A Multiagent Architecture for a Distributed Artificial Intelligence Application

G. Hartvigsen and D. Johansen

Department of Computer Science,
University of Tromsø,
N-9000 Tromsø, Norway

Abstract

Artificial intelligence applications are normally complex, demanding much computing power. This paper presents the multiagent architecture of StormCast, a distributed artificial intelligence application for severe storm forecasting. Important objectives as scaleability, robustness and reusability can be met by partitioning an overall task in subtasks, distribution of the subtasks on different nodes, parallel processing of the subtasks and cooperation between the subtasks.

Keywords

Distributed application; distributed artificial intelligence; expert system; weather forecasting.

1.0 INTRODUCTION

A basic goal in previous research projects within distributed artificial intelligence (DAI) has been the achievement of a high degree of time-critical computing. However, in several problem domains, such a requirement can be superfluous. By a limited reduction in the quality of the reasoning process, a lot of problems might be avoided.

This approach has been at focus in the design and implementation of the StormCast DAI application predicting severe storms in the Nordic part of the Northern hemisphere (Hartvigsen and Johansen, 1988, 1989). Our main strategy has been to decompose the application in a set of cooperating modules to meet the distributed nature of this kind of application. Scaleability, robustness and reusability are important objectives of this kind of application.

This paper presents how the StormCast application has been designed and implemented to meet these objectives. The design and the source code have been kept as simple as possible without main losses in the functionality. A control approach for this kind of DAI applications is presented, along with a discussion of this approach.

2.0 A MULTIAGENT ARCHITECTURE

In the StormCast project, a multiagent architecture has been developed to meet the distributed nature of this kind of application. As shown in fig. 1, StormCast is hierarchically organized into two main layers; a data collection layer and a knowledge layer (Hartvigsen and Johansen, 1988). The data collection layer consists of the monitoring and synthesizing modules (Johansen, 1988). The knowledge layer includes the transmission and expert module.

StormCast has been built on a software base where each module is implemented in C or CommonLisp as a UNIX process using current standards as TCP/IP and the X Window system. The hardware consists of Motorola 68030 workstations (Hewlett-Packard 9000/360 CH) using Ethernet-based LAN locally and X.25 for WAN connections. Altogether this makes a proper platform to build distributed applications.

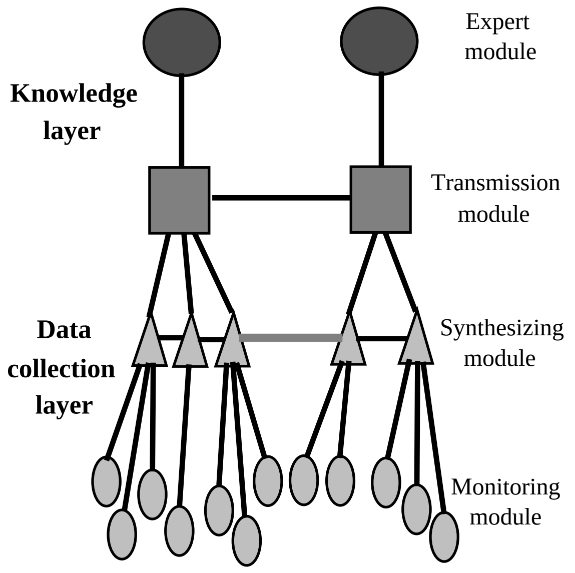

FIGURE 1. The multiagent architecture of StormCast.

2.1 Partitioning

The StormCast application is partitioned in a set of cooperating modules (agents) to match the distributed nature of this kind of application. The cooperating modules are continuously collecting and processing weather data from a fixed geographical area. In each geographical area, there is an expert module (i.e. expert system) responsible for the prediction of, in this case, severe storms. Each expert module has the knowledge and intelligence needed to make a severe storm forecasting. This severe storm forecast is based on the results achieved from the monitoring modules in their own area. In this way, the problem solving emphasizes intelligent local control of each expert module (problem solver) (Hartvigsen and Johansen, 1988, 1989).

The different expert modules are primarily concerned with the forecast of severe storm in its own region. Each problem solver is thus self-interested because the local data is considered to be most important in the problem solving.

2.2 Distribution and parallelism

The partitioned tasks in StormCast are distributed on nodes in the different domains StormCast operates in. The idea is to have enough modules in each domain to solve problems naturally belonging here, but where each domain can operate in parallel. Monitoring of weather data and predicting of weather for the different domains can be done simultaneously.

The conceptual basis for concurrent problem solving underlying artificial intelligence has been heavily discussed. In DAI, which is concerned with problem solving in which groups solve tasks, these fundamental aspects are closely related to the problem of cooperation. A general suggestion within distributed applications involving highly interdependent tasks, has been the requirement for sophisticated control mechanisms to promote effective cooperation.

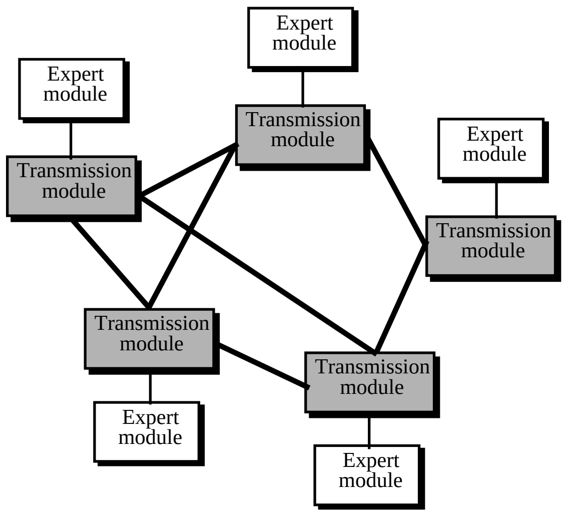

FIGURE 2. The network of expert modules.

As illustrated in fig, 2, a network of expert modules performs distributed problem solving by cooperating as a team to solve the same problem more properly. The cooperation requires that each expert module knows which solutions to communicate. Such networks are typically utilized in this kind of distributed sensor networks (Lesser and Erman, 1980; Smith, 1980; Wesson et al., 1981; Lesser and Corkill, 1983; McClelland and Rumelhart, 1987).

2.3 Cooperation

Cooperation among self-interested problem solvers is based on the assumption that this is in their own interest (Durfee, Lesser and Corkill, 1987a). In weather forecasting, the meteorologists working in regional institutes make their predictions upon their local

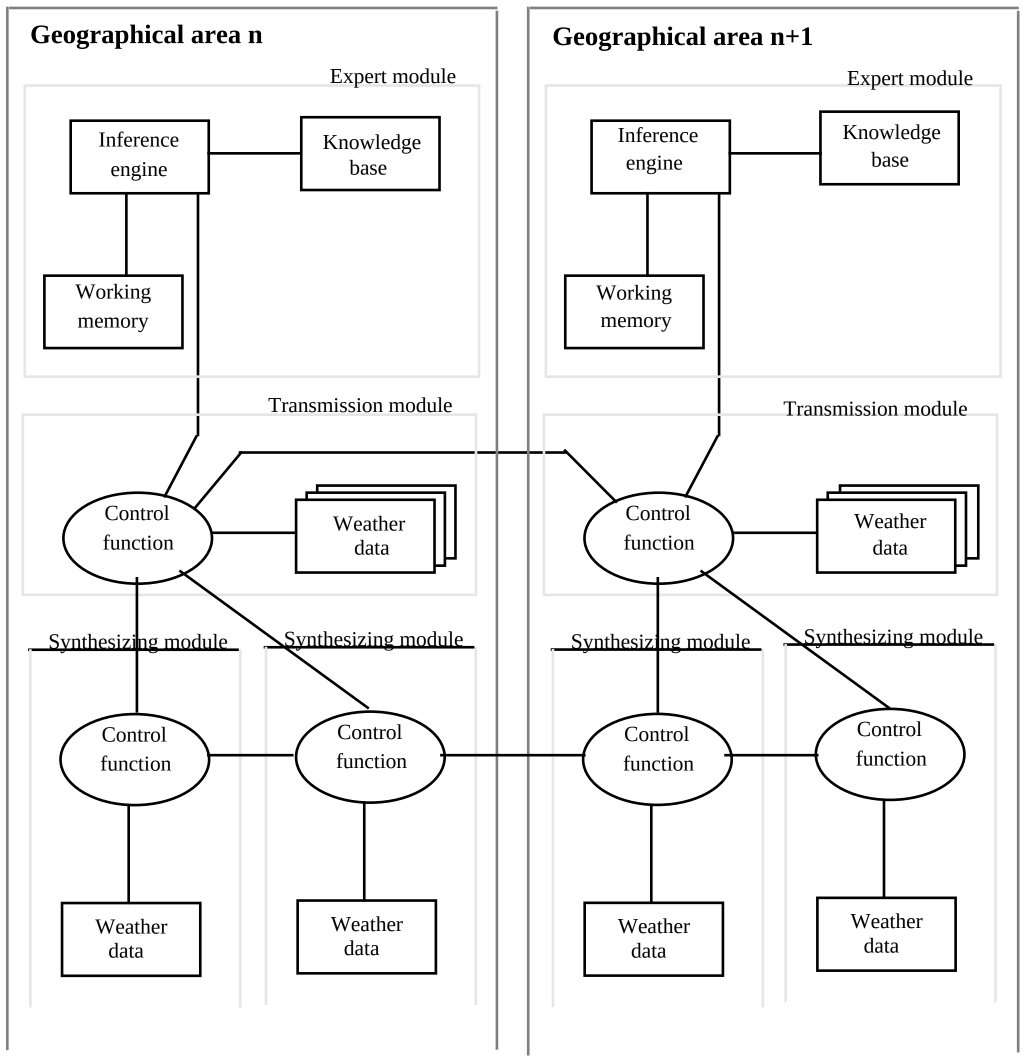

FIGURE 3. Parts of the system architecture in StormCast.

weather observations and data and weather forecasts received from other institutes. The employment of the results received from other institutes depends on how the results fit their own results. The task of obtaining such coherent cooperation is considered to be a difficult task (Lesser and Corkill, 1981; Davis and Smith, 1983).

Cooperation is a fundamental aspect in DAI, and lot of research has been done (e.g., Erman and Lesser, 1975; Smith and Davis, 1981; Fikes, 1982; Cammarata, McArthur and Steeb, 1983; Axelrod, 1984; Genesereth, Ginsberg and Roseschein, 1984; Durfee and Lesser, 1987; Durfee, Lesser and Corkill, 1985, 1987a, 1987b, 1989; Durfee, 1988; Zachary, Robertson and Black, 1988). Cooperation in DAI means to have expert systems working together towards solving (a) common problem(s). This calls for the development of *cooperative interaction mechanisms* that allow multiple expert systems to participate as a teamwork (see fig. 3). The design of a distributed system requires the selection of an organizational structure, i.e. processes (modules) and communication paths, and a control regime (Fox, 1981).

As similar to Herasay II (Fennell and Lesser, 1977), we have utilized a passive data structure (the blackboard) which contains the current state of the problem solution. Due to the physical distance between the expert modules, StormCast employs several such *simple blackboards,* instead of one global blackboard. Physical access to the blackboards is taken care of by a control function (in Hearsay II; the blackboard handler module), whose primary function is to accept or reject (i.e. postpone) requests from the active processing elements to read and write parts of the blackboard. The major difference between a blackboard in the traditional sense and the blackboard in StormCast, is that we do not use the blackboard directly in the reasoning process as a working memory. In StormCast, we view the blackboard more as an intermediate working memory.

3.0 SIMPLIFYING APPROACH

As mentioned previously, in StormCast we have left the centralized cooperation control mechanisms and focused on a distributed approach. This have forced the development of our own cooperation approach, which we refer to as the simplifying approach. Simplified refer to the fact that a complex module is decomposed in a set of much less complex modules. However, this strategy requires a more complex control regime due to the need to communicate data and synchronize different events.

The core concept in the simplifying approach is the time stamping of data with the local time at that node, and then accept both delivery within the near future without explicitly to promise the reception of the data, as well as the acceptance that the data can get lost, e.g., as a result of a local break down. The different clocks are loosely synchronized. In our approach, it is assumed that the clocks are synchronized within 60 seconds. If experience with more weather domains shows that this loose synchronization is inappropriate over time, approaches as (Lamport, 1978) can be taken.

In StormCast, sufficient ordering of events is done by time-stamping the data with local-time values. There is small problems with updates arrived at a node not in time-stamped order since the real time clock is used. Normally, a message from time t_0 might arrive at a node later than a message sent at time $t_0 + 1$. It is up to the expert system to determine whether data time-stamped at time t_0 or $t_0 + 1$ is to be used in the weather prediction. In fact, both values might be valid if not too old.

In addition, our approach is based on the philosophy of exchanging partial solutions. This means that a solution is reached by an iterative process of exchanging preliminary, partial solutions, and that the solutions, i.e. severe storm forecasts, in StormCast are partial for the receivers but not for the senders. If we look at the research arenas which the simplifying approach can be based on, distributed problem solving (DPS) and multiagent (MA) systems, applications in both arenas seem to explore complex time-critical cooperating approaches. However, the utilization of such approaches may not be necessary in all situations, as in our weather forecasting application.

The simplifying approach represents a suitable solution for weather forecasting. Since the data is exchanged over a very wide geographical area (Hartvigsen and Johansen, 1988), it is not recommendable to have a tight connection between the expert modules. However, each expert module must contain information concerning the others (organizational knowledge), e.g. geographical location in order to give the received results the correct weight in its own forecasting process. In this way, an expert module becoming isolated does not affect the rest of the system as it would if the forecasting process in other parts of the system were delayed or even halted as a result of this accident.

There is no problem with information growing over time exhausting available storage at a node. Updates older than a given time-stamp no longer needed can be discarded. In addition, old data is also discarded by simply overwriting with the new value and its time-stamp.

4.0 DISCUSSION

A fundamental part of the design phase in the StormCast project was the design of the “database” for the intermediate storing of weather data and forecasts from other expert modules. Initially, a blackboard architecture was chosen. However, unlike the traditional blackboard architecture model which is centralized, we have chosen a distributed blackboard architecture. In a centralized blackboard, no comparative autonomy exists between the knowledge sources (KS’s). Even if the KS’s were executed on different processors, the access to the key resources that drives the system’s computational behaviour is inevitably serialized (Cromarty, 1987). This calls for frequent, high-bandwidth access to the shared data space. The part of the blackboard which operates as a working memory is moved into the expert module running on the same machine. The utilization of this kind of philosophy in a distributed environment has several important implications:

- *Minimal restrictions of the implementation.* Through the specification of the data structure which are exchanged between the different modules, the modules themselves can easily be rewritten in another language, e.g. to meet future requirements to execution speed.
- *Reduction of synchronization.* Since there is no global shared data, the system do not need to consider shared-access synchronization operations. However, reduced synchronization have led to duplication of data.
- *Parallel execution of cooperating modules.* The StormCast architecture provides true parallel execution by running on physically separated nodes.
- *Reusability.* Scaling of the application is done by reuse of existing modules downloaded on nodes in the new domain. Reuse of existing code is done whenever a new weather domain is added to StormCast. However, the knowledge base in the expert module has to be adapted according to local weather conditions.
- *Robustness.* One approach is to reduce the dependency of predictions from other nodes. However, this approach is not sufficient to obtain a fault-tolerant application. Our current approach is to use N-version programming replicating fault-potential modules (Chen and Avizienis, 1978).

In StormCast, control of cooperation among the modules is decentralized and *implicit* in the autonomous behaviour of the individual modules. Each expert/transmission module utilizes local knowledge on how to act if certain situations should occur and what information to multicast. In a more sophisticated approach, the system could possibly warn (or invoke) its neighbours if the local weather conditions change rapidly.

A paradox in the implementation of the simplifying approach is that despite the overall goal of simplification, the severe storm forecasting process itself has become more complex than if a rendezvous coordinating mechanism should have been employed. As illustrated in fig. 4, this problem may be presented by using sets.

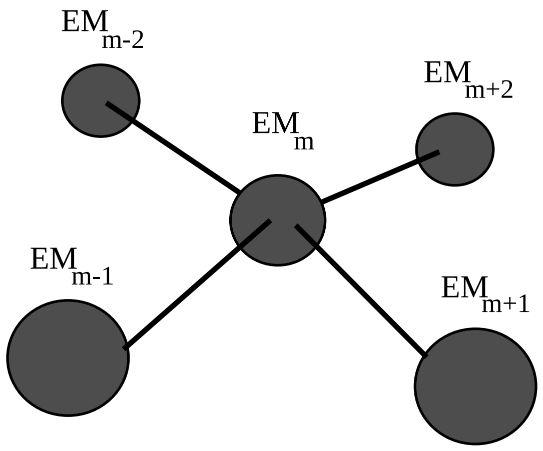

FIGURE 4. Relevance of multicasted information.

Given the set

$$S = \{EM_1, ..., EM_m, ..., EM_n\},\ 1 < m < n \qquad \text{(EQ 1)}$$

Then the distribution of information from expert module m (EM_m) at time t partly constitutes the basis for the predictions of its neighbours $S - \{EM_m\}$ at time $t + \varepsilon$. But at time $t - \varepsilon$, the results multicasted from the neighbours $S - \{EM_m\}$, were utilized by the EM_m. This indicate that the expert module in its own forecasting process must pay less attention to its neighbours forecasts, even if the physical distance between the EMs should be close, i.e. that neighbour results isolated would be of great interest.

5.0 CONCLUDING REMARKS

In the StormCast project, our main strategy in the design and implementation of the system has been to keep the design and the source code as simple as possible without main losses in the functionality. A modular design together with the utilization of de facto industrial standards has appeared to be sufficient to build a DAI concept. In addition, this modular design also makes scaling, reusability of software, and fault-tolerance possible. Adding a new domain to StormCast is straightforward, existing code is mainly copied and downloaded to the new domain of interest.

Some AI applications normally demanding high-performance mainframes are distributed in its nature born to true parallel processing. Weather forecasting is one computational domain where a complex task can be partitioned in subtasks, distributed on several nodes for parallel execution. Cooperation between these subtasks can be done to meet the functional requirements of the application. This cooperation increases network traffic, but as a whole, this multiagent architecture is well suited to a hardware base of interconnected computers as workstations.

As a result, complex AI applications can be constructed for a loosely coupled environment of i.e. workstations in stead of expensive mainframes. StormCast is design to effectively utilize the aspects of true parallel processing of distributed tasks. This is achieved through a simple, modular design. We believe that this kind of multiagent architecture can be regarded as desirable in the design of similar AI applications.

ACKNOWLEDGEMENT

Availability of the NAVF (Norwegian Researh Council for Science and the Humanities) supported distributed system laboratory at the Department of Computer Science, University of Tromsø, has stimulated the development of distributed applications. Discussions with members of the TRACE project at the University of Tromsø have been valuable, especially comments from O.J. Anshus and A. Munch-Ellingsen.

REFERENCES

Axelrod, R. (1984). *The evolution of cooperation.* Basic Books.

Cammarata, S., D. McArthur, and R. Steeb (1983). Strategies of cooperation in distributed problem solving. In Proceedings of the 8th International Joint Conference on Artificial Intelligence. Morgan Kaufmann, Los Altos, California. pp. 767-770.

Chen, L., and A. Avizienis (1978). N-version programming: a fault-tolerance approach to reliability of software operation. In Proceedings of the Eight Annual International conference on Fault-Tolerant Computing (Toulouse, France, 21-23 June 1978). IEEE, New York. pp. 3-9.

Cromarty, A.S. (1987). Control of processes by communication over ports as a paradigm for distributed knowledge-based system design. In L. Kerschberg (ed.), *Expert Database Systems. Benjamin Cummings.* Menlo Park, California. pp. 91-103.

Davis, R., and R.G. Smith (1983). Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence, 20,* 63-109.

Durfee, E.H. (1988). *Coordination of Distributed Problem Solvers.* Kluwer, Boston.

Durfee, E.H., V.R. Lesser, and D.D. Corkill (1985). Increasing coherence in a distributed problem solving network. In Proceedings of the 9th International Joint Conference on Artificial Intelligence. Morgan Kaufmann, Los Altos, California. pp. 1025-1030.

Durfee, E.H., and V.R. Lesser (1987). Using partial global plans to coordinate distributed problem solvers. In Proceedings of the 10th International Joint Conference on Artificial Intelligence. Morgan Kaufmann, Los Altos, California. pp. 875-883.

Durfee, E.H., V.R. Lesser, and D.D. Corkill (1987a). Cooperation through communication in a distributed problem solving network. In P.M. Huhns (Ed.), *Distributed Artificial Intelligence.* Pitman, London. pp. 29-58.

Durfee, E.H., V.R. Lesser, and D.D. Corkill (1987b). Coherent cooperation among communicating problem solvers. *IEEE Transactions on Computers, C-36,* 1275-1291.

Durfee, E.H., V.R. Lesser, and D.D. Corkill (1989). Trends in cooperative problem solving. *IEEE Transactions on Knowledge and Data Engineering, KDE-1,* 63-83.

Erman, E.D., and V.R. Lesser (1975). A multilevel organization for problem solving using many, diverse, cooperating sources of knowledge. In Proceedings of the 4th International Joint Conference on Artificial Intelligence. Morgan Kaufmann, Los Altos, California. pp. 483-490.

Fennell, R.D., and V.R. Lesser (1977). Parallelism in artificial intelligence problem solving: A case study of Hearsay II. *IEEE Transactions on Computers, C-26,* 98-111.

Fikes, R.E. (1982). A commitment-based framework for describing informal cooperative work. *Cognitive Science, 6,* 331-347.

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

Genesereth, M.R., M.L. Ginsberg, and J.S. Rosenschein (1984). Cooperation without communication. Technical Report HPP-84-36, Stanford Heuristic Programming Project, Stanford University, Stanford, California, September 1984.

Hartvigsen, G., and D. Johansen (1988). StormCast - A distributed artificial intelligence application for severe storm forecasting. In M.G. Rodd and T.L. d'Epinay (Eds.), Distributed Computer Control Systems 1988. Proceedings of the Eight IFAC Workshop (Vitznau, Switzerland, 13-15 September, 1988). Pergamon Press, Oxford, England, 1989. pp. 99-102.

Hartvigsen, G., and D. Johansen (1989). Cooperation through information interchange in StormCast. In Proceedings of the 9th IFAC Workshop on Distributed Computer Control Systems (Tokyo, Japan, 26-28 September, 1989). Pergamon Press, Oxford, England. (In press)

Johansen, D. (1988). Weather forecasting - distributed in nature. Network Information Processing Systems. Proceedings of the IFIP TC6/TC8 Open Symposium (Sofia, Bulgaria, 9-13 May 1988). North-Holland, Amsterdam, 1989. pp. 197-203.

Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM, 21* (7), 558-565.

Lesser, V.R., and D.D. Corkill (1981). Functionally-accurate, cooperative distributed systems. *IEEE Transactions on Systems, Man, and Cybernetics, SMC-11,* 81-96.

Lesser, V.R., and D.D. Corkill (1983). The distributed vehicle monitoring testbed: A tool for investigating distributed problem solving networks. *AI Magazine, 4,* 15-33.

Lesser, V.R., and L.D. Erman (1980). Distributed interpretation: a model and experiment. *IEEE Transactions on Computers, 29,* 1144-1163.

McClelland, J.L., D.E. Rumelhart, and the PDP Research Group (1987). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition (2 Volumes).* M.I.T. Press, Cambridge, MA.

Smith, R.G. (1980). The contract-net protocol: high-level communication and control in a distributed problem solver. *IEEE Transactions on Computers, C-29,* 1104-1113.

Smith, R.G., and R. Davis (1981). Frameworks for cooperation in distributed problem solving. I*EEE Transactions on Systems, Man, and Cybernetics, SMC-11,* 61-70.

Wesson, R., F. Hayes-Roth, J.W. Burge, C. Statz, and C.A. Sunshine (1981). Network structures for distributed situation assessment. *IEEE Transactions on Systems, Man, and Cybernetics, SMC-11,* 5-23.

Zachary, W., S. Robertson, and J. Black (1988). *Cognition, computation, and cooperation.* Ablex, Norwood, New Jersey.