UAH Research Report No. 804

FINAL REPORT
COOPERATING INTELLIGENT SYSTEMS

NCC-813

Prepared by:
Daniel Rochowiak
Johnson Research Center
University of Alabama in Huntsville
Huntsville, AL 35899

Prepared for:
Walt Mitchell
Systems Software Branch
Information and Electronic Systems Laboratory

Marshall Space Flight Center
National Aeronautics and Space Administration
Marshall Space Flight Center, AL 35812

August, 1989

TABLE OF CONTENTS

INTRODUCTION

This report is about cooperation among intelligent agents. It is assumed that such cooperation is desirable and that the problem is to generate some way in which that cooperation can be generated.

An earlier report established one way in which cooperation might be achieved in the effort to explain things. The system CHARLIE was designed to illustrate some of the principles being developed. CHARLIE was deficient in many ways. In attempting to link the abilities of a hypertext system to a rule based system it became clear that there should be some principled division of labor. In particular it was clear that some communications expert was needed. In order to remedy these sorts of difficulties the theoretical frame evolved into that of bureaucratic organizations

The central idea of the bureaucratic system is that it is structured around a distributed database that serves as the central and coordinated information system. As in any traditional database system in an organization it serves to provide a uniform transmission information from one agent to another. In the ordinary case these agents are human beings that are set to some task. In the distributed artificial intelligence environment these are intelligent computational agents. The agents are separated into bureaucratic agents and expert agents. The bureaucratic agents deal with the information system and the communications as well as dispense task to various expert agents. The expert agents contain the specific reasoning strategies that are needed to solve problems where the problems are either diagnostic, knowledge gathering, or explanation. The bureaucratic system is presented as an abstract model an has not yet been implemented.

The Research has been color by the notion of layering cooperation onto existing agents and facilities. This seems reasonable since it may be assumed that specialized developments will go on in many fields, and that computational agents will be added gradually to any system. Since this is the case some effort has also been made to provide a basis for agents to reason about other agents.

VARIETIES OF DISTRIBUTED ARTIFICIAL INTELLIGENCE

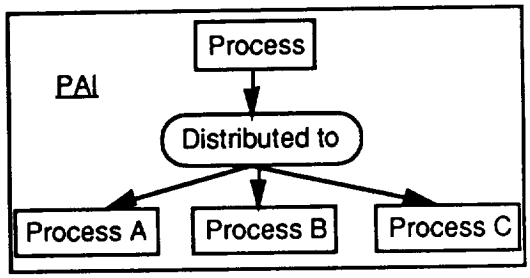
INTRODUCTION

Distributed artificial intelligence is beginning to grow as a field of both theoretical and practical inquiry. Within DAI there a number of striking features, but the most striking is the shift in paradigm or mind set from a psychological view of knowledge and knowledge based systems to a sociological view. Traditionally, knowledge based systems (KBS) attempted to represent or encapsulate the knowledge of some expert. The challenge of knowledge acquisition (KA) was marked by the effort to gain the knowledge in the individual expert's "head" or to acquire the knowledge in several expert's "heads." Within DAI the KA effort is geared to determining the ways in which groups interact, communicate, and come to decisions. The knowledge exists as a sort of shared resource which all agents can use. How each agent uses the knowledge may be particular to the agent, but the overall processing of the knowledge is a shared or social event. Somewhat more technically DAI is concerned with issues of concurrency in artificial intelligence.

Much of DAI can be divided into three areas: Parallel AI (PAI), Distributed problem solving (DPS), and Multiagent Systems (MAS). PAI focuses on the development of parallel architectures, languages, and algorithms for AI. DPS focuses on how the work of solving some particular problem can be distributed across a number of nodes that share knowledge and share in the solution of the problem. MAS focuses on the coordination or cooperation of multiple independently existing agents.

This report focuses on the KBS aspect of artificial intelligence, and will ignore many issues that are of interest to the wider AI community. This report will also ignore issues concerning parallel processors and parallel AI programming. This section of the report will motivate and clarify the focus on multiagent DAI systems (MAS) within the KBS framework.

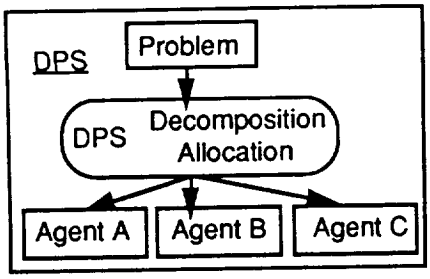
PAI



Much of the importance of parallel AI can be found in its promise of speed and performance. Indeed individualistic models of cognition implemented in AI systems will profit greatly from advances in PAI. This might also be true for more social conceptions of AI as well. However, the key issue concerns the level at which the PAI efforts are placed. They are not efforts at the knowledge or agent level. Rather they are efforts at the programming level. While the success of such efforts at the programming level will undoubtedly have a tremendous impact on all of AI, they will not solve the problems of cooperation at the knowledge level, and solutions to the problems of cooperation at the knowledge level neither require the solution of problems, nor solve problems at the PAI programming level. In this sense, the problems which PAI addresses are independent of the problems that are the focus of those concerned with cooperating intelligent systems. Thus, this report will largely ignore this avenue of research.

DPS

Distributed problem solving focuses on the ways in which a problem may be decomposed and solved. The central question may be though of as, "Which agent does what when?" It should be clearly understood that as a DAI task, this question is to be answered in a programmed, automated way. That is, the goal of DPS is to construct an automate system that answers the question. Such a solution would require that the tasks of the various agents be described and formulated in terms that both account for their being distributed and allow them to be distributed. In the former case a DPS system must take account of the already existing distribution of tasks, while in the latter case it must take account of the potential for a decomposition and distribution of the task to several agents. When a task requires more resources or knowledge that is possessed by any one agent, then the task must be decomposed until the decomposed tasks can be solved by the agents with their existing resources and knowledge. A DPS system then reconstructs the results into a solution of the problem. However, it should be carefully noted that the system that provides for some decomposition must also be able to allocate the tasks to the agents. The point here is that a DPS must first construct the decomposition. That decomposition, however, must also be allocated, since one or more of the agents in the decomposition may be such that the DPS cannot allocate the subtask to that agent.

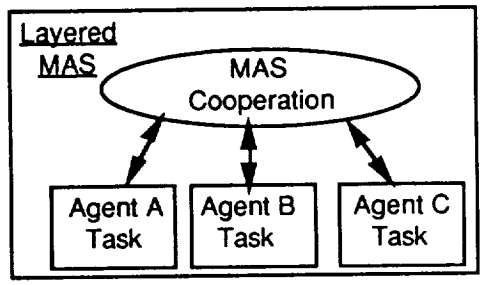


Decomposition and allocation are the central elements in the DPS, and these functions must be automated functions. In practice one may find that rather than having some system do the decomposition and allocation, the systems engineers, and designers have already performed the tasks of decomposition and allocation. In this case the decomposition and allocation are rather rigid and inflexible. However, what constitutes a good decomposition and allocation has already been determined and "built into" the network of systems and agents. It is not unreasonable to think that in many practical areas, and in particular the practical applications of DAI on the Space Station, the patterns of decomposition and allocation will be built into the system. This approach has the advantage of specifying in advance what is to count as good, but has a disadvantage insofar as it is not as flexible as a full and successful DPS would be. While it is to be hoped that future research will provide such flexibility, this report will assume that the patterns of decomposition and allocation are built into the network of agents under consideration.

MAS

Multiagent systems assume that the patterns of decomposition and allocation are already given to the system. These patterns may be given either by some automated system or by having them built into the network of the system. Hence, MAS while compatible with the efforts in DPS, does not demand a solution to its problems before it can begin to solve its own.

The central issues for MAS concern organization and modeling. The former concerns both the static (or 'snapshot' organization) of the agents, and the rules or procedures through which one agent can communicate with another. The latter concerns what one agent knows or can know about another. Together these issues can be thought of as the focus of cooperation, since they provide the resources through which several agents may cooperate in either solving a problem or providing information and knowledge that can solve a problem.



A MAS can be either a specialized system or a system layered onto a network of already existing systems. If the system is a specialized system, then the solutions to the organization and modeling problems are built into the construction of the agents of the system. If the system is a layered system, then the problems are to be solved at a level more abstract than the agents and the agents may be already existing computational systems

which from the MAS point of view can be understood as autonomous. In a specialized system the agents are a product of the MAS in the sense that they are either directly generated by the MAS or are designed to be parts of the MAS. In a layered system, already existing computational agents are either put to a cooperative task at some higher level of abstraction or function, or are the senders and receivers of information and knowledge that will facilitate or improve the task of the computational agent. This report focuses on layered MAS systems since it can be reasonably expected that MAS systems will be added to already existing collections of computational agents. Further, the layered approach is a reasonable first step toward richer DAI systems since it attacks basic DAI problems, but would not interfere with the ordinary operations of the agents in question. This latter point is important especially in the context of Space Station in which such systems would be evolving systems and would be systems such the "pulling the plug" would not jeopardize the overall system.

REFERENCES

• *Readings in Distributed Artificial Intelligence*, A.H. Bond and L. Gasser (eds). Morgan Kaufman: San Mateo, 1988.

• *Distributed Artficial Intelligence*, M. N. Huhns. Pitman / Moragn Kaufman: San Mateo, 1987.

• "Distributed artificial intelligence," L. Gasser. *AI Expert*, Vol. 4, No. 7 (1989): 26-33.

MULTIAGENT SYSTEMS

INTRODUCTION

Organization and modeling are central concerns for multiagent systems (MAS). Organization refers to the structure and rules through which agents communicate. Modeling refers to the ways in which agents gain knowledge about themselves and other agents.

Organizations vary in a number of ways. Organizations may operate by sharing tasks or results. Their structures may be static or dynamic and hierarchical or anarchic. These elements can be intermixed so that one organization may be task-sharing, dynamic, and hierarchical while another might be a result-sharing, static, and anarchic. It is generally accepted that in any system some organizations are better than others. Part of the problem in building a MAS, therefore, is finding the most satisfactory organization for the system. This requires that the purpose or purposes of the MAS be clearly specified.

Modeling requires an agent either to have or obtain expectations about itself or other agents. Ideally, the agent should be able to determine its function within the MAS. The expectations may be either built into the agents or may be acquired by the agents. The expectations may be discrete, categorical, or statistical. Agents may be able to represent the knowledge of other agents or be unable to do so. Finally, agents may be able to have global knowledge or be restricted to local knowledge. As in the case of the organization these aspects can be combined to form different sots of systems.

In this section the various aspects of the organization and modeling of MAS systems will be examined. The discussion, however, will focus upon layered MAS.

ORGANIZATION

In an organization an agent may or may not need to communicate with another agent. When communication is needed, it is assumed that the agent behaves in some way that it would not ordinarily behave if it were not part of the organization. Organizations, therefore, can be created from either atomic agents which do not ordinarily communicate, or social agents that ordinarily communicate with other social agents.

The patterns of communication in the organization may be geared to either task-sharing or result-sharing.

Task-sharing

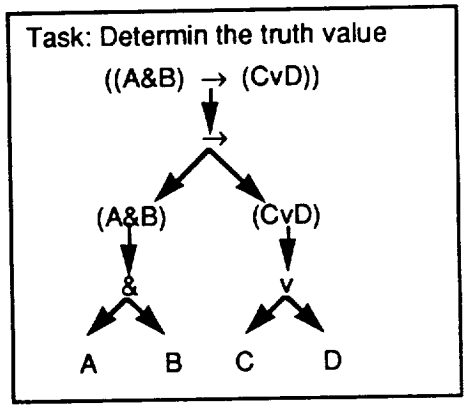
In task-sharing, the communication between agents is designed to divide the work of a large task among several agents. Large problems are decomposed into sub-problems until there are agents that can solve the sub-problems. Task sharing may be imposed or evolved. Imposed task-sharing applies to agents that are not inherently social. These agents have tasks imposed on them in virtue of their special knowledge or information. Alternatively, social agents may come to share a task by their own activity. In imposed task-sharing some agent imposes a task on another agent, while in evolved task-sharing some agent accepts a task published by some other agent. In evolved task-sharing at least some of the agents must be sufficiently general to take on different tasks at different times.

Task sharing fits most naturally into DPS. A DPS is, in a sense, a collection of agents engaged in task-sharing. The agents of a DPS are social and opportunistic, responding to published subtasks when they are able. A full DPS should be constructed with these requirements in mind, and should not be built as a compromise between inherently unsocial agents. In brief an, ideal DPS would be composed of social and opportunistic agents, and some of the agents would be capable of decomposing tasks and publishing subtasks.

For a full DPS, the decomposition of the task or problem is automated, and the code for the decomposition represents some decomposition strategy. In general, strategies such as hierarchical planning, load balancing, subgraphs, and aggregation operate on the notion that the decomposition is built into the statement or representation of the task or problem. For example, the task or problem might be represented as a list of lists. In such a case, the elements of the list

are the subtasks or subproblems. If the element is a list, then the decomposition is continued until the there are no more lists.

Or again, consider the figure to the right. A task or problem can be decomposed in the way that a logical formula is decomposed. Consider the formula ((A&B) → (CvD)). In this case the problem can be decomposed into three elements ' (A&B) ', ' → ', and ' (CvD) '. The first and last elements can again be decomposed into three elements. At that point the decomposition is terminated since all of the elements are either logical atoms or connectives. In a DPS, the atoms would then be published and some agents would solve for A, B, C and D.



Differing strategies for decomposition should produce logically equivalent results. Thus, the choice of the decomposition strategy should be made on grounds other than the logical correctness of the decomposition. These might include time, resource use, or memory space. However, as in the case of organization in general, the decomposition strategy that produces the most satisfactory results is a function of both the system and the task or problem to be decomposed. An ideal DPS would contain some method for selecting the most appropriate decomposition strategy.

In a rigid organization, task-sharing is established by the programmer-designer, and the agents in the system may be far more atomic than social. Thus, in rigid organizations it is reasonable to think of the decomposition as given in the construction of the system. This results in systems that are static. The system organization is the same for every task or problem that it receives.

Task-sharing in rigid organizations represents the programmer-designer's ideas about how problem are to be solved and which agents are best able to respond to the task or problem. The decisions of the programmer-designer can reflect a variety of concerns that are not inherent in the problem or task under consideration, however. For example, political problems and philosophical differences can be adjudicated by the programmer-designer by reaching compromises or other agreements with the human responsible for a particular computational agent. This can allow greater flexibility in the development of the MAS. It should be noted, however, that the efforts of the programmer-designer can be aided by automated tools, and these tools are likely to use the same strategies as would be used in an automated DPS. Such tools can produce evidence for a

particular decomposition and provide the programmer-designer with arguments to be used in the process of generating a compromise.

The advantages of rigid organizations are that the operations of the agents can be specified in advance, the agents can be optimized to particular tasks, and the system runs the same way every time it is given a task or problem. The disadvantages are that the system must respond in the same manner even if a different decomposition would give more satisfactory results, and since the agents are specialized redundancy must be built into the agents. Flexible organizations have the advantages that they are able to select the most appropriate decomposition strategy, and that some amount of redundancy can be built into the system of agents since at least some of the agents are generalists. The disadvantages of flexible organizations are that they are inherently more difficult to test since different tasks are handled in different ways, and the system is more difficult to construct since the agents must be designed as social agents. This report will focus on rigid organizations since it is reasonable to suppose that such organizations can be layered onto already existing groups of agents, and that as work progresses on automated decomposition, the results of that work can begin to be applied to static organizations. The latter point is important. It provides an evolutionary pathway for future development without having to abandon the existing population of agents. It is more restrictive than a full DPS system, however, since the decompositions are still given by the programmer. Only the choice of which scheme is automated. As noted above, the organization is still imposed on the agents since the agents do not select the problems or tasks on which they will work.

It will be assumed that the there exists some network through which the agents of a static organization can communicate. It should be noted that the term 'agent' is used in the sense of a computational agent that may be a program running on the same or different physical machines. Thus, a rigid organization might be applied to several computational agents running on the same physical computer through multitasking. On the other hand, the computational agents may be programs running on different physical machines. For the sake of simplicity, the communications paths between agents will be called the network. This level of abstraction allows many implementation issues to be put to the side.

It also will be assumed that the static organization imposed on the system will be such that the problem or task either will be successfully terminated or the system will be able to indicate that it cannot provide a satisfactory solution. Since the burden of decomposition is placed on the programmer, a system that violates this assumption or produces to great a ratio of nonsolutions to solutions must be redesigned and reprogrammed.

Given these assumptions and the focus on rigid systems it may appear that there is rather little cooperation possible among the agents. However, this is only so from the perspective of task-sharing. For the type of MAS that is the focus of this report, result-sharing provides the means for cooperation.

Result-sharing

In result sharing the partial results of the agents in the system are shared. Since it is assumed the organization is rigid and task decomposition is fixed, the agents in the system can use the partial results of other agents as data in their own computations. In this sense a MAS that focuses on result-sharing is data directed. The agents act on the data that they have available and share their results with other agents.

The agents in the system may be so constructed that they either share all of their results or some of their results with other agents. Assuming that the agents in question are more atomic than social it will be assumed that the agents share some but not all of their results. Further, the organization of the agents may be such that either any agent can communicate with any other or communication is in some ways restricted. It will be assumed that the agents in the system have restricted communication. These assumptions result in a hierarchical organization of possibly preexisting agents. However, it does not rule out the possibility that there are local groups of agents that are anarchic. The assumption only requires that if such local anarchies exist, they should place their results into a hierarchy. This is consistent with the layered approach adopted in this report.

An example will help to clarify the issues that are involved. Suppose that the task for a collection of computational agents is to determine if there is enough water on a space craft to continue normal operations. The simplest way to conceptualize the task is to assume that there is sufficient water unless it can be shown that there is not. The task can then be reduced to the ways in which the system might fail to provide enough water for normal operations. This in turn breaks into two parts. The first subtask is to determine what normal operations will be in the near future, and the amount of water needed for those operations. Here an agent in the hierarchy may need to consult plans, schedules, and the health status of the crew. Each of these items may represent the result of another agent. Sharing these items of information, therefore, would result in some measure of cooperation amongst the agents. Further, the agents below the level of the agent that determines what normal operations are may also need to communicate. For example, if a specific series of

physical exercises are scheduled for the crew members, but one of the members of the crew is sick, then the schedule may need to be altered. Both the new schedule and the fact that there is a sick crew member will have an impact on what constitutes ordinary operation, and consequently how much water is needed for normal operation. The second subtask concerns the status of the water treatment facility itself. The status of the system, the capabilities of the system, and the current reserves of the system are relevant to determining if the water treatment facility can meet the demand of the future normal operations. Here a sharing of results between the systems that monitor the equipment, and the performance trends of the system with the agent charged with making the determination of adequacy are important. Finally it should be noted that the the the agents in charge of the subtasks will also need to share results. For example, if the water treatment facility cannot provide the requisite amount of water, then some rescheduling must be done. The new schedule, then acts as a new partial result for the agent that determines if the facility can provide the water.

In this example it is important to note that some of the agents that are at work on subtasks are already agents that are at work on particular tasks. For example the system that monitors the status of the treatment facility equipment is a system that can be though to be an agent that is in continuous operation. To impose another task on this agent might force it to become less efficient or deliver unsatisfactory results. The idea of result sharing allows the system to continue to operate, and requires only that its results be shared. This shows the advantage of a layered MAS. It can work with existing specialists which must have there attention focused on a particular task.

MODELING

Agents in organizations, even in rigid organizations with relatively fixed decompositions, must know something about the other agents in the system. The minimal knowledge requirements are:

- what other agents can receive messages
- what kinds of messages can be received by other agents
- what sorts of things the other agents do.

To these can be added several knowledge elements which, while not essential, contribute to the actual operation of the organization. These are:

- how messages to other agents are structured
- how busy another agent is
- from what other agents messages are received.

Finally, the agent should have some idea of its own knowledge:

- from what agents it has received messages
- how busy the agent is
- what is the agent's goal

• how the agent fits into the organization.

Although there are various ways in which these requirements can be implemented. This section will focus on frame and script styles of representation. There are several reasons for this. First, these representational schemas have already meet with some success in AI applications. Second, even if these representational schemas are not fully implemented in a system, they can be simulated to some degree in classical databases. Finally, this style of representation appears to be extensible and applicable to other problem areas.

Frames

A Frame Structure Example

FRAME: ECLSS KB agent

TYPE:
RANGE: (water-recovery, air-recovery, fire)
DEFAULT: none

STATUS:
RANGE: (active, inactive, maintenance)
DEFAULT: active

RATE:
RANGE: (1-9)
DEFAULT: 5

MESSAGE_STATUS:
RANGE: (pending, none)
DEFAULT: none
IF_NEEDED: check-message-manager

A frame consists of a series of slots. Each slot is identified by the name of the attribute slot, a value or range of values, and procedural attachments. Consider the frame structure to the left. The name of the frame is ECLSS KB agent. It contains slots for TYPE, STATUS, RATE, and MESSAGE_STATUS. For each attribute a range and allowed default is provided. The DEFAULT slot indicates the value the slot will have if a value is not explicitly provided. It should be noticed that the MESSAGE_STATUS slot has a special IF_NEEDED attribute. This is an instance of a procedural attachment. If some operation needs to know the value of the MESSAGE_STATUS attribute, then the procedure check-message-manager will be used to get that value. This is one of the two forms of attachment. The second form will be examined in the case of a sub-frame.

One of the advantages of using the frame representation is that this style of representation supports inheritance. One frame can inherit the attributes of another by declaring it to be a specialization of some frame. An intuitive example is the relation of a passenger car to a motor vehicle. The concept of a passenger car contains the concept of motor vehicle. The containment is such that the concept of the passenger car contains all of the attributes of the concept of motor vehicle plus the attributes appropriate to the concept of a passenger car.

The example to the right will further clarify this point. The FRAME water recovery is a specialization of the FRAME ECLSS KB agent. Thus, this subframe inherits the attributes of that frame. This means that the RATE and MESSAGE_STATUS attributes apply to this subframe implicitly. However, the additional attributes in this subframe differentiate the subframe water recovery both from other specializations of ECLSS KB agent and other frames. Note that in this example the GOAL attribute is uniquely specified. Further, it should be noticed that the property UNIT_PROBLEM has a special procedural attachment. Unlike the IF_NEEDED procedural attachment in the parent frame, this attachment is used whenever a value is set for the attribute. The attachment contains two parts. The first issues an alert and the second advises the diagnostic unit. Within the context of this example, the message to the diagnostic unit would indicate that the water recovery system has had some problem and that problem needs to be diagnosed.

A Subframe Examaple

FRAME: water recovery

SPECIALIZATION_OF: ECLSS KB agent

GOAL: water-quality

IODINE:
RANGE: safe, yellow, red
DEFAULT: safe

TOC:
RANGE: safe, yellow, red
DEFAULT: safe

BIOLOGICAL:
RANGE: safe, yellow, red
DEFAULT: safe

UNIT_PROBLEM:
RANGE: component-failure, yellow-line, red-line
DEFAULT: none
IF_ADDED: allert and advise diagnostic unit

Frames provide a useful way of modeling the agents of the MAS. Each agent would have access to the frames and subframes appropriate to its organization. Note that the frames can be altered and expanded to fit the the specifications presented at the beginning of this section. As it stands the frames structure is a data structure that accords with ordinary database structures. The novelty is in the notion of procedural attachments and inheritance. I will assume that the mechanisms for these features are in place and that a frame-base exists. The central issue is how reasoning about the frames-base can be incorporated into the system.

Reasoning in a frame system can proceed in two ways. In the first way the frames simply act as databases that can be interrogated. That is, one agent can ask for the value of an attribute in a designated frame. However, this presupposes that the second mode of reasoning has already produced an instantiation of the frame. An instantiation of the frame converts the frame from a template into an actual object. Consider the example above. The object water recovery does not exist until it is instantiated. From the MAS point of view, this is important since not all of the

agents of the organization will instantiate all of the frames. Further, it should be noticed that upon the instantiation of the FRAME water recovery, all of the attributes of the frame ECLSS KB agent are inherited by the instantiated object. Thus, if the instantiated object, say water recovery one, is interrogated about its STATUS and no explicit value has been set for this attribute in water recovery one, then the response will be that water recovery one is active since this is the default value. Further, the agent in which water recovery one is instantiated has knowledge implicitly of the GOAL and UNIT_PROBLEM. The latter attribute it should be noticed also contains the knowledge about what to do if the value of the UNIT_PROBLEM attribute is not none.

Frames will provide useful devices for storing knowledge about the agents of the organization, and can be further extended to a discovery process of other agents with which they are communicating. This latter point may be important in systems that are built to configure themselves to their circumstances. Such agents, although not social agents, would need to discover which agents are sending and receiving messages relative to their place in the organization. This can be done by attempting to match the information known about another object to the frames that the agent knows or has access to. In this sense it could be claimed that the one agent discovers the exist and type of other agents in their organization.

Scripts

Scripts can also be considered as a form of modeling. Rather than modeling the agents in the system, however, scripts model the activities in the system. Instead of the slots for attributes and values in an agent frame, the script representation would include features such as the entry conditions, results, props, roles, and scenes for a typical activity. As in the case of frames the elements of the script contain default values and features that may or may not be used in the activity. This can be illustrated by considering the way in which an activity such as checking the water supply on the space station can be cast in terms of a typical sequence of scenes. The process may first check the holding tank, then check the processing equipment, and final check the output tanks. To enter the scene in this case would be to enter the water processing loop. If one agent were to attempt to get information from another agent in the water processing loop, the script could provide the background information against which the activities of the quizzed agent would be made intelligible. The use of certain processes would be codified in the props and the various roles played by other agents would be contained in the roles.

The script would contain background information that can aid an agent in modeling the activity of the system and establish reasonable defaults about what other agents are doing and why they are doing it. As in the case of scripts one can reason either forward, when the script becomes known, or backward, when the script is being looked for.

Paradigms

As a general means of expressing the modeling in MAS the notion of a "paradigm" will be helpful. A paradigm is simply a structure that provides for both defaults and procedural attachments. A paradigm can be created from other paradigms either through inheritance mechanism or forcing of inheritance. In this sense a paradigm can be implemented in either an object oriented environment where the the system itself takes care of inheritance or in a modular programing environment in which the raw materials for inheritance are available, but the programmer must explicitly construct the inheritance.

Paradigms can be implemented in various ways. What is important is that the paradigm represent the core the core characteristics of the entity or process it represents and provides for a way of manipulating or gaining its own values. In this sense, frames, scripts, and objects can all be used as paradigms. In the remainder of this report the term paradigm will be used to refer generically to these.

REFERENCES

• "Frameworks for coopertation in distributed problem solving," R. G. Smith and R. Davis. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-11 (1981): 61-70.

• "An organizational view of distributed systems," M. Fox. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-11 (1981): 70-80.

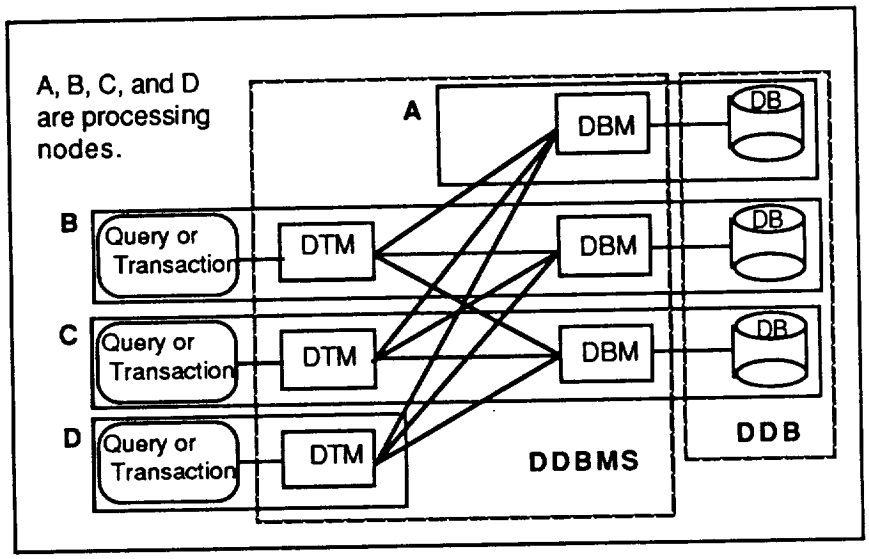
• "Communication and interaction in multi-agent planning," M. P. Georgeff. *Proceedings of AAAI '83* (1983): 125-129.

• *Principles of Artificial Intelligence and Expert Systems Development*, D. W. Rolston. McGraw-Hill, 1988.

DISTRIBUTED DATA SYSTEMS

INTRODUCTION

Distributed databases have been well researched, and a discussion of them will help to focus both further examination of distributed knowledge based systems, and the ways in which such systems can be implemented.



Distributed database systems (DDBMS) are systems that support the execution of transactions and the retrieval and updating of data across two or more independent computers. The management of DDB systems consists of the collection of distributed transaction managers (DTM) and database managers (DBM) for all the computers in the net. The DTM is a program that receives processing requests from query or transaction programs and translates them into action commands for the DBM. The DBM is a program that processes some portion of the DDB. Nodes in the distribution net may be either transaction nodes or database nodes where these are differentiated by whether the node can make transactions or queries (B,C,D), or store the database (A,B,C,). A single computer can have both node functions (B,C), but it need not (A,D). For database nodes, the database at the node may or may not replicate the DDB, and if it does replicate the DDB the replication may be partial or full. For transaction nodes, the level of authority may be that of equality, in which case the node can insert, modify, delete, or read data, or servitude, in which case the manipulations of the data are in some way restricted. These considerations provide a basic framework for this examination of DDBs.

Advantages	Disadvantages
Better performance	Worse performance
Increased reliability	Decreased reliability
Easily scaled in size	Increased complexity
Readily tailored	Difficult to control

The advantages and disadvantages of the DDB framework are set out in the table at the left. Although the table may appear to contain contradictory items, a closer inspection reveals that that the apparent contradictions are actually functions of other constraints. For example, in a well-behaved distributed system, the overall performance of the system will be better than for a centralized system. This is especially so when a local transaction matches a local database. However, if the system is not well-behaved or if there are system problems the performance can be worse than that of a centralized system. For example, if the distributed system uses a partial replication system and the partial distribution is not well tuned to demands of transaction and query nodes, the system performance will degrade. In a similar manner, the reliability of the distributed and centralized system can vary. Given the potential for duplication of data elements and the ability of distributed nodes to continue operation while other nodes are "down," the distributed system would be more reliable. On the other hand, recovering from problems may be far more difficult for a distributed system. Attempting to determine what changes have been made to the database while a node was down is more difficult than simply restoring the database program from as saved file.

DDB AND SPACE STATION

A distributed database system is envisaged as being part of the Standard Services of Space Station Freedom. The Software Requirements Specification for Standard Services indicates that Standard Services "will manage data and commands using a Distributed Data Base model." In particular, a "Runtime Object Data Base (RODB) can be thought to exist at each node of the DMS," and that "Standard Services will provide location transparent access to data in remote RODB's via directories." In the RODB model data are written into the RODB by the data supplier and are read out by data users. Further application command requests are written into the data base by command requestors. Standard Services either issues the command to the effector or notified the effector of a command request. Options exist within Standard Services Data Processing to perform exception checking on data and notify applications of exception conditions. Such exceptions will include local yellow line and red line limits.

ISSUES

Ideally, the operation of a DDB system should be location, replication, concurrency, and failure transparent. A DDB is location transparent, if the transaction managers determine the location of

data and issue commands to the appropriate DBMs. A DDB is replication transparent if transactions can be processed without knowing how many time data is replicated. A DDB is concurrency transparent, if all concurrent transactions are identical to transactions executed in serial order. A DDB is failure transparent if all transactions are atomic in the sense that either all or none of the transaction is processed. In this report failure transparency will not be examined, but the following discussion will focus on transaction that are atomic

The central issues for DDB development concern the ways in which and the degree to which the ideal can be achieved.

Location and replication transparency

Location and replication transparency are related in that data directories are used to ensure them. A data directory can be thought of as a database about a database. It is akin to a data dictionary but maintains a database of the locations at which data are stored. It should be remembered that in a DDB some data elements may be replicated and some nodes may not replicate the whole database.

	Local node	DTM	Directory	Actions
Before	Read D1	Look at directory to find location of D1; select appropriate location	D1 at N1 D1 at N2 D2 at N2 D2 at N3	DTM requests the DBM at either N1 or N2 for reading of the value of D1
	Write D1	Look at directory to find all locations of D1	D1 at N1 D1 at N2 D2 at N2 D2 at N3	DTM requests the DBMs at N1 and N2 to write new value of D1
After	Read D1	Look at directory to find location of D1; select appropriate location	D1 at N2 D1 at N3 D2 at N1 D2 at N2	DTM requests the DBM at either N2 or N3 for reading of the value of D1
	Write D1	Look at directory to find all locations of D1	D1 at N2 D1 at N3 D2 at N1 D2 at N2	DTM requests the DBMs at N2 and N3 to write new value of D1

Location transparency is needed to manage the DDB effectively. This is even the case where the the database is wholly replicated. If the DTMs are responsible for determining the locations of data and issuing actions to the appropriate DBMs, then location transparency can be supported. By allowing

the DTMs access to directories of data locations transactions can be isolated from changes in location.

If a transaction can be processed without knowing how many times the data is replicated, then the transaction is replication transparent. There are two cases. For reads, the DTM can pick any node at which the data is located. The read can be optimized in many ways. The DTM could pick, for example, the node that is the shortest distance from itself or pick the statistically least active node. For writes the DTM must issue the write request to every node at which the data is stored. The directory provides the information to the DTM on which nodes have which data elements. This leads to a demand on the directory management. To avoid errors, the presence of a data item must not be posted until the it is available (a read problem) and the item must be posted to the directory when it is available (a write problem.)

Concurrency transparency

Concurrency and failure transparency are related. Each is far more complicated than location or replication transparency. Further, it appears that gains in either concurrency or failure transparency lead to losses in the other. Concurrency requires that all concurrent transactions yield results that are identical to those which the same transactions performed serially would have yielded. Failure transparency requires that either all of a transaction is processed or none of it is. Together these entail the joint requirement that all transactions yield results identical to those of serial transactions or do not yield any results. One might consider three situations. A perfect node is a node that always processes a transaction correctly. A sane node is not perfect but fails in predictable, defined ways. An insane or Byzantine node fails in unpredictable and undefined ways. If predictable failures can be trapped and all nodes are either sane or perfect, then the joint requirements can be meet. If there is any insane node, then the requirements cannot be satisfied.

A more detailed examination of the joint requirement is in order. In order to construct this examination at bit of standard symbolism is helpful. Let '$r_i(N_I)$' be read as 'a read by transaction i of the value of N stored at node I.' Similarly for '$w_i(N_I)$' where 'w' stands for a write transaction. The arrows '$\leftarrow$' and '$\rightarrow$' are used for read the value and write the value respectively. '$r_1(N_A) \leftarrow 4$' means 'transaction 1 reads 4 for the value of N at node A.' Unless the DDB system is well structured inconsistencies can be generated by failures of the joint requirement. Two particular difficulties are lost updates and inconsistent reads. An example may help to illustrate the problems.

In the first case consider what happens in the case where two transactions attempt to process the data stored at node A.

Transaction Sequence	N_A
Start	5
$r_1(N_A) \leftarrow 5$	5
$r_2(N_A) \leftarrow 5$	5
$w_1(N_A) \rightarrow 4$	4
$w_2(N_A) \rightarrow 3$	3

Such a situation might occur when 1 unit of water is drawn from a tank and then another 2 units are drawn form the tank. The first transaction T_1 decreases the value of N at A by 1 and the second transaction T_2 decreases the value of N at A by 2. The final value of N_A should clearly be 2 (5-1-2). However, in this case the last value placed in N_A is 3.

In the second case consider what happens when a value changes at one node, but there is a compensating change at another node.

Such a situation might occur when one is moving 4 units of water from one tank to another. Suppose that the volumes of water for the three tanks are held at nodes A, B and C. Let T_3 be the transaction that moves 4 units of water from C to B. Let T_4 be the transaction the determines the total amount of water in all three tanks. In this case the the three reads of T_4 are completed before the adjustments caused by T_3. The result would be that T_4 would report a total of 8 units (2+1+5) rather than the correct 12 units.

Transaction Sequence	N_A	N_B	N_C
Start	5	1	6
$r_3(N_C) \leftarrow 6$	5	1	6
$w_3(N_C) \rightarrow 2$	5	1	2
$r_4(N_C) \leftarrow 2$	5	1	2
$r_4(N_B) \leftarrow 1$	5	1	2
$r_4(N_A) \leftarrow 5$	5	1	2
$w_3(N_C) \rightarrow 5$	5	5	2

Both kinds of problem can be avoided if the DDB system provides only executions that are serial equivalent.

Serial equivalence

Two transactions are serial equivalent if (1) each read reads data values produced by the same write in both executions, and (2) the final write of a data value is the same in both executions. Transactions may, however, conflict. Two operations conflict if they operate on the same data item and one of the operations is a write (read-write, write-read, and write-write conflicts). Such conflicts may be either intratransaction or inter transaction. It will be assumed that the DBM can handle intratransaction conflicts.

A transaction executes an ordered sequence of requests by the DTM. The ordered sequence is called a schedule, and a serial equivalent schedule is a consistent schedule. The fundamental theorem of serialization specifies the relation between conflicts, schedules and transactions. For any ordered list, **T**, of transactions, T_1, T_2, ..., T_n, a schedule, **S**, of **T** is consistent if for any two conflicting requests REQ_i and REQ_j, REQ_i precedes REQ_j in **S** if and only if T_i precedes T_j in **T**. This theorem can be applied to the DDB case by allowing that the order of conflicting requests must mirror the order of conflicting transactions no matter where the transaction is processed nor how many time it is processed. This implies that all of the nodes of the DDB have the same schedule. Since this may not be true, other problems may be generated. Two-phase locking provides a means to satisfying the conditions of the theorem.

Two-phase locking

Two phase locking as means of concurrency control assumes that the system under consideration uses two-phase commitment.

In a DDB system every transaction is given a private workspace. During the transactions updates a made in the workspace, but are not committed to the database. When the transaction is completed the results can be committed to the database. At completion the DTM sends out commit actions to the various DBMs. However, in a DDB a problem is encountered if several DBMs are involved and one or more of them cannot commit the changed to the database they manage. In consistency would result if only some of the databases were updated.

In order to prevent this the DDB uses two-phase commitment. During the first phase the DTM processing the transaction issues a pre-commit action to all DBMs holding data that is updated by the transaction. If the DBM can commit the updates of the transaction to the database, it writes log records and then responds with a YES to the DTM. If all of the DBMs answer YES, then the DTM issues a commit action. If any DBM responds with NO, then the transaction is aborted. This preserves the atomic character of transactions.

With two-phase commitment the fundamental theorem can be enforced by using distributed two-phase locking. This method requires DTMs to obtain locks before reading or writing data. Thus, before reading a data item the appropriate DBM must grant a read lock. When updating a data item every DBM that stores the data item must grant the DTM a write lock. There are three rules on the granting of locks.

(1) A read lock is granted if no write lock has been granted for the data item.

(2) A write lock is granted if no other read or write lock exists for the data item.

(3) If a DTM has released a lock of either type, no further locks can be granted to the DTM.

Under these conditions a transaction has two phases. In the growing phase the transaction acquires locks, and in the shrinking phase the transactions releases locks. It has been shown that read and write locks will generate consistent schedules just in case they proceed in these two phases. The clearest way to accomplish this two-phase locking is to release all locks only when the transaction is terminated either by the results being committed or by aborting the transaction.

In a DDB system each DBM must have a subsystem that grants and releases locks and each DTM must conform to rules noted above. The DTM can issue a read request as soon as any read lock has been granted but cannot issue a write command until all DBMs have granted a write lock. Under these conditions conflicts are avoided. If any DBM has issued a read lock on a data item and a new transaction requests a write lock, the DBM at issue will not grant it. Thus, since all the DBMs have not granted a write lock, the transaction cannot write and read-write conflicts are avoided. If all DBMs where a data item is stored have issued write locks, then no transaction will be granted a read lock for that data item. Thus, write-read conflicts are avoided. Finally, if all DBMs where a data item is stored have granted write locks to a transaction, then no other transaction can gain a write lock for that data item. Thus, write-write conflicts are avoided.

Consistent Concurrent Schedule for T_1, T_2, T_3, T_4

	N_A	N_B	N_C
Start	5	1	6
$r_3(N_C) \leftarrow 6$	5	1	6
$r_1(N_A) \leftarrow 5$	5	1	6
$r_3(N_B) \leftarrow 1$	5	1	6
$w_3(N_C) \rightarrow 2$	5	1	2
$r_4(N_C) \leftarrow 2$	5	1	2
$w_1(N_A) \rightarrow 4$	4	1	2
$w_2(N_B) \rightarrow 5$	4	5	2
$r_2(N_A) \leftarrow 4$	4	5	2
$r_4(N_B) \leftarrow 5$	4	5	2
$w_2(N_A) \rightarrow 2$	2	5	2
$r_4(N_A) \leftarrow 2$	2	5	2

The foregoing conditions for two-phase locking generate consistent schedules. However, this procedure raises the possibility of deadlock. Deadlocks occur when there is a circular series of requests for locks. There are two tactics for attempting to deal with deadlock, deadlock prevention, and deadlock detection.

TOWARD A BUREAUCRACY

The importance of a distributed database system can be found in the role it can play in a bureaucratic system. Within a bureaucratic system there is a need to share information among more or less independent intelligent agents. These agents are more-or-less specialists in a particular task. Within the bureaucracy the tasks are initially composed by the designers and managers of the bureaucratic system. In such a system there is little need for a system that engages in task sharing since the bureaucracy is intended to establish a rationalized distribution of labor for task specialists. However, with such an *a priori* task distribution system, the burden of the systems operation falls upon the sharing of the results of the specialists. Further, it should be noted that a bureaucracy is layered onto the collection of intelligent agents. This layered approach is designed to ensure that each agent is able to continue in his task. However, for this to be the case the bureaucratic system must provide access to the information that the specialist agent. Again the bureaucratic system is deigned to make that information available.

With computational cooperating intelligent agents, these conditions for a bureaucracy can be meet by the distributed database concept. In the DDB all agents of the system can post their results to the database, and all agents can read the database. The posting of the results amounts to the more traditional result sharing conception. It should be noticed that the mechanism examined here leverages the technology of the DDB in enforcing consistent write procedures. The other side of the issue is the conditions placed upon reads in a DDB. These conditions insure that an intelligent agent will have a consistent reading of the available information.

In the following section the bureaucratic model will be examined more closely.

REFERENCES

• *Database Processing*, D. K. Kroenke and K. A. Dolan. SRA: Chicago, 1988.

• "Transactions and consistence in distributed database systems," I. L. Traiger, J. Gray, C. A. Galtieri, and C. A. Lindsay. *Transactions on Database Systems*, 7(1982).

• "Applications of Byzantine agreement in database systems," H. Garcia-Molina, F. Pittelli, and S. Davidson. *Transactions on Database Systems*, 11(1986).

• "Transaction management in the R* distrubuted database management system," C. Mohan, B. Linsay and R. Obermarck. *Transactions on Database Systems*, 11(1986).

BUREAUCRATIC SYSTEMS

INTRODUCTION

A bureaucratic model for DAI is predicated on the differences between the individual expert agent (E-agent) and the intelligent bureaucratic agent (B-agent).

The paradigm of an E-agent is a diagnostician. Foe the moment the vast differences between a machine operator and a medical doctor will be ignored. It can be fairly assumed that the E-agent amasses knowledge through his or her interaction with the domain. At one step remove from the E-agent can be found the working scientist. As will be indicated below the working scientist or engineer shares characteristics of both the E and B-agents. The paradigm of a B-agent is the welfare agent of the Internal Revenue service agent. It can fairly be assumed that such agents have amassed knowledge about procedures and regulation as wells as ways in which these can be applied badly and well. Between the diagnostician and the bureaucratic agent

THE EXPERT AGENT

E-agents cane be separated into diagnosticians and scientists. Although this is a rough classification it does seem reasonable. These two classes of E-agents differ in both their goals and their bureaucratic interaction.

The Diagnostician

The novice's experiences with the domain become knowledge in several ways. An E-agent will most often receive some formal training in the domain. This formal and declarative knowledge furnishes a base on which addition knowledge can be built. Although such knowledge is not sufficient for the E-agent to have expertise, it is a necessary contributory factor. This explicit training can take place in two ways, however. One way is document drive. In this case the novice is presented with explicit documents. Such documents should be understood broadly to include not just printed materials, but also movies and computerized instruction. The other way is through an

apprenticeship. Apprenticeship training provides some measure of domain interaction, but always under the guidance of a mentor. In both cases the training provides a common body of knowledge. In either case the novice is provided with a background against which further experiences can generate individualized expertise.

The novice can gain expertise thorough failure experiences. That is, given the common body of knowledge failures in the adequacy of that knowledge create domain failures. The failures generate the opportunity for the E-agent to become even more familiar with the documents and existing experts in the domain. At this stage the novice comes to know that if a particular kind of failure occurs, then certain documents or experts have the knowledge needed to recover from the failure. Further the novice E-agent learns that if there is a failure of a specified type, then specified operations must take place to recover from that failure. A second way in which failures generate knowledge is through the novice E-agent reflecting on the conditions that preceded the failure. In this case the novice E-agent comes to that if specified conditions occur, then a specific kind of failure may occur. Further in combination with the former sort of learning the E-agent may also come to know that if certain condition occur preparations should be made for certain procedures of recovery. However, the novice E-agent may also come to know that if certain conditions obtain, and if the agent takes certain actions, then certain failures may not occur. Such a jump would be based on both the link between the conditions and the failure and the link between the failure and the recovery. The agent would in fact find that an early intervention into the system might avoid the failure.

In brief the novice becomes a an expert E-agent by amassing knowledge through experience. This knowledge is manifest in several generic kinds of rules:

- if a particular kind of failure occurs,
  then certain documents or experts have the knowledge needed to recover from the failure
- if there is a failure of a specified type,
  then specified operations must take place to recover from that failure
- if specified conditions occur,
  then a specific kind of failure may occur
- if certain condition occur,
  then preparations should be made for certain procedures of recovery
- if certain conditions obtain, and if the agent takes certain actions,
  then certain failures may not occur.

The Working Scientist

The working scientist deviates from the E-agent in several ways.

The working scientist most often focuses upon a domain problem. The working scientist can be understood to represent a domain problem as the inability of the current explanatory or theoretical resources of his or her discipline to adequately account for or manipulate some domain phenomenon. Thus the problem is a function of the description of the domain, the knowledge resources, and a measure of adequacy. In order to solve the problem the working scientist may either alter the description, add to the resources, or change the measure. In quite revolutionary periods all three elements might be changed, while in more normal time only one of the elements are altered. This is rather different from what the diagnostician does. The diagnostician uses the knowledge that he has in an effort to identify, recover, and remedy problems. The working scientist on the other hand attempts to solve problems which might require the alteration of knowledge. It is the alteration of the knowledge that distinguishes the scientist from the diagnostician.

Even though the task of the working scientist is the alteration of existing knowledge, some parts of that task are similar to the efforts of the diagnostician. Like the diagnostician the research scientist must go through a period of training. What is learned during that period is akin to what is learned by the diagnostician. However, in addition learning the rules for the detection, recovery, and avoidance of failures, the working scientist also learns how to manipulate knowledge structures. In particular the working scientist learns how to extend and amend existing structures, and how to present them.

Extending and amending knowledge structures has many aspects. Such efforts may for example focus on new mathematical or logical techniques. Such a focus provides new representational tools for older knowledge items. Such tools may allow the scientist to get around the perceived problems. Another focus may be the addition and deletion of aspects of the objects in the domain. Such reasoning may change the status of the properties of an object. Properties that were once though essential may displaced, and properties not though to be taxonomic may give rise to new classes. A third focus may be the extension of one set of knowledge elements to another set by analogical reasoning. Such reasoning is often useful in providing templates for solving perceived problems. In brief the effort to solve problems may lead the working scientist to develop a repertoire of techniques that can be used to solve other problems. In this case what the scientist

learns is that if a certain problem can be solved by extending or amending knowledge, then certain techniques of representation, definition, and analogy should be used.

Although the extending and amending knowledge structures is a principle part of the scientist's work there is a related but distinct part that is far less mechanistic. The process of discovery while sharing some of the features of the extending and amending of knowledge, often adds a very nonmechanical, insightful or serendipitous element. While it might be possible to learn some of the rules of discovery, it may always remain the case the some discoveries will by the product of the special blend of elements that the scientists brings to a phenomenon. In some sense all of the scientist's training prepares the ground for discovery, but in no way necessitates it.

In addition to the knowledge focused elements, the scientist also learns how to present his efforts, and it is in this effort to present his research that the scientist begins to participate in a more bureaucratic style of reasoning.

One of the principle components of the scientific ethos is that scientific knowledge is public knowledge. This idea should be understood in the broad sense that includes both the funding of the research and the presentation for critical review of the results of that research. Contemporary research is dominated by the idea of funded research and the funding for the research in one way or another traces to a governmental agency. In a sense the scientist is almost always performing his or her research at the approval of the populace. Admittedly the connection is indirect, proceeding through many funding and refunding agents. However, it is the particular characteristic of this sort of research that it must be proposed to a body for evaluation before it is funded. This leads to the somewhat curious practice of defending the validity of a research proposal before the research is done. More importantly it leads to the learning of how to construct the proposal. Learning to construct the proposal is learning how to communicate with other agents. Thus, the scientist learns that if the proposal is to be acceptable, then certain protocols of communication are to be used. Some of these protocols are explicitly presented by the funding body, others are implicit and learned only through experience. In a similar way the scientist must learn the communication protocols for the sharing of his results with other agents. Just as in the case of the proposal some of the protocols are explicit while others are implicit.

In brief the working scientist amasses techniques and procedures for solving problems and presenting research. These techniques and procedures can be thought of as encapsulated in some general rules:

- if a certain problem can be solved by extending or amending knowledge,
  then certain techniques of representation, definition, and analogy should be used
- if a proposal or report of research is to be acceptable,
  then certain protocols of communication are to be used.

THE BUREAUCRATIC AGENT

The B-agent differs from the scientist and the diagnostician. In a sense, the diagnostician and the scientist are both E-agents. They differ insofar as their expertise focuses on different aspects of the world. The diagnostician attempts to make devices run properly while the scientist attempts to make knowledge run properly. The diagnostician is successful if failures cane detected and prevented or can be recovered from; the scientist is successful if perceived problems can be solved. The focus of the B-agent is different. The B-agent seeks to create and apply procedures that allow both diagnosticians and scientists, as well as a myriad of others, to work properly.

The B-agent can be thought of as the agent that applies the rules that allow other agents to operate. In a very classical, Weberian sense, the bureaucracy rationalizes the large social actions of individuals. It will be assumed that the bureaucracy is a legitimate one, and that the B-agents carry out the policies and procedures of the bureaucracy in accord with an established distribution of labor.

Typically a B-agent amasses knowledge about the bureaucracy itself, and applies that knowledge to both the clients of the bureaucracy and the other B-agents in the bureaucracy. Unlike either the diagnostician or the scientist, the E-agent focuses exclusively upon the directive artifacts of the social organization. The scientist to a greater degree than the diagnostician encounters these in the communication protocols, especially those connected to proposals, but as scientist neither formulates nor applies the code of protocols, rules, and policies established by the bureaucracy.

The bureaucracy provides the B-agent with a set of protocols and rules that allow the B-agent to perform the application of a policy. What the B-agent learns is that if the current case is of this specific sort, then these particular protocols and rules ought to be used. As the B-agent quickly finds some measure of intelligence is needed to interpret and apply these, however. Further, the B-agent also finds that as the bureaucracy grows larger, either the B-agent must master ever increasing amounts of protocols and rules, or find that a new bureau with more restricted scope is created. In either case the B-agent must cope with changes in the materials that he must interpret

and apply. The skilled B-agent learns that if a bureaucratic change is of a certain type, then particular strategies for interpretation and application should be used.

The B-agents of the bureaucracy are most often organized into a hierarchy. This again is unlike the situation for E-agents. E-agents form a sort anarchy. Even in the case of the working scientist the organization of the scientists efforts are more anarchic than hierarchical. The scientist does, however, encounter a hierarchical and increasingly bureaucratic organization when he presents his research either as a proposal or a report. In both of these cases reviews and criticisms are often structured in such a way that various parts of the scientists efforts are examined by various B-agent to assure that protocols and polices are observed. These can range from the physical way in which the material is presented, through the use of terms and sources, to the content of the material. In each case the working scientist will encounter a specialized B-agent that is layered onto his work. The purpose of these agents is to ensure the quality of the uniformity and quality of the work. Thus, while a scientist may work in any way he deems appropriate on ay project he deems appropriate, the presentation of the research both for funding and acceptance demands that the scientists abide by certain protocols and policies. Various B-agents are both ready and able to ensure that this happens.

B-agents also participate in the establishment of policies and protocols and to some degree create a division of labor among both clients and other B-agents. In establishing the policies and protocols the B-agents are guided by both the goals and the previous history of the bureaucracy. They are often further guided by goals of efficiency and perhaps fairness. It should be clear that in the establishment of a policy some decisions must be made about which of these goals or perhaps other goals is primary. If the goals can be ranked then the rank ordered list of goals can be used as measure for proposed extensions and amendments of the protocols and procedures, as well as proposal for wholly new items. This process though apparently akin to that which the scientist used in solving problems, differs in important respects. Perhaps the most important respect in which there is a difference is in the normative quality of the protocols and policies. These require that the B-agent be familiar with the bureaucracy, but that familiarity does not necessitate any particular decision. The element of responsibility is a kin to that of insight in discovery. Thus, although policy making may in some ways resemble scientific problem solving, it is actually more akin to discovery.

The upshot of this examination of bureaucratic systems is that they can serve as a good model for larger systems in which there is a distribution of intelligent agents. The model can be flexibly applied since no reference is made to a specific domain. Finally, the division of function between B-agents and E-agents allows for systems modifications without severe disruption.

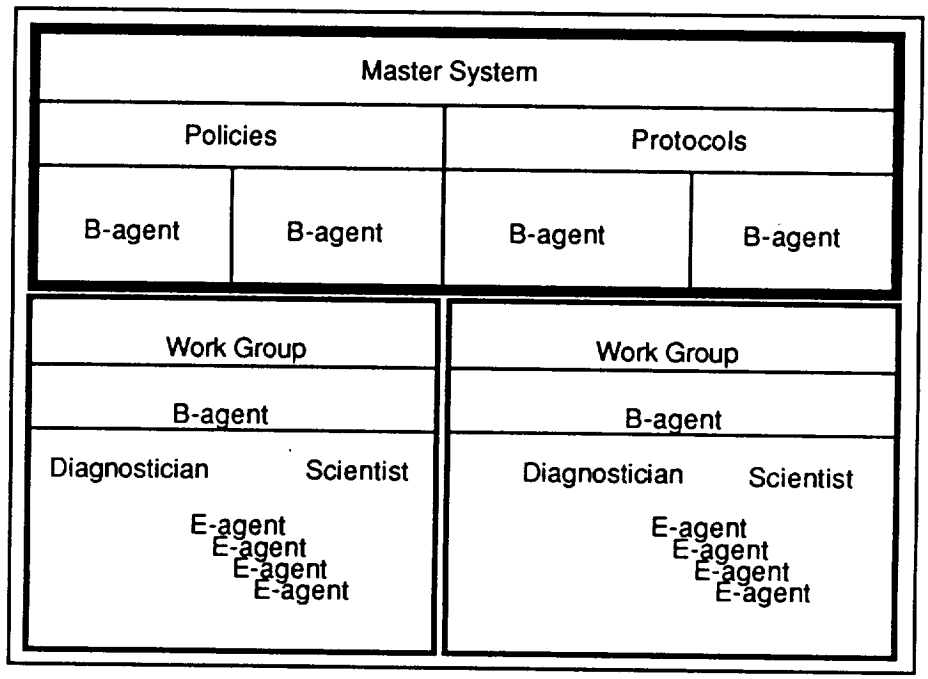
REFERENCES

- "Extensibility and completeness: an essay on scientific reasoning," D. Rochowiak. *The Journal of Speculative Philosophy*, 2(1988): 241-266.
- *Human Understanding*, S. Toulmin. Princeton University Press: Princeton, 1972.
- *Objective Knowledge*, K. Popper. Oxford University Press: Oxford, 1972.
- *Representing and Intervening*, I. Hacking. Cambridge University Press: London, 1983.
- "Scientific problems and constraints," T. Nickles. In *PSA 1978*, Philosophy of Science Association: East Lansing, 1978, pp. 134-148.
- *The Structure of Scientific Revolutions*, T. Kuhn. University of Chicago Press: Chicago, 1962.

A BUREAUCRATIC MODEL FOR MUTIAGENT SYSTEMS

INTRODUCTION

Distributed intelligent systems can be distinguished by the models that they use. The model developed in this report focuses on layered multiagent systems conceived of as a bureaucracy in which a distributed database serves as a central means of communication. In this section the various generic bureaus of such a system will be described and a basic vocabulary for such systems will be presented. In presenting the bureaus and vocabularies special attention will be given to the sorts of reasonings that are appropriate.



The bureaucratic system (B-system) is composed of a collection of E-agents and B-agents that operate in a cooperative way through a collection of protocols and policies. The E-agents like diagnosticians and scientists perform specialized services. These specialized services are monitored and facilitated by B-agents. Within the B-system that task decomposition is more-or-less fixed in terms of the bureaus or work groups of the bureaucracy.

In brief a bureaucratic model has a hierarchy of master system and work group that organizes the E-agents and B-agents.

The master system provides the administrative services and support facilities for the work groups. The goal of the master system is to stay in a stable state in which the communications between work groups can continue and the results of the work groups can be shared. The workgroups are collections of agents. The minimal workgroup would be composed of one B-agent and one E-agent. The administrative oversight and communications of the E-agent would be provided by the B-agent. Additionally, if there were several E-agents, the B-agent would provide the facilities for cooperation and especially result sharing among the E-agents.

In order to accomplish these goals the use of a DDB is essential. The DDB acts a central repository for information communicated by both the builder/designers of the system and the current states of the agents in the bureaucracy. The distributed database also serves to account for the actions of the agents and provide each agent with some account of the general nature of other agents. In order to generate cooperation among the agents of the system there must be specified protocols and policies. The protocols provide for the exchange of information and knowledge with out which the bureaucracy would collapse into chaos. Without the protocols no agent would know how to communicate and no agent would be able to share its knowledge. The policies of the master system establish the distribution of the tasks and the general modes of interaction among work groups. With out these no work group would know either what it was supposed to do nor the manner in which a task was to be done.

B - AGENTS

B-agents are characterized in terms of protocols and policies. The protocols are what allow for communication and the policies determine under what conditions and with whom the B-agent should communicate.

Protocols

The bureaucratic database protocols are simply the protocols that ensure transparency in a DDB. Since these tasks are essentially bureaucratic they would be performed by a B-agent within a work group. Such a B-agent would have at least the information and knowledge needed to act as a transaction manager. Thus one of the tasks performed by the B-agent in the work group is to implement the protocols of DDB. The READ and WRITE command terms would be the major ways in which a particular E-agent would address the workgroup's B-agent. The B-agent would in turn know how to issue requests and obtain the locks required by the DDB. The second set of

protocols are more complex insofar as they require some way in which other agents can be identified and decisions can be made about having access to them.

Policies

Given that a B-agent knows how to communicate the decision of when and with whom to communicate become important. These decisions are the policies of the B-system. Each B-agent will need to know the policy on which it should act.

As suggested earlier one way in which the other agents can be identified is through a frame like system in which the relevant data for each sort of agent is represented as a value for an attribute. Common sorts of agents would have common parent frames and these frames would be particularized as the specialization of the agent increases.

E-AGENTS

E-agents perform various functions. As indicated in the previous chapter E-agents might be thought of as either diagnosticians or scientists. In the model developed in this chapter that distinction will be preserved largely in the sorts of reasoning that the agents do. A diagnostic E-agent does two things: diagnosis problems and plans remedies. The scientific E-agent does several things some of which are akin to what the diagnostic E-agent does. The particular task of the E-agent is to simulate a physical system and solve problems. The problem solving strategies available to the E-agent may include case based reasoning, constraint propagation, and induction. Additionally, the scientific agent should have strong explanation facilities.

Diagnostic E-agents

A diagnostic E-agent corresponds most closely to a combination of classical expert systems and planning systems. Such systems might be able to operate in a largely rule based environment. The planning component can be handled by fitting into and amending a global schedule created in some other way. This is consistent with the assumption of layering for the MAS. The global schedule is given and the variations must be fit into it. In this sense the problem has already undergone some measure of decomposition.

The reasoning of diagnostic E-agents is largely rule reasoning. The two tasks of diagnosis and planning within a global plan, could be handles by the forward and backward chaining

mechanisms used in an opportunistic way. In this way the difference would be represented in the rule collections rather than the reasoning mechanisms.

Scientific E-agents

Scientific E-agents must have several means of reasoning at their disposal. The scientific E-agent works in a larger time frame than the diagnostic E-agent and primarily gathers knowledge about a physical system and attempts to explain its activity. In general the knowledge is acquired for the purposes of discovery and problem solving and the explanations are generated in response to requests by other agents. In general such requests would come from either another scientific E-agent or some B-agent.

The reasonings used by the scientific E-agent are diverse. However, they are in general directed by being appropriate to the task. Thus, case based reasoning might be used in cases of historical, analogical, or approximate reasoning, while inductive strategies might be used in the case of discovery. In most case there will also be some sort of simulation of the physical system which the agent can use as a base line. Ideally this simulation would be tuned as that scientific E-agent gains more knowledge of the physical situation. Finally, the agent should be able to explain the activities of the physical system to both a human user and other appropriate agents.

AGENTS

All agents would be generated from a primitive agent frame. The primitive agent frame would require only two slots:

```
FRAME: agent
        TYPE:
                RANGE: (B-agent or E-agent)
                DEFAULT: (none)
        LEVEL:
                RANGE: (master or workgroup)
                DEFAULT: (none)
```

B-agents

The frames for the B-agents would bring together the generic descriptions of the features that any B-agent must have. Among the generic features for a B-agent are the following:

```
FRAME: B-agent
SPECIALIZATION_OF: (agent)
        FUNCTION:
                RANGE: (protocol or policy)
                DEFAULT: (none)
        PROTOCOLS:
                RANGE: (list of protocols available)
                DEFAULT: (distributed database protocol)
        POLICIES:
                RANGE: (list of available policies)
                DEFAULT: (none)
        TASKS:
                RANGE: (list of available tasks)
                DEFAULT: (transaction manager)
```

One step lower in the inheritance structure would be the particular features of the specific B-agents. The frame for specific B-agents would include:

```
FRAME: specific B-agent name
SPECIALIZATION_OF: (of B-agent)
        PROTOCOLS_KNOWN:
                RANGE: (subset from PROTOCOLS attribute of B-AGENT)
                DEFAULT: (check DEFAULT attribute of B-AGENT)
                IF_ADDED: (check PROTOCOLS attribute of B-AGENT)
        POLICIES_KNOWN:
                RANGE: (subset from POLICIES attribute of B-AGENT)
                DEFAULT: (check DEFAULT attribute of B-AGENT)
                IF_ADDED: (check  POLICIES attribute of B-AGENT)
        WORKGROUP_ASSOCIATION:
                RANGE: (list of associated WORKGROUPS)
                DEFAULT: (none)
        E-AGENT_ASSOCIATIONS:
                RANGE: (unknown)
                DEFAULT: (none)
                IF_NEEDED: (get from WORKGROUP frames)
        TASKS_PERMITTED:
                RANGE: (subset from TASKS attribute of B-AGENT)
                IF_ADDED: (check TASKS attribute of B-AGENT)
        TASK:
                RANGE: (single element)
                DEFAULT: (none)
                IF_ADDED: (assemble POLICIES_KNOWN & PROTOCOLS_KNOWN and do TASK)
```

Within this general schema additional structures would contain the complete set of protocols and policies. The structure would be accessed when a specific B-agent is created. The specific B-agent would then acquire the needed protocols and polices. Since these would be held in a central structure there is a guarantee that all B-agents would use the same protocols and policies. In the simplest case a specif B-agent would acquire the protocols for acting as a transaction manager. If the transaction protocols were to change then whenever the B-agent went to act on these the current protocols would be used. Further, communications can be restricted between various workgroups

by restricting the protocols. This would also be a function of the level. For example, a B-agent at the master level would know protocols for sharing results among other master level B-agents, but not have a protocol for sharing results among other agents at the workgroup level. Or again, a master level B-agent would know the protocol for assigning tasks to workgroup B-agents, but would not have a protocol for assigning tasks to other master level B-agents.

The various policies that the B-agents can know would focus upon the decision to communicate, when to communicate, and with whom to communicate. These policies are the decision rules appropriate to the B-agent. Some B-agents might be given a policy of always communicating their results to the master level. Other policies may be imposed to enforce communication only when certain conditions are meet. Yet other policies may force a B-agent to communicate with a specific E-agent or force a specific E-agent to redo its task with new or additional information.

The reasoning structure for B-agents has two prongs. The first prong is more-or-less traditional. The knowledge represented in the policies and the protocols is used as a basis for doing some task. The inside of the policies and protocols will be calls to other frames or objects. For example, one policy may need information from an E-agent in a work group. The policy would for a message through a protocol to the E-agent in that workgroup. What is returned by that E-agent would the be given to the policy in the the B-agent through the protocol. The process of using the policies and procedures inside of the B-agent could represent different computational paradigms. All that is required is that the bureaucracy continue to function through policies and protocols. The second prong of reasoning concerns the ways in which one B-agent can reason about other B-agents. Again the assemblage of policies and protocols controls the activity of the B-agent. However in this case the reasoning is about the frame themselves rather than the information that can be provided from some other agent through its frame. In this sort of case there may be policies that allow an agent to make new associations, or to search for another agent with the right attributes and values; or to find an agent that is not busy that can do a particular task. These sorts of reasoning are distinct from the former sort since they require no direct access to the workings of another agent. It should also be remembered that in almost all case the B-agents will have to reason about the DDB, since by default any B-agent would know how to be a transaction manager.

E-agents

An E-agent is a specialization of the agent primitive.

FRAME: E-agent
SPECIALIZATION_OF: (agent)
    FUNCTION:
        RANGE: (diagnostic or scientific)
        DEFAULT: (none)
    DOMAIN:
        RANGE: (list of domains available)
        DEFAULT: (none)

Since E-agents are divided into kinds there will be general frames and specializations for each.

FRAME: E-agent kind
SPECIALIZATION_OF: (of E-agent)
    KIND:
        RANGE: (diagnostic or scientific)
        DEFAULT: (none)
    REASONING_PATTERNS_KNOWN:
        RANGE: (list of pattern names)
        DEFAULT: (none)
        IF_ADDED: (check DOMAIN attribute for consistency)

The further specialization of the E-agent kind generates a specific E-agent.

FRAME: specific E-agent name
SPECIALIZATION_OF: (of E-agent kind)
    WORKGROUP_ASSOCIATION:
        RANGE: (list of associated WORKGROUPS)
        DEFAULT: (none)
    B-AGENT_ASSOCIATIONS:
        RANGE: (unknown)
        DEFAULT: (workgroup B-agent)
        IF_NEEDED: (get from WORKGROUP frames)
    REASONING_PERMITTED:
        RANGE: (subset from REASONING_PATTERNS_KNOWN attribute)
        IF_ADDED: (check REASONING_PATTERNS_KNOWN attribute)
    CURRENT_REASONING:
        RANGE: (single element)
        DEFAULT: (none)
        IF_ADDED: (check REASONING_PERMITTED)
    CURRENT_TASK:
        RANGE: (unknown)
        DEFAULT: (none)
        IF_ADDED: (find and load relevant information)
    CURRENT_STATUS:
        RANGE: (busy, available, suspend)
        DEFAULT: (available)
    REPORTS_TO:
        RANGE: (agent in workgroup)
        DEFAULT: (B-agent for workgroup)
    REPORTS_FROM:
        RANGE: (agent in workgroup)
        DEFAULT: (B-agent for workgroup)

The idea of the specialized E-agent is that it acts as a shell for any particular task. Once the task is assigned and a determination is made of whether the task is one for a scientist or diagnostician, the specialized E-agent would conform to the dictates of the specifications. This would allow the B-agent for the workgroup to give tasks to the particular E-agent skeletons, if their current status is available. The reports are given to the workgroup B-agent if no other agent is specified. Further the E-agent receives information form the B-agent if no other agent is assigned.

With in the skeletal idea of the E-agent is also the notion that it permits certain kind of reasoning and is assigned certain kinds of tasks. It is the matching of one with the other that would allow an E-agent to respond to a request. This is in some ways a special consideration since the B-agent for the work group already knows what its E-agents can do and assigns the task accordingly. However, making the E-agents skeletal in the foregoing way allows for some notion of recovery or critical cases in which an E-agent might have to respond to the B-agent on the basis of what it knows about itself.

REFERENCES

• "Modeling coordination in organizations and markets," T. Malone, *Management Science* 33(10):1317-1332, 1987.

• "Offices are open systems," K. Hewitt. *ACM Transactions on Office Information Systems*, 4(3): 271-287, 1986.

• "An organizational view of distributed systems," M. Fox. *IEEE Transactions on Systems, Man and Cybernetics*, SMC_11:70-80, 1981.

• "The scientific community metaphor," W. Kornfield and K. Hewitt, *IEEE Transactions on Systems, Man and Cybernetics*, SMC_11:24-33, 1981.

CONCLUSION

This report has examined some of the issue connected to the development of a bureaucratic system. It has emphasized a layer multiagent approach to DAI and focused on the division of labor in a bureaucracy.

The bureaucratic model seems to be a fertile model for further examination since it allows for the growth and change of system components and system protocols and rules.

The first part of implement the system would be the construction of a frame based reasoner and the the appropriate B-agents and E-agents. The agents themselves should act as objects and the E-objects in particular should have the capability of taking on a different role.

The report has made no effort to address the problems of automated failure recovery, problem decomposition, or implementation. Instead what has been achieved is framework that can be developed in several distinct ways, and provides a core set of metaphors and issues for further research.