ICDM: Integrated Cooperative Decision Making - in Practice

Leonard Myers

CDM Technologies, Incorporated

Jens Pohl

California Polytechnic State University

Abstract

Multi-agent systems provide an attractive architecture for the implementation of complex systems. Much of the research is focussed on complete automation of the decision making process as a means of duplicating human abilities for working with new problems and environments. There is also a need for systems that employ the human as an agent and rely on human abilities for common sense and deep thought. The CAD Research Center at Cal Poly and CDM Technologies have significant experience in building systems of the latter type that assist human users in solving complex problems in planning, design and economics. This experience has generated a set of guidelines and a software development framework that have collectively been identified as ICDM. This paper presents a brief history of major applications built with the ICDM framework, and proposes some basic precepts.

1 Introduction

Making decisions is central to the course of action in any dynamic system. Much of the research in computerized decision making is intended to produce systems that can duplicate the decisions made by humans, particularly in the solution of complex problems. Such problems, commonly encountered in planning, design, economics, and management situations, are characterized by a set of distinctive attributes that do not lend themselves to a strictly sequential solution approach. These characteristics include many related variables, some of which are largely undefined, and dynamic changes in both solution requirements and strategies throughout the decision making process. The complexity of such decision making environments is not due to a high level of difficulty in any one area but the multiple relationships that exist among the many issues that impact the desired outcome. Since a decision in one area will tend to influence several other areas there is a need to consider many factors at the same time. This concurrency requirement places a severe burden on the human cognitive system and provides a strong incentive for computer-based assistance.

The models for decision making are typically a group of humans, such as a committee, board, or software development team, or the human brain, which may also be described in terms of member components. Commonly the components are called 'agents' and the focus of the research is on how the agents reason. A primary interest in the research is to make agents sufficiently independent to provide potentially important and unique contributions to the solution of a problem; yet, to see that these contributions are efficiently refined into a coherent response.

Multi-agent systems may provide an environment which is hostile to an individual agent. For example, a war game may support common communication between friend and foe agents, but the content of messages from a foe may be neither friendly nor reliable. Similarly, a multi-agent system that simulates a world-wide financial market must presume that agents are not necessarily working for the common good of the market players, but are likely to favor their own interests.

On the other hand, it is useful to develop systems in which the agents do cooperate to promote the common good of the system. These are 'friendly' cooperative systems. Further, within the set of friendly cooperative systems there is great variation in decision making facilities.

ICDM (Integrated Cooperative Decision Making) is a name used to reference the general guidelines and software framework employed by CDM Technologies and the Cal Poly CAD Research Center for the development of particular multi-agent applications.

1.1 History of ICDM

ICDM is the result of experience gathered during the development of a variety of applications using several multiple expert system platforms. A brief history will show how the ICDM guidelines and framework

1063-6730/94 $4.00 © 1994 IEEE

evolved. Then the key ideas will be stated and discussed.

1.2 PEBBLE

The implementation ideas for ICDM began with the development of the PEBBLE (Parallel Execution of Blackboard-Linked Experts) system [12]. Expert systems written in the PEBBLE language can interact with a data-blackboard housed in shared memory. PEBBLE was implemented on a parallel processing (multi-CPU) system, in order to permit expert systems to execute simultaneously. A required control expert developed for each PEBBLE application posts facts to the data-blackboard that identifies the current state of the problem solution and triggers actions within other experts.

These other expert systems, called domain experts, read each fact immediately after it has been posted to the data-blackboard and execute rules in parallel whenever their conditions are satisfied. The use of multiple CPUs in PEBBLE makes it unnecessary to explicitly schedule the execution of domain experts. The multiple CPUs quickly produce a rich set of reactions to trigger facts placed on the data-blackboard and provide a simple setting for control.

The primary lesson learned from PEBBLE was that the use of multiple CPUs with fast communication can eliminate programming concerns for scheduling process execution, which was the primary task of early blackboard systems This permits application developers to concentrate on problem-level, as opposed to machine-level, events. Experience in writing PEBBLE applications helped to formulate patterns by which large problems can be partitioned into domains and rule strategies for cooperation between those domains.

1.3 MARBLE

MARBLE (Multiple-Accessible Rete Blackboard-Linked Experts) [11] is a major extension of PEBBLE. In order to provide a more robust environment for programming rule-based expert systems, the CLIPS ('C' Language Integrated Production System) is used in MARBLE to replace the PEBBLE language and inference engine [13]. The graph-based inference engine developed for PEBBLE makes it possible for the rules in all experts to directly reference shared memory. However, the CLIPS expert system shell is intended for single CPU execution. Use of the CLIPS shell and the very nature of the Rete network [6] it implements makes a single shared factlist impractical. Therefore, MARBLE imposes a new paradigm for the data-blackboard.

In MARBLE the data-blackboard is distributed. A CLIPS shell runs in shared memory on each CPU. In addition, each shell is associated with a shared memory structure called an 'action descriptor' (AD). The action descriptor provides pointers, buffers, as well as status and action request flags that are used for communication between the shells. In addition to providing the actual communication data, flags are used as semaphores to guarantee reliable access and use of the data in all possible events within the parallel environment.

To post a fact to the data-blackboard, an expert invokes a function to place the address of the buffered fact into its AD. A control expert that constantly monitors the shared memory ADs manages the process of copying the request information into the ADs of all active experts, which then assert the fact into their own factlists. In effect, all of the experts share what is on the data-blackboard, even though they have their own copies. The implementation supports assertion, deletion, and modification of facts. Further, the logic is verified by a dynamic event state-transition diagram, to ensure that communication actions are finished before the information can be destroyed and to prevent deadlocks from occurring.

The distributed data-blackboard in MARBLE demands consideration of potential problems when an action requested by a domain expert is out of phase with respect to a data-blackboard fact. Thus, a problem might occur if a particular domain expert requests that a blackboard fact be modified, while in parallel the control expert asks for deletion of the fact. The domain expert might receive the request for deletion of the fact from control immediately after it has recommended the modification. On the other hand, this need not be a problem at all! When the domain expert receives the deletion request, it will simply delete the fact, and the modification request will be ignored by control when it discovers that no fact exits.

The potential for problems, such as the one described above, forces MARBLE application developers to consider how they are going to determine when a decision should be made. Here, this will be refered to as the 'timeliness' problem. MARBLE guarantees the integrity of the actions it is requested to perform with respect to the data-blackboard. It very loosely synchronizes the order in which the domain experts and the control expert can deal with data-blackboard facts. No expert can execute more than one local rule before processing all communication actions that are

pending.

Different applications developed in MARBLE use different approaches to deal with the timeliness problem. These applications demonstrate that the expert systems for some problems can be written so that timeliness is of little concern. Such systems are primarily forward-driven by discrete transitions in response to specific facts on the blackboard. The blackboard facts define the state of the system and are rarely deleted, although they are frequently modified. Time is involved only in terms of how long it takes for a transition to occur. Under these circumstances the system may simply wait for certain facts to be generated. (This is not unusual when the monotonicity of systems such as DENDRAL [7] is recalled.)

Differences in the PEBBLE and CLIPS-MARBLE capabilities force the application programmer to think of different ways to accomplish the same ojective. The differences also generate ideas for the implementation of new facilities. In general, the differences in language call attention to the influence that representation has on process.

Experience with MARBLE shows that the blackboard concept is a useful model for agent communication even when the blackboard data is distributed among the agents. The greater convenience in using MARBLE naturally leads to the development of more complex applications. The greater complexity of application in turn generates more interaction and a need for more types of interaction between agents.

1.4 ICADS-DEMO1

ICADS-DEMO1 is the first of several working models of ICADS, an Intelligent Computer-Assisted Design System [15]. It demonstrates a design system in which the user orchestrates the evolution of a building design solution with assistance from the computer. Emphasis is placed on the early design stages during which conceptual solution strategies are formulated and the framework for the entire design process established.

ICADS-DEMO1 is designed as a distributed CPU implementation of MARBLE, a multi-agent environment with a single coordination expert. Socket communication code replaces the use of shared memory for communication between the CPUs, which may be heterogeneous [17]. Otherwise, the concept of the distributed data-blackboard is the same. Specifically, ICADS-DEMO1 comprises: an existing commercial computer-assisted drawing system; a Geometry Interpreter (GI) capable of formulating architecturally meaningful geometric objects from point/line data schemas; relational databases incorporating prototypical building type information, site and neighborhood descriptions, and various reference data; an Expert Design Advisor consisting of six domain experts that automatically monitor the evolving design solution; and a Coordinator responsible for finding values that are compatible with the suggestions from the domain experts. The primary task of the system as a whole is to provide dynamic checking of constraints and dynamic generation of inferred results in the domain areas of knowledge about building design.

The size and complexity of the ICADS-DEMO1 application motivated the development of a protocol for CLIPS facts within a generic frame representation [1]. It also required the inclusion of the human user as the primary authority for resolving inconsistencies among the domain agents. The introduction of the user as an agent prompts the important, but subtle, reidentification of the MARBLE control agent as a 'coordination' agent. (This helps emphasize that ICADS-DEMO1 is an assistant to the user and not an autonomous designer.) Further, the resolution of difference between agents in DEMO1 is so complex that it requires new resolution strategies [19].

ICADS-DEMO1 provides many types of problems in producing decisions that converge to system solution. In particular: it requires a philosophy for treating the user as an agent; it focuses attention on determining when differences in domain suggestions are significant; it requires a method for preventing infinite cycles of resolution; and, it motivates a concern for the scalability of a single coordinator in a multi-agent environment.

1.5 ANT-FARM

ANT-FARM is a series of projects used to study various communication and coordination methods for a multi-agent environment [2]. Execution of the system produces a realistic graphic portrayal of ant behavior. Currently there is a single nest from which ants emerge. Each ant is implemented as an agent with the same instinctive behavior facilities. The action of an ant is somewhat random, except for response behavior that is triggered by the recent presence of ants in the near vicinity. Scent and food excitement levels are dynamically adjusted with respect to time and decrease with distance. As ants discover a source of food and find their way back to the nest, the path from nest to food and back becomes easier for the ants to determine. Within a few minutes of operation, there is a fairly regular line of ants between the nest and food. The instincts are modeled from the

literature on ant behavior and produce a very realistic result.

In addition to a variety of experiments in driving the ant agents from parallel, distributed, and single CPU platforms, ANT-FARM serves to demonstrate how the appearance of cooperative behavior can take place without making the agents aware of their contributions to the rest of the system. Similar to research in autonomous agents reported by Brooks [3], Wilson [18], Meyer [9], and Maes [8], the ANT-FARM experience demonstrates how clever encoding of basic actions can achieve behavior goals that are not explicitly identified.

1.6 ICODES

ICODES (Integrated Computerized Deployment System) refers to a continuing project which is currently in a proof-of-concept working system form [4]. It is a comprehensive system that will help both novice and experienced 'stowers' to plan the loading of cargo onto ships. Like the ICADS prototypes, ICODES features interactive graphics within a framework of multiple expert systems that can either monitor the manual stowing of cargo or automatically stow a subset of the entire cargo list. Cargo icons are automatically displayed within the ship drawing or dragged into position by the user. The user selects the actions to be performed and can limit the automatic operations to alternate between manual and assisted actions. Violations generated by manual actions are presented graphically and a rich set of query and report facilities are supported. Manual actions can undo any of the assisted actions.

The implementation of ICODES required new ICDM facilities. In addition to a richer frame representation than that employed in DEMO1, more flexible communicatioin between agents is employed. A new programmer-friendly distributed communication package based on the independently developed Mercury Message System [16] is being developed. Specifically, the access agents that determine whether a given cargo item can be driven, pulled or lifted into a ship compartment benefit from direct communication with each other. Only the result of their discussion needs to be known by other agents. As a result the framework for ICODES is more characterized as a multiple agent architecture than specifically a blackboard system.

1.7 Object-Agents

The newest research in the CAD Research Center is associated with the concept of 'object-agents' [10]. In this work there is not necessarily a data-blackboard. The agents are more like those in standard cooperative, distributed computing systems [5], but with one significant difference. In the object-agent applications, the implementation of software objects in languages such as C++ is enhanced with code that permits objects to act as agents.

Currently the ICDM framework from the ICADS prototypes is being used to support experimentation with object-agent applications to architectural design [14]. The basic ideas collectively called ICDM continue to provide the guidelines for the behavior of objects in an agent environment. Hopefully this approach will make it easier to incorporate as agents objects that have been previously defined in languages such as C++. If so, a significant increase in productivity could be realized when existing object systems are extended into a cooperation paradigm.

2 ICDM Guidelines and Framework

The following guidelines are based on what was found to work well in the applications that the CAD Research Center has implemented in recent years. They are presented for consideration particularly by others who are considering the implementation of multi-agent systems that can work efficiently in large systems. Human agents are expected to provide the common sense reasoning and deep reasoning that may be required when the system has a problem. The system agents themselves use strong specific inferencing. One warning is appropriate. Context is essential! A particular precept may be inappropriate if the goals or implementation environment are substantially different from the applications listed above.

1. Model objects with natural language.

Generate an object model of the application in which a complete vocabulary of words natural to the application is embedded. In particular, identify every item or attribute that must be known in order to describe a solution to the problem. The communication among agents must be within the common vocabulary. Of course, the agents may use local domain words to make inferences or compute values for the common vocabulary, but the local words should not be communicated to other agents. Generally, the common vocabulary consists of what the system must be able to report to its users and what must be communicated between agents.

2. Create object or frame representations.

The representation must be efficient for computation, inferencing, and communication. Preferably it should be possible to easily translate the representation into a relational database format. Minimize the amount of information that must be transmitted in order to make a suggestion for the value of a single item or attribute. Provide a 'source' slot or attribute to identify the source agent of the transmission, provide a 'type' slot or attribute to identify whether the value is a suggestion or the current solution value. (The set of attributes whose type slots are marked as current solution values identifies the state of the system solution at any point in time.) Do not permit any attribute to receive a current solution type from more than one source agent. If necessary, create new attribute names to maintain this sole source precept.

3. Provide robust communication facilities.

Agents can translate the common representation into other internal forms, but the requirement of common communication makes it possible to dynamically link agents.

4. Minimize communication bottlenecks.

Partition the common communication. There is no need for every agent to receive all of the common vocabulary. Develop a mechanism by which agents identify the input they can use and constrain communication to this set. There are a variety of architectures with which this can be achieved. It is best to provide agent-to-agent as well as broadcast (to all agents) communication. A router built within the communication hub for each CPU in the system could handle the messages between the agents it hosts. For external agents, the same router can communicate directly with the hubs on their CPUs. Each agent can register an 'input template' to identify its input values with its hub.

In a single CPU system there is simply one hub and all routing takes place within it. In a multiple CPU system the assignment of agents to CPUs must take into consideration the expected frequencies of their communication with other agents. Also in multiple CPU systems, broadcasts can be performed by a single hub-to-hub message to each of the other hubs which then distribute the information to the agents they host, in accordance with the input templates.

5. Group units of communication.

In order to activate the best response in an agent, it is often necessary to prohibit the agent from reacting to individual message information in the order it is received. Instead, the sender can group the information units together and expect the receiver to react only after the group has been received. In a rule-based system this makes it possible to write rules that respond to many possible combinations of input and use priorities to select the best of all the rules activated by grouped input.

6. Make decisions correctly.

Make decisions at the lowest level possible. Be sure that all necessary input will be available to the extent possible. Refrain from making a decision when insufficient information is available. Instead, let the absence of a decision be available to the system explanation facilities that can help the user correct the problem, or make a decision for the agent. Capture and analyze the reasons for inadequate agent suggestions. Often, an agent simply requires input that it has been denied.

7. Generate feed forward waves.

Suggestions and decisions should feed forward in terms of setting the system solution attributes. Of course, there are many occasions when new actions within the system will generate values for previously unassigned attributes or new values for previously assigned attributes. These new values should also feed forward and form a new wave of response within the system.

The vocabulary and the agents should be designed to generate forward dependencies. The 'line of command' management practice, still used in many American institutions, is simply not efficient. However, current bottom-up and top-down communication practices are effective if the top-down information proceeds all the way to the bottom and the response is returned through the natural bottom-up activities. Some circumstances may permit a breaking of the feed forward directive, but such deviations from the general rule should be implemented only after careful consideration. In most cases the need for breaking the directive can be eliminated by moving some decisions to lower levels, or by introducing new information to lower level decision makers, or by delaying a lower level decision until more information is received. These actions will usually allow better decisions to be made at lower levels.

8. Prevent repetition of cyclic behavior.

It is likely that 'making waves' can result in a situation where one or more successive waves produce a current solution attribute that will cycle through the same set of values. This is easily detected and prevented. A count field and a limit field may be included in the representation of the attribute. Whenever the coordinating agent that is authorized to set the current value of the attribute makes a change in its value field, it can also update the count field and check the limit field. If the limit has been exceeded, help can be requested from a human agent. The application should include friendly, interactive facilities to help the human user understand the problem and gather sufficient information to make a decision. Furthermore, the user may select a reset of the count field or maintain its value. If the count is reset then, depending on the actions of the user, it is of course possible that the cycle will repeat. If the limit violation is retained, it can be used to prohibit further changes.

The use of forced values set by a user is generally an action encouraged only to permit continuation of the currently executing system. The system should log the forced incident and this record should be used to provide a more permanent improvement.

9. Isolate the consideration of alternatives.

Even when good decisions are made by a system, it is often desirable to ask 'What if?' types of questions. The isolation of alternative computational sequences helps to minimize the use of system resources. In particular, the performance of expert system inference engines is typically degraded by large increases in facts. Thus it may be preferable to use multiple engines, rather than a single engine that uses pattern matching to focus attention on one alternative.

Explicit facilities for handling alternatives should be provided. For example in SARDINE [10], a system that the CAD Research Center is currently developing as part of an ICDM-Kernel project, the design calls for a duplicate ICDM environment across a distributed network by managing forked copies of the common processes. Any copy can be executed with different actions to explore alternatives, but the forked processes that are not part of the executing copy are asleep and use few system resources. If the system resources permit, it might even be desirable to concurrently execute more than one copy of the whole system.

10. Maximize simultaneous response.

The use of multiple agents with properly selected domains can maximize the suggestions to be used in making a decision in a small amount of time. Long running algorithms may require new formulations that will provide intermediate values that can be shared with other agents to detect problems at an early stage.

11. Also generalize decision making.

A major emphasis of cooperative multi-agent computing research is the development of strategies for making decisions in general. Specifically, the intention is to produce mechanisms that will provide solutions to many problems, including 'new' problems that were not identified when the agents were created. Often, the low-level decisions made by an agent are separated from the high-level decisions that are made for the whole system. The low-level, or application domain, decisions are characterized by the use of application-specific information. The high-level, or system, decisions are characterized by more general decision making strategies that do not depend on application-specific information.

The same guidelines and framework that have been described for low-level decisions apply to high-level decisions. However, the domains of high-level agents may include vocabulary items that are not usually identified with the application itself.

3 Summary

The ICDM framework and its implementation in the various applications described in this paper, are indicative of the authors' conviction that decision support systems in which human users and computer-based agents work in partnership are highly desirable. The decision making activity presumes an element of the unknown, a problem that has to be solved through a decision making process that cannot be completely predefined because of incomplete information and dynamic information changes. Under such conditions, the ability of the human partner to apply intuition is a welcome and necessary complement to the logical capabilities of the computer-based agents.

In order to build large interactive cooperative systems a number of guidelines are necessary in order to drive implementation decisions. The guidelines presented in this paper have been successfully used in a series of working applications developed over many years of work.

References

[1] H. Assal and L. Myers, "Implementation of a Frame-Based Representation in CLIPS," *Proc. First CLIPS Users Conference*, Johnson Space Center, NASA, Houston, TX, August 13-15, 1990.

[2] W. Bock, *A Simulation of the Food Gathering Behavior of Ants,* Senior Project, Computer Science Dept., Cal Poly, San Luis Obispo, CA, 1994.

[3] R.A. Brooks, "Intelligence without Reason - Computers and Thought Lecture," *Proc. IJCAI-91*, Sydney, Australia, 1991.

[4] CAD Research Center, *ICODES: Integrated Computerized Deployment System,* Final Report of Stage(1), prepared for Naval Civil Engineering Laboratory and Military Traffic Management Command, US Department of Defense (contract CAD Research Center, College of Architecture and Environmental Design, Cal Poly, San Luis Obispo, CA, March, 1994.

[5] E. Durfee, *Coordination of Distributed Problem Solvers* Kluwer Academic, Boston, 1988.

[6] C.L. Forgy, "Rete: A fast Algorithm for the Many Pattern," *Artificial Intelligence*, 19(1), 1982.

[7] R. Lindsay, B.G. Buchanan, E.A. Feigenbaum, and J. Lederberg, *DENDRAL,* McGraw-Hill, New York, 1980.

[8] P. Maes and R. Kozierok, "Learning Interface Agents," *Proc. AAAI-93: The Eleventh National Conference on Artificial Intelligence*, MIT Press, 1993.

[9] J.A. Meyer and A. Guillot, "Simulation of Adaptive Behavior in Animats: Review and Prospects," *Proc. First International Conference on the Simulations of Adaptive Behavior*, Meyer and Wilson (eds.), MIT Press/Bradford Books, 1991.

[10] L. Myers, J. Pohl, J. Cotton, J. Snyder, K. Pohl, S. Chien, S. Aly, and T. Rodriguez, *Object Representation and the ICADS-Kernel Design,* Technical Report, CADRU-08-93, CAD Research Center, Design Institute, College of Architecture and Environmental Design, Cal Poly, San Luis Obispo, CA, January, 1993.

[11] L. Myers, C. Johnson, and D. Johnson, "MARBLE: A System for Executing Expert Systems in Parallel," *Proc. First CLIPS Users Conference*, Johnson Space Center, NASA, Houston, TX, Aug.13-15, 1990.

[12] L. Myers, T. Cheng, D. Erickson, L. Nakamura, T. Rodriguez, R. Russett, and T. Sipantzi, "PEBBLE: Parallel Execution of Blackboard-Linked Experts," *Proc. SURF Conference*, Newport Beach, CA, Sept, 1988.

[13] NASA, *CLIPS Architecture Manual (Version 4.3),* Artificial Intelligence Section, Lyndon B.Johnson Space Center, TX, May, 1989.

[14] J. Pohl and L. Myers, "A Distributed Cooperative Model for Architectural Design," *Symposium on Knowledge-Based Systems for Architectural Design*, Consiglio Nazionale Delle Ricerche, Rome, Italy, May 13-14, 1993: in Knowledge-Based Computer-Aided Architectural Design, (eds. Carrara and Kalay), Elsevier, Amsterdam, (pp.205-242), 1994.

[15] J. Pohl, A. Chapman, L. Chirica, R. Howell, and L. Myers, "Implementation Strategies for a Prototype ICADS Working Model," *Technical Report, CADRU-02-88, CAD Research Unit, Design Institute*, School of Architecture and Environmental Design, Cal Poly, San Luis Obispo, CA, 1988.

[16] K. Pohl, "MERCURY: A Real-Time Message Management Facility for Distributed Cooperative Computing Environments," *Proc. Advances in Computer-Assisted Building Design Systems*, Focus Symposium: 4th International Symposium on Systems Research, Informatics and Cybernetics, Baden-Baden, Germany, August 2-5, (pp.155-164), 1993.

[17] J. Taylor and L. Myers, "Executing CLIPS Expert Systems in a Distributed Environment," *Proc. First CLIPS Users Conf.*, Johnson Space Center, NASA, Houston, TX, August 13-15, 1990.

[18] S.W. Wilson, "The Animat Path to AI," *Proc. First International Conference on the Simulations of Adaptive Behavior*, Meyer and Wilson (eds.), MIT Press/Bradford Books, 1991.

[19] W. Wuilleumier, *A High Level Strategy Approach to Knowledge Acquisition,* Master Thesis, Computer Science Dept., Cal Poly, San Luis Obispo, CA, 1992.