Using GRATE to Build Cooperating Agents for Industrial Control

N.R.Jennings

Department Electronic Engineering, Queen Mary and Westfield College,
Mile End Road, London E1 4NS.

Abstract. Communities of cooperating problem solvers have recently begun to emerge as a paradigm for overcoming the complexity of building large software systems in the area of process control. Each agent is capable of solving some problems by itself, but its power can be extended by sharing information and tasks with others. Also, more importantly, the community as a whole exhibits some desirable problem solving characteristics (eg graceful degradation of performance, robustness, etc.) as well as offering the opportunity of connecting and integrating existing problem solvers. GRATE is a general purpose cooperation environment which enables groups of interacting problem solvers to be built for the domain of industrial control. It has been applied to two real-world problems in this area: electricity transport management and diagnosis in a particle accelerator beam controller. We reflect upon GRATE's functional architecture, its underlying principles and the insights gained during this process.

Keywords. Artificial Intelligence; Distributed Control; Multi-Agent Systems; Electricity Transport Management.

INTRODUCTION

As computing systems are being applied to ever more demanding and complex domains, so the infeasibility of constructing a single monolithic problem solver becomes more apparent. To combat this complexity barrier, system engineers are starting to investigate the possibility of using multiple, cooperating problem solvers in which both control and data is distributed. Each *agent* has its own problem solving competence; however it needs to interact with others in order to solve problems which lie outside its domain of expertise, to avoid conflicts and to enhance its problem solving.

To date, two types of multi-agent system have been built: those which solve particular problems (eg air traffic control (Cammarata et al., 1983), vehicle monitoring (Lesser & Corkill, 1983) and acting as a pilot's aid (Smith & Broadwell, 1988)) and those which are general (eg MACE (Gasser et al., 1988) and ABE (Hayes-Roth et al., 1988)).

The general systems either provide a language with which a system can be constructed or a "shell" which the application developer is able to instantiate with the appropriate cooperation and control knowledge. In the former case, the application designer has complete flexibility over the system to be built, but expends a substantial amount of effort imposing the desired structure, because each application must be constructed from scratch. In the latter case, the structure and the mechanisms available are determined by the shell and the designer has to use the languages and tools provided to build the working system.

However, as yet, there have been few attempts to construct multi-agent systems for real-world or complex domains (Jennings & Wittig, 1992). One of the reasons for this lack of progress is the nature of the development environments. They fail to provide the support to cope with the complexities of real-size problems (Bond & Gasser, 1988). The research described here sought to address this fundamental issue by constructing a multi-agent development environment in which some of the knowledge required to build a working system is already embedded. For reasons of comprehensibility, it was decided to encode the inbuilt knowledge in a declarative manner using generic rules. The rules aim to represent high level knowledge and reasoning which is applicable for most multi-agent systems, but is often only represented implicitly. Thus the developer can utilise it directly, rather than constructing the system

from scratch and coding this knowledge himself. This is a step forward because a large corpus of the knowledge which must be brought to bear is already coded, thus the application designer can build upon this and concentrate on defining knowledge and structures specific to the application at hand.

This general description of cooperative agent behaviour, represented by GRATE's built in knowledge, is possible because all the domain-dependent information, which is obviously necessary to define individual behaviour, is stored in specific data structures called *agent models*. These models provide an explicit representation of other agents in the community (Gasser et al., 1988) - including knowledge about the state of the system, the capabilities and aims of the individual agents and evaluative knowledge which enables alternatives to be distinguished between (Jennings et al., 1992). The information which may be maintained in the models (i.e. their structure) is consistent across all applications. However some parts may be left unfilled in particular cases (eg the goals of a database system may not be represented, whereas for an expert system they may be an integral component). Obviously the particular instantiation of an agent model is highly domain dependent and must be carried out by the application builder. The generic knowledge built into the system, however, is able to operate on the homogeneous structure of the agent models rather than the idiosyncracies and domain dependent level of their specific *contents*. This approach is an extension of the notion from conventional AI that generic structure can be utilised when building specialised systems (Chandrasekaran, 1986; Steels, 1990).

A further innovation of GRATE is in the type of problem which is being tackled. Early Distributed AI (DAI) systems concentrated on communities which were purpose built for cooperation and typically had one overall problem to achieve. In such systems (often called distributed problem solving systems) the main emphasis was on techniques for problem decomposition and assigning agents to tasks (Smith & Davis, 1981). Within the domain of industrial process control, such an approach is infeasible because of the large number of systems which are already in existence and the complexity of the problem being tackled (Jennings, 1991). To address this problem the ARCHON project (Jennings & Wittig, 1992), in which some of the work described here took place, focussed on getting possibly preexisting and independent intelligent systems (eg knowledge/data bases, numerical systems, etc.) to cooperate with each other on a variety of goals. The fact that there is no longer just one aim for the whole system, requires explicit reasoning about the process of coordination and means that multiple, unrelated social activities may be taking place concurrently.

GRATE ARCHITECTURE

GRATE agents have two clearly identifiable components: a *cooperation and control layer* and a *domain level* system (see fig. 1). The domain level system may be preexisting or purpose built and solves problems such as detecting disturbances in electricity networks, locating faults and proposing remedial actions. The cooperation and control layer is a meta-controller which operates on the domain level system in order to ensure that its activities are coordinated with those of others within the community. Communication between agents is by the passing of messages.

The diagonal shading indicates those components which are inbuilt (i.e. require the builder to do nothing with them), the lightly dotted boxes those structures which the developer must instantiate and the domain level system which the developer must build. The thinner arrows represent control and the thicker ones data flow.

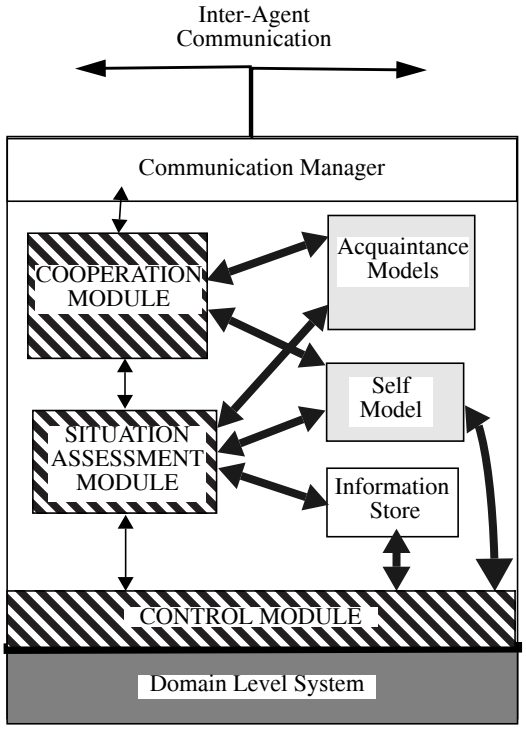

Fig 1: GRATE Agent Architecture

The information store provides a repository for all domain information which the underlying system has generated or which has been received as a result of interaction with other agents in the community. Each agent has two types of agent model: acquaintance models represent other agents in the community while self models represent an abstracted view of the local domain level system.

GRATE communities have a "flat" organizational structure - there is no centralized or hierarchical

structure and also there is no predefined authority structure. A global controller was not considered because interagent communication has a limited bandwidth, meaning that each agent could only maintain a restricted view of the overall problem solving process. Secondly a global controller may be a severe communication and computational bottleneck. Finally reliability criteria require that community performance degrades gracefully if one or more agents fail - which would certainly not be the case if the global controller failed.

By having control distributed within the community, an individual agent plays two distinct roles. Firstly it has to play the role of a *team member* acting in a community of cooperating agents and secondly the role of an *individual*. It also means that there may be more than one goal being pursued by the community - for example there may be agents which are trying to detect faults, agents locating faults and agents proposing remedial actions. Much of the early work on DAI concentrated almost exclusively on the former view and paid scant regard to the latter. However contemporary DAI, with its greater emphasis on autonomous agents, also highlights the role of the individual. Therefore when designing a cooperation framework both aspects should be accounted for. Such a system must:

- Direct local problem solving

  decide which tasks to launch, when they should be launched, their relative priorities and how best to interleave their execution

- Coordinate local activity with that of others within the community.

  when and how to initiate cooperative activity, how to respond to cooperative initiations and which activities require interagent synchronization.

When defining GRATE's modular architecture, it was initially appealing to try and reflect this binary distinction directly. However because of the multiple cooperation contexts within the community, caused by the lack of a single unifying goal, there is a significant class of activities which fall into a grey area between the two. These activities are concerned with *situation assessment*; for example deciding: which activities should be carried out locally and which should be solved with aid of others, what cooperation requests should be honoured and which should not, the relative priority of activities which have to be performed and so on. Therefore to promote a clean separation of concerns, GRATE has three main modules in which the situation assessment module acts as an interface between the local and social control mechanisms. The control module is informed by the situation assessment module of the tasks which should be performed and their relative priorities; it is then the control module's responsibility to ensure that this is carried out. Similarly the need to initiate social activity is detected by the situation assessment module and then the responsibility for realising this activity is left to the cooperation module.

So, for example, the agent's control module may require information *i* in order to execute a particular task. If this information is not available in its information store then it would send the request "provide *i*" to the situation assessment module. This module would, in turn, identify whether *i* could be provided locally and also whether another community member could provide it. If both options are viable, the module decides whether to generate *i* by launching a local task or by asking an acquaintance. If the latter option is chosen, the request will be passed to the cooperation module which will use its acquaintance models to make the request to an agent which it believes is capable of supplying *i*.

Each of the three main modules is implemented as a separate forward-chaining, production system with its own inference engine and local working memory. The generic rules are written in a standard if-then format. The following rule taken from the situation assessment module expresses the condition that if the agent is unable to produce a piece of information locally, then it should try and determine whether an acquaintance is capable of supplying it. Thus the need for social interaction is detected by the situation assessment module and passed onto the cooperation module to enact.

```
(rule situation-assessment-5
    (IF (INFO-NEEDED ?INFO ?TASK)
        (CANNOT-PRODUCE-LOCALLY ?INFO))
    (THEN (TELL-MODULE COOP-MODULE
                INFO-REQUIRED ?INFO ?TASK)))
```

As the TELL-MODULE statement indicates, communication between modules is by message passing, there is no shared memory. At present all three inference engines are identical. However to meet the requirements of future applications, one or maybe all of the inference engines might need to be customised. For example, the control module may need to respond rapidly to certain key events and hence need to be more sophisticated than that of the cooperation module in which events can be handled on a first come first served basis in most circumstances.

BUILDING GRATE APPLICATIONS

At present, GRATE applications embody only completely generic knowledge and knowledge required to control activity in a particular application. The generic rules define an agent's default behaviour (i.e.

given no information to the contrary an agent's activity will be governed by generic rules). However in certain well defined instances this default behaviour is overridden by behaviour tailored to the specific situation at hand. These two types of knowledge can be viewed as opposite ends of a spectrum which could be brought to bear in the problem of ensuring coherent behaviour between cooperating agents. The former being applicable to all cooperative scenarios and the latter to one specific problem. In between, however, are several others layers which represent varying levels of generality (see fig. 2). Ideally a cooperation shell would provide built-in knowledge for all levels but the individual problem, which must obviously be provided by the application developer.

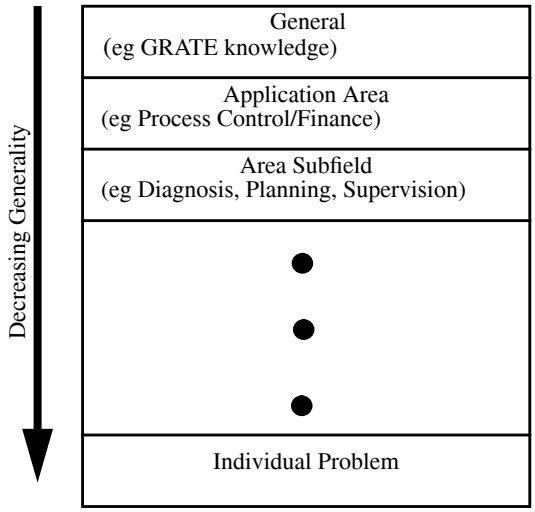

Fig 2: Spectrum of Cooperation Knowledge

For such an approach to succeed, it must firstly be possible to identify and characterise general areas of problem solving. The feasibility of constructing generic tasks models has been demonstrated by Chandrasekaran (1986) and libraries of such tasks have been constructed in the KADS project as a means of simplifying the domain modelling process (Hickman et al., 1989). Similar approaches in conventional AI have been championed as a mechanism for making software development easier (by supplying programs which solve classes of problems (McDermott, 1990)) and form the basis of the knowledge sharing vision of building conventional expert systems (Neches et al., 1991).

As such models appear feasible for conventional AI, there is no reason to doubt that it is possible to construct similar descriptions of generic social interactions. In the above hierarchical knowledge model the application area knowledge for control would define typical cooperative scenarios for process control systems and the area subfield diagnosis would provide general models of cooperation between systems working on diagnosis, and so on.

As an example of such a general model of interaction, consider the problem of diagnosis. In this application the following are illustrations of high level interactions, agents may:

- divide the problem domain into non-overlapping parts and each work separately
- both perform the same diagnosis using different data or problem perspectives
- cross-check diagnoses of the same problem
- focus each others problem solving by exchanging highly rated hypotheses

If such knowledge could be assimilated (and the success of GRATE in defining some generic knowledge is an initial step in this direction) then a new paradigm is required for building multi-agent systems. Rather than constructing the system afresh for each new application, the developer starts from a state in which much of the knowledge required for building multi-agent systems is available in various "knowledge libraries". Thus he has to select the required knowledge, configure it for his particular system and then augment it with any necessary application specific knowledge (as shown below). Such reasoning may be necessary to provide a shortcut in the general reasoning process in order to meet the desired performance characteristics or to reflect truly domain dependent reasoning

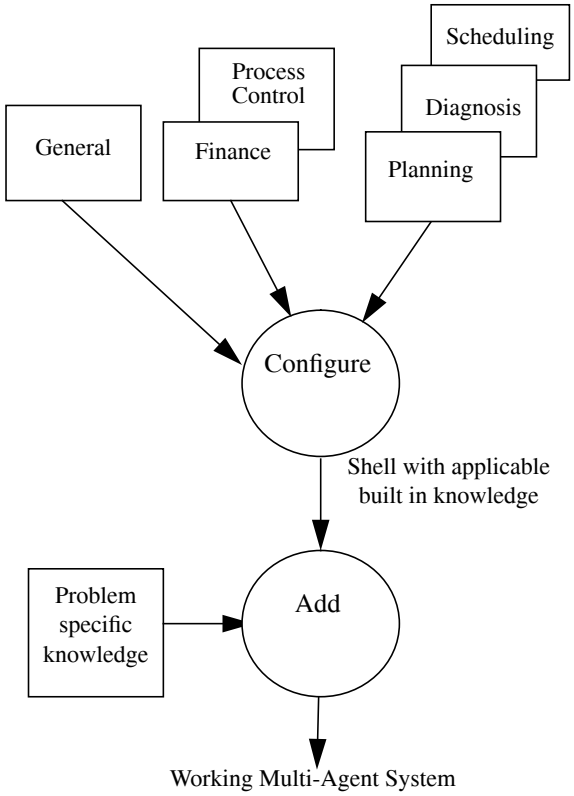

Fig 3: Building Multi-Agent Systems Using Levels of General Knowledge

The process of configuration, in this instance, involves two steps. Firstly selecting a subset of the available knowledge, for the problem at hand. For example the application builder may never want to use a contract net (Smith & Davis, 1981), in which case he would remove the general knowledge associated with this protocol. Also he may be building an application for process control, in which case knowledge from the finance domain is not appropriate. Secondly the control strategies of the problem solving modules may need to be fine tuned to meet the desired performance characteristics. For instance in the present implementation of GRATE equal weight is given to each of the three modules; however in applications which require more sophisticated local control and less interagent interaction the control and situation assessment modules may need to be given more resources than the cooperation module.

This paradigm has significant advantages over conventional means of constructing multi-agent systems - including the reuse of problem solving components (increasing reliability, decreasing risks and development time and making effective use of specialists (Horowitz and Munsen, 1984)) and provides the ability to fall back on increasingly general knowledge (Lenat and Feigenbaum, 1991). It also follows the lead of other disciplines which engineer complex artifacts (eg planes, cars), in that product development would consist predominantly of assembling components (Stefik, 1986).

GRATE IN INDUSTRIAL CONTROL

GRATE has been applied to two distinct domains: electricity transport management (Jennings et al., 1992) and diagnosis in a particle accelerator beam controller (Fuchs et al., 1992). In both cases the applications have been constructed rapidly and without the need to significantly augment GRATE's inbuilt knowledge. In both instances the designer has merely filled in the agent models and provided the appropriate interface functions to the underlying domain level system. The number of agents in the community has been three to five agents, the domain level systems have been mainly expert systems and in the latter application they were running on different machines and in different languages.

The types of cooperation encountered in these two applications were fairly simplistic in nature. Two main forms were observed: firstly agents would spontaneously send information to other agents that they believed (based on their acquaintance models) would benefit from receiving it (result sharing). Secondly, agents were able to make requests of each other - asking for tasks to be performed or information to be supplied (task sharing).

The generic knowledge embodied in GRATE's control module was sufficient for these two applications because the control exerted over the domain level systems is fairly rudimentary in nature - consisting of stopping, starting, suspending and aborting tasks - and the performance criteria demands have not been too high. Also the types of domain level system were limited to expert systems - not the full range of systems (eg databases, numerical systems, etc.) which would be expected in a full industrial control environment. For the reasons of performance, domain dependence and heterogeneity of the underlying domain level system, we doubt whether it is possible to continue to use such generic control knowledge in all future applications. Due to these reservations, within the ARCHON project it was decided that the component responsible for controlling the domain level system should predominantly consist of generic mechanisms *not* generic knowledge.

In contrast with the control level, functions associated with social activity are more or less independent of the application domain and are relatively few in number. Therefore the approach advocated by this work is kept for these functions.

CONCLUSIONS

We have outlined a general purpose development environment for the domain of industrial process control. This environment was designed to speed up the process of building multi-agent systems by providing a shell which has a significant amount of inbuilt knowledge related to cooperation and control. This approach, and its logical extension to general classes of cooperative problem solving, requires a paradigm shift for application builders. Rather than constructing a system from scratch and continually re-coding the same basic knowledge - the designer is faced with pre-built libraries of knowledge. The process of building applications then becomes one of configuring this knowledge and augmenting it with any application specific knowledge which is required.

At present, the general knowledge embodied in GRATE has no formal theoretical grounding. That is, there is no deeper model of coordination or cooperation represented by the generic rules. However, as a result of the generality and explicit representation of the knowledge embodied in GRATE, it was possible to devise such a theory (Jennings, 1991; Jennings & Mamdani, 1992). This theory (called *joint responsibility*) is based on the notions of intentions and is particularly useful for ensuring coordinated behaviour in complex, dynamic environments in which agent's beliefs may change, wrong decisions may be taken and unanticipated events may occur (i.e. situations often typical of industrial control applications). We are currently coding this theory in terms of generic rules and they will form the basis of the situation

assessment and cooperation modules in future versions of GRATE.

ACKNOWLEDGMENTS

The work described in this section has been partially carried out in the ESPRIT II project ARCHON (P2256) whose partners are: Atlas Elektronik, JRC Ispra, Framentec, Labein, IRIDIA, Iberdrola, EA Technology, Amber, Technical University of Athens, University of Amsterdam, Volmac, CERN and University of Porto. In particular discussions with, and comments from, Abe Mamdani (QMW), Erick Gaussens (FTC) and Thies Wittig (Atlas) have been particularly useful in shaping the ideas presented here. Also thanks are due to the people at CERN and to Rob Aarnts (Volmac) for building the multi-agent particle accelerator application using GRATE.

REFERENCES

Bond,A.H. & Gasser,L., (1988), "Readings in Distributed Artificial Intelligence", Morgan Kaufmann.

Cammarata,S., McArthur,D. & Steeb,R., (1983), "Strategies of Cooperation in Distributed Problem Solving", in Proc. of IJCAI, pp 767-770.

Chandrasekaran,B., (1986), "Generic Tasks in Knowledge Based Reasoning: High Level Building Blocks for Expert System Design", IEEE Expert, 1 (3), pp 23-30.

Fuchs,J., Skarek,P., Varga,L. & Malandain,E., (1992) "Distributed Cooperative Architecture for Accelerator Operation" in Second Int. Workshop on Software Engineering, Artificial Intelligence and Expert Systems for High Energy and Nuclear Physics.

Gasser,L., Braganza,C. & Herman,N., (1988), "MACE: A Flexible Testbed for Distributed AI Research", in Distributed Artificial Intelligence (ed M.N.Huhns), pp 119-153 Pitman Publishing.

Hayes-Roth,F., Erman,L.D., Fouse,S., Lark,J.S. & Davidson,J., (1988), "ABE: A Cooperating Operating System and Development Environment", in Readings in Distributed Artificial Intelligence (eds A.H.Bond & L.Gasser), pp 457-490, Morgan Kaufmann.

Hickman,F.R., Killin,J.L. Land,L., Mulhall,T., Porter,D. & Taylor,R., (1989), "Analysis for Knowledge-Based Systems", Ellis Horwood.

Horowitz,E. & Munsen,J.B., (1984) "An Expansive View of Reusable Software", IEEE Trans. Software Engineering, 10 (5), pp 477-487.

Jennings,N.R. & Mamdani,E.H., (1992), "Using Joint Responsibility to Coordinate Collaborative Problem Solving in Dynamic Environments", in proc AAAI-92.

Jennings,N.R. & Wittig,T (1992) "ARCHON: Theory and Practice", in Distributed Artificial Intelligence: Theory & Praxis, (Ed L.Gasser and N.Avouris), Kluwer Academic Press (forthcoming).

Jennings,N.R., Mamdani,E.H., Laresgoiti,I., Perez,J. & Corera,J., (1992), "GRATE: A General Framework for Cooperative Problem Solving", Journal of Intelligent Systems Engineering, Vol. 1, (forthcoming).

Jennings,N.R., (1991) "Cooperation in Industrial Systems" Proc. ESPRIT Conference, Brussels.

Jennings,N.R., (1991) "On Being Responsible", Proc. Modelling Autonomous Agents in a Multi-Agent World, Third European Workshop, Kaiserslautern, Germany.

Lenat,D.B. & Feigenbaum,E.A., (1991), "On the Thresholds of Knowledge", Artificial Intelligence 47, pp 185-250.

Lesser,V.R. & Corkill,D.D., (1983), "The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks", AI Magazine, pp 15-33.

McDermott,J., (1990), "Developing Software is Like Talking to Eskimos about Snow", in proc AAAI-90, pp 1130-1133.

Neches,R., Fikes,R., Finin,T., Gruber,T., Patil,R., Senator,T., & Swartout,T., (1991), "Enabling Technology for Knowledge Sharing", AI Magazine, pp 36-56.

Smith,D. & Broadwell,M., (1988), "The Pilot's Associate: An Overview", in SAE Aerotech Conference, Los Angeles, CA.

Smith,R.G. & Davis,R. (1981), "Frameworks for cooperation in Distributed Problem Solving", IEEE Trans. on SMC, 11, 1, pp 61-70.

Steels,L., (1990), "Components of Expertise" AI Magazine, 11 (2), pp 29-49.

Stefik,M., (1986), "The Next Knowledge Medium", AI Magazine 7 (1), pp 34-46.