Designing a reusable co-ordination module for co-operative industrial control applications

N.R.Jennings
J.A.Pople
E.H.Mamdani

Indexing terms: Distributed artificial intelligence, Co-operative information systems, Knowledge-based systems, Industrial control applications

Abstract: Distributed artificial intelligence (DAI) systems, in which multiple agents communicate and co-operate with one another to achieve their individual and collective goals, are a promising enabling technology for constructing large, real-world industrial control applications. To facilitate the development of such systems a number of generic DAI frameworks have been devised. These frameworks typically aid the development process by providing a language, a set of structures, and/or some tools with which the necessary infrastructure and support mechanisms for interacting agents can be instantiated. The paper reports on one such framework, called ARCHON™, which has been used to build DAI systems in the following industrial control domains: electricity distribution management, electricity transportation management, cement factory control, particle accelerator control and flexible assembly robotic cells. A distinguishing and novel feature of the ARCHON framework is that it extends the level of support offered to the system builder — it provides generic and reusable knowledge about the process of co-operation, in addition to the more standard development facilities. This generic knowledge is embedded in a domain-independent co-ordination module and it is the rationale, design, implementation and evaluation of this module which forms the major contribution of the paper.

1 Introduction

Systems composed of multiple, interacting components (agents) are becoming an increasingly popular means of building complex industrial control applications [1]. The majority of these systems are functionally distributed and have subcomponents with clear, predefined communication links which are ordered in some hierarchical fashion. Although this modular approach increases the maintainability of the system, it keeps the overall control at a central location (i.e. a global controller co-ordinates the activities of all the subcomponents). This centralisation has two particular drawbacks for industrial control applications [2]. Firstly, for large applications with a number of distinct supervisory and control subcomponents, the activation of tasks in the subsystems and the decision of what data to exchange between them depends on the state of the entire process. In a centrally controlled system this assessment requires the controller to take into account the different views of all the relevant subsystems and can, therefore, lead to severe delays while the relevant information is assembled and the appropriate decisions are taken. Secondly, it is difficult (sometimes impossible!) to perform the modifications required to integrate the large number of pre-existing (legacy) systems which are often found in industrial applications into a unifying whole.

To alleviate the decision-making bottleneck, increase the flexibility of data exchange and task activation, and facilitate software reuse, the next stage in system design is to decentralise the control and allow the components to interact directly with one another. This approach not only allocates more responsibility to the subsystems, but also requires them to co-ordinate their tasks if the whole system is to interact in a coherent manner. Such co-ordination can be hand-crafted for each and every application or it can be undertaken in a more structured manner by developing a framework which can be reused in a number of different scenarios (the approach described in this paper). The ARCHON™ (architecture for co-operative heterogeneous online systems) framework [3], which provides the context for this work, has been used to build co-operative, multiple agent applications in the domains of electricity distribution management [4, 5], electricity transportation management [6], cement factory control [7], flexible assembly robotic cells [8] and particle accelerator control [6, 9]. A summary of all of these applications is presented in [10].

Within the ARCHON framework, each agent is composed of a number of functional components, one of which is responsible for co-ordination in a decentralised environment. During the design and development of this planning and co-ordination module (PCM) [11] a number of crucial issues needed to be addressed: (i) what are the requirements for co-ordination in real-world industrial applications? (ii) what types of facilities should a general-purpose framework provide an application developer? (iii) how can the reasoning of the co-ordination module be controlled so that the

IEE Proceedings online no. 19960186

Paper first received 13th December 1994 and in revised form 27th November 1995

The authors are with the Department of Electronic Engineering, Queen Mary & Westfield College, University of London, Mile End Road, London E1 4NS, UK

agent's objectives are satisfied? (iv) how can the co-ordination module be designed so that it responds rapidly to important events but also deals with events in a fair manner avoiding resource starvation? (v) how can a generic co-ordination module be tailored to fit a particular application? (vi) how can such a co-ordination module be implemented so that it meets all these desiderata?

This paper describes how the above issues were tackled and solved within the ARCHON framework. These experiences and insights are important for a number of different reasons. From the perspective of distributed artificial intelligence (DAI), this work represents one of the first serious attempts to build a generic co-operation framework for large-scale, real-world industrial applications. From the perspective of industrial control applications, this work highlights the feasibility of employing a co-operating systems metaphor and enables the problems associated with building decentralised control systems to be clearly stated and evaluated. From a system engineering perspective, this work enables the approach of constructing and utilising libraries of reusable problem-solving know-how to be evaluated in a realistic setting.

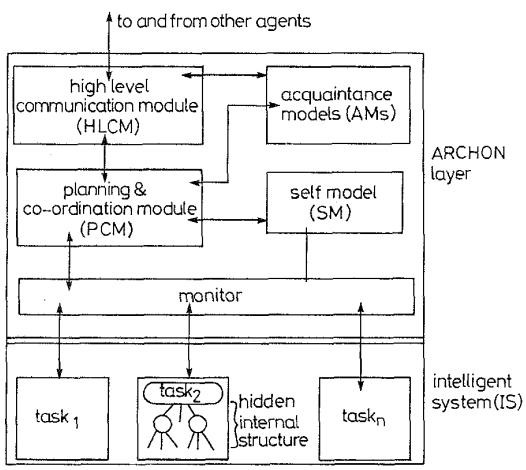

Fig. 1 *ARCHON agent architecture*

2 Structure of an ARCHON agent

ARCHON agents have two distinct components: an intelligent system (IS) and an ARCHON layer (Fig. 1). The former may be pre-existing or purpose built and solves domain-level problems such as detecting disturbances in electricity networks or controlling the blower of a cement factory kiln. In the majority of ARCHON's applications, the co-operating community contains a number of different types of IS, including expert systems, databases and conventional numerical software. From the ARCHON layer perspective, the IS is composed of a number of atomic executable tasks, although in terms of their actual implementation the tasks may involve branching, sophisticated reasoning and control actions [5]. The ARCHON layer is a meta-level controller which operates on the IS to ensure that its activities are co-ordinated with those of the others within the community. The separation of the domain and co-operation know-how into the IS and the ARCHON layer, respectively, allows pre-existing systems to be incorporated into the multiple agent community with relatively few modifications and allows the co-operation know-how to be reused in a number of applications. Without this demarcation, extensive changes would be required to the existing systems in order to provide them with the necessary knowledge to interact with, and benefit from, the other agents in the community.

Communication between agents is via message-passing and is controlled through the high-level communication module (HLCM). This module is deemed high-level since it not only provides standard communication facilities (achieved through a session layer implementation) but also embodies services such as intelligent addressing and filtering. A message-passing paradigm was chosen because of the physical distribution of the problem solving agents and the desire to conform to OSI standards.

The acquaintance models (AMs) are a representation of other agents in the community. Information maintained includes an acquaintance's skills, interests, current status, workload and so on [12]. These models are essential when co-ordinating activity because they provide a characterisation of the social problem-solving context in which the agent has to operate. Much like the AMs represent other agents in the community, the self model (SM) is an abstract characterisation of the agent's underlying IS. It contains information about the current state of the IS and embodies a representation of the sequences of actions which can be executed by the ARCHON layer in its underlying IS.

The monitor organises locally executable activities and is responsible for passing information to and from the IS. Skills are the coarsest granularity at which these activities are described. Other ARCHON layer components deal exclusively on the level of skills, but within the monitor they are given a finer structure, corresponding to an OR-graph in which the named branches specify alternative solutions. The nodes of the graph are called monitoring units and they correspond to the invocation of individual tasks within the IS (see [5, 6] for more details of this structure).

The PCM reasons about the agent's role in terms of the wider co-operating community. It has to assess the agent's current status and decide which actions should be taken in order to exploit interactions with others whilst ensuring that the agent contributes to the community's overall well-being. Specific examples of the functionality supported include: deciding which skills should be executed locally and which should be delegated to others, directing requests for co-operation to appropriate agents, determining how to respond to requests from other agents, and identifying when to disseminate timely information to acquaintances who would benefit from receiving it.

The ARCHON approach, to construct a generic framework which can be instantiated in a number of different application domains, is now becoming an accepted way of building DAI systems. Other exemplar systems which have adopted this methodology include MACE [13] and DASEDIS [14], although, to date, no other framework of this genre has been applied to as wide a range of real world applications as ARCHON. Other paradigms for developing DAI applications include: (i) DAI programming languages (e.g. AGENT0 [15] and MAIL [16]); (ii) testbeds designed specifically for a particular domain (e.g. DVMT [17]). In this work, the former approach was eschewed because of the difficulty of designing a coherent and

usable language which covers concepts from both traditional distributed computing (e.g. communication protocols, interoperation across heterogeneous platforms, etc.) and agent systems (e.g. co-operation protocols, situation assessment, negotiation, etc.). The latter approach was rejected because there was a need to develop systems for a number of different applications without having to start from scratch in each case.

3 ARCHON's planning and co-ordination module

3.1 Reusable generic co-operation know-how

Analysis of a number of industrial control applications highlighted a surprising degree of commonality in terms of their status and their characteristics. In the majority of cases studied, there were a number of automated components which were responsible for a well defined portion of the overall process. Although the subsystems made reference to the same environment, and hence decisions and actions by one component influenced those of another, they were not integrated. However, when major events occurred (e.g. lightning storms in the electrical management domains) the operators of the individual components interacted verbally with one another to co-ordinate their problem-solving activity [2].

In addition to this conceptual similarity at the operator level, the problem solving entities also had a number of broadly common characteristics. Most important from the PCM's point of view was the fact that the subsystems were able to undertake significant amounts of processing in their own right — a consequence of the fact that most of them were originally intended to operate alone or with minimal intervention from an operator. In terms of the co-operating system metaphor, this meant that agents would spend the majority of their time engaged in domain level computations and substantially less time on co-ordination activities and interagent communication. Also, the number of co-operative interactions which would be needed were relatively small in comparison to the number of activities undertaken within the domain level system. However, interaction with other agents was needed to accomplish tasks that could not be performed locally and to supply information which was needed for problem solving but which could not be readily accessed. As well as these necessary interactions, there were a number of other new interactions, made possible by the subsystem integration, which could enhance the problem-solving of the participating agents [4, 6]. Examples include: receiving relevant information which helps an agent prune its search space; cross-checking results by performing tasks which produce the same information using different data or a different approach; providing more timely/accurate information for injection into the problem-solving process. In general, the mandatory interactions mirror those between the stand-alone system and its operator, whereas the new ones are similar to the types of interactions which took place between the operators when exceptional circumstances arose.

Within these well defined constraints, it was decided that the greatest degree of support could be offered to the developers of ARCHON applications if a significant portion of the co-operative functionality could be provided as a core of in built knowledge. Thus, rather than providing the developer with just programming features, he is presented with a library of knowledge about co-operation with which the application can be constructed[1]. This core can then be augmented, if necessary, with domain-specific co-operation knowledge in order to build the co-ordination mechanism for a particular multiagent system (Fig. 2). This approach contrasts with the conventional means of fabricating multiagent systems in which the application developer is forced to continually recode a large proportion of essentially the same knowledge in each and every case (Fig. 3).

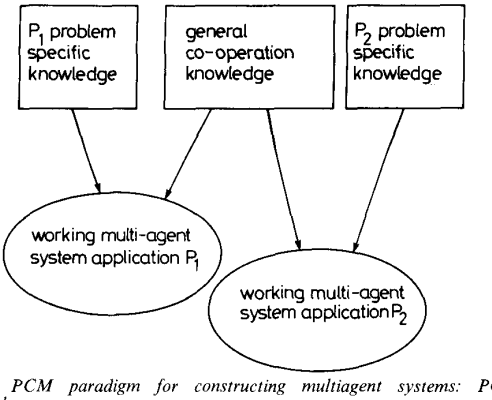

Fig.2 *PCM paradigm for constructing multiagent systems: PCM approach*

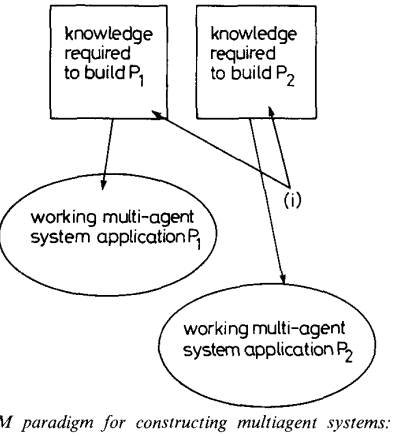

Fig.3 *PCM paradigm for constructing multiagent systems: traditional approach*
(i) Note that there will be an overlap of the knowledge required to build P_1 and P_2

The reusable knowledge approach means that each agent has the same core know-how about co-operation encoded in its PCM. The majority of the domain-dependent data, which is obviously needed to define individual behaviour, is then located in the agent models. Examples of three such generic rules are as follows:

$Rule_1$: if an agent has generated a piece of information i and it believes that i is of use to an acquaintance then send i to that acquaintance

$Rule_2$: if an agent has a skill to perform and it is not able to perform it locally then seek assistance from another agent

$Rule_3$: if an agent has finished executing skill s and s was undertaken because of a request from an acquaintance then inform the acquaintance that s has finished and return any results which have been produced

[1] This approach has been advocated by a number of researchers concerned with the inherent difficulties and inefficiencies in the present software engineering development process (see [18–21])

In the case of $Rule_1$, the acquaintance models contain a list of information that the other agents are interested in receiving and a condition under which they are interested. If the condition is met, then the second clause of the rule will be satisfied and the information will be sent. In the case of $Rule_2$, the self-model is used to determine that the agent cannot complete the skill locally and the acquaintance models are used to identify those agents who are able to furnish the necessary skill. $Rule_3$ is triggered when the monitor indicates that a skill has finished. At this point, the self-model is examined to determine the reason for executing the skill. If this reason indicates that the skill was initiated as a result of a request from an acquaintance, then the information that the skill has finished and any relevant results which have been produced are returned to the originator. All of these rules are application-independent and are tailored to a specific domain by the appropriate instantiation of the agent models.

3.2 Design decisions

Being a key functional component of the ARCHON architecture, it is important that the PCM's design rationale and philosophy are made explicit and are open to scrutiny. This allows the factors which influenced its internal structure, its representations and its control mechanism to be evaluated and assessed for appropriateness. Throughout the entire design process, the primary objective was to develop a domain-independent, reusable mechanism whose operation would be as transparent and extensible as possible. More details about how this design was realised are contained in [11].

Given that the PCM is the overall director of, and broker between, the activity of the underlying IS and that of the agent's acquaintances, it has two obvious spheres of influence. First, to interact with other agents there must be an interface to the HLCM so that messages can be sent and received across the community. Likewise, an interface to the monitor is needed so that the PCM can influence the activities of the IS. This separation of concerns meant that the PCM's operations could be divided into two distinct groups — those related to managing the agent's local activity in a co-operative environment and those related to controlling the agent's social activities *per se*. For reasons of software modularity and clarity of design, these distinct functional roles were implemented within the PCM as separate problem solving modules, the former as the situation assessment module (SAM) and the latter as the co-operation module (CM).

In more detail, the SAM is responsible for the following: deciding how data needed by the IS can be supplied (start activity locally or enlist the help of an acquaintance?); determining whether a request for the performance of a skill should be carried out locally; evaluating which skills should be started, in what order and with what data; deciding whether external requests should be met by starting a new skill or by exploiting an already active one; evaluating whether new information should be passed on to the relevant active skills.

The CM has three primary objectives. First, it has to establish social interactions. This involves deciding how requests from the SAM can be best satisfied. Two forms of co-operation are currently supported: skill and information sharing. In the former case the agent asks an acquaintance to execute a skill or produce a specified piece of information; in the latter case the agent spontaneously volunteers information to acquaintances who will benefit from receiving it (based on information specified in the acquaintance models). In both forms of interaction, the CM has to decide with which acquaintances the interaction should take place (i.e. which agents to request aid from and which agents to disseminate information to). With skill sharing, the CM has to additionally decide between the clientserver protocol and the contract-net protocol [22] as the means of determining how the task should be awarded to an acquaintance. With the client-server protocol, the request is directed to just one acquaintance. With the contract-net protocol, the agent advertises the activity it would like to be performed to all of those acquaintances who are capable of providing it. Upon receipt of the request, each acquaintance puts together a bid which specifies when and with what quality it could provide the service. When the originating agent receives all the bids, it evaluates them and establishes a contract for the activity with the most appropriate agent.

Secondly, it has to maintain ongoing co-operative activities. So, for example, in the case in which an agent agrees to perform a skill because an acquaintance has asked it to, the social action's progress must be tracked to ensure that any relevant intermediate results are returned and that upon completion a final report describing the status and results of the requested activity is sent back to the originator (see, for example, $Rule_3$ of the preceding sub-Section). Finally, the CM has to respond to co-operation initiations from other agents.

An early prototype of the PCM, called GRATE [12], which implemented the SAM and the CM as concurrent processes, was built for evaluation purposes and applied to the domains of electricity transportation management [12] and particle accelerator control [9]. As a consequence of this prototyping activity, three important points pertaining to the design of the PCM were highlighted [23]. First, the process of controlling the reasoning about co-operation and situation assessment needed to be significantly improved (GRATE just had a simple looping structure and consequently did not respond quickly to important events). Secondly, some organisational structure needed to be imposed on the knowledge embodied within the SAM and the CM if the application developer was to be able to add any domain specific know-how (in GRATE all the co-operation and situation assessment knowledge was intermingled). Finally, it was deemed necessary to specify the objectives of the PCM so that important events could be more easily recognised. With respect to the final point, GRATE did not enable the application developer to introduce any bias into the reasoning process. So, for example, it was not possible to reflect the fact that the agent's main role in the community may be to provide services for the others (e.g. a database agent which contains large amounts of static information about the process being controlled). Nor was it possible to reflect the fact that another agent carries out such an important task that it should not be interrupted by low priority external requests (e.g. an expert system planning how the network can be repaired after a major fault should not be distracted by the receipt of unrequested information which is probably out of date). In the former case, the developer

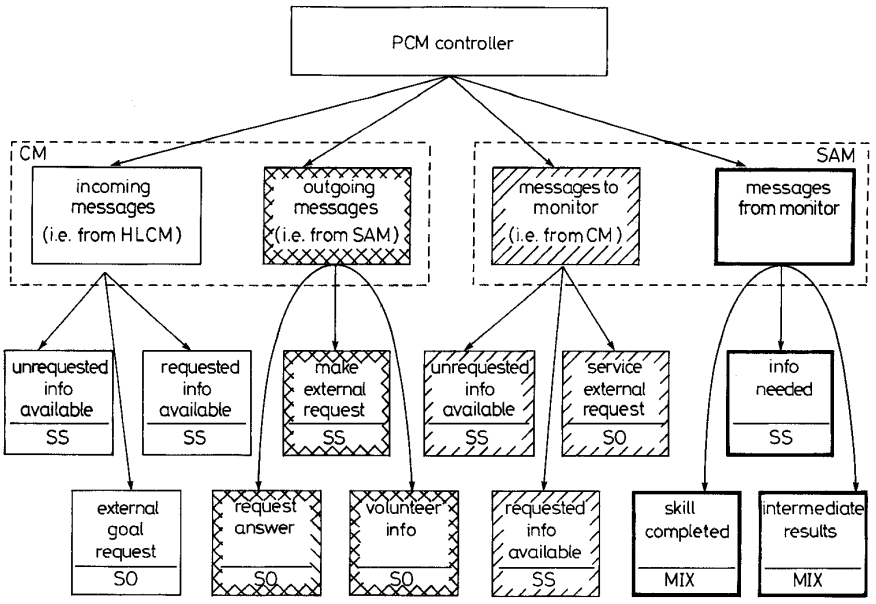

Fig. 4 *Detailed structure of PCM*

needs the facility to specify that external requests should be given a higher priority than locally generated ones and in the latter case that local activities should take precedence.

To rectify these problems it was decided that the PCM should be decomposed into smaller, more modular units and that some explicit reasoning about the invocation of situation assessment and co-operation functions needed to be introduced. First, rather than allowing the CM and the SAM to run as concurrent processes, and hence having no real control over their relative resource usage, an overall controller was introduced into the PCM (Fig. 4). This controller maintains a high-level description of all the processing which the PCM has to undertake and decides whether situation assessment or co-operative functionality should be invoked next. Secondly, the CM and the SAM were further divided into two submodules according to the interface which initiated their action. For the SAM this resulted in one sub-module for dealing with messages arriving from the monitor and another for dealing with messages from the CM. Likewise for the CM, one group of operations were activated by messages arriving from the HLCM and a separate group were related to messages arriving from the SAM. These submodules act on the overall controller's instructions and use their more detailed knowledge of that subarea of the PCM's operation to decide which types of functionality should be invoked and for what duration. As functionality invocation is now to occur as a result of reasoned activity, rather than being purely data-driven, the messages arriving at a submodule needed to be stored in a buffer. Rather than having just one buffer, in which the structure of the activities to be performed would be lost, each submodule maintains its own buffer for the messages that it has to process. Thirdly, the individual functionalities of the PCM were represented as distinct blocks, called operational rule blocks. Thus the CM submodule which processes messages from the HLCM is responsible for controlling the operational rule blocks which deal with the arrival of unrequested information, with requests to carry out problem-solving activity for other agents, and with the return of information which has been requested from other agents[2]. To facilitate the reasoning about invocation, each operational block is designated as having a particular orientation (which mirrors the overall orientation of the agent): serves-self (SS) means that it progresses the agent's own local objectives; serves-others (SO) means that it progresses the processing of other community members and mixed (MIX) means that it has elements of both.

3.3 Meta-level control of co-ordination process

Ensuring that agents act coherently in an environment in which control decisions are decentralised is a difficult task which has resulted in the development of a variety of co-ordination mechanisms [24]. In terms of the PCM, the major decision which affects the coherency of the system is the decision of what operational functionality to invoke, at what time, and for how long. To take this decision a number of factors needed to be taken into consideration, ranging from long-term and relatively static information about the agent's objectives, to the immediate and constantly varying status information. The PCM's objectives are determined by examining the designated role of the agent in the community. Three alternatives are available; an agent's primary role may be: (i) serves-self, in which case its main objective is to complete its own problem solving; (ii) serves-others, in which case the agent is predominantly a server for the other community members; (iii) mixed, in which case the agent has a mixture of objectives (some of which are related to serving its own needs and some of which are related to helping others).

As well as invoking the appropriate operational functionality in order to fulfil the agent's role within the system, the control regime of the PCM has two other desiderata. First, it must avoid resource starvation and ensure that messages do not remain in the system for an unacceptable amount of time without being processed. Secondly, because ARCHON is to be used in

[2] For reasons of clarity, only 12 of the PCM's operational blocks are shown in Fig. 4. Those not shown are related to the rejection of co-operation requests, the resolution of conflicts and the contract-net protocol

industrial applications, the decision making process which determines the functions to be invoked should not consume significant amounts of resource. This means that a 'satisficing' [25] approach to control decisions is required in which relatively simple (and computationally cheap) criteria are applied to produce decisions which are 'good enough'. Optimal decisions, though desirable, may consume considerably more resources to make only marginally better decisions and may compromise ARCHON's time-criticality objectives.

Table 1 PCM control algorithm

```
CONST
    AgentOrientation ∈ {SERVES-SELF, SERVES-OTHERS, MIXED};
    SubModuleList ∈ {Incoming-Messages, Outgoing-Messages,
                     Messages-To-Monitor, Messages-From-Monitor};
    SelectionCriteria ∈ {ROUND-ROBIN, SHORTEST-FIRST, BUSIEST-FIRST};
LOOP FOREVER
    NextActive = select (SubModuleList, SelectionCriteria);
    IF NextActive ≠ nil THEN
      BEGIN
        PCMWorkloadStatus = EvaluateWorkload (SubModuleList);
        FORAll OperationBlk(i) ∈ NextActive DO
            CASE PCMWorkloadStatus OF
              CLEAR-BACKLOG: Process all messages in buffer;
              NORMAL: IF orientation (OperationBlk(i)) = AgentOrientation
                        THEN process all associated messages
                        ELSE process first associated message;
              BUSY: IF orientation(OperationBlk(i)) = AgentOrientation OR
                        high-priority(OperationBlk(i))
                      THEN process first associated message;
            ENDCASE
      ENDTHEN
ENDLOOP
```

The PCM's overall controller is responsible for selecting which of the four submodules should be processed at any one time and also for determining the amount of resource that should be consumed during this processing. The decision about submodule activation is based on the policy set by the application developer:

ROUND-ROBIN: select the successor of the currently active submodule until the end of the ordered list is reached, in which case restart with the first element.

SHORTEST-FIRST: select the submodule with the fewest messages to process.

BUSIEST-FIRST: select the submodule with the most messages to process.

The amount of resource which should be consumed during a particular submodule invocation depends on the loading of the PCM. If this load is high, then processing should be evenly spread between the submodules to ensure that all the important events are dealt with in a reasonable amount of time. If the PCM's load is relatively light, then some effort can be dedicated to processing less important messages and hence ensuring that long backlogs do not build up. The three choices which the controller can pass onto the chosen submodule are as follows:

CLEAR-BACKLOG: clear up any backlogs which have built up.

DEFAULT: process important messages first but also ensure that no messages are waiting too long before being receiving attention.

IMPORTANT-TASKS-ONLY: only process those message types which are important.

Within the constraints set by the controller, the chosen submodule has to decide which of its associated operational blocks will be invoked and how much processing each should undertake. So, for example, if the submodule processing messages from the HLCM is chosen, it may decide to process all of the messages corresponding to replies for information which have been made to acquaintances, one message providing unrequested information, and no messages which are requests from other agents for the local agent's services. This selection will be based on the policy set by the controller, the priority of the individual operational blocks, the orientation of the operational blocks, and the agent's orientation. Table 1 gives a more detailed description of the algorithm controlling this process.

Other work has also highlighted the importance of utilising metalevel control knowledge to produce more dynamically adaptable behaviour. Of particular prominence in this respect is the blackboard control architecture [26] which views the problem of control as one of multiple task planning. It proposes the use of a dedicated control blackboard where solution elements may be elaborated at various levels of abstraction under the direction of both domain independent and domain-specific control knowledge sources. There are obvious parallels between this approach and that of the PCM's

control mechanism. Both integrate domain and control problem solving, and allow for the modification of the control structure according to prevailing problem-solving situations. The PCM combines knowledge of the utility of various domain actions, when compared to an overall orientation, with knowledge about its workload, to ensure that efficient execution policies are adopted, and hence its overall performance is maintained. Similarly, the blackboard architecture depends on knowledge of global and temporary objectives (derived from strategies adopted at run time) to ensure coherent behaviour of the overall system according to those objectives. For both approaches, the need to make explicit control decisions and the ability to adopt variable grain control heuristics is of primary importance. However, the blackboard control architecture incurs a significantly greater overhead, since it is inherently more complex. This makes it inappropriate for incorporation into the already complex ARCHON layer.

Still further work in this area has stressed the utility of metalevel knowledge as a means of dynamically adapting a parameterised control mechanism to changing problem-solving situations. Whether this is through the use of dedicated meta-rules for resolving control decisions [27] or through the diagnosis of system behaviour to select appropriate parameter settings [28], the central theme is to ensure a correspondence between the control strategies employed and the state of problem-solving at any given point. The PCM's meta-level control mechanism aims to address the issue of adaptability through a flexible control strategy capable of updating certain parameters. These dictate run-time behaviour of the agent and allow it to respond effectively, under a variety of circumstances, according to its particular bias. There is, however, the possibility of extensions that would allow high-level monitoring, not only of load characteristics, but also of the problem-solving state of the agent and its acquaintances. This abstract view could be used as the motivation for dynamically setting the bias introducing parameters (e.g. agent orientation, ruleset orientation and priority) that are currently fixed on initialisation.

3.4 Instantiating the PCM for a particular application

The first step when instantiating the PCM is to analyse the in-built generic knowledge to determine whether it contains all the functionality and reasoning required to build the application. In all of the ARCHON applications which have been built so far, this generic knowledge has been sufficient and has not needed modification. However, in general, the application builder may wish to augment this knowledge with co-operation know-how which is specific to the application being developed. In the present implementation, this process is limited to the modification of existing functionality (i.e. the developer can change the way in which unrequested information is processed, but a new message type cannot be added to the system, nor can the PCM structure be altered other than in the modification of its control parameters). See [11] for a more detailed explanation of how domain-specific reasoning can be added to the system.

This corpus of knowledge (generic plus application-specific) together with its associated structure (as described in Section 3.3) then forms the basis of the working PCM for a given application (Fig. 5). Although the generic knowledge is widely applicable, the functionality which it supports will vary in usefulness between scenarios. So, for example, in some cases the main form of co-operation will be through the volunteering of unrequested information to relevant agents [6]; whereas in others, most co-operative interactions will be through explicit requests for services and information [4, 5]. To provide the necessary degree of flexibility, there are various parameters which the application builder is able to tweak in order to tailor the PCM to best fit the given problem. At the finest level, the developer may alter the relative priorities of the operational blocks. So, for example, the block which deals with unrequested information may be given a higher (lower) priority than that of explicit requests. The developer also has to specify what constitutes a high priority, since it is important that these functions are processed in a timely and efficient manner.

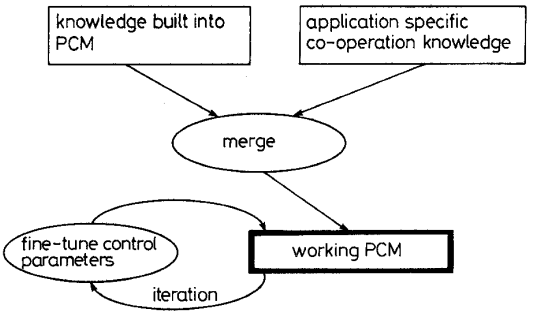

Fig. 5 *Instantiating the PCM*

The application designer then has to specify what constitutes a small number of messages for the PCM to process and what constitutes a large number. In between these two values, the PCM is operating in normal mode. These parameters are important because, as Table 1 indicates, the PCM behaves differently if it has a large number of messages to process from when it has a normal amount and when it has a small number. Finally, the policy for selecting the next submodule needs to be fixed.

As Fig. 5 highlights, the process of tuning the parameters is iterative. The designer sets up a first approximation for each of the agents, based on the experimental findings of Section 4, and then tests how they perform in their operational environment. As a result of this analysis, the parameters of one or more of the agents will be modified. This process continues until the community attains a satisfactory level of performance across a broad range of tasks.

4 Experiments with the PCM's meta-level control

The purpose of these experiments is to offer a quantitative means of evaluating how the setting of the various parameters of the PCM's control regime affect its performance. In evaluating a particular configuration the following main objectives should be borne in mind:

(i) the PCM should ensure that the agent's local objectives are met (Section 4.1)

(ii) the PCM should ensure that the agent assists its acquaintances in their processing where necessary and appropriate (Section 4.2)

(iii) the PCM should ensure that no messages are left unprocessed for a significant length of time (fairness criterion) (Section 4.3)

(iv) the PCM should ensure that as many messages as possible are processed in the available time (Section 4.4).

All of these objectives are interrelated and to a certain extent inconsistent with one another. Thus, for example, if the PCM decides to devote a large proportion of its time to furthering its local needs, then this may be detrimental to the attainment of its acquaintances' objectives. Similarly, when there are a large number of messages to process, the PCM may decide to focus on high-priority tasks in order to ensure that it maximises its local and global processing throughput, in which case certain less important message types may remain unprocessed within the PCM for a considerable period of time (contravening the fairness criterion).

The experiments described in the remainder of this Section are designed to assess the affect of the following parameters on the PCM's performance:

(i) the orientation of the agent

(ii) the submodule selection criterion

(iii) the relative effect of spending time in 'clear backlog' mode against 'normal' mode against 'busy' mode.

To offer a fair means of comparing the different configurations, the following factors remained constant in all the experiments: the duration of the experiment, the number of operational rule packages, the orientation of the operational rule packages, the priority of the operational rule packages, and the setting of what constitutes a high-priority rule package. The following assumptions were also made: message arrival rate[3] is uniformly distributed over the duration of the experiment (there are no sudden bursts of activity), each message is processed by only one operational rule package, the time taken to process a particular message is constant across all rule packages, the amount of time making control decisions is negligible in comparison to the time that the operational rule packages take to process a message, and the arrival of requests from acquaintances and the generation of new local goals are uniformly spread out over the duration of the experiment. So that the results reflect an unbiased evaluation of each configuration, it was important that the PCM did not start from scratch in each experiment. To overcome this problem, the configuration was allowed to reach a stable state (messages in all the buffers, tasks running in the underlying IS, outstanding requests made to acquaintances, etc.) before the measurements started. The values plotted in each of the following graphs are averaged over ten runs.

To provide an additional yardstick for comparison, two other common (but simple) control regimes were included in the experiments. The first-come, first-served approach had a common buffer for all the submodules (rather than the four separate ones) and messages were processed in the order in which they arrived. The depth-first approach placed a unique ordering on the operational rules within the PCM (i.e. the submodule level was removed) and then processed them in a round-robin order. When a particular operational rule package was invoked, *all* its associated messages were processed (irrespective of their arrival time or the load of the PCM).

[3] Arrival rate refers to unsolicited messages only. This would include, for example, an acquaintance spontaneously volunteering information or asking for a service to be provided, but not the case where the message is the result of a request that the agent has made

4.1 Achieving local objectives

In these experiments, agents only acquire new local objectives when they receive a piece of information which triggers one of their skills (i.e. there is no goal-driven activation of local skills). The number of locally motivated activations is directly proportional to the number of pieces of information which arrive at the HLCM; the percentage of message arrivals which generate new local goals is ~ 30 in all experiments. The chosen means of measuring how well an agent achieves its local objectives is to measure the percentage of its local goals that it completes. This percentage is computed from the number of potential local goals, rather than the actual number of local goals which are recognised; these two diverge when the agent receives information which would trigger a local skill but which it does not have the opportunity to process. Most skills require certain information to be present before they can be executed; in these experiments there was a uniform distribution between cases in which the skill could be activated immediately because the necessary information was already available (either because the agent had generated it from previous activity or because an acquaintance had sent it) and cases in which the necessary information was unavailable and so skill activation had to be delayed while the PCM initiated a social interaction to obtain it.

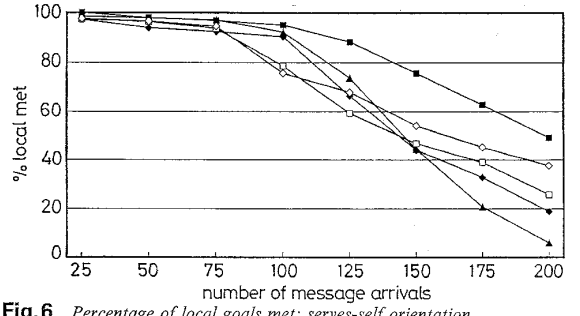

Fig. 6 *Percentage of local goals met: serves-self orientation*
–■– round robin –◆– busiest first –▲– shortest first
–□– first come, first served –◇– depth first

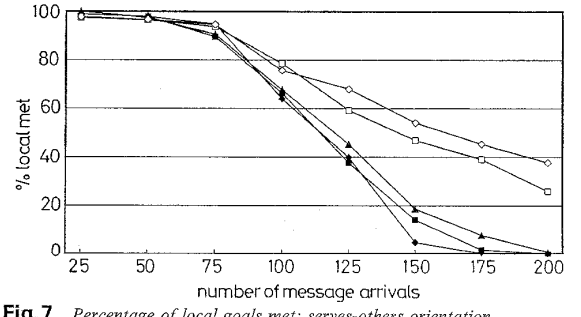

Fig. 7 *Percentage of local goals met: serves-others orientation*
–■– round robin –◆– busiest first –▲– shortest first
–□– first come, first served –◇– depth first

As can be seen from Figs. 6–8, all the configurations complete a very high percentage of their local goals when the number of message arrivals is low (less than 75). This is because there are so few messages in the buffers at any one time that the overriding selection

criterion is that of finding a rule package with a message to process; hence virtually all messages are dealt with.

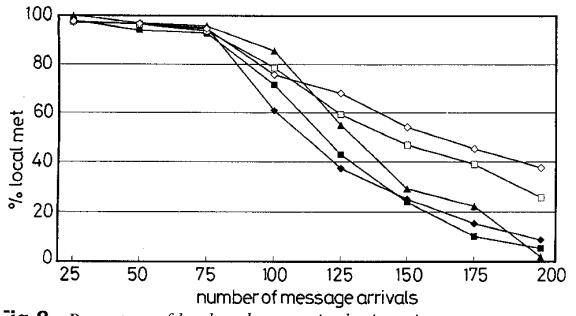

Fig. 8 *Percentage of local goals met: mixed orientation*
-■- round robin -◆- busiest first -▲- shortest first
-□- first come, first served -◇- depth first

For arrival rates above 75, the bias introduced by the agent orientation becomes the dominant factor. For the serves-others (Fig. 7) and mixed orientations (Fig. 8), the percentage of local goals completed drops markedly as the PCM starts to discriminate against those rule packages which are necessary for the completion of local goals. As this discrimination becomes worse (more time spent in busy mode) the number of local goals completed continues to fall until none of them are met. The depth-first and first-come, first-served policies fare better than both the mixed and the serves-others policies precisely because they do not discriminate against these rule packages. Depth-first is marginally better than first-come, first-served because it is less distracted by the large volume of new message arrivals which build up in the HLCM buffer as the simulation progresses. With the serves-self orientation (Fig. 6), the percentage of local goals met falls less sharply and remains at a higher overall value because as it becomes busier the PCM chooses to favour those rule packages which facilitate the completion of locally activated skills.

The submodule selection criterion is a less dominant factor in determining the amount of local processing which is completed. For the mixed and serves other orientations, there is very little difference between the three selection criteria. In both cases, however, shortest-first is the best choice because it concentrates the PCM's processing effort on the agent's ongoing activities at the expense of the range of new activities which arrive at the HLCM. With the serves-self orientation, on the other hand, by far the best selection criterion is round-robin, as this ensures that a significant amount of potential local goals which arrive later in the simulation are actually dealt with and result in new local activities which are subsequently completed (this can be achieved because of the bias towards rule packages which further local processing needs; with the other orientations round-robin spreads the PCM's resources too thinly). With a serves-self orientation, busiest-first performs better then shortest-first for large (greater than 150) arrivals because it ensures that more potential local goals become actual local goals (again this is only possible because of the discrimination in favour of the rule packages which assist this process).

7.2 Helping acquaintances achieve their objectives

When agents require assistance from their acquaintances they make a direct request (either for a particular piece of information to be provided or for a particular skill to be executed). The chosen means of gauging an agent's degree of helpfulness towards others is to measure the percentage of external requests which it completes. In this case, completion is defined as providing the desired service and returning the result to the originating agent. The number of external requests rises linearly with the number of message arrivals and they account for ~ 20% of the total in all the experiments. As with local goals, requested skills may require a social interaction to obtain the information which is needed to carry out its processing.

As with local processing, there is a relatively high completion rate for all orientations when there is a small number of messages arriving (Figs. 9–11); in most cases it is not as high as with the local processing because external requests require more activity to initiate and also because they are not deemed to be complete until the desired result has left the agent who is providing the service (local goals are deemed to be finished when the monitor returns the result to the PCM and the PCM starts to process it).

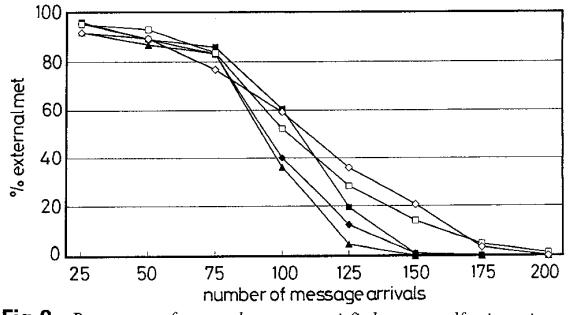

Fig. 9 *Percentage of external requests satisfied: serves-self orientation*
-■- round robin -◆- busiest first -▲- shortest first
-□- first come, first served -◇- depth first

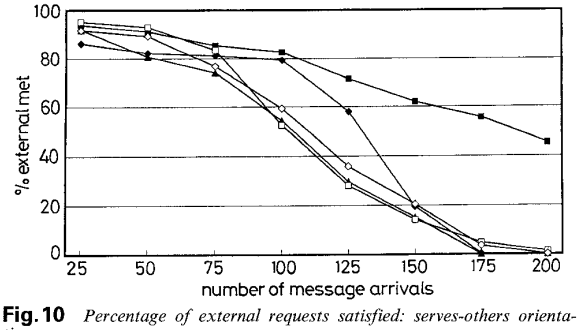

Fig. 10 *Percentage of external requests satisfied: serves-others orientation*
-■- round robin -◆- busiest first -▲- shortest first
-□- first come, first served -◇- depth first

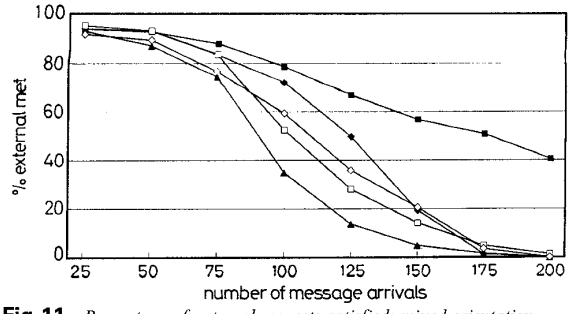

Fig. 11 *Percentage of external requests satisfied: mixed orientation*
-■- round robin -◆- busiest first -▲- shortest first
-□- first come, first served -◇- depth first

Again in these experiments, the dominant parameter is the orientation of the agent, although the submodule selection criterion plays a more important role than it did in the local processing measurements. With the serves-self orientation (Fig. 9), the completion rate falls very sharply and to a very low value once the arrival rate is greater than 75 (it reaches zero much more quickly than it does for the local processing measurements because more rule packages need to be invoked in order to complete an external request). In fact, the serves-self orientation performs worse than both the first-come, first-served and the depth-first ones because of its policy of active discrimination against those rule packages which are needed to process requests originating from acquaintances. First-come, first-served performs better than depth-first for small numbers of arrivals because it ensures that more of the external requests are recognised and acted upon. Above this arrival rate, however, the depth-first mode of operation is better because it is not unduly distracted by the large numbers of new messages which are from the HLCM and it ensures that the service completion messages which are needed to count external requests are dealt with in a systematic fashion.

With the serves-others (Fig. 10) and the mixed (Fig. 11) orientations, the percentage of external requests satisfied falls off much more gradually as the number of arrivals increases. Serves-others outperforms the mixed orientation as it places greater emphasis on those rule packages which assist with the processing of external requests.

In all cases in which there are a significant number of arrivals (greater than 75), the best submodule selection criterion is round-robin; this strikes a good balance between maintaining ongoing activities and starting new ones (with the serves-others and mixed orientations, this policy is far superior to the others because more time is devoted to rule packages which help with the processing of external goals, and hence it is important to obtain a balance of new and ongoing activities). Busiest-first is better than shortest-first (especially in the serves-others and mixed orientations) because it focuses processing on the two HLCM buffers; this not only ensures that more new external requests are brought into the system, but also that messages which report successful completion are dealt with promptly (this is necessary before an external request can be counted as finished).

4.3 Fairness of processing

The chosen means of assessing the fairness of a given PCM configuration is to determine the percentage of messages which remain within its internal buffers for a 'significant amount of time' — in this case greater than 50 time units. Messages which remain within the buffers for longer than this threshold value are deemed to have been starved of processing and thus have been dealt with unfairly. The graphs show the percentage of all the messages which the PCM has had to process which fall into this category.

As Figs. 12–14 illustrate, none of the PCM configurations process messages unfairly when there are a small number (less than 75) of messages to deal with. The overall fairest policies are first-come, first-served and depth-first because they do not discriminate against any message types and hence spread their processing around evenly. Of the two, depth-first performs better because all message categories are processed at regular intervals and related functionality is processed in close temporal proximity (in first-come, first-served, once a message has been processed it has to go to the end of the queue, which means there will be a significant delay before it is dealt with again if there are a large number of messages in the system).

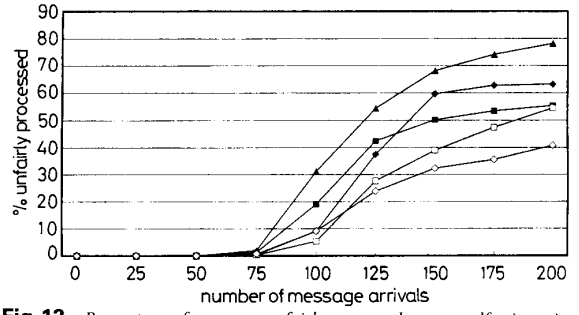

Fig. 12 *Percentage of messages unfairly processed: serves-self orientation*
–■– round robin –◆– busiest first –▲– shortest first
–□– first come, first served –◇– depth first

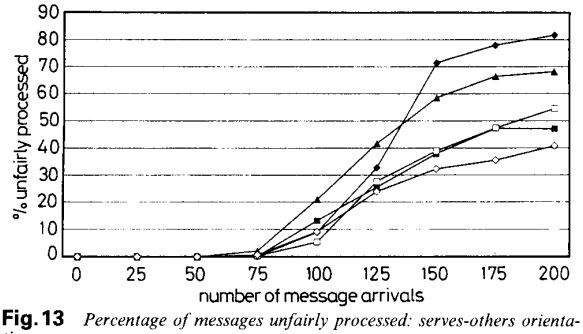

Fig. 13 *Percentage of messages unfairly processed: serves-others orientation*
–■– round robin –◆– busiest first –▲– shortest first
–□– first come, first served –◇– depth first

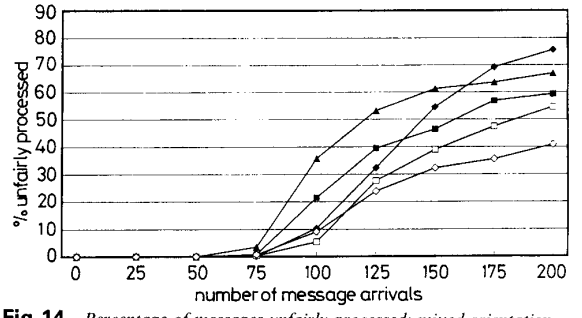

Fig. 14 *Percentage of messages unfairly processed: mixed orientation*
–■– round robin –◆– busiest first –▲– shortest first
–□– first come, first served –◇– depth first

When there are a medium number of message arrivals (between 75 and 125) the fairest submodule selection criterion is busiest-first. This policy ensures that large backlogs of unprocessed messages do not build up because it directs the PCM to those buffers which are the busiest. In this range, the worst performance configuration is shortest-first; this policy results in a large build-up of unprocessed messages at the HLCM buffer which are only started on when the PCM has very few other activities to perform. The busiest first policy becomes counterproductive as the number of messages becomes large (greater than 125) because it means that the PCM concentrates on getting new activ-

ities under way at the expense of those which it has already managed to get started. On this performance index the most consistent overall strategy is round-robin; it ensures that each of the submodules is dealt with in turn and thus reduces the likelihood of starvation. Shortest-first does particularly badly with the serves-self orientation because it means that virtually all of the external requests which are made are ignored by the PCM as it becomes busier.

4.4 Processing throughput

This metric is designed to give an indication of the overall efficiency of the PCM configuration. It measures the number of messages which the PCM is able to complete the processing of in the available time; thus, for example, with messages containing volunteered information they must be processed by both the CM's and the SAM's unrequested information available operational rule package before they can be deemed as completed. Partially processed messages (i.e. those processed by only a subset of the necessary rule packages) are deemed to be unprocessed for these purposes.

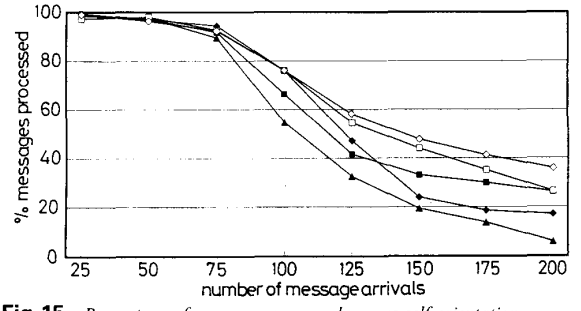

Fig. 15 *Percentage of messages processed: serves-self orientation*
■ round robin -◆- busiest first -▲- shortest first
-□- first come, first served -◇- depth first

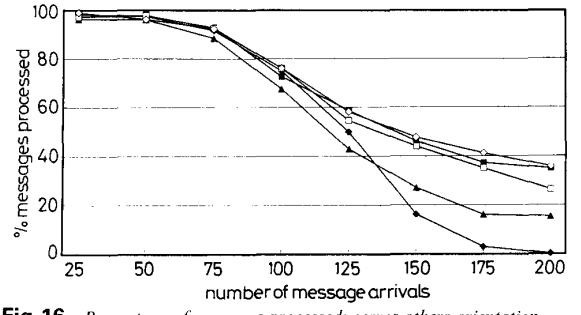

Fig. 16 *Percentage of messages processed: serves-others orientation*
-■- round robin -◆- busiest first -▲- shortest first
-□- first come, first served -◇- depth first

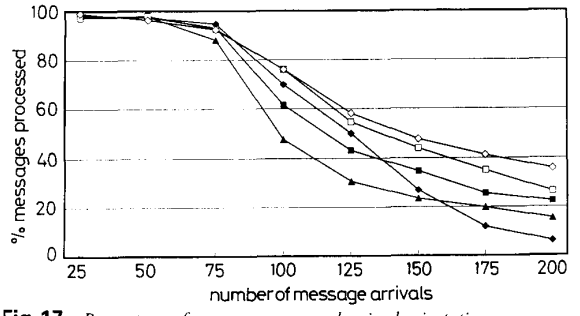

Fig. 17 *Percentage of messages processed: mixed orientation*
-■- round robin -◆- busiest first -▲- shortest first
-□- first come, first served -◇- depth first

As Figs. 15–17 show, all of the PCM configurations process a very high percentage of their messages when the arrival rate is low (less than 75). However, as the arrival rate increases, so the percentage of messages processed gradually begins to decline. All three orientations exhibit broadly similar patterns of behaviour: busiest-first is marginally the best policy in the medium ranges (because a large number of the quickest to process message type (unrequested information arrived) can be processed); round-robin is the best for large numbers of messages (effort spread out over a number of activities — not just initiating the processing of a few message types); shortest-first is the worst policy most of the time (relatively few unrequested information messages dealt with). However with the serves others (Fig. 16) and mixed (Fig. 17) orientations, busiest first performs the worst for large numbers of messages. This is because too much time is spent getting new messages into the system at the expense of devoting resources to activities that could be completed if they were allocated slightly more processing time. The overall best performance on this metric is given by a serves-others orientation and setting the submodule selection criterion to round-robin. This configuration ensures that processing is divided equally between all of the four submodules and that most of the really time-consuming activities (dealing with external requests) are not unduly delayed. The first-come, first-served and depth-first policies do best on this metric because they process messages in a systematic manner. Of the two, depth-first is marginally better because it means messages are not unduly delayed by the large queues which can build up.

4.5 Discussion

These experiments show how the PCM can be made to exhibit different behaviour simply by changing a few of its key control parameter settings. Such flexibility is essential because the PCM has been designed to be used in a number of different application contexts and to control a number of different types of agent (e.g. databases, expert systems, planners) which play a different role in the multiagent community. For example, a database agent is typically a provider of information to the others in the community, whereas an expert system agent typically has the role of solving the problem for which it was designed and its inclusion in a multiagent context is to exploit the opportunities for interactions provided by its acquaintances. The builder of an ARCHON application can use these results to broadly give individual agents their desired properties and then fine tune the settings to produce the optimal configuration for his particular application (as described in Fig. 5).

The experiments show that there is no universally best configuration, each combination of settings gives varying degrees of satisfiability along the key performance dimensions of achieving local goals, being helpful to others, processing messages fairly, and having a large throughput of messages. Setting the orientation to serves-self ensures that a high percentage of local goals are met, but that a low percentage of external ones are dealt with. A serves others orientation has the opposite properties. A mixed orientation ensures a reasonable number of external and local goals are met, that fewer messages on average are significantly late, but that there is a lower throughput. The submodule selection

criteria have a similarly radical affect on the PCM's performance: round-robin ensures that all the different message types are dealt with in a systematic and fair manner; busiest-first ensures that the newly arriving messages from the HLCM are dealt with promptly and not left to build up; shortest-first ensures that ongoing activities are given priority over starting fresh ones.

5 Conclusions

This paper has described the rationale, design and implementation of ARCHON's planning and co-ordination module. This module has been used to instantiate co-operative problem-solving in a number of real-world control applications — at the time of writing there are approximately 17 PCMs running in four different industrial settings [11]. The novel approach of utilising a corpus of in-built generic knowledge about co-operation and situation assessment has been explained and a number of empirical experiments have been undertaken to assess the quantitative affect on a number of key dimensions of changing the PCM's control parameters. This analysis is a significant aid to the agent designer in that it provides guidance on the tradeoffs involved in configuring the PCM for a given application.

For the future, there are a number of issues which require further investigation. First, the co-operation paradigms encoded in the PCM are relatively straightforward; how will the reusable knowledge approach cope with more sophisticated scenarios? Secondly, the corpus of generic knowledge has been devised from the perspective of a particular class of actions (i.e. industrial control); will it also be appropriate in domains such as office systems, telecommunications network management, concurrent engineering and enterprise integration? Thirdly, the prospect of the PCM adapting itself to its environment at run time needs to be explored. Finally, there is a need to develop a model which relates the control decisions of individual agents to the performance of the overall community so that the application developer can devise optimal global policies.

6 Acknowledgments

The work described in this paper was carried out in the ESPRIT II project ARCHON (P2256), whose partners were: Atlas Elektronik, Framentec-Cognitech, Labein, Queen Mary and Westfield College, IRIDIA, Iberdrola, EA Technology, Amber, Technical University of Athens, FWI University of Amsterdam, CAP-Volmac, CERN and University of Porto.

7 References

1 JENNINGS, N.R.: ' Cooperation in industrial multi-agent systems' (World Scientific Press, 1994)
2 JENNINGS, N.R., and WITTIG, T.: 'ARCHON: Theory and practice', *in* AVOURIS, N.M., and GASSER, L., (Eds.): 'Distributed artificial intelligence: Theory and practice' (Kluwer, 1992), pp. 179–196
3 WITTIG, T. (Ed.): 'ARCHON: An architecture for multi-agent systems' (Ellis Horwood, 1992)
4 VARGA, L.Z., JENNINGS, N.R., and COCKBURN, D.: 'Integrating intelligent systems into a cooperating community for electricity distribution management', *Expert Syst. Appl.*, 1994, 7, (4), pp. 563–579
5 COCKBURN, D., and JENNINGS, N.R.: 'ARCHON: A distributed artificial intelligence system for industrial applications' *in* O'HARE, G.M.P., and JENNINGS, N.R., (Eds.): 'Foundations of distributed artificial intelligence' (Wiley, 1996), pp. 319–344
6 JENNINGS, N.R., CORERA, J., LARESGOITI, I., MAMDANI ,E.H., PERRIOLAT, F., SKAREK, P., and VARGA, L.Z.: 'Using ARCHON to develop real-word DAI applications for electricity transportation management and particle accelerator control', *IEEE Expert*, **11**
7 STASSINOPOULOS, G., and LEMBESSIS, E.: 'Application of a multi-agent cooperative architecture to process control in the cement factory'. ARCHON technical report 43/3-93, 1993
8 OLIVEIRA, E., CAMACHO, R., and RAMOS, C.: 'A multi-agent environment in robotics', *Robotica*, 1991, **4**, (9)
9 JENNINGS, N.R., VARGA, L.Z., AARNTS, R.P., FUCHS, J., and SKAREK, P.: 'Transforming standalone expert systems into a community of cooperating agents', *Int. J. Eng. Appl. Artif. Intell.*, 1993, **6**, (4), pp. 317–331
10 JENNINGS, N.R.: 'The ARCHON system and its applications'. Proc. of second international working conference on *Cooperating knowledge based systems*, Keele, United Kingdom, 1994, Invited paper, pp. 13–29
11 JENNINGS, N.R., and POPLE, J.A.: 'Design and implementation of ARCHON's coordinating module'. Proc. workshop on Cooperating knowledge based systems, Keele, United Kingdom, 1993, pp. 61–82
12 JENNINGS, N.R., MAMDANI, E.H., LARESGOITI, I., PEREZ, J., and CORERA, J.: 'GRATE: A general framework for cooperative problem solving', *J. Intell. Syst. Eng.*, 1992, **1**, (2), pp. 102–114
13 GASSER, L., BRAGANZA, C., and HERMAN, N.: 'Implementing distributed artificial intelligence systems using MACE'. Proc. Third IEEE conference on *Artificial intelligence applications*, 1987, pp. 315–320
14 BURMEISTER, B., and SUNDERMEYER, K.: 'Cooperative problem-solving guided by intentions and perception', *in* WERNER, E., and DEMAZEAU, Y., (Eds.): 'Decentralized A.I., vol. 3' (Elsevier, 1992), pp. 77–92
15 SHOHAM, Y., THOMAS, B., SCHWARTZ, A., and KRAUS, S.: 'Preliminary thoughts on an agent description language', *Int. J. Intell. Syst.*, 1991, **6**, pp. 497–508
16 HAUGHENDER, H.: 'IMAGINE final project report'. IMAGINE, Esprit Project 5362, 1994, Siemens
17 DURFEE, E.H., LESSER, V.R., and CORKILL, D.: 'Coherent cooperation among communicating problem solvers', *IEEE Trans.*,1987, **C–36**, pp. 1275–1291
18 BLUM, B.I.: 'The fourth decade of software engineering: some issues in knowledge management', *J. Intell Cooperative Inf. Syst.*, 1992, **1**, (3, 4), pp. 475–514
19 COX, B.J.: 'Planning the software industrial revolution', *IEEE Softw.*, 1990, **7**, pp. 25–33
20 NECHES, R., FIKES, R., FININ, T., GRUBER, T., PATIL, R., SENATOR, T., and SWARTOUT, T.: 'Enabling technology for knowledge sharing', *AI Mag.*, 1991, **12**, pp. 36–56
21 STEFIK, M.: 'The next knowledge medium', *AI Mag.*, 1986, **7**, (1), pp. 34–46
22 SMITH, R.G.: 'The contract net protocol: high level communication and control in a distributed problem solver', *IEEE trans.*, 1980, **29**, (12), pp. 1104–1113
23 JENNINGS, N.R.: 'Using GRATE to build cooperating agents for industrial control'. Proc. IFAC/IFIP/IMACS international symposium on *Artificial intelligence in real time control*, Delft, The Netherlands, 1992, pp. 691–696
24 JENNINGS, N.R.: 'Commitments and conventions: The foundation of coordination in multi-agent systems', *Knowl. Eng. Rev.*, 1993, **8**, (3), pp. 223–250
25 SIMON, H.A.: 'Artificial intelligence: An empirical science', *Artif. Intell.*, 1995, 77, (1), pp. 95–127
26 HAYES-ROTH, B.: 'Intelligent control', *Artif. Intell.*, 1993, **59**, pp. 213–220
27 DAVIS, R.: 'Meta-rules: Reasoning about control', *Artif. Intell.*, 1980, **15**, pp. 179–122
28 HUDLICKA, E., and LESSER, V.R.: 'Meta-level control through fault detection and diagnosis'. Proc. 1984 conf. of the American Association of Artificial Intelligence, 1984, pp. 153–161