the problem and the result send back to the requesting node using MESSAGE-SERVER.

5.2.4 THE MESSAGE-SERVER

The MESSAGE-SERVER performs basically two main functions:

- It checks the received message to see if it is meant for the host node. If the message is meant for the Host Node then it gives it to the PLANNER for appropriate planning as described in the preceeding section.

- The other function deals with sending messages to other nodes.

If destination address of message is known then the message is transmitted using the function TRANSMIT . If the destination address of the message is '* ' then the message is broadcast to all other nodes on the network using the function BROADCAST.

The EXPERTNET, as seen from the above discussion, uses both Broadcast and Directed transmissions. The type of transmission chosen depends on the intentions of the host node and its knowledge about the capabilities of other nodes. When the capabilities of other nodes is known, then the host node simply selects the node with the best match and chooses a directed transmission mechanism. Otherwise a broadcast mechanism is used. Broadcast communication also serves as a method to collect information about other nodes on the network. Hence messages transmitted among nodes during the problem solving process provide a rich source for knowledge acquisition. Based on the newly received information, the related existing knowledge is updated. The updated knowledge is then used to improve the system's communication and planning process.

CHAPTER 6

PROOF BY RESOLUTION

The EXPERTNET, as discussed in the previous section, solves a problem with the help of certain rules. In this section a proof procedure called resolution is used. The resolution is a simple iterative process (RICH83, NILSSON80), at each step of which two clauses, called the parent clauses are compared (resolved), yielding a new clause that has been inferred from them. The new clause represents the way the two parent clauses interact with each other. In order to carry out the resolution technique the various inference rules represented in the predicate calculus form must be converted into clausal form .

6.1 KNOWLEDGE REPRESENTATION USING PREDICATE LOGIC

The various rules that are used by the EXPERTNET for solving a problem given to a particular workstation , are represented in Predicate Logic as follows :

RULE 1 :

$\forall x \forall p$ (CANSOLVE(X,p) --> SOLVED (X,p))

where X = Node / Workstation with a problem to be solved

p = Problem to be solved

The above rule states that, " If 'X ' can solve 'p' then the problem given to the node 'X ' is solved. In otherwords the node has the expertise, locally, to solve the problem 'p' ".

RULE 2 :

$\forall X \forall p \exists Y$ ($\neg$ CAN_SOLVE(X,p) $\wedge$ SOLVER_KNOWN (X,Y,p)

--> REQUEST_SERVICE (X,Y,p))

Where X = Node which has a problem to solve

p = Problem to be solved

Y = Node from which 'X' requests service for the solution of problem 'p'

This rule states that, " If 'X' cannot solve 'p' and if 'X' knows that there exists a node 'Y' which can solve the problem, then 'X' will request service from 'Y' for the solution of the problem."

RULE 3 :

$\forall X \forall p \forall Y$ (REQUEST_SERVICE (X,Y,p) $\wedge \neg$ RESTRICTED_NODE (X,Y,p)

--> GET-SERVICE (X,Y,p))

This rule states that, " If 'X' request service from 'Y' for the solution of problem 'p' and if 'X' is not a restricted node of

'Y ' (i.e. 'Y' has not restricted 'X' from receiving services) then 'X' will get service from 'Y' .

RULE 4 :

$\forall X \forall p \forall Y$ (CAN_SOLVE (Y,p) $\wedge$ RESTRICTED_NODE (X,Y,p) --> $\neg$GET_SERVICE (X,Y,p))

This rule states that, " If 'Y' can solve the problem 'p' and if 'X' is a restricted node of 'Y' then 'X' will not get service from 'Y' .

RULE 5 :

$\forall X \forall p \forall Z$ ($\neg$CAN_SOLVE (X,p) $\wedge \neg$SOLVER-KNOWN (X,Z,p) --> BROADCAST (X,Z,p))

This rule states that, " If 'X' cannot solve 'p' and if 'X' does not know of any node, on the network, which can solve 'p' then 'X' will broadcast for service to all the nodes ."

RULE 6 :

$\forall X \forall p \forall Z$ (BROADCAST (X,Z,p) $\wedge \neg$RESTRICTED_NODE (X,Z,p) --> GET_SERVICE (X,Z,p))

This rule states that, " If 'X' has broadcast for service to all the nodes on the network and if 'X' is not a restricted node of

any node that can solve 'p', then 'X' will get service."

RULE 7 :

$\forall x \forall p \forall z$ (CAN-SOLVE (Z,p) $\wedge \neg$RESTRICTED_NODE (X,Z,p) --> $\neg$GET_SERVICE (X,Z,p))

This rule states that, " If 'p' can be solved by any node on the network and if 'X' is restricted node of any node that can solve 'p', then 'X' will not get service."

6.2 CONVERSION TO CLAUSE FORM

In order to carry out the Resolution procedure the rules expressed in Predicate Logic, in the Previous section, must be converted to clause form. The conversion process consists of the following steps :

(1) Eliminate the Implication '--->' symbols, using the fact that a ---> b = $\neg$a $\vee$ b

(2) Reduce the scope of '$\neg$' , using

- the fact that $\neg(\neg p) = p$
- deMorgan's laws that

$\neg(a \wedge b) = \neg a \vee \neg b$

$\neg(a \vee b) = \neg a \vee \neg b$

- and the standard correspondence between quantifiers

$\neg \forall x \; P(x) = \exists x \neg P(x)$

$\neg \exists x \; P(x) = \forall x \neg P(x)$

(3) Standardise variables so that each quantifier binds a unique variable

(4) Eliminate existential quantifiers. A formula that contains an existentially quantified variable asserts that there is a value that can be substituted for the variable that makes the formula true.The quantifier is eliminated by substituting for the variable a reference to a function that produces the desired value.

If existential quantifiers occur within the scope of the universal quantifiers, then the value that satisfy the predicate may depend on the values ofthe universally quantified variables. For example, in the the formula

$\forall x \exists y$ REQUEST_SERVICE(x,y)

the value of y that satisfies REQUEST_SERVICE depends on the particulr value of x. Thus we must generate functions with the same number of arguements as the number of universal quantifiers in whose scope the expression occurs. So this example would be transformed into

$\forall x$ REQUEST_SERVICE(Φ(x),x)

(5) Convert to Prenex Form. At this stage, there are no remaining existential quantifiers and each universal

quantifier has its own variable. All the quantifiers are moved to the left of the formula.

(6) Eliminate universal quantifiers. At this point all the variables are universally quantified, so the prefix can be dropped.

(7) Remove the parentheses to give us conjunction of disjuncts, this enables us to exploit the associative property of OR

With the help of the above steps, the rules described in the previous section are converted into clause form as follows :

RULE 1

$\forall X \forall p$ (CAN_SOLVE(X,p) --> SOLVED (X,p)

Eliminate '-->' using a --> b = $\neg a \vee b$

Note : For convenience abbreviations will be used for the Predicates, so CAN_SOLVE = CS and SOLVED = S

$\forall X \forall p$ ($\neg$CS(X,p) $\vee$ S(X,p))

Removing the Universal Quantifiers we get conjunction of disjuncts

$\neg$ CS(X,p) $\vee$ S(X,p)

RULE 2

The following abbreviations are used

CS = CAN_SOLVED

S = SOLVED

SK = SOLVER_KNOWN

RS = REQUEST_SERVICE

$\forall X \forall p \exists Y \; (\neg CS(X,p) \wedge SK(X,Y,p) \dashrightarrow RS(X,Y,p))$

Remove '-->' using $a \dashrightarrow b = \neg a \vee b$ and $\neg(a \wedge b) = \neg a \vee \neg b$

$\forall X \forall p \exists Y \; \neg(\neg CS(X,p) \wedge SK(X,Y,p)) \vee RS(X,Y,p)$

$\forall X \forall p \exists Y \; (CS(X,p) \vee \neg SK(X,Y,p) \vee RS(X,Y,p))$

Removing Existential Quantifiers (Skolemisation)

$\forall X \forall p \; (CS(X,p) \vee \neg SK(X,\Phi(X,p) \vee RS(X,\Phi(X,p),p))$

Removing the Universal Quantifier we get

$CS(X,p) \vee \neg SK(X,\Phi(X,p),p) \vee RS(X,\Phi(X,p),p)$

RULE 3

The abbreviations in addition to those used in the above rule are

RN = RESTRICTED_NODE

GS = GET_SERVICE

$\forall X \forall p \forall Y \; ((RS(X,Y,p) \wedge \neg RN(X,Y,p)) \dashrightarrow GS(X,Y,p))$

Eliminate '-->' using $a \dashrightarrow b = \neg a \vee b$

$\forall X \forall p \forall Y \; \neg((RS(X,Y,p) \wedge \neg RN(X,Y,p)) \vee GS(X,Y,p))$

Using $\neg(a \wedge b) = \neg a \vee \neg b$

$\forall X \forall p \forall Y \; (RS(X,Y,p) \vee RN(X,Y,p) \vee GS(X,Y,p)$

Removing Universal quantifiers we get

$\neg RS(X,Y,p) \vee RN(X,Y,p) \vee GS(X,Y,p)$

RULE 4

$\forall X \forall p \forall Y$ (CS(Y,p) $\wedge$ RN(X,Y,p) --> $\neg$ GS(X,Y,p))

Eliminate '-->'

$\forall X \forall p \forall Y \neg (\neg (CS(Y,p) \wedge RN(X,Y,p)) \vee \neg GS(X,Y,p))$

Using $\neg\neg(a \wedge b) = \neg a \vee \neg b$

$\forall X \forall p \forall Y (\neg(CS(Y,p) \vee \neg RN(X,Y,p) \vee \neg GS(X,Y,p))$

Removing Universal Quantifiers we get

$\neg CS(Y,p) \vee \neg RN(X,Y,p) \vee \neg GS(X,Y,p)$

RULE 5

B = BROADCAST

$\forall X \forall p \forall Z$ ($\neg SK(X,Z,p)$ --> B(X,Z,p))

Removing '-->'

$\forall X \forall p \forall Z$ (SK(X,Z,p) $\vee$ B(X,Z,p))

Removing the Universal quantifiers we get

SK(X,Z,p) $\vee$ B(X,Z,p)

RULE 6

$\forall X \forall p \forall Z$ (B(X,Z,p) $\wedge \neg$ RN(X,Z,p) --> GS(X,Z,p))

Eliminating '-->'

$\forall X \forall p \forall Z$ ($\neg$ B(X,Z,p) $\vee$ GS(X,Z,p)) $\vee$ RN(X,Z,p))

Removing the Universal quantifiers we get

$\neg$ B(X,Z,p) $\vee$ RN(X,Z,p) $\vee$ GS(X,Z,p)

RULE 7

$\forall X \forall p \forall Z$ (CS(Z,p) $\wedge$ RN(X,Z,p) $\rightarrow \neg$ GS(X,Z,p))

Eliminate '-->'

$\forall X \forall p \forall Z$(CS(Z,p) $\wedge$ RN(X,Z,p) $\vee \neg$ GS(X,Z,p)

Removing the Universal quantifiers

$\neg$ CS(X,Z,p) $\vee \neg$ RN(X,Z,p) $\vee \neg$ GS(X,Z,p))

6.3 RESOLUTION PROCEDURE

The rules converted into clause form, shown below, can now be used in the Resolution procedure.

Rule 1 : $\neg$CS(X,p) $\vee$ S(X,p)

Rule 2 : CS(X,p) $\vee \neg$SK(X,$\overline{\text{e}}$(X,p),p) $\vee$ RS(X,$\overline{\text{e}}$(X,p),p))

Rule 3 : $\neg$ RS(X,Y,p) $\vee$ RN(X,Y,p) $\vee$ GS(X,Y,p)

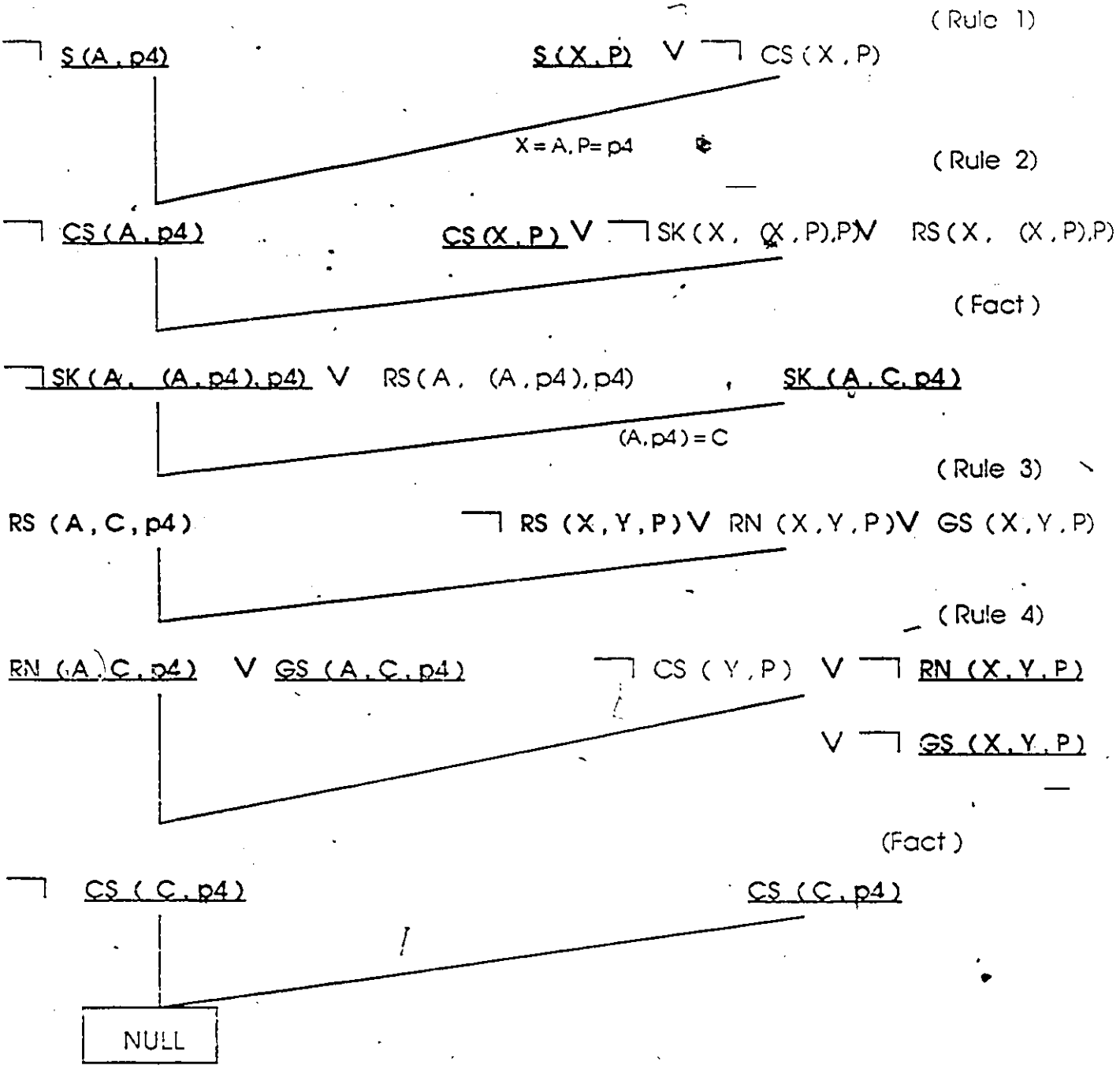
Rule 4 : $\neg$ CS(X,Y,p) $\vee \neg$ RN((X,Y,p) $\vee \neg$GS(X,Y,p)

Rule 5 : SK(X,Z,p) $\vee$ B(X,Z,p)

Rule 6 : $\neg$ B(X,Z,p) $\vee$ RN(X,Z,p) $\vee$ GS(X,Z,p)

Rule 7 : $\neg$ CS(X,Z,p) $\vee \neg$RN(X,Z,p) $\vee \neg$GS(X,Z,p)

Resolution produces proof by refutation. In other words to prove a statement (i.e., show that it is valid), resolution attempts to show that the negation of the statement produces a contradiction with the known statements (i.e., that it is unsatisfiable). Let us know see how Resolution works with the following examples



GOAL : S(A, p4)

NEGATION : ¬ S(A, p4)

FIGURE 6-1 : PROBLEM TRANSMITTED TO A SPECIFIC NODE

Example 1 - FIGURE 6-1

Node 'A' is given a problem 'p4' to be solved. The goal/objective is to find out whether 'A' can provide the solution for 'p4'. This goal can be achieved by either

- 'A' solving the problem by itself if it has expertise available locally
- or by requesting service from some other node which has the required expertise to solve 'p4'

RESOLUTION PROCEDURE

Goal S(A,p4) - Can A solve p4

In order to prove the above goal we have to prove that its negation is false i.e. $\neg S(A,p4)$ is False.

Various steps involved in this proof are :

Step1

The goal $\neg S(A,p4)$ is matched with Rule 1 which is $S(X,p) \vee \neg CS(X,p)$ The variables 'X' and 'p' in rule 1 are bound (known as unifaction) to 'A' and 'p4' respectively to give us

$S(A,p4) \vee \neg CS(A,p4)$

Now we have $\neg S(A,p4)$ and its compliment $S(A,p4)$ this results in resolution and the result is the resolvent $\neg CS(A,p4)$ which says that 'A' cannot solve 'p4'. If it cannot solve a problem then it has to request service from some other node which has the

expertise to solve 'p4'. Rule 2 deals with such a situation and we shall see what happens in the next step.

Step 2

In this step the resolvent ¬CS(A,p4) from step1 is unified with rule 2 to give

$CS(A,p4) \vee \neg SK(A,\exists(A,p4),p4)\ RS(A,\exists(A,p4),p4)$

Now CS(A,p4) from the above clause resolves with the resolvent ¬CS(A,p4) from step 1 to give us the resolvent

$\neg SK(A,\exists(A,p4),p4) \vee RS\ (A,\exists(A,p4),p4)$

Step3

In the knowledge base we have a fact SK(A,C,p4) please refer to Figure 6-2. This fact states that the node 'A' knows that node 'C' can solve 'p4'. We can use this fact for resolution and we get the resolvent RS(A,C,p4) , which results from resolution of SK(A,C,p4) with ¬SK(A,C,p4) the resolvent from step 2.
The resolvent RS(A,C,p4) states that 'A' requests service from 'C'for solution of 'p4'

Note : (A,p4) in the resolvent from step gets bound to 'C' when matched with the Fact SK(A,C,p4)

Let us summarize what we have seen so far.

(1) Node 'A' was given a problem to be solved

(2) Node 'A' cannot solve the problem but it knows that 'C' can

FIGURE 6 - 2 FACTS

CAN-SOLVE (A , p1)

CAN-SOLVE (A, p2)

CAN-SOLVE (B , p3)

CAN-SOLVE (C , p4)

RESTRICTED-NODE(B , A , p1)

RESTRICTED-NODE (NIL , A , p2)

RESTRICTED-NODE (D , B , p3)

RESTRICTED-NODE (NIL , C , p4)

SOLVERKNOWN (A , C , p4)

SOLVER-KNOWN (B , C , p3)

solve 'p4'

(3) Node 'A' requests service from node 'C'.

Step 4

Resolvent RS(A,C,p4) from step 3 is matched with Rule 3. The variables 'X','p','Y' in rule 3 get bound to A,C,p4 respectively to give us

$\neg RS(A,C,p4) \vee RN(A,C,p4) \vee GS(A,C,p4)$

Resolution occurs in this step due to RS(A,C,p4) and $\neg RS(A,C,p4)$. The Resolvent is $RN(A,C,p4) \vee GS(A,C,p4)$ which states that 'A' is not restricted from receiving services from 'C' and that it will get services from 'C' for solution of 'p4'.

Step 5

When the resolvent from step 3 is matched with Rule 4 the variables X,Y,p in Rule 4 get bound to A,C & p4 respectively

Resolution occurs due to

RN(A,C,p4) and $\neg RN(A,C,p4)$

GS(A,C,p4) and $\neg GS(A,C,p4)$

and the resulting resolvent is $\neg CS(C,p4)$

Step 6

In the Knowledge base we have a Fact CS(C,p4) which states that 'C' can solve 'p4'. This fact resolves with the reslovent

¬CS(C,p4) from step 5 to give us NULL.

So we have proved that ¬(S(A,p4) is False which means S(A,p4) is True.

Therefore the problem 'p4' which was given to the node 'A' was solved by node 'C' on request from.

Example 2 FIGURE 6-3

In this example we want to find out if node 'A' can solve or get problem 'p3' solved.

RESOLUTION PROCEDURE

Goal : S(A,p3)

Goal Negation : ¬S(A,p3)

Step 1

Match negated goal with Rule 1 and we get resolvent ¬CS(A,p3) which states that A cannot solve 'p3'

Step 2

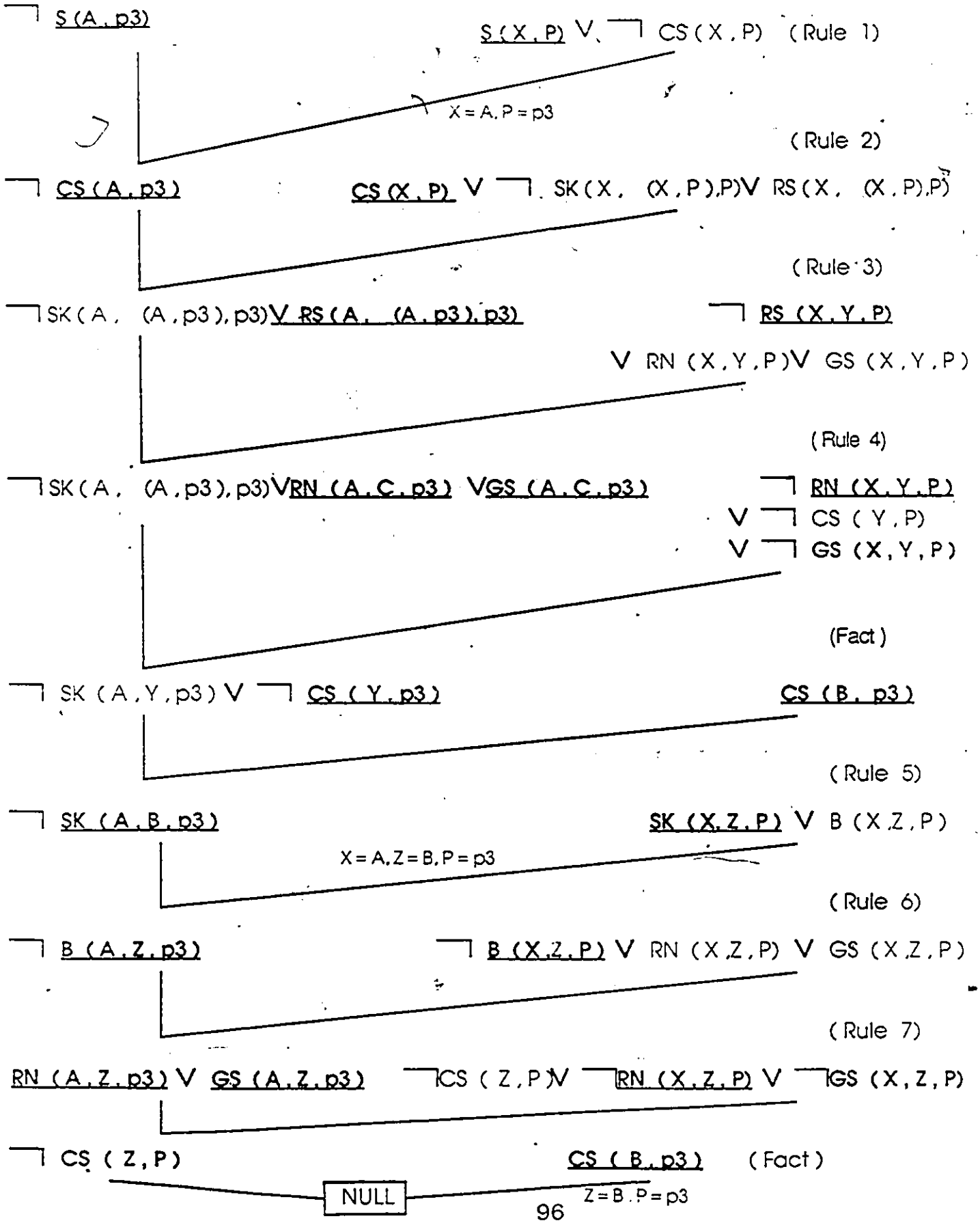
Match resolvent from previous step with Rule 2.

The Resolvent obtained is

¬SK(A,∃(A,p3),p3) ∨ RS(A,∃(A,p3),p3)

Which states that A does not Know who can solve 'p3' , A will have to request for service from any node that can solve 'p3'

FIGURE 6 - 3 : PROBLEM BROADCASTED TO ALL THE NODES



Step 3

Resolvent from Step 2 is matched with Rule 3 to give us

$\neg SK(A,F(A,p3),p3) \vee RN(A,C,p3) \vee GS(A,C,p3)$

Step 4

Rule 4 is matched with resolvent in Step 3 to give us

$\neg SK(A,Y,p3) \vee \neg CS(Y,p3)$

which states that A does not know who can solve 'p3'

Step 5

The Fact CS(B,p3) which says B can solve 'p3' is matched with the resolvent from step 4 to give us

$\neg SK(A,B,p3)$ this says that A does not that B can solve 'p3'

Let us know summarize what has happened so far

(1) Node 'A' was given problem 'p3' to be solved.

(2) Node 'A' does not have the expertise to solve 'p3'

(3) There is Node 'B' which can solve 'p3'

(4) Node 'A' does not Know that 'B' can solve 'p3'. Implying that 'A' must broadcast for service

Step 6

The resolvent $\neg SK(A,B,p3)$ from step 5 is matched with Rule 5 to give us $\neg B(A,Z,p3)$

Step 7

Resolution of ¬B(A,Z,p3) with Rule 6 gives us

RN(A,Z,p3) ∨ GS(A,Z,p3)

which says that if 'A' is not restricted from receiving services from a node that can solve 'p3' then 'A' will get service

Step 8

Resolution of the resolvent from step 7 with Rule 7 gives

¬CS(Z,x)

Step 9

There is Fact in the knowledge base which states that node 'B' can solve 'p3' i.e., CS(B,p3).

Resolving this fact with resolvent in step 8 gives us NULL.

So we have proved the negation of goal S(A,p3) is False.

Therefore the goal S(A,p3) is true.

CHAPTER 7

AN APPLICATION

Expertnet was implented in Common Lisp and simulated on an IBM PC. An office function performed by network engineering department of an organization, marketing packet switched networks, was chosen as an application.

The function of the network engineering department is to configure packet switches based on the information provided by a customer. The customer information normally consists of terminal population, type of terminals, distribution of terminals and communication line speeds, packet (data units) length, etc.

Packet switches allow terminals of different types (produced by different vendors) to share common communication facilities. In other words there is no requirement for dedicated facilities for each type of device. The terminals can communicate with each other or talk to different hosts over the packet switched network

The network engineer after configuring the switches based on customer input,has the task of costing the equipment. The cost information is maintained by pricing group in the marketing department. The cost information is obtained from the pricing group and the total cost for the equipment is prepared and the

information is sent out to the customer as a bid.

Inorder to illustrate the working of Expertnet in the above environment, lets assume node 'A' represents the workstation of of a network engineer and node 'B' the workstation of a pricing specialist.

As described in Chapter 5.0 , each workstation/processing node consists of a Planner, Meta Knowledge Base (KB), Process Server, Message Server, Response and Request Handlers and Knowledge Source/s (KS) . The KSs contain domain specific knowledge. Hence the KSs in node 'A' will be different from those in node 'B'.

On receiving the request for proposal from the customer, the network engineer prepares the input information required for configuration. He now gives to node 'A' the problem description SWITCH-CONFIG and the input data.

The Planner checks in the Meta KB to see if there is a frame with problem ID as SWITCH-CONFIG. It finds a frame with problem ID CONFIG-SWITCH . This frame is now retrieved (copy of the frame is shown in Figure 7-1 A) . The first slot of the Meta Frame contains the problem ID.

The process name which is KS-CONFIG ,is retrieved (by Planner) from the next slot in the frame and handed over to Process Server along with the input data. THe Process Server invokes KS-CONFIG and passes to it the input data as the arguement.

KS-CONFIG acts as the Task Manager. It decomposes the task into sub problems. In doing so it invokes the knowledge sources KS-X25 and KS-ITI. These KSs contain domain specific knowledge and inference mechanism to solve the configuration problem. The domain knowledge is represented in frames. The knowledge base contains sufficient information to solve the problem . In other words if the user does not provide some information such as line utilization, packet length, etc. , the KS will use the default values stored in the knowledge base.

KS-X25 produces equipment list (number of line processors, trunk processors, memmory boards, etc.) due to X25 lines. On the other hand KS-ITI produces equipment list associated with ITI lines. KS-CONFIG combines the two lists and gives it to the user of node 'A'.

Once the equipment list is obtained, the network engineer has to assign cost to the equipment. So he now gives to node 'A' the equipment list as input data and COST-INFO as problem ID. The Planner checks in the Meta KB and retrieves a frame (Figure 7-1B) with problem ID as COST-INFO. The second slot of the frame is checked for process name. This slot has a nil value meaning node 'A' cannot solve this problem. So the next slot is checked to find out who can solve the problem. This slot has the value 'Node B' specified in it. Now the Planner passes the node ID, problem name and the input data to the Message Server. A message is created with the following information : originating and

destination node IDs, problem ID, Request type (meaning it is a message of type request) and input information (equipment list). The completed message is now sent to node 'B' .

On receiving the message, the Message Server in node 'B' checks if the message received is request type or response type. Since it is request type , it is handed over to Request Handler who checks in the meta KB to see - if requested problem can be solved by 'B' (frame retrieved from meta KB is shown in Figure 7-1 C). This is done by looking at slot of the frame. Since this slot contains a non nil value i.e. KS-COSTING, it implies that node 'B' can solve the problem COST-INFO.

The process name KS-COSTING is handed over to Process Server , who in turn invokes KS-COSTING and passes to it the equipment list as arguement.

The result produced by KS-COSTING is handed over to Response Handler. A message of type Response is created and send over to node 'A'.

The message received in node 'A' by Message Server is given to Response Handler who gives the result to the user. This describes the application of Expertnet to an office environment.

CHAPTER 8

CONCLUSION

In this thesis a technique, called EXPERTNET, for resource sharing among a network of workstations was discussed. Through this technique the following benefits can be realized:

(1) Workstations can request for services, not available locally from other workstations on the same network.

(2) Utilize resources of idle workstations on the network.

(3) Update local workstation knowledge base, on an on-going basis, about the expertise and capabilities of other workstations.

In the Expertnet, communication between workstations is carried out with the help of two types of messages, question-type and answer-type. All the requests are treated as question-type messages. And all the responses are considered as answer-type messages.

Broadcast transmissions and directed transmissions are used based on the intention of the host workstation and its knowledge about the capabilities of other workstations. When the capabilities of the other workstations are known, then the host workstation

selects the node with the best match and chooses a directed transmission mechanism. Otherwise, a broadcast mechanism is used. Broadcast communication also serves as a method to collect information about other nodes on the network.

HEARSAY-II system and Contract Net are suitable for real time applications such as air trafific control, robotics, speech understanding, missile tracking, remote sensing, etc. On the other hand in the Expertnet, the emphasis is not so much on the solution of a problem in real time. The emphasis is more on the effective utilization of idle resources (workstations) in an office environment. As well as requesting of services, not available locally, from other workstations.

Limitations of the Proposed Technique

The system does not have the capability to interact with the user. For example, the system expects the users to know all the problem IDs as specified in the Meta Knowledge Base. In the event of a user specifying a problem name different from the one known in the knowledge base, the system cannot introgate the user to find out if he meant this instead of that. In order to implement such a capability one would require the use of natural language processing technique or some form of directory which associates a particular problem ID with various possible/probable IDs.

Expertnet uses Broadcast mechanism for the purpose of

requesting services when the requesting node does not know which particular node can solve a problem. This has the potential of wasting computing resources of certain workstations. For example, Node 'A' which has a problem 'X' to be solved would broadcast to all the nodes belonging to the same group (as Node 'A') if it does not know of any node that can solve 'X'. If for instance nodes 'C' and 'D' have the expertise to solve the problem 'X', both would attempt to solve the problem. If suppose 'C' solves it earlier than 'D', then 'A' would receive the solution from 'C' and reject the solution of 'D' received later. As result of this the computing resources of 'D' were wasted. A mechanism which would enable 'A' to choose between the nodes 'C' and 'D' (such as bid mechanism used in Contract Net) or a process by which the two nodes (C and D) can agree between themselves as to who would solve the problem, could prevent wasting of computing resources.

The Expertnet does not provide the requesting node the capability to include a time parameter in the request message. If this facility is available then the node does not have to wait for a response after a time period specified in the message. On expiry of this period it can request services from another node. Implementing this facility will require time synchronization on all the nodes .

The proposed technique can decompose a problem into sub problems and distribute them to knowledge sources within the same node.But it does not have the capability to distribute the sub problems

among different nodes. This ability will enhance the usage of Expertnet in office environment. For example, there may be certain problems which require to be decomposed into various sub problems. The host node may not have the expertise to solve all the sub problems. In such an event, the node will have to request services from some other node, for the solution of the sub problem for which expertise is not available locally. Such a capability will increase the problem solving capabilities of a processing node.

Further research work is required in the areas identified above. The Expertnet provides basic framework which can be used for building various enhanced capabilities.

REFERENCES

(1) BAER73 : BAER J.L. , " A survey of some theoretical aspects of multiprocessing. " Computing Surveys 5 (1) , 1973 , pp 31-80.

(2) BONELL83: BONELL RONALD D., HUHNS MICHAEL N., STEPHEN LARRY M., "Control and Co-operation in Distributed Expert Systems" , IEEE SOUTH EAST CON '83 Conference Proceedings, pp 241-245 , April 1983.

(3) CHAND81 : CHANDRESEKARAN B., " Natural and Social System Metaphors for Distributed Problem Solving ": IEEE Transactions on Systems, Man and Cybernetics , VOLUME SMC-11 NO.1 , pp 1-5 , JAN 1981 .

(4) DAVIS83 : DAVIS RANDALL , SMITH G.R. , " Negotiation as a Metaphor for Distributed Problem Solving ", Artificial Intelligence , Vol. 20 , No. 1 , pp 63-109, JAN 1983.

(5) DAVIS81 : DAVIS R. , SMITH G.R. , " Frameworks for Co-operation in Distributed Problem Solving " , IEEE Trahsactions on Systems , Man and Cybernetics, Vol. SMC-11 , No.1 , pp 61-70 , JAN. 1981 .

(6) ERMAN80 : LESSER V.R., ERMAN L.D., "Disrtibuted Interpretation : A Model and Experiment " , IEEE Transactions on Computers, Vol. C-29 , No. 12, pp 1144-1163 ,December 1980.

(7) FEIGENBAUM78 : FEIGENBAUM E.A., Nii H.P., " Rule-Based Understanding of signals in Pattern-Directed Inference Systems, WATERMAN D.A. and HAYES-ROTH F. Eds New York: Academic Press, 1978, pp 483-502.

(8) GESCH80 : GESCHWIND N., "Neurological Knowledge and Complex Behaviors " Cognitive Science, Vol. 4, No. 2, pp 185-193, 1980.

(9) GEVINS83 : GEVINS A.S., " Overview of Human Brain as Distributed Computing Network"

(10) HANSON78 : , HANSON A.R., RISEMAN E.M., " Visions: A Computer System for interpreting scenes," Computer Vision System, New York, Academic Press, pp 303-333 , 1978 .

(11) HAYES-ROTH77 : HAYES-ROTH F., " Focus of Attention in Hearsay II Speech Understanding System ", Proceedings 5th IJCAI, pp 27-35, 1977 .

(12) HAYES-ROTH80 : HAYES-ROTH F., ERMAN L.D., LESSER V.R., " Hearsay II Speech Understanding System : Integrating Knowledge Vol. 8, pp 323-364, 1977.

(13) HEWITT77 : HEWITT C. , "Viewing Control Structures as Patterns for Passing Messages ," Artificial Intelligence , Vol. 8, pp. 323 - 364 , 1977.

(14) HO86 : HO C.S., HONG Y.C., KUO T.S., "A Society Model for Office Information Systems ", ACM Transactions on Office Information Systems, Vol. 4, No. 2, April 86.

(15) LESSER80 : LESSER V.R., ERMAN L.D., "Distributed Interpretation : A Model and Experiment", IEEE Transactions on Computers, VOL. C-29 , NO. 12, DEC. 1980.

(16) LESSER81 : LESSER V.R., CORKILL D. D.," Functionally Accurate Co-operative Distributed Systems," IEEE Transactions on Systems,Man and Cybernetics, SMC-11 (1),pp 61-70, Jan 1981.

(17) LESSER82A: LESSER V.R., CORKILL D. D., HUDLICKA E., "Unifying Data-directed and Goal-directed Control : An Example and Experiments," Proceedings of the 2nd National Conference on AI, pp 143-147, August 1982 .

(18) LESSER82B : LESSER V.R. et al, "A High Level Simulation Test Bed for Co-operative Distributed Problem Solving, "Proceedings of 3rd International Conference on Distributed Computing , pp 341-350 , October 1982 .

(19) LESSER83 : LESSER V., CORKILL D. D., " The Use Of Meta Level for Co-ordination in a Distributed Problem Solving Network, Proceedings of the 8th IJCAI, Karlsruhe , West Germany, Vol.2 , pp 748-756 , August 1983.

(20) LOCHOVSKY86 : LOCHOVSKY F.H., WOO C.C., " Supporting

Distributed Office Problem Solving in Organizations " ACM Transactions on Office Information Systems, Vol. 4, No. 3, July 1986

(21) MALONE87 : MALONE T.W., GRANT K.R., " Intelligent Information Sharing Systems " , Communications of the ACM, Vol. 30, No. 5, May 1987

(22) NARAIN84 : CAMMARATA S., NARAIN S., "Distributed Intelligence for RPV Fleet Control ", Proceedings IEEE EASTCON '84, 17th Annual Electronics and Aerospace Conference , pp 233-236, September 1984 .

(23) Nii86 : Nii H.P., " Black Board Systems from Knowledge Engineering Perspective " , The AI Magazine, pp 82-103, August 1986.

(24) NILSSON80 : NILSSON N., Principals of Artificial Intelligence , Tioga Publishing Company, Palo Alto, California , 1980 .

(25) RICH83 : RICH E., " Artificial Intelligence ", McGRAW-HILL SERIES IN ARTIFICIAL INTELLIGENCE, 1983

(26) SMITH80 : SMITH G.R. , "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver," IEEE Transactions on Computers, Vol. C-29, pp 1104-1113, December 1980 .

(27) STEEB82 : STEEB R. , McARTHUR D. , CAMMARATA S. , " A Framework for Distributed Problem Solving ", Proceedings of the AAAI-82 , pp 181-184 , 1982.

(28) STEEB83 : STEEB R., McARTHUR D., CAMMARATA S., Proceedings of the Eighth International Joint Conference on Artificial Intelligence , Karlshruhe , West Germany , Vol. 2, pp 767-770 , August 1983.

(29) STEELE84 : STEELE JR. GUY L. , Common Lisp The Language, Digital Press ,1984.

(30) WINSTON84A : WINSTON P. H. , Lisp ,Second Edition, Addison Wesley Publishing Company , 1984.

(31) WINSTON84B : WINSTON P. H., Artificial Intelligence ,Second Edition, Addison Wesley Publishing Company, Massachusetts, 1984 .

L I S P P R O G R A M S

```
;*************************************************************************
;* Purpose of this function is to retrieve problem name from a FRAME     *
;*                                                                       *
;*************************************************************************

(defun GET-PROBLEM-NAME (frame)
   (cadr (car (car frame))))


;*************************************************************************
;* Purpose of this function is to retrieve the process name from a       *
;* FRAME                                                                 *
;*************************************************************************

(defun GET-PROC-NAME (frame)
   (cadr (car (cdr (car frame)))))


;*************************************************************************
;* Purpose of this function is to retrieve the list of candidates        *
;* which can solve a particular problem                                  *
;*                                                                       *
;*************************************************************************

(defun GET-CANDIDATE (frame)
  (cadr (car (cdr (cdr (car frame))))))


;*************************************************************************
;* Purpose of this function is to retrieve the list of candidates        *
;* which are restricted from use of a process of the Host Node           *
;*                                                                       *
;*************************************************************************

(defun GET-RCAND (frame)
    (cadr (car (cdr (cdr (cdr (car frame)))))))
```

```
;*************************************************************************
;*                                                                       *
;* Purpose of this function is to add a new FRAME  to the Meta-          *
;* Knowledge Base                                                        *
;*                                                                       *
;*************************************************************************


(defun ADD-MK-FRAME (prb prc cand r-cand)
  (setf prob-list  (list (cons 'prob (list prb))))
  (setf proc-list   (list (cons 'proc (list prc))))
  (setf cand-list  (list (cons 'candidates (list (list cand)))))
  (setf r-cand-list (list (cons 'restr-cand (list (list r-cand)))))
  (setf mk-new-frame (append (append (append prob-list proc-list)
                       cand-list) r-cand-list)) ; Create new mk-frame
  (setq mk-frames  (cons mk-new-frame mk-frames))) ; Add new mk-frame
                                          ; to Meta Knowledge



;*************************************************************************
;*                                                                       *
;* The purpose of this function is to add to the Candidate List the      *
;* Node which can solve a given problem                                  *
;*                                                                       *
;* Functions called by this Function are :                               *
;*  GET-PROC-NAME  - To Retrieve Process Name from the Matched Frame     *
;*  GET-RCAND      - To Retrieve Restricted Candidates from the          *
;*                     Matched Frame                                     *
;*  ADD-MK-FRAME   - To Add the Updated Frame to the Meta-knowledge      *
;*                                                                       *
;*************************************************************************


(defun UPDATE-CAND-LIST (rcvd-msg mk-pointer mk-frames)
   (setf prb (message-task rcvd-msg))
   (setf prc (GET-PROC-NAME (list mk-pointer)))
   (setf cand (message-org-node rcvd-msg))
   (setf r-cand (GET-RCAND (list mk-pointer)))
   (setf mk-frames (delete mk-pointer mk-frames))
   (setf mk-frames (ADD-MK-FRAME prb prc cand r-cand)))
```

```
;*****************************************************************************
;*                                                                           *
;* The purpose of this function is to update the attribute value of          *
;* the Process Slot in a Frame in the Meta Knowledge Base                    *
;*                                                                           *
;*  Functions called by this Function are :                                  *
;*   COMPARE        - To retrieve the Matched Frame from the Meta            *
;*                  - Knowledge Base                                         *
;*   ADD-MK-FRAME   - To add the Updated Frame to Meta Knowledge Base        *
;*                                                                           *
;* DESCRIPTION OF VARIABLES :                                                *
;* prb       - Problem Name                                                  *
;* prc       - Process Name                                                  *
;* cand      - List of Nodes which can solve the Problem                     *
;* r-node    - Node Restricted from receiving service for the solution       *
;*           - of the Problem , from the Host Node                           *
;* mk-pointer - Matched Frame                                                *
;*                                                                           *
;*****************************************************************************


(defun UPDATE-PROC-NAME (prob proc &optional r-cand)
   (cond ((COMPARE prob mk-frames)
             (progn
                (setf prb prob)
                (setf prc proc)
                (setf cand nil)
                (setf r-node r-cand)
                (setf mk-frames (delete mk-pointer mk-frames))
                (setf mk-frames (ADD-MK-FRAME prb prc cand r-node))))
         (t 'nil)))
```

```
;*****************************************************************
;*                                                               *
;* Purpose of the functions DEFSTRUCT and MAKE-MSG is to construct *
;* structered messages                                           *
;*                                                               *
;*  DESCRIPTION OF FIELDS :                                      *
;*   type         - Message of Type Request or Response          *
;*   task         - Problem to be Solved                         *
;*   org-node     - Node Originating the Problem to be Solved or the *
;*                - Solution to the Problem                      *
;*   dest-node    - Node Receiving the Problem or the Solution   *
;*   data         - Data or Input Associated with the Problem    *
;*   result       - Solution of the Problem                      *
;*                                                               *
;*****************************************************************


(DEFSTRUCT (message)
  (type nil)
  (task  nil)
  (org-node nil)
  (dest-node nil)
  (data nil)
  (result nil))



(defun MAKE-MSG (type task  org dest data result)
  (make-message :type type
                :task task
                :org-node org
                :dest-node dest
                :data data
                :result result))
```

```
;*****************************************************************
;*                                                               *
;* Purpose of this function is to plan the manner in which  the given  *
;* problem should be solved                                      *
;*                                                               *
;* Functions called by this Function are :                       *
;*  COMPARE          - To retrieve the Matched from the Meta-Knowledge  *
;*  GET-PROC-NAME    - To obtain the Process Name which can solve the   *
;*                     given Problem                             *
;*  GET-CANDIDATE    - To Retrieve the Node which can solve the given   *
;*                     Problem                                   *
;*  MESSAGE-SERVER   - To take appropriate actions on Message Received  *
;*                     and Message to be Transmitedd or Broadcast     *
;*  RESPONSE-HANDLER - To Handle the Response Received from the other  *
;*                      Node                                     *
;*  REQUEST-HANDLER - To provide appropriate service to Message       *
;*                     Received from other Nodes                 *
;*  MAKE-MESSAGE    - To Construct Message                        *
;*                                                               *
;*  DESCRIPTION OF VARIABLES :                                   *
;*  prob           - problem given to the Host Node              *
;*  user-data      - data given to the Host Node by the user     *
;*  mk-frames      - Meta-Knowledge                              *
;*  mk-pointer     - Matched Frame which contains the problem name    *
;*                   specified by the user or the Requesting  Node    *
;*  proc-name      - the Process Name in the Matched Frame which can  *
;*                   can solve the problem                     *
;*  candidate      - the Node which can solve the problem        *
;*  rcvd-msg       - Message received by the Host Node from other Nodes *
;*  Xmit-Msg       - Message to be Transmited to a specific Node      *
;*  Broadcast-msg  - Message to be Broadcast to other Nodes      *
;*                                                               *
;*****************************************************************

(defun PLANNER (prob mk-frames &optional user-data)
   ;
   ; ** PLANNING FOR THE PROBLEM PROVIDED BY THE USER TO THE HOST NODE *
   ;
  (IF (and prob user-data)
      (progn
       (Cond ((COMPARE prob m)
                 (setf proc-name (GET-PROC-NAME
                                          (list mk-pointer)))))
      ;
      ; Is there a match if yes ,then get the Process Name of the
      ; Matched Frame
      ; IF Process Name found then give it to the Process Server
       (IF proc-name (PROCESS-SERVER proc-name user-data)
```

```
;
; ELSE get the list of candidates ,from the matched frame which ca
; solve the problem
                (progn
                 (setf candidate (GET-CANDIDATE
                                  (list mk-pointer)))
                 (IF candidate
                           (MESSAGE-SERVER (setf xmit-msg
                                           (MAKE-MSG 'request prob
                                            'node1 (car candidate)
                                             user-data nil)))

      ;
      ; IF Candidate Node found then give the Xmit-Msg to the Message
      ; Server to be Transmitted.
      ; ELSE give the Message to the Message-Server to be Broadcast
      ;

                        (MESSAGE-SERVER (setf broadcast-msg
                                              (MAKE-MSG  'request p
                                                'node1 '*  user-dat
                                                 nil)))))))

;
;  ** PLANNING FOR MESSAGE RECEIVED FROM OTHER NODES **
;
; ELSE if it is a Received Message check Destination address of the
; Received Message is either equal to the Host Node or '*'
;
   (cond ((OR (equal 'node1 (message-dest-node rcvd-msg))
              (equal '* (message-dest-node rcvd-msg)))
   ;
   ; Next check if Message is of Type 'RESPONSE' or 'REQUEST'
   ;
         (cond ((equal 'response (message-type rcvd-msg))
                   (RESPONSE-HANDLER rcvd-msg mk-frames))
   ;
   ; If Message is of type 'RESPONSE' then give the Received Message
   ; to RESPONSE-HANDLER for Update Meta Knowledge
   ;
               (t (equal 'request (message-type rcvd-msg))
                     (REQUEST-HANDLER rcvd-msg mk-frames))))
        (t ( nil)))))
   ; If Message is of type 'REQUEST' then give the Message to
   ; to Request-Handler to check whether the problem can be solved by
   ; the Host Node or not .
   ;
```

```
;*************************************************************************
;*                                                                       *
;* Purpose of this function is to check if the user problem  is in the   *
;* Meta-Knowledge base. If it is present then the Matched Frame is       *
;* is returned                                                           *
;*                                                                       *
;*  Function called by this function is :                                *
;*   GET-PROBLEM-NAME   -  To retrieve the Problem name from the         *
;*                         Matched Frame                                 *
;*                                                                       *
;*  DESCRIPTION OF VARIABLES :                                           *
;*  mk             - Meta-Knowledge                                      *
;*  mk-pointer     - Matched Frame                                       *
;*  prob           - Problem to be Solved                                *
;*************************************************************************


(defun COMPARE (prob mk)
  (setq mk-pointer (car mk))
  (cond ((null mk) 'prob-not-found) ; if end of meta-knowledge base
        ((equal prob (GET-PROBLEM-NAME (list mk-pointer))) mk-pointer)
         ; if user-input matches the problem name then return the
          ; matched frame
        (t (COMPARE prob (cdr mk))))))  ; recurse
```

```
;*****************************************************************
;*                                                               *
;* Purpose of this function is to Retrieve the MK Frame which has the *
;* problem specified in the Received Message and Include in the Host  *
;* Node candidate List the Message sending Node                  *
;*                                                               *
;*  Functions called by this function are :                      *
;*   COMPARE        - To Retrieve the Frame from the Meta-Knowledge  *
;*                    which contains the problem specified in the  *
;*                    Message Received from other Node             *
;*   UPDATE-CAND-LIST - To Update the Candidate List of the Matched  *
;*                      Frame in Meta-Knowledge                    *
;*                                                               *
;*****************************************************************


(defun RESPONSE-HANDLER (rcvd-msg mk-frames)
     (cond ((COMPARE (message-task rcvd-msg) mk-frames) ; Retrieve the M
              ; Frame which matches the Task specified in the Received M
              ;
            (UPDATE-CAND-LIST rcvd-msg mk-pointer mk-frames))) ; Includ
              ; Node form which the Message has been received in the
              ; Candidate List of the Host Node
```

```
;*****************************************************************************
;*                                                                           *
;* Purpose of this function is to provide appropriate Service to             *
;* the Message received from other Nodes                                     *
;*                                                                           *
;* The functions called by this function are :                               *
;*  COMPARE        - to check if Task Reqested is in the Meta-Knowledge      *
;*  GET-RCAND      - to check if the Requesting Node is in the Host          *
;*                   Node's Restricted Candidate List                        *
;*  GET-PROC-NAME - To retrieve the Process Name from the Matched            *
;*                   Frame                                                   *
;*  PROCESS-SERVER - To apply the appropriate Process on the Data            *
;*                                                                           *
;*  DESCRIPTION OF VARIABLES  :                                              *
;*  rcvd-msg           - Message Received from other Node                    *
;*  message-task       - Problem to be Solved for the Requesting Node        *
;*  message-org-node   - Message Originating Node                            *
;*  mk-pointer         - Matched Frame                                       *
;*  proc-name          - Process which can Solve the Problem                 *
;*                                                                           *
;*****************************************************************************


(defun REQUEST-HANDLER (rcvd-msg mk-frames)
    (cond ((COMPARE (message-task rcvd-msg) mk-frames)
              ; Get the Matched Frame from the Meta Knowledge
            (cond ((member (message-org-node rcvd-msg)
                       (GET-RCAND mk-pointer)) nil) ; If the Message
               ; Sending Node is in the Restricted Candidate List of the
               ; Host Node then Discard the message
                  (t (progn
                       (setf proc-name (GET-PROC-NAME (list mk-pointer)))
               ; If not get the Process name from the Matched Frame and g
               ; to Process-Server along with Data in Received Message
                       (cond ((not (equal nil proc-name))
                         (PROCESS-SERVER proc-name rcvd-msg))
                            (t ( (problem-cannot-be-solved-by-host)))))
             ; IF Process name not found in the Matched Frame then Disca
             ; Message since the Host node cannot solve the  problem
```

```
;*****************************************************************
;*                                                               *
;* Purpose of this function is to take appropriate actions on the *
;* Messages Received and Messages to be Transmitted.             *
;*                                                               *
;* Functions called by this Function  are :                      *
;*  PLANNER     - To Determine appropriate action to be taken    *
;*  SENDER      - To Transmit Message to Specific Node           *
;*  BROADCAST   - To Broadcast Message to all other Nodes        *
;*                                                               *
;*****************************************************************


(defun MESSAGE-SERVER (message)
 (Progn
  (cond ((and (equal (message-type message) 'request)
              (or (equal (message-dest-node message) host-node)
                  (and (equal (message-dest-node message) '* )
                       (not (equal (message-org-node message) host-node)
                       (PLANNER message mk-frames))
        (t 'nil))
    ;
    ;  If Message is of type Request and the Destination Node of the
    ; of the Message is either the Host node or '* then give it to
    ; the Planner for further action
    ;
  (cond ((and (equal (message-type message) 'response)
              (equal (message-dest-node message) 'node1)
                 (PLANNER message mk-frames)))
        (t 'nil))
    ;
    ; If Message is of type Response and the Destination Node of the
    ; Message is Host node then give it to the Planner for further actio

  (IF (and (or (equal (message-type message) 'response )
                  (equal (message-type message) 'request ))
              (and (not (equal (message-dest-node message) host-node))
                   (not (equal (message-dest-node message) '*))))
                 (SENDER message)
    ;
    ; If Message is either Response or Request Type and is meant for
    ; other Node then give the Message to the function SENDER
    ;
              (cond ((and (equal (message-type message) 'request)
                          (equal (message-dest-node message) '*))
                 (BROADCAST message))))))
   ; ELSE if the Message is of type Request and meant for all other Node
   ; then give the Message to the function Broadcast
```

```
;*************************************************************************
;*                                                                       *
;* Purpose of this function is to retrieve the proper Process/Function   *
;* from the Process-List and apply it to the Data.                       *
;* If the Data is from the Host Node then the Result is returned to      *
;* to the user on the Host Node.                                         *
;* If the Data is from a Message received then the Result is send to     *
;* the Requesting Node with the help of MESSAGE-SERVER function          *
;*                                                                       *
;*  Functions called by this Function are :                              *
;*   MATCH-PROC     -  To Check if the Process which can solve the       *
;*                     Problem is found in the Process List              *
;*   MESSAGE-SERVER - For appropriate action on Received Messages and    *
;*                     Messages to be Transmitted                        *
;*   MAKE-MSG       - To Construct Messages                              *
;*                                                                       *
;*************************************************************************


(defun PROCESS-SERVER (proc-name data )
  (cond ((MATCH-PROC proc-name process-list) proc-name))
        ; Check if the Process Name is in the Process List
          (IF (and proc-name (listp data))
              (progn
                (setf result (FUNCALL proc-name data))
                (HERE IS THE RESULT ===> ,result))
        ; Apply the Process to the user data and
               (progn
                    (setf result (FUNCALL proc-name (message-data rcvd-ms
        ; Apply the Process to the Data in the Received Message. Place th
        ; result in the Message.Change the originating Node to Destinatio
        ; Node and give the Message to the Blackboard-Server to be Xmited
        ;
                    (MESSAGE-SERVER (setf xmit-msg
                                          (MAKE-MSG 'response
                                              (message-task rcvd-msg) host
                                               (message-org-node rcvd-msg)
                                               nil result))))))
```

```
;*****************************************************************
;* Purpose of this function is to check if a given Process Name  *
;* is in the Process List                                        *
;*****************************************************************

(defun MATCH-PROC (proc-name process-list)
      (cond ((null process-list) nil) ; End of Process list ?
            ((equal proc-name (car process-list)) proc-name)
        ; Check if Process name is found the Process list
            (t (MATCH-PROC proc-name (cdr process-list))))) ; Recurse
```

```
;*************************************************************
;*                                                           *
;* Purpose of the functions PUSH-THROUGH-FRAME and FOLLOW-PATH *
;* is to push through the frame while searching for slot  and  *
;* facets                                                    *
;*                                                           *
;*************************************************************


(defun FOLLOW-PATH (path a-list)
   (cond ((null path) a-list)
         (t (FOLLOW-PATH (cdr path) (PUSH-THROUGH-FRAME (car path) a-lis


(defun PUSH-THROUGH-FRAME (key a-list)
   (cond ((assoc key (cdr a-list)))
         (t (cadr (rplacd (last a-list) (list (list key)))))))


;*************************************************************
;*                                                           *
;* Purpose of this function is to information. It requires the  *
;* user to to supply an access path consisting of a frame, a slot*
;* and a facet                                               *
;*                                                           *
;*************************************************************


(defun GET-FRAME (frame slot facet)
       (cdr (assoc facet (cdr (assoc slot (cdr (get frame 'frame)))))))
```

```
;*****************************************************************
;*                                                               *
;* Purpose of the function ADD-FRAME is to add new information *
;* existing frame . If a frame does not exist then it uses      *
;* function FGET-FRAME to add a new frame                        *
;*                                                               *
;*****************************************************************

(defun ADD-FRAME (frame slot facet value)
    (let ((value-list (FOLLOW-PATH (list slot facet)
                                   (FGET-FRAME FRAME))))
      (cond ((member value value-list) nil)
            (t (rplacd (last value-list) (list value))
               value))))

(defun FGET-FRAME (frame)
   (cond ((get frame 'frame))
         (t (setf (get frame 'frame) (list frame)))))
```

```
;*****************************************************************
;*                                                               *
;* Purpose of this function is to calculate throughput requirements*
;* and Line Processor Utilization for ITI lines. From this is    *
;* calculated the number of line processors required             *
;*                                                               *
;*****************************************************************


(defun KS-ITI (input)
   (setq iti-speed (ITI-LINE-SPEED input)) ; Retrieve line speed from us
                                           ; provided input
   (setq iti-lines (ITI-LINES input)) ; Number of lines from user input
                                      ;
 (cond ((equal (ITI-LINE-UTIL input) nil)
         (setq iti-line-utils (car (GET-FRAME 'ITI 'line-util 'value)))
             ; If line utilization not provided by user then retrieve i
             ; from knowledge base.
             ; Else use the input provided by the user
       (t (setq iti-line-utils (ITI-LINE-UTIL  input))))
             ;
 (cond ((equal (ITI-PAC-SIZE input) nil)
         (setq iti-pac-len (car (GET-FRAME 'ITI 'pac-len 'value))))
             ; If packet size not provided by the user then retrieve in
             ; from knowledge base .
             ; Else use the user input.
       (t (setq iti-pac-len (ITI-PAC-SIZE input))))
             ;
    (setq iti-op-time (car (GET-FRAME 'ITI 'over-proc-time 'value)))
    (setq iti-pp-time (car (GET-FRAME 'ITI 'pac-proc-time 'value)))
    (setq iti-pac-over (car (GET-FRAME 'ITI 'pac-overhead 'value)))
             ; throughput per ITI line
    (setq iti-thruput (* 2 (/ (* iti-speed (/ iti-line-utils 100))
                              ( * iti-pac-over iti-pac-len))))
             ; total throughput for all ITI lines
    (setq tot-iti-thruput (* ITI-THRUPUT iti-lines))
    (setq iti-tot-pac-proc-time ( + iti-op-time (* iti-pp-time
                                                   iti-pac-len)))
              ; calculate total line processor capacity
    (setq tot-HSLP-cap-iti ( / 1000 iti-tot-pac-proc-time))
              ; calculate the number of line processors required
    (setq num-of-HSLP-iti (/ tot-iti-thruput tot-HSLP-cap-iti)))
```

```
;*****************************************************************
;*                                                               *
;* Purpose of this function is to calculate throughput requirements*
;* and Line Processor Utilization for X.25 Lines. From this is   *
;* calculated the number of processors required.                 *
;*                                                               *
;*****************************************************************


(defun KS-X25 (input)
   (setq x25-speed (X-25-LINE-SPEED input)) ; Retrieve the line speed fr
                                            ; user input
   (setq x25-lines (X-25-LINES input))      ; Retrieve the number of lin
                                            ; from user input
                                            ;
  (cond ((equal (X-25-LINE-UTIL input) nil)
           (setq x25-line-util (car (GET-FRAME 'X-25 'line-util 'value)
                ; If line utilization not provided by user then Retriev
                ; from the knowledge base .
                ; Else use the user provided data
        (t ( setq x25-line-util (X-25-LINE-UTIL input))))
                ;
  (cond ((equal (X-25-PAC-SIZE input) nil)
         (setq x25-pac-len (car (GET-FRAME 'X-25 'pac-len 'value))))
                ; If packet size not provided by the user then retrieve
                ; from knowledge base.
                ; Else use the user provided data
      (t ( setq x25-pac-len (X-25-PAC-SIZE input))))
                ;
   (setq x25-op-time (car (GET-FRAME 'X-25 'over-proc-time 'value))); Re
             ; overhead processing time from knowledge base
   (setq x25-pp-time (car (GET-FRAME 'X-25 'pac-proc-time 'value))); Ret
             ; packet processing time from KB
   (setq x25-pac-over (car (GET-FRAME 'X-25 'pac-overhead 'value)));Retr
             ; packet overhead from KB
             ;
             ; calculate throughput per x25 line
   (setq x25-thruput ( * 2 ( / ( * x25-speed  ( / x25-line-util 100))
                                ( * 8 ( + x25-pac-len x25-pac-over)))))
            ; total throughput for all X.25 lines
   (setq tot-x25-thruput (* x25-thruput (X-25-LINES input)))
   (setq x25-tot-pac-proc-time (+ x25-op-time (* x25-pp-time
                                                 x25-pac-len)))
            ; calculate total line processor capacity
   (setq tot-HSLP-cap-x25 (/ 1000 x25-tot-pac-proc-time))
            ; calculate total number of line processors
   (setq num-of-HSLP-x25 (/ tot-x25-thruput tot-HSLP-cap-x25)))
```

```
;*************************************************************
;*                                                           *
;* Primitives used by KS-X25 and KS-ITI to retrieve infor-   *
;* mation from the frames                                    *
;*                                                           *
;*************************************************************

(defun X-25-LINE-UTIL (input-list)84  (cadr (cadr (cdr (cdr (cdr (car in

 (defun x-25-pac-size (input-list)
   (cadr (car (cdr(cdr(cdr(car input-list)))))))

(defun SERV-X-25 (input-list)
  (cadr (car (car input-list))))


(defun ITI (input-list)
   (cadr (car (cadr input-list))))


(defun X-25-line-speed (input-list)
    (cadr (car (cdr (cdr (car input-list))))))

(defun X-25-lines (input-list)
     (cadr (cadr (car input-list))))

(defun ITI-lines (input-list)
     (cadr (cadr (cadr input-list))))

 (defun ITI-line-speed (input-list)
       (cadr (cadr (cdr (cadr input-list)))))


(defun ITI-LINE-UTIL (input-list)
    (cadr (cdr(cdr(cdr(cdr(cadr input-list)))))))

(defun ITI-PAC-SIZE (input-list)
    (cadr (car (cdr(cdr(cdr(cadr input-list)))))))
```