Engng Applic. Artif. Intell. Vol. 6, No. 4, pp. 317–331, 1993
Printed in Great Britain. All rights reserved

Contributed Paper

Transforming Standalone Expert Systems into a Community of Cooperating Agents

N. R. JENNINGS
Queen Mary and Westfield College, University of London

L. Z. VARGA
Queen Mary and Westfield College, University of London

R. P. AARNTS
Volmac Nederland B. V., Utrecht

J. FUCHS
PS Division, CERN, Geneve

P. SKAREK
PS Division, CERN, Geneve

(Received October 1992; in revised form February 1993)

Distributed Artificial Intelligence (DAI) systems in which multiple problem-solving agents cooperate to achieve a common objective is a rapidly emerging and promising technology. However, as yet, there have been relatively few reported cases of such systems being employed to tackle real-world problems in realistic domains. One of the reasons for this is that DAI researchers have given virtually no consideration to the process of incorporating pre-existing systems into a community of cooperating agents. Yet reuse is a primary consideration for any organisation with a large software base.

To redress the balance, this paper reports on an experiment undertaken at the CERN laboratories in which two pre-existing and standalone expert systems for diagnosing faults in a particle accelerator were transformed into a community of cooperating agents. The experiences and insights gained during this process provide a valuable first step towards satisfying the needs of potential users of DAI technology—identifying the types of changes required for cooperative problem solving, quantifying the effort involved in transforming standalone systems to ones suitable for cooperation and highlighting the benefits of a cooperating systems approach in a realistic industrial application.

Keywords: Distributed Artificial Intelligence, multi-agent systems, cooperating expert systems, fault diagnosis, particle accelerator control.

1. INTRODUCTION

As computer hardware and software becomes increasingly powerful, so applications which used to be considered beyond the scope of automation come into reach. To cope with these increased demands, software systems are becoming correspondingly larger and more complex. However the problems encountered in building these large systems are not simply scaled-up versions of those faced when constructing small ones. Since the late 1960's, when the "software crisis" was first noted, it has been realised that large systems require radically different techniques and methods.

One paradigm for overcoming the complexity barrier is to build systems of smaller, more-manageable components which can communicate and cooperate.[1–3] Such a Distributed Artificial Intelligence (DAI) approach has several potential advantages. Firstly, divide and conquer has long been championed as a means of constructing large systems because it limits the scope of each processor. The reduced size of the

Correspondence should be sent to: Dr N. R. Jennings, Department of Electronic Engineering, Queen Mary and Westfield College, University of London, Mile End Road, London E1 4NS, U.K.

input domain means the complexity of the computation is lower, thus enabling the components to be simpler and more reliable. Secondly, decomposition aids problem conceptualisation; many tasks appear difficult because of their sheer size. Other benefits include reusability of problem-solving components, greater robustness in the case of component failure, speed-up due to parallel execution, enhanced problem solving due to the combination of multiple paradigms and sources of information and finally, increased system modularity.[1,3]

In DAI systems, individual problem-solving entities are called *agents*; agents are grouped together to form *communities* which cooperate to achieve the goals of the individuals and of the system as a whole. It is assumed that each agent is capable of a range of useful problem-solving activities in its own right, has its own aims and objectives and can communicate with others. The ability to solve some problems alone (*coarse granularity*[4]) distinguishes components of DAI systems from the components of neural network systems in which individual nodes have very simple states (either on or off) and only by combining many thousands of them can problem-solving expertise be recognised.

Agents in a community usually have problem-solving expertise which is related, but distinct, and which frequently has to be combined to solve problems. Such joint work is needed because of the dependencies between agents' actions, the necessity of meeting global constraints and the fact that often no one individual has sufficient competence to solve the entire problem alone. There are two main causes of such interdependence (adapted from Ref. 5). Firstly, problem partitioning may yield components which cannot be solved in isolation. In speech recognition, for example, it is possible to segment an utterance and work on each component in isolation, but the amount of progress which can be made on each segment is limited. Allowing the sharing of hypotheses is a far more effective approach.[6] Secondly, even if subproblems are solvable in isolation, it may be impossible to synthesize their results because the solutions are incompatible or because they violate global constraints. For instance, when constructing a house, many subproblems are highly interdependent (e.g. determining the size and location of rooms, wiring, plumbing, etc.). Each is solvable independently, but conflicts which arise when the solutions are collected are likely to be so severe that no amount of work can make them compatible. It is also unlikely that global constraints (e.g. total cost less than £70,000) would be satisfied. In such cases, compatible solutions can be developed only by having interaction and agreements between the agents during problem solving. It is the need for significant amounts of cooperation to achieve tasks and the relative autonomy of the agents to determine their own activities which distinguishes DAI from more-conventional distributed-systems work.

Despite these potential advantages, there are still relatively few DAI applications working on realistic problems in real-world domains.[7] One of the reasons for this is the mismatch between the needs of organisations who require solutions to their problems and the research objectives of the DAI community. Typically, organisations which have problems amenable to a cooperating systems approach already possess computer systems in which they have invested substantial amounts of time and money. Naturally enough, they want the return on this investment to be maximised, meaning the systems should be utilised until they become obsolete, or significantly better alternatives become available. However, most work in DAI assumes that the agents have been purpose-built using tools and techniques designed solely for cooperative problem solving. Virtually no consideration is given to the process of incorporating pre-existing systems into a cooperating community – yet this must be a central concern if DAI is to leave the laboratory and progress to real applications.

To help redress the balance, this paper reports on an experiment carried out at the CERN laboratory, under the auspices of the ARCHON project,[8,9] in which two standalone and pre-existing expert systems for diagnosing faults in a particle accelerator were transformed into a community of cooperating agents. The problems faced during this process and the insights which emerged are recounted as an important first step towards tackling the larger issue of providing a methodology for describing how pre-existing systems can be incorporated into a cooperating community. The framework used to facilitate the cooperative problem solving was GRATE[10,11] which is described more fully in the following section. This paper also indicates the types of social interaction which can be expected between large, coarse-grained agents—an important indication of the appropriateness of theoretical research into techniques for cooperation and coordination for real-size industrial applications. Finally because DAI (and AI in general) techniques are so rarely applied to real applications, this experiment offers valuable insights into the problems and constraints encountered during this process—such issues often fail to emerge when idealised problems or simulated environments are used.[12]

2. GRATE: A GENERAL FRAMEWORK FOR COOPERATIVE PROBLEM SOLVING

GRATE (Generic Rules and Agent model Testbed Environment) is a general framework for constructing communities of cooperating agents for industrial applications. GRATE agents have two major components—a *cooperation and control layer* and a *domain level system* (see Fig. 1). The domain-level system solves problems such as detecting and diagnosing faults, proposing remedial activities and checking the validity of

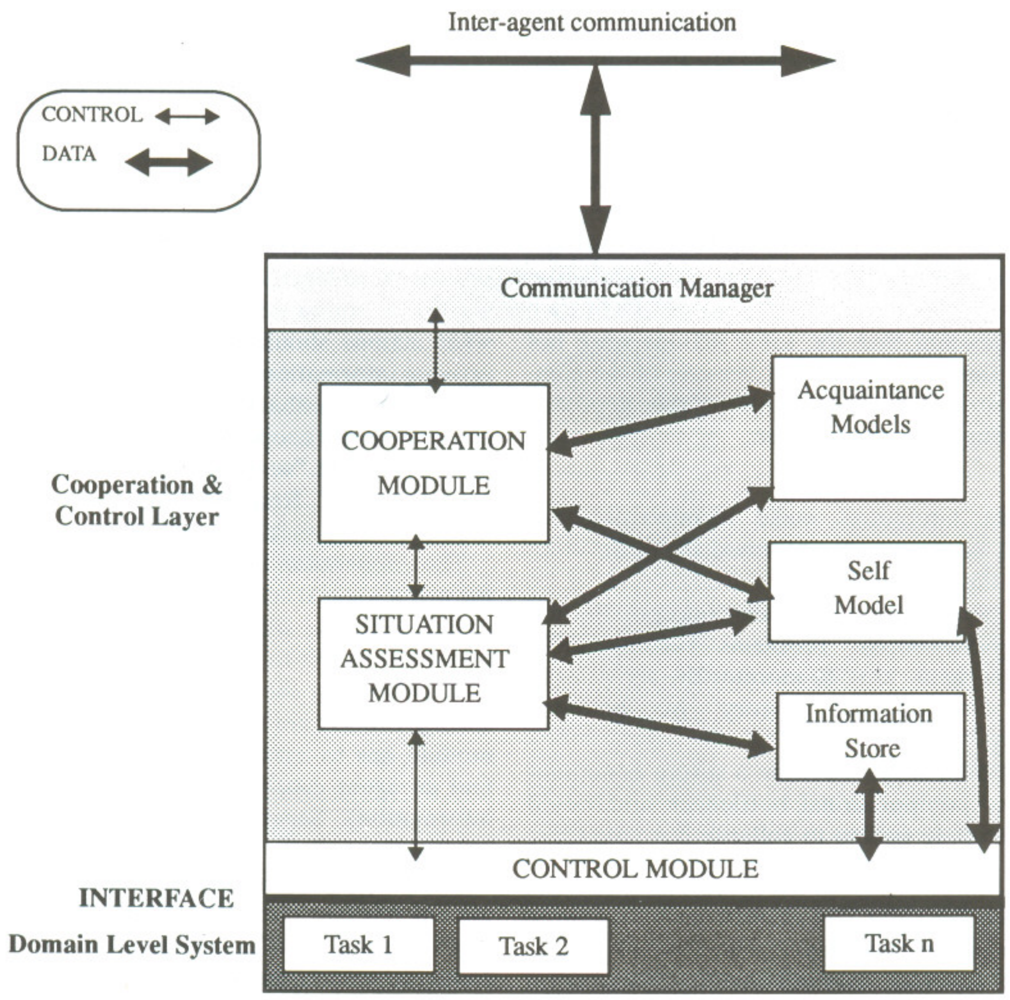

Fig. 1. Detailed GRATE agent architecture.

operator actions. These problems are expressed as tasks—atomic units of processing when viewed from the cooperation and control layer. The cooperation layer is a meta-level controller which operates on the domain-level system; its objective is to ensure that the agent's activities are coordinated with those of others within the community. It decides which tasks should be performed locally, determines when social activity is appropriate, receives requests for cooperation from other community members, and so on.

GRATE's clear delineation of domain problem solving and knowledge related to cooperation and control has several advantages. Firstly, it increases software reusability in that the cooperation layer can be deployed in multiple applications without having to disentangle the knowledge used to guide social activity from that used to solve domain-level problems. Secondly, the domain and cooperation layers can be developed independently, provided that they respect the interface definition. This is especially important with respect to pre-existing software because it places very few restrictions or constraints on the types of system which can be incorporated. Thus the domain-level systems may be written on different hardware and software platforms, use different knowledge representation formalisms and have different control regimes.[13] By providing a standard interface to the domain-level system, all of the underlying heterogeneity can be masked; enabling the cooperation layer to be used in a wide range of applications. The division also simplifies the domain-level system because it can continue to act on the basis of local information; it need not be concerned with the activities of other agents because any influence they have on its behaviour will be exerted through the cooperation layer.

The disadvantage of the delineation, with respect to pre-existing systems, is that the control regime and some interface interpretation commands may need to be written to enable the cooperation layer to exact the appropriate control—this issue is discussed further in the section describing the implementation details. Also, for systems which are purpose-built for cooperative problem solving in a particular environment, this architecture may involve creating an artificial barrier in its local control regime if the interface definition is too rigid.

In GRATE communities control is completely distributed, there is no hierarchy and all agents are equal.

Agents have a degree of autonomy in generating new activities and in deciding which tasks to perform next. A global controller would have been the easiest way of ensuring the cooperating community acted in a coherent way; with its knowledge of the goals, actions and interactions of all community members the controller could have ensured: that misleading and distracting information was not spread, that multiple agents did not compete for unshareable resources simultaneously, that agents did not unwittingly undo the results of each other's activities and that the same actions were not carried out redundantly. However because of the complexity of most industrial applications this approach was deemed inappropriate because:

- Bandwidth limitations make it impossible for agents to be constantly informed of all developments in the system.[5]
- A local perspective simplifies conceptualisation and implementation. Problems become more complex if agents have to monitor the activities of all the others—the theory of bounded rationality.[14]
- The controller would become a severe communication and computational bottleneck and would cause the whole system to collapse if it failed.[15]

GRATE's cooperation and control layer has three main problem-solving modules. Each module is implemented as a separate forward-chaining production system with its own inference engine and local working memory. Communication between the modules is via message passing. The rules built into each module are generic; they are applicable for controlling activities in a broad class of industrial applications. Some credence to this claim is given by the fact that GRATE has been applied to the domain of electricity transportation management[10] and detecting overloads in a telecommunication network[16] as well as the problem of diagnosing faults in a particle accelerator which is reported here. A more comprehensive description of GRATE can be found in Ref. 17.

The *control module* is GRATE's interface to the domain-level system and is responsible for managing all interactions with it. This interaction is controlled through the following set of primitives:

- From GRATE to the domain-level system:

```
<START task inputs>   <STOP task>
<SUSPEND task>        <KILL task>
<REACTIVATE task>
<MORE-INFO-AVAILABLE task info>
```

- From the domain-level system to GRATE:

```
<DIRECTIVE-FAILED directive>
<RESULTS-PRODUCED task>
<DIRECTIVE-EXECUTED directive>
<FINISHED task>
```

The *situation assessment module* makes decisions which affect both of the other modules. It decides which activities should be performed locally and which should be delegated, which requests for cooperation should be honoured, how requests should be realised, what actions should be taken as a result of freshly arriving information and so on. It issues instructions to, and receives feedback from, the other modules. Typical requests to the cooperation module include "get information X" and "send out information Y to interested acquaintances". Requests to the control module are of the form "stop task T1" and "start task T2".

The *cooperation module* is responsible for managing the agent's social activities. The need for such activity is detected by the situation assessment module, but its realisation is left to this module. Three primary objectives related to the agent's role in a social problem-solving context are supported. Firstly, the cooperation module has to establish new social interactions (e.g. find an agent capable of supplying a desired piece of information). Secondly, the module has to maintain cooperative activity once it has been established, tracking its progress until successful completion (e.g. sending out relevant intermediate and final results to interested agents). Finally, the module has to respond to cooperative initiations from other agents.

The cooperation layer's other components provide support for the activities of the main problem-solving modules. The *information store* provides a repository for all the data which the underlying domain-level system has generated or which has been received as a result of interaction with others. *Acquaintance and self models* are representations of other agents and of the local domain-level system respectively. They describe the agent's current problem-solving state, the tasks it is able to solve, the goals it is working towards and so on—a fuller description is given in Ref. 10 and an illustration of their use in the CERN experiment is given in the next section. The agent models and the information store contain all the domain-dependent data needed at the cooperation and control layer—thus enabling the rules of the problem-solving modules to be application-independent.

Agents communicate with one another via a message-passing paradigm. This form of communication has several advantages over a shared-memory approach (such as a blackboard[18,19]). Firstly, message passing has well understood semantics and offers a more abstract means of communication.[20] No hidden interactions can occur; so there is greater comprehensibility, reliability and control over access rights. Secondly, message passing makes fewer assumptions about system architecture. Finally, shared-memory systems do not easily scale up. If only a single blackboard exists then it becomes a severe bottleneck and if several exist the semantics revert to message passing.[21]

3. DIAGNOSING FAULTS IN A PARTICLE ACCELERATOR

This section describes the diagnosis problem of accelerator operation, details two pre-existing expert systems used for this task which are running at the CERN laboratories and outlines the potential benefits of cooperation in this application.

3.1. Particle accelerator operation

The CERN Proton Synchrotron (PS) accelerators are one of the world's most sophisticated high-energy research tools. The PS complex is at the heart of CERN's experimental facilities acting as an injecter for the larger accelerators—the Super Proton Synchrotron and the huge Large Electron Positron rings. In the PS, nuclear particles are focused into particle beams, accelerated, and directed through several linear and ring accelerators using electromagnetic fields. These beams are then used in the physicists' experiments. Different experiments require different types of beam; the variations are provided by the accelerator's different operational modes.

The PS complex is controlled by a team of operators who maintain the beam performance and the operational modes of the accelerators. Accelerator operation is a task demanding technical competence, experience, diagnostic skill and judgement. The operator has to handle information coming from control-system modules, accelerator components and the acceleration process itself. Observations are made via measurement devices and range from simple status displays of specific accelerator components to complicated application programs and graphical information. The operator can change setpoint control values for the different components directly or he can use application programs which present the beam properties on a higher conceptual level. Different sections of the accelerator are controlled from different consoles in the same control room. In the present system if there is some suspicion about a problem in an overlapping area, operators communicate by talking to their colleagues on other consoles.

The operator's workload is constantly increasing as new accelerators and different operational modes are added to the system. To cope with this increase, automated tools are becoming an integral part of the control process. CERN has already equipped their control room with several supporting tools—ranging from simple ones that display status information to high-level software based on expert-system technology.[22–26] This experiment on cooperative problem solving concentrated on two of the high-level tools, BEDES and CODES, which employ expert-systems technology for diagnosing faults in the accelerator.

3.2. BEDES and CODES

The main goal of *BEDES* (BEam Diagnostic Expert System) is to diagnose operational faults at the beam level for the PS Booster injection part of the PS complex. Operational faults occur, for example, if the intensity of the particle beam falls below a certain level or if the beam deviates considerably from its ideal trajectory. Such problems can be caused by the incorrect setting of a control parameter (e.g. wrong timing is set for a switching magnet), a breakdown in a controller (e.g. the switching magnet is not working), or an error in the control system (e.g a module that controls the switching magnet is down). BEDES can diagnose the first two types of fault if the underlying control system is still working correctly. If such faults are detected, BEDES tries to recover from them by resetting the correct control value or by optimizing a control value respectively.

CODES (COntrol Diagnostic Expert System) has a similar control structure to BEDES, the main difference being in the domain knowledge. Whilst BEDES works on the beam level, CODES operates on the level of the accelerator's control system. This control system consists of thousands of hardware and software modules in a large computer network and is used by the operator to ensure that the accelerator process works successfully. A fault in the control system usually manifests itself in terms of a deviation of the particle beam and so the problem will be picked up by BEDES. In such cases BEDES and CODES could work together to determine the source of the fault. BEDES could help to detect whether the problem is caused by wrong parameter settings or a breakdown of a controller, while CODES could determine whether the fault is caused by the control system itself.

3.3. Preparing for cooperative problem solving

When the particle accelerator is running, the operator receives a myriad of information from which he has to make an overall judgement of the situation and take reasonable decisions. As humans are resource-bounded, the operator is able to use only a limited number of tools and correlate very few pieces of information in real time. Therefore assistance is required in producing a consistent interpretation of the information. At present BEDES and CODES report information independently to the operator, who then has to translate it to a common domain, determine whether it is consistent and decide upon the appropriate course of action. If the tools were capable of exchanging information directly, they would be capable of producing a consistent view automatically, leaving the operator to concentrate on the more-cognitive activities (e.g. interpreting and acting upon diverse sources of information, tweaking the system to enhance performance, etc.) for which he is better suited. From here stems the motivation for introducing cooperation between the expert systems.[27, 28]

BEDES and CODES have knowledge about the diagnosis of the same particle accelerator from different perspectives. In certain situations, faults can be identified or even recovered from using only one of the systems, but in many instances contributions from both of them are needed. By exchanging intermediate and final results, the expert systems are able to focus each other's problem-solving activity on promising areas and draw each other away from unprofitable avenues of reasoning.

Initially cooperation between BEDES and CODES was studied by means of several paper exercises. Later practical exercises started and the expert systems were enhanced with an application-specific cooperation software which sent hypotheses directly from BEDES to CODES.[29] These preliminary studies identified some key design issues which needed to be addressed. These include: "How are local actions performed in one expert system?", "What is the common language of the expert systems?", "How does one expert system model the other expert system?" and "How does one expert system model itself?" Each of these questions is addressed in turn before the GRATE experiment is described in detail.

Local actions

The main unit of reasoning for both BEDES and CODES is the hypothesis. Hypotheses are stored on an agenda; the status of the agenda determines whether the expert system is active or just idling. At the beginning of each inference cycle the agenda is rearranged (see Fig. 2), which means that hypotheses are realigned according to their priority and any which have become obsolete are removed. The first hypothesis is then taken from the agenda, it is evaluated and possibly more-detailed descendent hypotheses are created and injected into the agenda. Evaluation requires that data describing the current status is gathered from the accelerator's control system (e.g. currently valid control values); reasoning about this data is then undertaken, the outcome of which is a change in state of the hypothesis being evaluated (e.g. from unconfirmed to confirmed). If the evaluated hypothesis is confirmed, the fault has been found and diagnosis stops—the results are then reported to the operator.

From this structure it is apparent that the natural unit of local activity is the basic inference cycle and that local action can be controlled through operations on the agenda (e.g. to stop diagnosis the agenda should be cleared and to focus on a promising hypothesis it should be moved to the beginning of the agenda). Using this strategy local and cooperative actions can be kept separate, and well-organised, cooperative features can be added to the expert systems without significantly modifying their existing reasoning mechanisms.

Common language

As both BEDES and CODES represent their hypotheses using a similar structure, it was decided to use this as the basis of a common language between the two agents. Hypotheses are assertions together with accompanying knowledge about how to prove them. The assertion is about an element or parameter of the accelerator which might be in an incorrect state or could go wrong in the near future. The related knowledge is composed of the necessary inference steps to prove the assertion and has two parts: *procedural steps* including data acquisition (e.g. read values from the control system, filter uncertainty and discrepancies and compute the derived parameter) and *declarative rules* operating on the structural description of the diagnosed system stored in the knowledge base of the expert

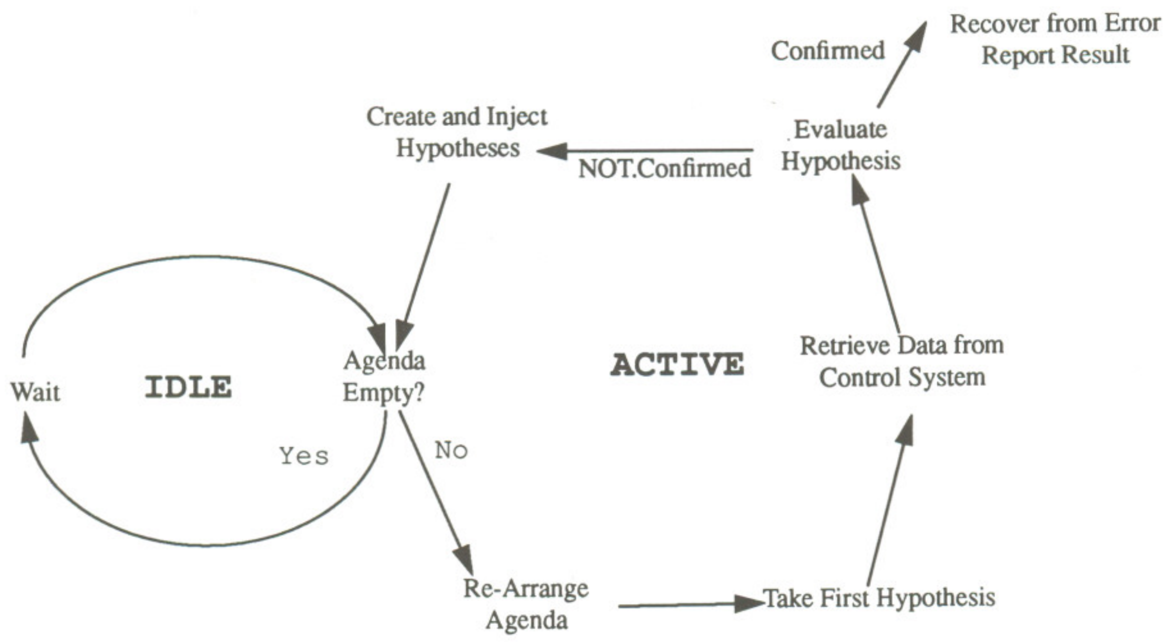

Fig. 2. Expert system control loop.

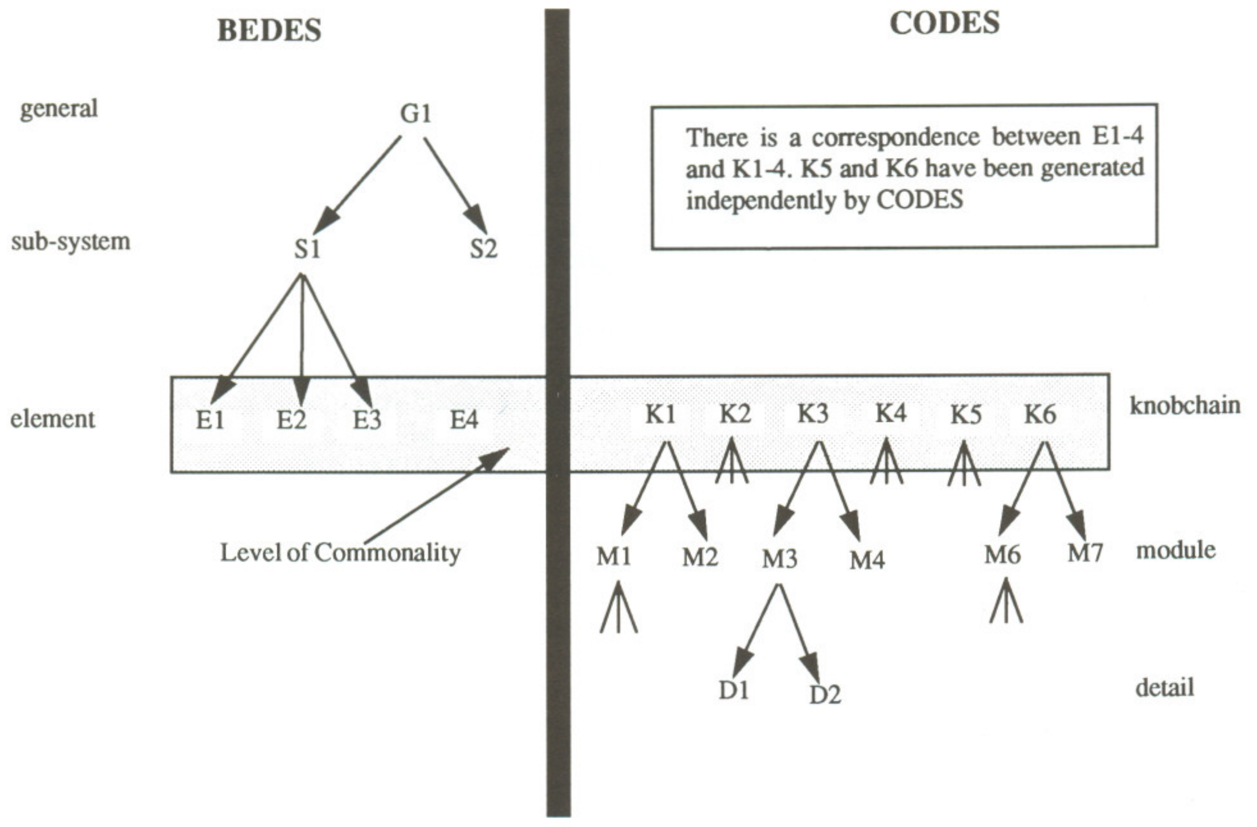

Fig. 3. Hierarchy of hypotheses.

system (e.g. "Is the value close enough to "ideal"? If not, then create derived hypotheses for those parts that can cause the deviation").

Hypotheses are implemented as frames. Although the structure of the hypotheses are the same for both expert systems, the contents of the slots are different. A *suspected-entity* slot describes the element which may be at fault. A *state-of-entity* slot provides detailed data about the state of a suspected entity—including information such as the element is in fault, is operating out of specification or is operating normally. During the evaluation cycle, this slot is used to indicate the progress of the fault finding. The *state-of-hypothesis* slot expresses the state of the hypothesis itself. Possible values include:

NOT.EVALUATED	The hypothesis is newly created and not yet evaluated.
NOT.CONFIRMED	The hypothesis could not be confirmed but no attempt has been made to deny it.
CONFIRMED	The hypothesis was confirmed but no recovery attempt has been made.

The state of a hypothesis is important for cooperation, because it contains information on the current phase of the diagnosis process. For example, if a hypothesis is confirmed by one agent, then the fault has been found and the other system should stop trying to locate it. Another example is that if a hypothesis is evaluated by an expert system because of a request from an acquaintance and it cannot be confirmed, then the originator should be informed since it affects its local problem-solving behaviour. A *rating* slot indicates the priority of a hypothesis and is used to order items on the agenda.

During evaluation the expert system might create new hypotheses of a more-detailed level resulting in a tree structure (see Fig. 3). BEDES and CODES are incapable of understanding each other's hypotheses directly because they refer to different domains—BEDES to sub-systems and elements, CODES to knobchains, modules and details. However there is a level of commonality, in that translation can be performed between element- and knobchain-level hypotheses. This process involves changing the value of some slots of the hypothesis and loading structural data into the knowledge base. For example, if BEDES suspects that something is wrong with a controller element (the element is the suspected-entity of a BEDES hypothesis), then CODES cannot directly use this. CODES has to map the element to that set of control system elements (knobchain) which operates this controller element. It also has to load into its knowledge base the structure of the knobchain. The structural knowledge is physically stored in a centralised and separate database for ease of maintenance and because of its sheer size. However for the purposes of this experiment, the database was regarded as part of CODES's domain-level system. So that the agents are not unnecessarily distracted by extraneous hypotheses which they cannot understand (e.g. CODES cannot use those of a general or sub-system level), agents represent the types of hypotheses that their acquaintances can process in their agent models (see the following section for more details). This knowledge is then used to guide hypothesis interchange.

The advantage of using the hypothesis as a common language is that it involves a minimal translation overhead. Also, it is close to the language used by the domain-level systems, which reduces the amount of modification required in the pre-existing systems. The disadvantage is that any new agents which may be added to the community at a later stage must also be able to represent and understand knowledge in this particular format. A better approach in terms of extensibility would be to construct a domain-independent interlingua in which assumptions about the knowledge-representation commitments are stated explicitly—see the work on the knowledge interchange format[30, 31] for a more-comprehensive discussion of this issue.

3.4. Benefits of a distributed AI approach

The benefits of a DAI approach in this particular domain include:

(1) As the accelerator and its control system consist of huge numbers of elements, corresponding to more than 10,000 setpoint control values, it would be extremely difficult to maintain and develop a centralised knowledge base for the whole process. Decomposing the problem into modules results in smaller subproblems which are much easier to tackle. A modularised approach also fits more naturally into the existing organisational structure—knowledge of different domains (located in different divisions or groups) can be separate, but can be combined by cooperative problem solving at runtime.
(2) The overall system will be *open*.[32] New agents covering different aspects of the particle accelerator process can be added when they are developed without having to alter the application's existing conceptual model. This is important because new accelerators are built and added during the lifetime of the accelerator complex, also new operational modes may be developed.
(3) The computing power of several workstations connected together through a network can be utilised. Thus the agents can work in parallel and produce results faster by sharing the workload.
(4) Some of the drudgery and non-cognitive aspects of the operator's job are removed, leaving greater time for the higher-level tasks which cannot be automated using the currently available technology.

4. THE GRATE CERN EXPERIMENT

This section describes how cooperative fault detection can be carried out using the methods and tools discussed above. A typical cooperative scenario involving BEDES and CODES is outlined, before the steps involved in transforming the stand-alone systems into a community of cooperating agents being controlled by GRATE are expanded upon.

4.1. A cooperative scenario

In the implemented scenario, the main form of cooperation manifests itself in terms of the intelligent sharing of information between the two agents. This information is used to indicate changes in the status of the particle accelerator and to direct agents' problem solving by sharing intermediate and final results.

There are three distinct phases to controlling the particle accelerator. Firstly, there are *normal operating* conditions in which no fault has been detected. In this phase BEDES monitors the accelerator system to identify possible discrepancies. Monitoring involves continually comparing measurable system properties (such as the particle beam's intensity, efficiency and trajectory) with their archived "ideal" values. If there is a significant discrepancy, then there is a possible fault in the accelerator. When a possible fault is detected, BEDES carries out a preliminary diagnosis phase and produces a list of hypotheses about suspected subsystem components.

Once BEDES has produced a list of hypotheses to explain the accelerator fault, the second phase of *verifying the cause* of the problem begins. The fault may have occurred as the result of a problem at the beam level or a fault with the control system. Therefore, as well as starting to verify the cause of the fault, BEDES also informs CODES that there is probably a faulty element in the accelerator. When CODES receives this notification, it starts a diagnostic process to determine whether the problem lies within the control system. At this stage BEDES and CODES share the common goal of trying to locate the accelerator's fault; they are looking at different aspects of the problem but their work is related by the fact that the hypotheses of CODES are further specialisations of BEDES's element-level hypotheses.

As the two agents proceed with their diagnoses, various possibilities for cooperation exist, based on the exchange of information about hypotheses. When such information is received, the recipient will undertake one of the following courses of action. A practical illustration of each case follows.

(1) Take no action if the information is not relevant to its current problem solving context.
(2) Use the information to deflect the focus of its problem-solving activity away from an unprofitable area.

(3) Use the information to concentrate its problem-solving activity on a promising area.

Case 1

As a result of its evaluation, BEDES creates some new element-level hypotheses. These hypotheses are sent to CODES, which translates them, before adding them to its agenda. As they are new hypotheses (status NOT.EVALUATED) they are merely added to the list of things to do, they do not affect the focus of CODES's current problem solving.

Case 2

BEDES evaluates an element-level hypothesis (denoted by H). As H is at the element level it would have already been sent to CODES when it was created (see Case 1). However, as a consequence of its evaluation task BEDES has produced more information about H. This additional information is either that H is NOT.CONFIRMED or that H is CONFIRMED (see Case 3 for the latter situation). In the former case, when CODES receives the NOT.CONFIRMED message it takes one of the following actions depending on its problem-solving context:

(a) CODES has not started working on H yet. Since BEDES was unable to confirm H, the chance that CODES will find a fault with the derivatives of H has been lowered. Knowing this, CODES's rating of H will be reduced, meaning that other, more-likely hypotheses will be dealt with first.
(b) CODES has started work on H or its derivatives. The probability that one of the hypotheses of the derived tree can be confirmed has been decreased. If there are other high-level hypotheses in its agenda, then CODES continues with those. CODES drops its attention on the hypothesis tree of H and will, after the next rearrange agenda, continue with a new tree.
(c) CODES has already finished the evaluation of H and all the hypotheses derived from it. If CODES was also unable to confirm any hypothesis in the tree of H, then this is further confirmation of BEDES's result and can be used to increase the operator's level of confidence in the information. However if CODES did find a fault, then there is a conflict. In this instance resolution is straightforward; since CODES works at a lower level than BEDES its results are assumed to be more reliable. The user is thus presented with the result of CODES and a short note about the confiict.

Case 3

If BEDES can confirm H, then the problem-solving effort of CODES now switches to concentrate on the hypothesis tree of H and its derivatives. If CODES also detects a fault in this tree then the user's confidence in the diagnosis will be heightened. If it does not find a fault then there is a conflict and the operator is informed.

The above cases describe situations in which information supplied by BEDES is used to direct the problem solving of CODES. However there is also a valuable flow of information in the opposite direction. The exchange of hypotheses from CODES to BEDES works differently because BEDES cannot translate nor understand the hypotheses of CODES directly. So knobchain level hypotheses created by CODES (e.g. K5 and K6 in Fig. 2) are of no direct relevance to BEDES. BEDES is only interested in results related to the hypotheses which it has previously sent to CODES (K1-K4 in Fig 2). There are two situations in which CODES should send a result to BEDES:

(a) CODES was able to confirm a hypothesis. Since BEDES cannot understand the hypothesis itself, a back-translation is needed. This involves moving up the tree to find the root node from which all the other hypotheses were derived and then sending this hypothesis back to BEDES. Thus if CODES finds a fault in the detail level (say D1 in Fig. 2) then its root hypothesis (K3) should be translated to (E3) and sent back to BEDES.
(b) CODES could not confirm any of the hypotheses derived from those sent by BEDES. This only occurs if all hypotheses from the tree are evaluated and none of them could be confirmed. The result sent back (after translation) will be the original hypothesis of BEDES with the status NOT.CONFIRMED.

In both of these cases, BEDES tries to integrate the information received into its own problem-solving activity. The situations which might occur now are quite similar to those of Cases 1–3, in that the attention of BEDES can be drawn or dropped to a certain hypothesis depending on the status of the information supplied by CODES. If BEDES was ahead of CODES then the results will be compared; in the case of a conflict it is assumed that the agent which found a fault is more reliable.

The cooperative fault-finding phase comes to an end if all hypotheses are evaluated and none of them could be confirmed (a transient fault or a false alarm) or a hypothesis has been confirmed. In either case the *recovery phase* will begin. This phase is outside the scope of this experiment.

4.2. Integrating pre-existing expert systems

Converting the standalone versions of BEDES and CODES into a community of cooperating agents being controlled by GRATE required three main activities to be carried out. Firstly, some adaptations to the control of BEDES and CODES were required and the domain-level tasks had to be defined. Secondly, the interface between the expert systems and GRATE had to be constructed. Finally, the acquaintance and self models of the agents needed to be populated—this process includes specifying the recipes* which control agent activity, enumerating the tasks which the domain-level system can perform and representing the information which other agents would benefit from receiving.

Expert-system adaptations

As Fig. 2 illustrates, the initial control of both BEDES and CODES was a non-interruptible loop. However for the purpose of controlling local problem solving from an upper layer, such a coarse granularity was inappropriate. To utilise the benefits of cooperation, GRATE has to be capable of influencing the rating of hypotheses and of injecting new items into the agenda. Therefore the control cycle needed to be split into more-manageable components. As the original coding of the control loop was carried out in a modular fashion it was relatively straightforward to decouple the cycling routines from the actual functionality which it drove.

Having identified the control regime, the next step is to determine the domain-level system tasks. When performing this analysis the overriding objective was to minimise the amount of change required to the structure of the pre-existing systems, whilst still permitting the benefits of interaction to take place. Inspection of the existing control loop suggested there should be six tasks—one for each node of the graph. However a deeper examination of the system structure revealed that "retrieve data" and "evaluate hypothesis" are virtually indivisible because the former is deeply embedded within the latter. Also, "create and inject" is intimately related to the evaluation process and also could not easily be separated. Selection of hypothesis from the agenda was regarded as the initialisation phase for evaluation, therefore it was decided to leave it hidden in the domain level system. Thus the control loop was collapsed into two basic tasks – evaluating hypotheses and rearranging the agenda.

- REARRANGE-AGENDA: re-arranges the agenda so that the highest-priority tasks are near the beginning. Also removes any superfluous hypotheses.
- EVALUATE-HYPO: takes the first hypothesis from the agenda and evaluates it.

This level of control was considered appropriate for two reasons. Firstly, because of the way the pre-existing systems were implemented; any other decomposition would have required significant modifications to the existing structure. Secondly, from the perspective of exploiting information gleaned from other agents, the advantages of a finer-level control would have been negligible. As a consequence of this reconceptualisation it is apparent that some of the control which resided originally in the expert system has migrated up into GRATE's cooperation and control layer. However, not all of the control has been moved; a significant amount of lower-level control remains with the domain-level system.

In this application GRATE exerts control over the domain-level system through its agenda. Therefore some of the manipulation functions which existed within the domain-level system needed to be made available at the cooperation and control layer if the benefits of information received from acquaintances is to be exploited. These functions include:

- INJECT-HYPO: inject a hypothesis into the agenda
- DELETE-HYPO: remove a hypothesis from the agenda
- GET-AGENDA: return the current contents of the agenda
- CHANGE -RATING: modify the rating slot of a hypothesis in the agenda.

The responsibility for ensuring that these commands are executed in a coherent manner resides with the control of the domain-level system. Thus if GRATE decides that the rating of a hypothesis should be modified, it issues the "change-rating" command to its domain-level system. Once received this directive will not be acted upon immediately since, for example, the expert system may be in the middle of performing an evaluation. Only when it comes to its rearrange agenda task will the modification actually take place. From a design perspective, it is important that such domain-specific control remains within the expert systems. Exporting it to the upper level would require GRATE's control module to be at least as sophisticated as the control of the domain-level system and would also mean that it was different for each and every application. Maintaining a clean separation of concerns allows GRATE's control module to be simpler and more generic.

Interfacing GRATE and the domain-level systems

BEDES and CODES run on separate workstations and are implemented in KEE making use of SUN Common LISP; GRATE is written in Allegro Common LISP. Because of incompatibilities between the different pieces of software, and also for efficiency reasons, it was decided to run GRATE on a third workstation. Thus all the agents' cooperation and control layers ran on one machine, this machine being different from the ones which were executing the domain-level systems. To allow an agent's control module to interact with its domain-level system, a communication package was utilised. This package established bidirectional communication between SUN Common LISP on one workstation and Allegro Common LISP on another. It was

* Recipes are sequences of actions known by an agent for achieving a particular objective.[33]

```
RECIPE NAME: (VERIFY-CAUSE-OF-FAULT)
TRIGGER: (ENTER-FAULT-FINDING-MODE)
ACTIONS: ( (START (REARRANGE-AGENDA (> FIRST-HYPO))
             (START (EVALUATE-HYPO (<FIRST-HYPO) (>STATUS) (>NEW-HYPOS)))
           (LOOP-UNTIL (FAULT-CONFIRMED (<STATUS))))
RESULTS: (NEW-HYPOS)
```

Fig. 4. Basic control loop for fault-verification phase. ">" means unbound variable and "<" indicates a bound variable. Thus the EVALUATE-HYPO task takes one input (called FIRST-HYPO) and produces two outputs (respectively named STATUS and new-hypos).

based on standard UNIX tools such as sockets and TCP/IP, and had been developed as part of the application-specific cooperation software used for preliminary experimental work. GRATE would issue commands such as: START(EVALUATE-HYPO). This directive would be picked up by the communication package which would send the message onto the workstation running the appropriate domain-level system.

In addition to interacting with the domain-level system, the cooperation and control layer needs to carry out reasoning about received and generated information. To facilitate this, some domain dependent functions needed to be written for inclusion into GRATE's recipes. In this experiment these functions were primarily related to providing an interpretation of the common language (i.e the hypotheses) and of presenting output to the operator. Examples of such functions include:

- (has-slot-value <hypo> <slot> <value>)
  Boolean function which verifies if a specified slot of a hypothesis contains a certain value.
- (has-equal-slots <hypo-1> <hypo-2> <slot>)
  Boolean function which verifies if the slots of the two hypotheses contain the same value.
- (find-related-hypos <hypo-list> <hypo>)
  Returns all members of the hypothesis list which have the same value in the SUSPECTED.ENTITY slot as the specified hypothesis.
- (confirm <hypo>)
  Displays message to the operator that a hypothesis has been confirmed by both agents.

Instantiating the agent models

When building a GRATE application a significant proportion of the knowledge required to control cooperative problem solving is built into the system. For these experiments, no additions were needed to the generic rule set. Thus each agent had exactly the same rules in its cooperation and control layer and the application builder was only concerned with the domain-dependent features of GRATE (i.e. the agent models).

Firstly, the self models need to be instantiated. This involves describing the tasks which the domain-level system is able to perform—including the name, the inputs it must receive in order to execute and the results which are produced. In this experiment the self models contained descriptions of the following tasks: rearrange-agenda, evaluate-hypo, inject-hypo, delete-hypo, get-agenda and change-rating. Two sample descriptions are given below:

```
TASK NAME: EVALUATE-HYPO
MANDATORY INPUTS: (HYPO)
RESULTS PRODUCED: (STATUS NEW-HYPOS)

TASK NAME: CHANGE-RATING
MANDATORY INPUTS: (HYPOS RATING-CHANGE)
RESULTS PRODUCED: NIL.
```

Tasks are grouped together into recipes. Recipes have trigger conditions which indicate when they should be activated, a body which describes the actions to be performed and a description of the results produced. The recipe which encodes the basic control loop for the agent's fault verification phase is shown in Fig. 4. This recipe is triggered when the accelerator monitoring phase detects a problem; it loops continuously until the cause of the fault has been ascertained whereupon a recovery mode recipe is invoked.

Figure 5 illustrates a recipe which describes how CODES utilises information about hypotheses received from BEDES. In particular it highlights the way in which information about the state of hypotheses can be used to draw or deflect CODES's attention from a particular branch of the search space. It encodes cases two and three of the cooperative scenarios highlighted earlier; note that to simplify the example, cases of conflict are not dealt with.

The recipe is triggered when CODES receives a hypothesis which BEDES has evaluated (status CONFIRMED or NOT.CONFIRMED). According to cooperative scenario case 1, CODES will already have received information about the hypothesis when it was first generated (status NOT.EVALUATED). To carry out the necessary reasoning, CODES has to identify those hypotheses in its agenda which are related to the one just received from BEDES. This matching process is carried out by the recipe's first two actions GET-AGENDA and FIND-RELATED-HYPOS.

The remaining recipe actions are conditional upon CODES's current problem-solving context. The first condition tests whether BEDES has also verified a hypothesis which CODES has already confirmed

(CONFIRMED.HYPO). If this is the case, then the level of confidence in the diagnosis is increased and the operator should be informed. The second conditional action draws the attention of CODES to the hypotheses related to the one confirmed by BEDES—this is achieved by increasing the rating of the related hypotheses by a value of 30. The final action deflects CODES's attention away from a hypothesis tree which appears less promising.

Once the self models have been completed the acquaintance models need to be populated. In the example cooperative scenarios the most important feature to model about another agent is the information which it is known to be interested in. For example, BEDES's model of CODES contains the information that CODES would benefit from receiving any newly generated hypotheses which are at the element level (cooperative scenario case 1), any element-level hypotheses that it has been unable to confirm (cooperative scenario case 2), or any element-level hypotheses that it has been able to confirm (cooperative scenario case 3).

```
INTERESTS:
  (..(HYPO(AND (AT-ELEMENT-LEVEL HYPO)
               (HAS-SLOT-VALUE HYPO
               STAT.OF.HYPO
               NOT.EVALUATED)))
    (HYPO (AND (AT-ELEMENT-LEVEL HYPO)
                (HAS-SLOT-VALUE HYPO
                STAT.OF.HYPO CONFIRMED)))
    (HYPO (AND (AT-ELEMENT-LEVEL HYPO)
                (HAS-SLOT-VALUE HYPO
                STAT.OF.HYPO
                NOT.CONFIRMED))) . . .
```

5. RESULTS AND EXPERIENCES

BEDES and CODES were successfully transformed from standalone expert systems to a community of cooperating agents under the control of GRATE. This transformation was achieved with minimal modifications to the pre-existing expert systems and with no augmentation to GRATE's generic knowledge. The cooperating system was tested using a special development mode of the accelerator in real time. As the accelerator operates in a time-sharing fashion it was possible to deliberately introduce faults into the system in the test mode without disturbing the other modes which were serving real physicists' experiments.

The results of this experiment highlighted some shortcomings in the design decision to map the BEDES and CODES expert systems directly into agents. This proved to be a less than optimal choice because of the large amount and diverse range of processing carried out in each system. As both systems were originally conceived as standalone pieces of software they contain a vast array of functionality which does not logically belong together—including monitoring, data acquisition, fault diagnosis and recovery. Also because of their sheer size, the expert systems were becoming unwieldy in their own right. Introducing such systems into a cooperating community merely exacerbated these problems.

Using the cooperation metaphor it is possible to divide the systems into a number of simpler and logically separate agents which could work on dedicated areas of the problem. A new design was proposed in which the functionality contained in BEDES and CODES was split into seven agents.[34] This new approach offers greater system modularity and allows the benefits of parallelism and interaction to be exploited to an even greater degree.

Firstly the data acquisition and treatment functionalities were separated out. It became apparent that the reasoning process in the different agents was heavily reliant on the correctness of the data. For this reason, the treatment of acquisitions is likely to become increasingly sophisticated in the future and so a dedicated agent is warranted. An additional advantage is that it is easier to provide treated data from a single

```
RECIPE NAME: (USE-EVALUATED-HYPO-INFORMATION)
TRIGGER: (AND(info-available HYPO)
              (not (has-slot-value HYPO STATE.OF.HYPO NOT.EVALUATED))
ACTIONS:
( (start(GET-AGENDA (> FULL-AGENDA)))
  (start (FIND-RELATED-HYPOS (< FULL-AGENDA)(< HYPO)(> RELATED-HYPOS)))
  (start-if
      (and (has-slot-value HYPO STATE.OF.HYPO CONFIRMED)
           (has-equal-slot HYPO CONFIRMED.HYPO SUSPECTED.ENTITY))
    (CONFIRM (< HYPO)))
  (start-if (has-slot-value HYPO STAT.OF.HYPO CONFIRMED)
    (CHANGE-RATING (< RELATED-HYPOS) 30))
  (start-if (has-slot-value HYPO STAT.OF.HYPO NOT.CONFIRMED)
    (CHANGE-RATING (< RELATED-HYPOS) -20)))
RESULTS: NIL
```

Fig. 5. CODES recipe for exploiting information received from BEDES.

source rather than having to do separate acquisition and treatment in each and every agent.

The next decision was to remove the user interface functionality from the individual systems and provide homogeneous presentation of data through a specialised agent. This provides the operator with one entry point to the entire agent community. It also allows him to be presented with high-level information about process parameters which can be obtained through interaction with the acquisition agent. This is not possible in the existing control system and so the operator has to rely on raw data from the process.

BEDES and CODES were originally conceived as diagnostic expert systems; it was some time before their recovery facilities were incorporated. Because of their add-on nature, it was decided to separate the recovery actions from the diagnostic part.

Finally, BEDES reasons on two conceptual levels: on the high level of beam parameters (which must be deduced from raw data rather than acquired directly) and on the direct level of the equipment (raw data). In the modified design these two levels are mapped into separate agents. This is beneficial because it frees the high-level reasoning from the shackles imposed by the strict mechanism of the hypothesis verification process.

6. CONCLUSIONS

The stated aim of this work was to take two standalone and pre-existing expert systems and construct from them a community of cooperating agents. This was achieved by using the GRATE system to control the cooperative activity and required only slight modifications to the expert systems. The cooperating community worked together to diagnose faults which occurred in the real particle accelerator process.

Most reported systems work with highly idealised problem solvers and simplified domains. This naturally makes experimentation easier, but is dangerous in that the assumptions which have to be made may hide important issues which need to be addressed if the technology is eventually to be used in realistic environments. This experiment used real expert systems working on a real-world problem. By adopting this approach the experiment provided many useful insights into the fundamental problem of incorporating pre-existing systems in a community of cooperating agents. When undertaking this activity, the structure of the pre-existing system needs to be analysed to ensure it is open enough for the cooperation and control layer to exploit the information gleaned during interactions with fellow community members. Necessary modifications may include defining a finer granularity of control, making previously hidden control functions explicit, or developing completely new functions.

From this experiment it is clear that some of the basic control which previously resided in the domain-level system needs to be moved into the cooperation and control layer. However, lower-level control which is more application-specific is best left at the domain level. When deciding the separation of concerns there is a tradeoff between the amount of restructuring of the domain level system and the desired granularity of control. A detailed analysis of the existing system must be undertaken so that a balance can be struck which avoids a significant reformulation system but which still allows the benefits of interaction to be realised.

A second important consideration which this experiment uncovered is the relationship between pre-existing systems and agents—it is not always best to adopt a simple one-to-one mapping. Often standalone systems contain a number of logically disparate tasks which can be split into separate agents. This decomposition typically allows greater parallelism to be exploited in problem solving and makes better use of the cooperating systems metaphor.

The types of interaction exhibited by BEDES and CODES are typical of a broad class of problem solving called Functionally Accurate, Cooperative (FA/C)[35]. In the FA/C paradigm agents asynchronously exchange partial results about the intermediate state of their processing to ensure that the community arrives at a consistent interpretation of the whole problem. From a traditional computer science perspective, this type of cooperative problem solving can be viewed as a form of distributed search which has multiple loci of control.[36] In the CERN experiment, the partial results are hypotheses and as further evidence emerges (e.g. hypotheses become CONFIRMED or NOT.CONFIRMED) so the problem solving behaviour of the community develops accordingly. The types of cooperation exhibited in this experiment are similar in nature to other industrial applications which have been studied within the context of the ARCHON project (e.g. electricity transport management[37] and management of electricity distribution[38]), which suggests that theoretical research into the FA/C paradigm has a practical use.

This experiment also provides further support for two of Lesser's observations about the FA/C paradigm.[36] Firstly, in comparison to a standalone version, an FA/C agent is more complex. This can be seen by the refashioning of and additions to the expert systems' control regimes. Secondly, it is observed that effective control of cooperative problem solving requires local control decisions to be influenced by the state of problem solving in other agents. In this experiment, the behaviour of other agents is monitored by the cooperation and control layer and influence is exerted through modifications of the agent's agenda.

The implemented cooperation schemes are relatively straightforward, but there is scope for greater sophistication which may enhance performance still further. At present when BEDES sends CODES its initial block of hypotheses, CODES starts processing them in the order in which they arrive. This means both agents start trying to verify the fault from the same position in the search space. As the hypotheses are unrated at this point, this focuses the community's efforts on an unnecessarily small portion of the accelerator. It would be better if the two agents worked on different areas of the search space while the information about the faults is limited, then only when further information becomes available would they focus their joint efforts on promising areas. This could be realised by a relatively straightforward approach in which CODES starts working on hypotheses from the end of the list or in a more elegant manner by a form of negotiation[39, 40] in which agents decide upon the portion of the problem space on which they will concentrate their initial efforts. This division of labour is possible because both systems are usually capable of detecting the same fault. If, however, neither of them is able to find the fault in their part of the search space then only at this stage should they start trying to work on areas which the other agent has already processed.

A final enhancement to the application would be to incorporate a more sophisticated mechanism for resolving conflicting opinions between the agents. For example, rather than simply assuming the result produced by CODES is correct, it would be better if some conflict-resolution expertise could be used to determine the source of the conflict and provide a means of resolving it.[41] Such a mechanism would enable better-quality information to be presented to the operator.

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

REFERENCES

1. Bond A. H. and Gasser L. (Eds) *Readings in Distributed Artificial Intelligence.* Morgan Kaufmann, San Mateo, CA (1988).
2. Gasser L. and Huhns M. N. (Eds) *Distributed Artificial Intelligence,* Vol. II. Pitman Publishing, London (1989).
3. Huhns M. N. (Ed.) *Distributed Artificial Intelligence.* Pitman Publishing, London (1988).
4. Sridharan N. S. 1986 Workshop on distributed AI. *AI Mag.* Fall, 75–85 (1987).
5. Davis R. and Smith R. G. Negotiation as a metaphor for distributed problem solving. *Artif. Intell.* **20**, 63–109 (1983).
6. Lesser V. R. and Erman L. D. An experiment in distributed interpretation. *IEEE Trans. Comput.* **29**, 1144–1163 (1980).
7. Jennings N. R. and Wittig T. ARCHON: theory and practice. In *Distributed Artificial Intelligence: Theory and Praxis* (Gasser L. and Avouris N. M., Eds), pp. 179–195. Kluwer Academic, London (1992).
8. Wittig T. *ARCHON: An Architecture for Multi-Agent Systems.* Ellis Horwood, Chichester (1992).
9. Jennings N. R. Cooperation in industrial systems. *Proc. ESPIRIT Conference,* Brussels, pp. 253–263 (1991).
10. Jennings N. R., Mamdani E. H., Laresgoiti I., Perez J. and Corera J. GRATE: a general framework for cooperative problem solving. *J. Intell. Syst. Engng* **1**, 102–114 (1992).
11. 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,* pp. 691–696, Delft, The Netherlands (1992).
12. Cohen P. R. A survey of the Eighth National Conference on Artificial Intelligence: pulling together or pulling apart. *AI Mag.* **12**, 16–41 (1991).
13. Roda C. and Jennings N. R. The impact of heterogeneity on cooperating agents. *Proc. AAAI Workshop on Cooperation among Heterogeneous Intelligent Systems*, Anaheim, Los Angeles (1991).
14. Simon H. A. *Models of Man.* Wiley, New York (1957).
15. Lesser V. R. and Corkill D. D. Distributed problem solving. *Encyclopedia of Artificial Intelligence* (Shapiro S. C., Ed.), pp. 245–251. Wiley, New York (1987).
16. Whitney C. Cooperating intelligent agents: a study of GRATE. BT Report MAIN-WP1008, BTRL Martlesham Heath, Ipswich (1992).
17. Jennings N. R. Joint intentions as a model of multi-agent cooperation in complex dynamic environments. Ph.D. Thesis, Dept Electronic Engineering, Queen Mary and Westfield College (1992).
18. Engelmore R. and Morgan T. (Eds) *Blackboard Systems.* Addison Wesley, Reading, MA (1988).
19. Hayes-Roth B. The blackboard architecture: a general framework for problem solving?, Stanford Heuristic Programming Project, HPP-83-30, Stanford University (1983).
20. Hewitt C. E. and Kornfield W. A. Message passing semantics. *SIGART Newslett.* p. 48 (1980).
21. Hewitt C. E. and Liberman H. Design issues in parallel architectures for artificial intelligence. *Proc. IEEE Computer Society International Conference*, pp. 418–423 (1984).
22. Malandain E., Pasinelli S. and Skarek P. A fault diagnostic expert system prototype for the CERN PS. *Europhysics Conference on Control Systems for Experimental Physics,* Villars-sur-Ollon (1987).
23. Skarek P., Malandain E., Pasinelli S. and Alarcon I. A fault diagnosis expert system for CERN using KEE. SEAS (SHARE European Association) Spring Meeting, Davos (1988).
24. Malandain E., Pasinelli S. and Skarek P. Knowledge engineering methods for accelerator operation. *European Particle Accelerator Conference*, Rome, Italy (1988).
25. Malandian E. and Skarek P. Linking a prototype expert system to an oracle database, IASTED. *International Conference on Expert Systems, Theory and Applications,* Zurich (1989).
26. Malandain E. An expert system in the accelerator domain. *International Workshop on Software Engineering, Artificial Intelligence and Expert Systems for High Energy Nuclear Physics*, Lyon Villeurbanne (1990).
27. Fuchs J., Skarek P., Varga L. and Wildner-Malandain E. Integration of generalized KB-systems in process control and diagnosis. *SEAS Conference*, Lausanne (1991).
28. Fuchs J., Skarek P., Varga L. and Wildner-Malandain E. Distributed cooperative architecture for accelerator operation. *2nd International Workshop on Software Engineering, Artificial Intelligence and Expert Systems for High Energy and Nuclear Physics*, L'Agelonde, La-Londe-les-Maures (1992).
29. Varga L. Cooperation between the two diagnostic expert systems BEDES and CODES. CERN Technical Report, PS/CO/WP 91–02 (1991).
30. Neches R., Fikes R., Finin T., Gruber T., Patil R., Senator T. and Swartout W. R. Enabling technology for knowledge sharing. *AI Mag.* Fall, pp. 36–56 (1991).
31. Ginsberg M. L. Knowledge interchange format: the KIF of death. *AI Mag.* Fall, pp. 57–63 (1991).
32. Hewitt C. E. The challenge of open systems. *BYTE* **10** 223–244 (1985).
33. Pollack M. E. Plans as complex mental attitudes. In *Intentions in Communication* (Cohen P. R., Morgan J. and Pollack M. E., Eds), pp. 77–105. MIT Press, Cambridge, MA (1990).
34. Fuchs J., Skarek P., Varga L. and Wildner-Malandain E. Distributed cooperative architecture for accelerator operation. ARCHON Technical Report 26, CERN, Geneva (1992).
35. Lesser V. R. and Corkill D. D. Functionally accurate, cooperation distributed systems. *IEEE Trans. Syst. Man Cybernet.* **11** 81–96 (1981).
36. Lesser V. R. A retrospective view of FA/C distributed problem solving. *IEEE Trans. Syst. Man Cybernet.* **21**, 1347–1362 (1991).
37. Aarnts R. P., Corera J., Perez J., Gureghian D. and Jennings N.

R. Examples of cooperative situations and their implementation. *Vleermuis J. Software Res.* **3**, 74–81 (1991).

38. Cockburn D., Varga L. Z. and Jennings N. R. Cooperating intelligent systems for electricity distribution. *Proc. Expert Systems 1992 (Applications Track)*, Cambridge (1992).
39. Laasri B., Laasri H. and Lesser V. R. An analysis of negotiation and its role in cooperative distribution problem solving. *Proc. Second Generation Expert Systems Conference*, Avignon (1991).
40. Conry S. E., Kuwabara K., Lesser V. R. and Meyer R. A. Multi-stage negotiation for distributed constraint satisfaction. *IEEE Trans. Syst. Man Cybernet.* **21**, 1462–1477 (1991).
41. Klein M. Supporting conflict resolution in cooperative design systems. *IEEE Trans. Syst. Man Cybernet.* **21**, 1379–1390 (1991).

AUTHORS' BIOGRAPHIES

Nick Jennings is a lecturer in the Department of Electronic Engineering at Queen Mary Westfield College (University of London). His main research interests are the theory and application of multi-agent systems to real-world problems. He is an active member of the ARCHON consortium, Europe's largest Distributed AI project, leading several workpackages concerned with cooperation techniques and architecture design. He has been involved with designing and developing applications in the fields of electricity transportation management, electricity distribution management, telecommunication networks and particle accelerator control. In his Ph.D. research he developed and implemented a novel model of joint intentions as a means of controlling cooperative problem solvers in complex and dynamic domains.

Laszlo Zsolt Varga received his M.Sc. and Doctoral Degrees from the Technical University of Budapest in 1984 and 1988 respectively. Since 1984 he has been with the KFKI Research Institute for Measurement and Computing Techniques, Budapest, Hungary, where he is now a Scientific Associate. He participated in the ARCHON p2256 ESPRIT project as a visiting Scientific Associate at CERN in Geneva from 1990 to 1991 and at the Queen Mary and Westfield College, University of London from 1992 to 1993. He worked on the problem of integrating cooperating intelligent systems within the framework of ARCHON. His research interests include computer communication protocols, distributed artificial intelligence, multi-agent systems.

Rob R. Aarnts received a MS in Mechanical Engineering from the Technical University of Twente, The Netherlands. In 1981 he joined Stork Boilers where his main tasks were the analysis of boiler performance and the implementation of several calculation programs. In 1983 he joined the Government at the Bridge Design department. As a project leader he was in charge of the design of bridges and sluices. Since 1986 he has been employed by Volmac where he has worked on several projects. His latest affiliation is the ARCHON project which started in 1989. He is responsible for the development of the C++ version of the ARCHON software.

Joachim Fuchs studied Electrical Engineering (Telecommunication and Information Systems) at the University of Karlsruhe (83–87), at the University of Essex (87–88) and ESIEE, Paris (88–89). After this he was employed at CERN as a technical fellow in the field of Expert Systems for Diagnosis of an accelerator, control system development, integation of AI techniques in "classic" control architecture. During this time he was involved with the ARCHON project. He is currently working for ESA/ESTEC (European Space Agency) where he is responsible for an AI Demonstration progamme.

Paul Skarek got his Ph.D. (Physics, Mathematics) in 1960 from the University of Vienna and has worked since 1962 at CERN. There he has participated in several nuclear physics experiments and in designing and implementing large-scale software (accelerator physics, beam optics). Between 1970–1976 he took part in the improvement program for the Synchro-cyclotron accelerator and the automation of magnetic field measurements by process-control computers. Since 1976 he has been in the Controls Group of the Proton-Synchrotron accelerator, where he has participated in the redesign, implementation and maintenance of the distributed computer control system. He is currently working on introducing and applying AI methods in the accelerator maintenance of the distributed computer control system.