scope: *architecture(p), diagnosis(s) & planning(s)*
contribution: *tools & techniques*

GRATE: a general framework for co-operative problem solving

by N. R. Jennings, E. H. Mamdani, I. Laresgoiti, J. Perez and J. Corera

As the deployment of expert systems has spread into more complex and sophisticated environments, so inherent technological limitations have been observed. As a technique for overcoming this complexity barrier, researchers have started to build systems composed of multiple co-operating components. These systems fall into two distinct categories; those that solve a particular problem, such as speech recognition or vehicle monitoring, and those that are general to some extent. GRATE is a general framework, which enables an application builder to construct multi-agent systems for the domain of industrial process control. Unlike other frameworks, GRATE embodies a significant amount of inbuilt knowledge related to co-operation and control which can be utilised during system building. This approach offers a paradigm shift for the construction of multi-agent systems, in which the role of configuring pre-existing knowledge becomes an integral component. Instead of starting afresh, the designer can utilise the in-built knowledge and augment it, if necessary, with domain-specific information. The GRATE architecture has a clear separation of concerns, and has been applied to real-world problems in the domains of electricity transportation management and diagnosis of a particle accelerator beam controller.

1 Introduction

After more than a decade of successful exploitation, there are now thousands of expert systems being used in hundreds of companies all over the world to solve complex problems in numerous domains [1]. Such systems have been particularly important and successful in the domain of industrial process control, where conventional software and teams of operators were unable to cope with the demands presented by rapidly changing, complex environments [2]. However, as expert systems technology has proliferated and individual systems have increased in size and complexity, new problems and limitations have been noted [3, 4]:

- **scaleability:** the complexity of an expert system may rise faster than the complexity of the domain.
- **versatility:** a complex application may require the combination of multiple problem-solving paradigms.
- **reusability:** several applications may have requirements for similar expertise, which has to be coded afresh in each new situation.
- **brittleness:** expert systems typically have a narrow range of expertise and are generally very poor at identifying problems that fall outside their scope.
- **inconsistency:** as knowledge bases increase in size, it becomes correspondingly more difficult to ensure that the knowledge they embody remains consistent and valid.

Several approaches have been proposed to circumvent these problems. Lenat *et al.* [5] have built CYC, a large expert system containing a substantial amount of 'common sense' knowledge. This work is based on the supposition that such a vast body of knowledge is needed to solve any non-trivial problem. Other researchers have followed a more traditional software engineering approach and have compartmentalised the problem solv-

ing into smaller, more manageable components that can communicate. Work adopting the latter philosophy is grouped under the collective term *distributed artificial intelligence* (DAI) [6–8] and is the main focus of this paper. From a system engineering perspective, multiple communicating problem solvers have several potential advantages, including 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 increased system modularity [6, 8].

In DAI, the individual problem-solving components are called *agents*; these are grouped together to form *communities*. Each agent is capable of a degree of problem solving in its own right,* has its own goals and objectives, and can communicate with others by passing messages. Agents within a community typically have problem-solving expertise that is related, but distinct, and that frequently has to be combined to solve problems. There are two main cases in which multiple agents need to work together in a co-ordinated and coherent manner; first, when individual subproblems cannot be solved in isolation. For example, in work on speech recognition, it is possible to segment an utterance and work on each individually. However, the progress which can be made in such circumstances is very limited [10]. Secondly, even if individual subproblems are soluble in isolation, it may be impossible to synthesise their results; for example, when building a house, subproblems such as determining the size and location of rooms, wiring, plumbing, and so on interact strongly and cannot be considered in isolation [11].

Two main types of software tool have been developed; *integrative systems* and *experimental testbeds* [6]. Integrative systems, such as GRATE, provide a framework to combine a variety of problem-specific tools and methods into a useful whole, whereas experimental testbeds have their main emphasis on controlled experiments in which both measurement and monitoring of problem solving activity are the prime consideration. Within the former category, two broad classes of system can be identified:

- systems that test a specific type of problem solver, a specific co-ordination technique or a particular domain; e.g. DVMT [12], ATC [13], HEARSAY II [14], YAMS [15].
- systems that are *general* to some extent; e.g. MACE [16, 17], ABE [18], AGORA [19].

The systems listed under the general category are considered general because the architecture or language they espouse is not targeted towards any particular application domain. They either provide a language with which a system can be constructed (e.g. AGORA or the family of ACTOR languages [20, 21]) or a 'shell' which the application developer is able to instantiate with the appropriate co-operation and control knowledge (e.g. MACE and ABE). In the former case, the application designer has complete flexibility over the system to be built, but expends a substantial amount of effort imposing the desired structure, because each application must be constructed afresh. With the latter approach, the structure and the mechanisms available are determined by the shell, and the designer has to use the languages and tools provided to build the working system.

If DAI is to progress, more powerful development environments are required to cope with the complexities of real-size problems [6]. The research described here sought to address this fundamental issue, by investigating the feasibility of constructing an environment in which some of the knowledge required to build a working system is already embedded. This novel approach means developers can use the *generic knowledge* that is embodied (often implicitly) in all multi-agent systems and is built into the shell, rather than constructing the system afresh and coding this knowledge themselves. In this paper, we discuss how such a system was built and reflect on our experiences of developing the **GRATE** (Generic Rules and Agent model Testbed Environment) system.

The general description of co-operative agent behaviour represented by the built-in knowledge is possible because all the domain-dependent information, which is obviously necessary to define individual behaviour, is stored in specific data structures (called *agent models*). This approach is an adaptation of the conventional AI notion that generic structures can be used in building specialised systems [22, 23]. Agent models provide an explicit representation of other agents in the world [16] and the information that may be maintained in them (i.e. their structure) is consistent across all applications. However, some parts may be left unfilled in particular cases (e.g. the goals of a database system may not be represented, whereas for an expert system they may be an integral component). Obviously, the particular instantiation of an agent model is highly domain-dependent and must be carried out by the application builder. The generic knowledge built into the system then operates on the *structure* of the models, rather than their specific *contents*.

A further innovation of GRATE is in the type of problem being tackled. Early DAI systems concentrated on communities purpose-built for co-operation and with typically one overall problem to solve. In such systems (often called distributed problem-solving systems [6]), the main emphasis was on techniques for problem decomposition and assigning agents to tasks [11]. Within the domain of industrial process control, such an approach is infeasible because of the large number of systems already in existence and the complexity of the problem being tackled [24]. To address this problem, the ARCHON project [25] (in which some of the work described here took place) focused on making possibly pre-existing and independent intelligent systems (e.g. knowledge/databases, numerical systems etc.) co-operate on a variety of goals. The fact that there is no longer just

* Each agent has an identifiable expertise (e.g. it may control a piece of machinery, diagnose faults, perform scheduling activities, and so on). This contrasts this with other forms of distributed problem solving, such as neural networks [9], in which individual nodes have very simple states (either on or off), and only by combining many thousands of these simple entities can problem-solving expertise be recognised.

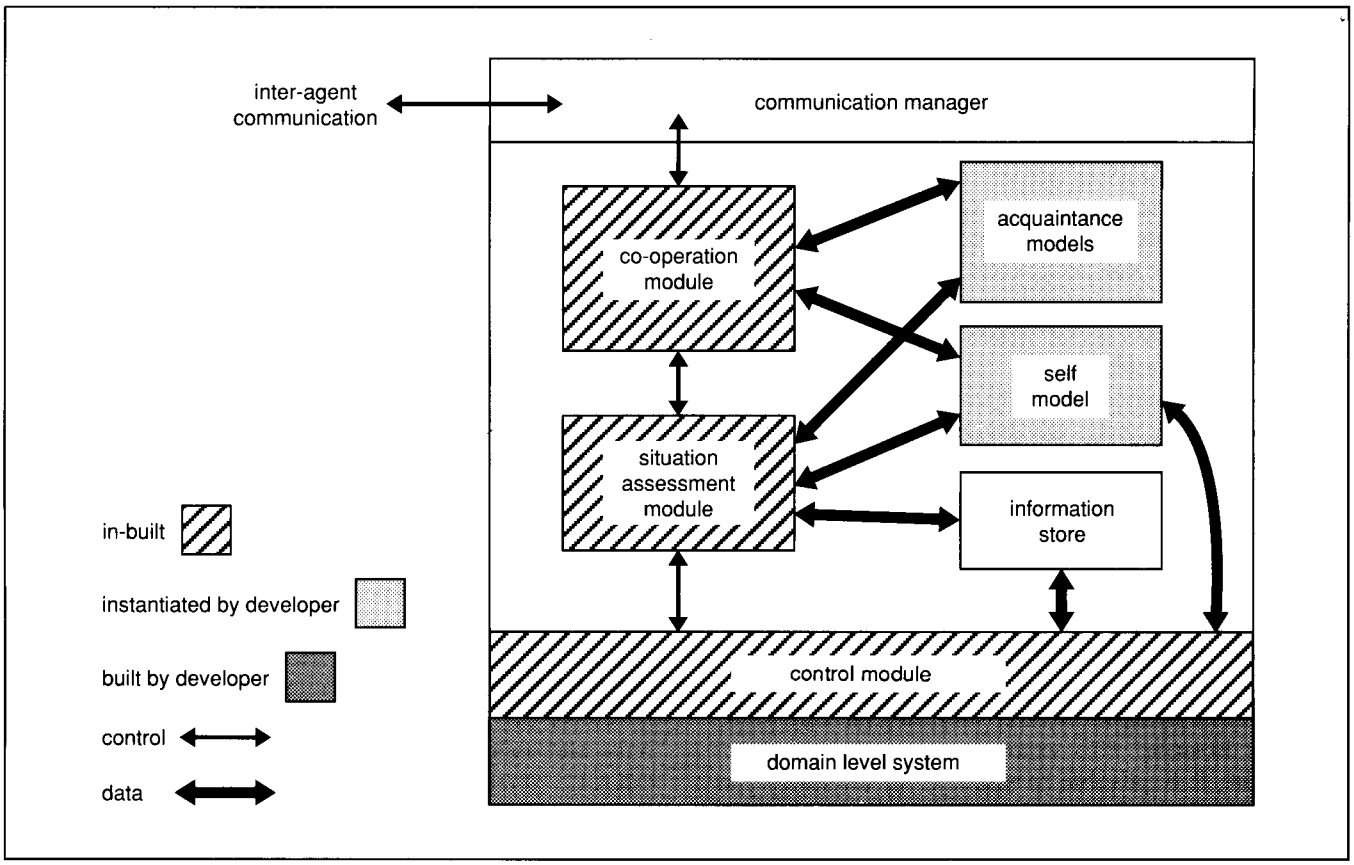

Fig. 1 GRATE agent architecture

one aim for the whole system requires the agent architecture to reflect the dual role of being an individual and a community member, requires explicit reasoning about the process of co-ordination, and means that multiple, unrelated social activities may be taking place concurrently.

2 The GRATE architecture

2.1 GRATE agents†

GRATE agents have two clearly identifiable components; a *co-operation and control layer* and a *domain level* system. The domain level system may be pre-existing or may be purpose-built, and solves problems such as detecting disturbances in electricity networks, locating faults and proposing remedial actions. The co-operation and control layer is a meta-controller, which operates on the domain level system in order to ensure that its activities are co-ordinated with those of others within the community.

GRATE communities have a 'flat' organisation; there is no centralised or hierarchical structure within the community and also that there are no predefined authority relationships. A global controller was not considered because interagent communication has a limited bandwidth, meaning that each agent could only maintain a restricted view of the overall problem solving process (i.e. each agent has *bounded rationality* [26]). Secondly, a global controller may be a severe communication and computational bottle-neck. Finally, reliability criteria require that community performance is degraded gracefully if one or more agents fail, which would certainly not be the case if the controller failed.

By having control distributed within the community, an individual agent plays two distinct roles. First, it has to play the role of a *team member* acting in a community of co-operating agents, and secondly the role of an *individual*. It is also means that there may be more than one goal being pursued by the community, for example, there may be agents trying to detect faults, agents trying to locate faults and agents proposing remedial actions. Much of the early work on DAI concentrated almost exclusively on the former and paid scant regard to the latter. However, contemporary DAI [27, 28], with its greater emphasis on autonomous agents, also highlights the role of the individual. Therefore, when designing a co-operation framework, both aspects should be accounted for. Such a system must

- direct local problem solving; decide which tasks to launch, when they should be launched, their relative priorities and how best to interleave their execution
- co-ordinate local activity with that of others within the community; when and how to initiate co-operative activity, how to respond to co-operative initiations and which activities require synchronisation.

When defining a modular architecture, it is intuitively appealing to reflect this binary distinction directly. However, because of the multiple co-operation contexts within the community, there is a significant class of activities that falls into a grey area between the two. These activities are concerned with *situation assessment*; for example, which tasks should be carried out locally

† GRATE is implemented in Allegro Common LISP and CLOS on a SUN SPARC I workstation.

and which should be solved with the aid of others, what co-operation requests should be honoured and which should not, the relative priority of activities to be performed, and so on. Therefore, to promote a cleaner separation of concerns, GRATE has three main modules in which the situation assessment module acts as an interface between the local and co-operative control mechanisms (Fig. 1). The control module is informed by the situation assessment module of the tasks to be performed and their relative priorities; it is then the control module's responsibility to ensure that this is carried out. Similarly, the need to initiate social activity is detected by the situation assessment module and then the responsibility for realising this activity is left to the co-operation module.

The information store provides a repository for all domain information which the underlying system has generated or which has been received as a result of interaction with other agents in the community. The acquaintance and self models are representations of other agents in the community and of the local domain level system, respectively; they are described in greater detail in Section 2.2.

Each of the three main modules is implemented as a separate forward-chaining production system, with its own inference engine and local working memory. The rules are written in a standard if-then format, and examples are given in Section 3. Communication between the knowledge bases is by message passing; there is no shared memory. At present, all three inference engines are identical, consisting of a continuous loop. However, to meet the requirements of future applications, one or maybe all of the inference engines might need to be customised. For example, the control module may need to respond rapidly to certain key events, and hence need to be more sophisticated than that of the co-operation module in which events can be handled on a first-come-first-served basis in most circumstances.

2.1.1 Control module: this is the sole interface to the underlying domain level system and is responsible for managing all interactions with it. The domain level systems activities are implemented as independent LISP processes, which means the control module has to deal with concurrent tasks. The general rules in this module can be divided into the following main categories:

□ controlling the execution of tasks in the underlying system based on directives from the situation assessment module (e.g. stop tasks, suspend tasks and reactivate tasks).
□ ensuring the essential information required by a task is available. There are three ways in which this information can be obtained:
- the situation assessment module provides it.
- the information may be available locally; in which case, it must be retrieved from the information store.
- the information may not be available locally; in which case, inform the situation assessment module that additional information is needed if the task is to proceed.

□ if a task is waiting for a particular piece of information and that information becomes available to the control module, update the information available for that task.
□ determining whether task execution can begin (e.g. checking that all the essential information is available).
□ detecting when the task has finished; at which point, gather any results produced and pass them to the situation assessment module for evaluation.
□ if optional information becomes available (i.e. information that could be used in task, but which is not mandatory) determine what to do with it:
- if the task has not yet started, add it to the list of information to be passesd.
- if the task has started, incorporate it into the process.

2.1.2 Situation assessment module: serves as a link and balancer between the agent's two primary roles; that of being an individual and that of being part of a community. It decides which activities should be performed locally and which should be delegated, which requests for co-operation should be honoured, how they should be honoured, which plans should be activated, what action should be taken as a result of freshly arriving information, and so on. The module issues instructions to both the control and the co-operation modules. Typical requests to the co-operation module include 'get information X' and 'get task Y executed by an acquaintance'. Requests to the control module are of the form 'STOP task T1' and 'execute task T2 with data D1'. These two perspectives are reflected in the types of rules this module maintains.

Controlling local activity

□ If the trigger conditions for a plan are satisfied and it is appropriate for that plan to be started, adopt the intention of completing that plan.
□ If the agent has a task to perform and it can do so, instruct the control module to execute that task.
□ If a task has finished and that task was part of a plan, perform the plan's next action.
□ If an information request has been made and it can be generated by executing a local task, determine which task to execute:
- if there is already an active task which will produce it, use it to produce the required information.
- if no task is currently active, select the most appropriate one for generating the required information.

□ If a goal request is made and that task is already active, use the existing task rather than duplicate activity.
□ If the control module indicates that information is needed before a task can be executed, and it is worth expending effort on generating the information and the information can be generated locally, then adopt the intention of producing that information locally.
□ If information becomes available which is useful to a task being executed, pass the information onto the control module.

□ If unrequested information arrives (information which has not been explicitly asked for), assess what it can be used for:

- if it is mandatory for a task which is to become active, pass it onto the control module so that it can be used by the appropriate task.
- if it is optional for a task which is (or is about to become) active, pass it onto the control module to determined whether it is worth making use of.
- see whether it is a trigger for a local plan.
- if it may be of use to the agent at a later stage, store it; otherwise discard it.

□ If it is necessary to alter the execution of a task (e.g. suspend, abort etc.), inform the control module of this point.

Controlling social activity

□ If the agent has a task to perform and it cannot do so, instruct the co-operation module to get help.
□ If a plan has finished, inform the co-operation module of this fact. Depending on the motivation for executing it, the co-operation module takes various actions.
□ If the control module indicates that information is needed before a task can be executed, and it is worth expending effort on generating the information and the information cannot be generated locally, then ask the co-operation module to find an agent capable of supplying it.

2.1.3 Co-operation module: has three primary objectives related to the agent's social role. It has to

- establish co-operation: once the need for social interaction has been established by the situation assessment module, the co-operation module must reason about how the request can be best satisfied. Two forms of co-operation are presently supported; an agent may ask for a task to be executed or a piece of information to be produced by another agent (*task sharing* [11]); or an agent may decide spontaneously to share information it has generated with other community members (*result sharing* [11]). In both cases, the agent originating the activity must decide with which other agents the interaction should take place, and in the task-sharing mode, a *co-operation protocol* must also be selected. Two such protocols could be used; *client-server* and *contract net* [11]. This means that when an agent decides it requires a task-sharing form of interaction to achieve a social objective, it may do so either by making a request directly to one agent (client-server) or by issuing the request to all community members, and waiting for and evaluating their bids before selecting the agent with which to establish the interaction (contract net). In order to distinguish between these two protocols, rules have been devised to select the protocol most appropriate for the particular case. Simple selection of the protocol could be based on priority if more than one agent is capable of completing a particular task. High-priority tasks make use of a contract net protocol to ensure they will be serviced in good time, and low-priority tasks use client-server. However, in the future, such a selection could be more sophisticated and make use of other properties of the two protocols, as shown in Table 1. Therefore, if low communication costs are the primary consideration, the client-server protocol is selected, whereas if the task requires that the best available agent is selected, the contract net is initiated. The client-server's fit of task to problem solver is put as 'average'. In some cases, it may make as good a decision as the contract net, but in others, it may make poor decisions. Its behaviour is heavily dependent on the number of agents capable of performing tasks (the greater the number, the less the chance of making the best decision) and on the system load distribution (heavier load means wrong choices are more expensive in terms of time before task completion).

Table 1

	communication overhead	time to establish	fit of task to problem solver	chance of finding agent to perform action
Contract net	high	high	good	high
Client server	low	low	average	low

- maintain co-operative activity once it has been established: once started, the agent must track its progress until completion. Therefore, for example, if an agent has agreed to do a task because another agent asked it to, this interaction needs to be monitored. Tracking in this case involves sending any relevant intermediate results to the requesting agent and also ensuring that, once the task has finished, a final report describing the status and results of the requested activity is sent to the originator. Tracking would also ensure that if the activity fails, for whatever reason, the originator is informed at the earliest opportunity so that replanning can commence.
- respond initially to co-operation requests from other agents: typically, the co-operation module is unable to determine if a co-operative initiation request should be honoured without referring to the situation assessment module. However, if requests are made for services that cannot be performed locally, they can be rejected at this level without reference to the situation assessment.

2.2 *Agent models*

In order to participate in co-operative activity, agents need to be able to reflect about their role and also that of others within the community. This leads to two distinct types of knowledge being maintained; knowledge about local capabilities (*self model*) and knowledge about other community members with which the agent may have to interact (*acquaintance models*).

Although the actual information to be maintained in both cases is similar, there is a clear distinction in the way in which these two types of knowledge are obtained. An agent's self knowledge is assumed to be specified by the designer of the domain level system; therefore, it can be regarded as an abstraction of this system. The reason

why the designer needs to specify this knowledge is that it would require substantial learning and reasoning power to infer this from the underlying system. It would also require the underlying system to be able to reflect about itself (a capability which would not be present in pre-existing systems). However, it is more reasonable to expect agents to build up models of acquaintances in a dynamic and flexible manner. There are several approaches to this latter problem, which are arranged in order of increasing system complexity (or decreasing application builder effort):

□ 1 application builder completely specifies static acquaintance models.
□ 2 each agent transfers its entire self description to all interested parties before any co-operative activity is started.
□ 3 agents go through an initiation phase, in which each agent asks others questions to obtain the information it needs (e.g. which agents can complete task *t*?, which agents can supply information *i*?, and so on)
□ 4 agents assume others have the same structure and capabilities as themselves§.
□ 5 models of other agents are built up dynamically during the course of problem solving, and no explicit learning phase is entered into*.

In any domain which exhibits complex behaviour, the first and last options in their pure form are impractical because of the excessive effort required by the designer and the necessity of sophisticated non-monotonic reasoning capabilities, respectively. The assumption of similarity (point 4) is also not applicable because agents are typically heterogeneous in nature [33]. The merits of options 2 and 3 depend on the relative costs of communication and computation, and also the percentage of overlap between the agents. In GRATE, a compromise solution was adopted; in the main, the application designer specified much of the acquaintance models, and then some general rules for dynamically updating the agent models were also included, to ease this task and provide some degree of flexibility.

2.2.1 Classification of the agent model knowledge: the exact nature of the information to be maintained about agents (and the self) is one of the major open issues in contemporary DAI [6], and at this stage, there has been little or no attempt to define it in a rigorous manner. Therefore, like others before us [16, 34–36], we have defined this knowledge through experimentation. In GRATE, the type of information maintained about the self and acquaintances are the same (although they represent different levels of abstraction), and so separate classifications are not given. Examples are provided from the self model of the AAA agent in the electricity transport management scenario. The exact meaning of the domain names is not relevant here but is expanded in Section 4.

• State knowledge (e.g. solution progress): indicates the activities which are currently being carried out, how far they have progressed and when they are likely to be completed.

SOLUTION PROGRESS:
Task Name: (HYPOTHESIS-GENERATION)
Information Passed:
((BLOCK-ALARM-MESSAGES (BR HERNANI 3 (CO) LOCAL BR HERNANI 4 (CO) LOCAL BR HERNANI 6 (CO) LOCAL))
(DISTURBANCE-DETECTION-MESSAGE (DISTURBANCE)))
Status: EXECUTING Priority: 10
Rationale: ((PLAN-ACTION (DIAGNOSE-FAULT) G1919 G1907))

• Capability knowledge (e.g. task descriptions, recipes): knowledge about actions which can be performed, how they are combined to achieve particular results, the information they require and the results which can be expected.

Task Description:
Identifier: (HYPOTHESIS-GENERATION)
Necessary Requirements: (DISTURBANCE-DETECTION-MESSAGEBLOCK-ALARM-MESSAGES)
Results Produced: (FAULT-HYPOTHESES)
Time to execute: 15
Recipe Name: (DIAGNOSE-FAULT)
Trigger: (INFO/AVAILABLE DISTURBANCE-DETECTION-MESSAGE) (FAULT-DETECTED))
Actions: (((START (HYPOTHESIS-GENERATION ?FAULT-HYP))
(GET-INFO INITIAL-ELTS-OUT-OF-SERVICE))
((WHILE-WHEN† (BLACK-OUT-AREA-NOT-AVAILABLE) DO
(((START (HYPOTHESIS-VALIDATION ?FAULT-HYP ?VALIDATED/HYP))))
(WHEN ((SUSPEND (HYPOTHESIS-VALIDATION)))
((START (HYPOTHESIS-REFINEMENT ?FAULT-HYP ?REFINED-HYP)))
((REACTIVATE (HYPOTHESIS-VALIDATION ?REFINED-HYP)))))))
Time to Execute: 64 Priority: 10
Outcome: (FAULT-HYPOTHESES)
Instantiations: Identifier: G1907
Component of:
(SATISFY-LOCAL-GOAL (DIAGNOSE-FAULT))

§ 'Stereotyping' is often used as a way of installing user models in interface design [29–31].

* Some dynamism is achieved in GRATE by using relatively simple inference mechanisms (e.g. if agent *a* asks for information *i*, assume *a* is interested in *i*). The type of adaption from which agents infer the models of their acquaintances based on their behaviour [32] would not be suitable for any real-size problem because of the complex nature of the agents involved and the sophistication of their inter-relationships.

† WHILE-WHEN p DO q WHEN r means while p is true do q; when (if) p becomes false do r.

Current Action:
(((GA1919 ((START (HYPOTHESIS-GENERATION ?DISTURBANCE-DETECTION-MESSAGE ?BLOCK-ALARM-MESSAGES ?FAULT-HYPOTHESES)))) (TOP-LEVEL (G1919 ((START (HYPOTHESIS-GENERATION ?DISTURBANCE-DETECTION-MESSAGE ?BLOCK-ALARM-MESSAGES ?FAULT-HYPOTHESES)))) NIL)))
Local Bindings: ((FAULT-HYPOTHESES ?))

- Intentional knowledge (e.g. intentions): provides a high-level description of the targets an agent is working towards. Intentions are fairly stable in nature as they represent long-term objectives§. Complex agents are likely to have several intentions active at any one time, and the situation assessment module must ensure they are honoured in the most efficient manner possible.

Intention
Name: (ACHIEVE (DIAGNOSE-FAULT))
Motivation: ((SATISFY-LOCAL-GOAL (DIAGNOSE-FAULT)))
Chosen Recipe: (DIAGNOSE-FAULT)
Start Time: 8 Maximum End Time: 72
Duration: 64 Priority: 10
Status: ACTIVATED
Outcome: (FAULT-HYPOTHESES)

- Evaluative knowledge (e.g. time to complete task, intention end time): when faced with several alternatives for achieving the same objective, evaluative knowledge provides a means of distinguishing between them.
- Domain knowledge (e.g. recipe priority, recipe triggers): facts and relationships that hold true of the environment in which the agent is operating and that are relevant for defining an agent's behaviour.

3 Achieving generality

Having specified the types of knowledge to reside in the agent models and that this knowledge is manipulated by generic control mechanisms, it is important to see how these two factors can be combined to produce useful behaviour both within and between agents. The underlying assumption of this approach is that meaningful behaviour can be described in terms of the *type* of information maintained in the agent models (i.e. the agent model structure), rather than the information itself*. To illustrate this, several examples are given; the module from which the rule is taken is indicated by its name.

3.1 Social activity

Within the acquaintance model definition, there is a slot to express an agent's interest in particular information (capability knowledge):

INTERESTS: $\{\langle I_1, Cond_1\rangle, \langle I_2, Cond_2\rangle \ldots\}$

Intuitively, this can be interpreted as meaning that, if the modelling agent has information Ω and within one of its acquaintance models it has a tuple involving Ω (e.g. $\langle\Omega, \omega\rangle$), then if ω is true, Ω may be of use to that acquaintance. Expressing this interpretation in a declarative form leads to the following production rule:

```
(rule cooperation-module-5
  (IF (INFO-GENERATED ?AGENT ?INFO ?GOAL))
  (THEN (FIND-INTERESTED-ACQUAINTANCES
              ?AGENT ?INFO ?GOAL ?ACQS)
        (SEND-TO-INTERESTED-
              ACQUAINTANCES ?AGENT ?INFO
              ?GOAL ?ACQS)))
```

This rule is independent of any application and can be considered relevant for all co-operative situations. It illustrates the fact that the rule interprets the slot structure and can be defined without actually knowing what information Ω is or the function ω.

When tasks are activated by an agent, this fact is recorded in a solution progress description, as illustrated in the previous Section. Associated with each task is the information which has been passed to it, its execution status, priority and the reason for executing it. The rule shown below makes use of this descriptor to obtain the reason why a task and its associated plan was activated. If it was to satisfy an information request made by another agent, a generic rule is needed to ensure that the request is honoured if the information is available:

```
(rule cooperation-module-19
  (IF (PLAN-FINISHED ?PLAN ?RESULTS
      (SATISFY-INFO-REQUEST ?INFO ?ORIG)
      ?ID))
  (THEN (EXTRACT-DESIRED-VALUE ?RESULTS
        ?INFO ?DESIRED-VALUE)
        (ANSWER-INFO-REQUEST ?INFO
        ?DESIRED-VALUE ?ORIG)
        (DELETE-MOTIVATION SELF ?PLAN
        (SATISFY-INFO-REQUEST ?INFO
        ?ORIG))
        (SEE-IF-FURTHER-MOTIVATONS ?PLAN
        ?RESULTS ?ID)))
```

If an agent has to perform a task as part of one of its local plans, the first thing for the situation assessment module is to determine whether the activity can be performed locally. If it can, the situation assessment module must

§ Intentions have been used to describe many different concepts [36–38]; however, within this context, they refer to a desired state without reference to how it can be reached. There is a fine distinction between **plans** and **intentions**; plans correspond to recipes for performing particular actions or for achieving particular goal states and are 'known' by an agent. Intentions are adopted and are used to guide an agent's problem-solving activity [39].

* Expressed in more conventional terms, the agent models and the control rules can be seen as an abstract data type. The information is actually stored in the agent models, and the control rules provide the procedural semantics.

decide whether to perform it locally or whether to delegate it. If, however, it cannot be performed locally, the agent has no choice but to ask an acquaintance to carry it out. Such generic knowledge about how to behave in a social environment requires that the agent has a representation of the activity which can be performed locally: GRATE agents encode this in task descriptors.

```
(rule situation-assessment-14
  (IF (HAS-GOAL ?GOAL ?INPUTS ?MOTIVE
      ?PRIORITY)
      (CANNOT-PERFORM-LOCALLY ?GOAL))
  (THEN
      (TELL-MODULE  COOPERATION-MODULE
      GOAL-AID-NEEDED
      ?GOAL ?MOTIVE ?PRIORITY)))
```

As all these rules illustrate, representing information in the agent models is of no use unless it influences an agent's behaviour. If no action is ever taken on the basis of believing a proposition, there is no point in storing or even deducing it in the first place. This offers a consistency constraint for the developer of the general rules. All information slots of the agent model must appear in the antecedent of at least one generic production rule.

3.2 *Controlling domain level system tasks*

One important function of the control module is to launch tasks in the underlying domain level system. The responsibility for deciding which tasks should be executed rests with the situation assessment module; however, the initiation and exact interleaving of the tasks is the responsibility of the control module. Control rule 6 provides an illustration of a generic rule associated with the execution of domain level tasks. The agent's self model task descriptors define the information required to initiate the task, and hence forms the basis of the missing information clause:

```
(rule control-module-6
  (IF (EXECUTE-TASK ?AGT ?TASK)
      (MISSING-INFORMATION ?AGT ?TASK
      ?INFO))
(THEN (TELL-MODULE SITUATION-
            ASSESSMENT INFO-MISSING ?AGT
            ?TASK ?INFO)))
```

This rule places an interpretation on task descriptor's 'necessary requirements'. It states that if there is 'missing information', a request should be made to the situation assessment module to provide this piece of information. In the situation assessment module, the request may initiate a social activity (if the information cannot be produced locally), may result in a new task being launched locally, or may result in no action if the task is not an important one. The role of generic knowledge at this level is again to define situations which must always be true before other actions can occur.

Another illustration of controlling domain level activity relates to how co-operation requests should be honoured. If an agent accepts a request to provide information for an acquaintance, it first checks whether it is currently undertaking an activity that produces the desired information (by looking at the outcome slot of the intention descriptors). If this is the case, this existing activity should be utilised, rather than starting a new one:

```
(rule situation-assessment-30
  (IF (INFO-REQUEST-MADE ?INFO ?ORIG
      ?PRIORITY)
      (ACTIVE-INTENTION-PRODUCING-
      INFO SELF ?INFO ?INTENTION))
  (THEN (ADD-ADDITIONAL-MOTIVATIONS
      SELF ?INTENTION (SATISFY-INFO-
      REQUEST ?INFO ?ORIG))))
```

One of the important tasks of the situation assessment module is to track the execution of tasks in the underlying domain level system. The rule below illustrates one facet of this tracking. It states that if the control module has reported that a goal has finished, but that the results produced were not what was expected, then the situation assessment module should try and find an alternative method for achieving the goal. This may involve trying another local activity if one exists or requesting aid from an acquaintance. This rule again makes use of the motivation slot for carrying out a goal, the expected outcome slot of the recipe to determine what should have been produced and task descriptors to determine alternative sources.

```
(rule situation-assessment-20
  (IF (GOAL-FINISHED ?GOAL ?RESULTS
      ?ORIG)
      (IS-MOTIVATION ?ORIG ?GOAL (PLAN-
      ACTION ?PLAN-NAME ?MARKER ?ID))
      (UNEXPECTED ?RESULTS ?PRODUCED
      ?GOAL ?RESULTS))
  (THEN (DETERMINE-WHETHER-
      ALTERNATIVE-SOURCES ?GOAL
      ?RESULTS
      ?PLAN-NAME ?MARKER ?ID)))
```

3.3 *Building multi-agent systems using GRATE*

The motivation for building GRATE was to investigate the feasibility of constructing a new type of multi-agent development environment; one in which significant amounts of knowledge related to co-operation and control was embodied. This novel approach requires a new paradigm for building multi-agent applications, in which the application builder has to configure pre-existing knowledge and augment it with any necessary domain-specific information. In present environments, the whole system has to be constructed afresh, meaning that a builder is continually re-coding the same knowledge. Domain-specific information may be required either to provide a short cut in order to meet the desired

performance characteristics or to reflect truly domain-dependent reasoning.

This leads to a situation in which generic rules define an agent's default behavour (i.e. given no information to the contrary, an agent's activity is governed by generic rules). However, in certain well defined instances, the default could be overridden by behaviour tailored to the specific situation at hand. The application builder obviously needs to provide the appropriate domain-specific rules, but applications could still be built very rapidly because much of the general behaviour is already defined. We are currently working on strategies for incorporating domain-specific reasoning into the GRATE architecture; the solution being pursued is to specify a meta-control strategy that ensures application-specific rules take precedence over generic ones.

The configuration process may involve selecting a subset of the available knowledge, for the problem at hand. For example, the application builder may never want to use a contract net, in which case they would remove the general knowledge associated with this protocol. Configuration may also involve fine-tuning the control strategies of the problem-solving modules. In the present implementation, equal weight is given to each module. However, in applications requiring more sophisticated local control and less inter-agent interaction, the control and situation assessment modules may need to be provided with more resources than the co-operation module.

This approach has significant advantages over conventional means of constructing multi-agent systems, including the reuse of problem-solving components (increasing reliability and decreasing risks), decreasing development time (some knowledge acquisition and coding has already been carried out) and making effective use of specialists [40]. It also follows the lead of other disciplines which engineer complex artifacts (e.g. planes, cars), in that product development would consist predominantly of assembling components [41].

4 Using GRATE in electricity transport management

To verify that the concepts embodied in GRATE are useful and applicable, the domain of electricity transport management was used and our experiences are described in this Section. Recently, GRATE has also been applied to the co-operative diagnosis of a particle beam accelerator controller at the CERN laboratories [42]. These problems were chosen because contacts made within the ARCHON project afforded the opportunity to work on real-world problems with all that involves; from a DAI point of view, the applications are of appropriate complexity [43]; and finally, they are both typical of a large class of process control systems.

In order to be available at the required consumption sites, electrical energy has to pass through a series of complex processes; *generation*, *transportation* and *distribution*. Generation is the process of transforming 'raw' energy (e.g. hydraulic, thermal, nuclear and solar) into a more manageable and useful form such as electrical energy. Ideally, the consumption sites would be near

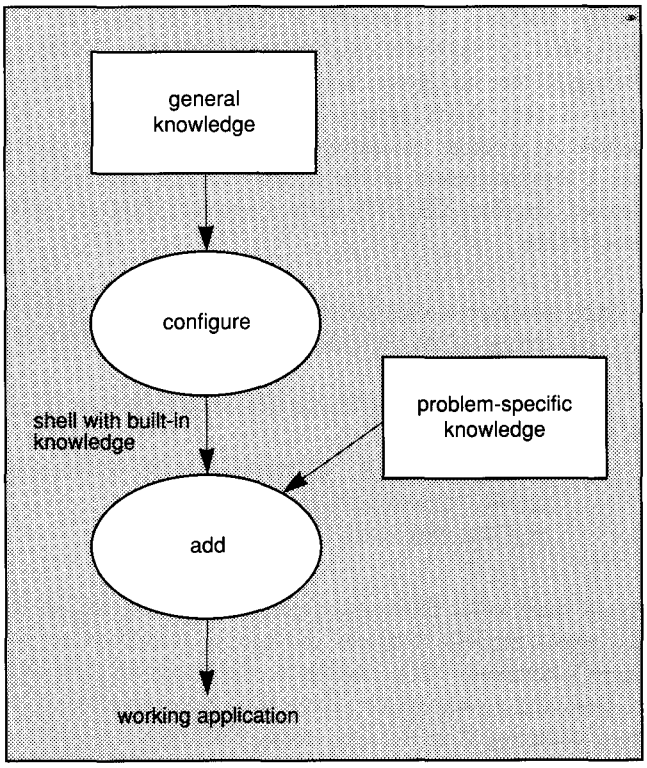

Fig. 2 GRATE paradigm for building multi-agent systems

these generation sites. However, due to various economic, social and political factors, this is often impossible. Hence, energy needs to be transported from its generation site to the customer. To minimise losses during transportation, the electrical voltage is made high (132 kV or above) before it is placed on the *transport network* and sent over many hundreds of kilometres. The final operation necessary to make energy available to the customers is the lowering of the voltage and the distribution to the consumers. This is achieved using an electricity *distribution network*, which involves many kilometres of network (all below 132 kV) spread over a much smaller area.

To ensure the electricity transportation network remains within the desired safety and economical constraints, it is equipped with a sophisticated data acquisition system. This system acquires enormous amounts of information, which it sends to a central dispatching control room (DCR) for analysis by control engineers (CEs). As a result of this analysis, a CE may detect a disturbance and deem it necessary to alter the configuration of the network; achieved by sending commands from their console through the telemetry system to the network elements, or by giving orders over the phone to engineers in the field (in places where telemetry is not available).

Whenever a non-planned and sudden disconnection of the network occurs, an emergency situation is signalled to the operators in the DCR by displaying alarm messages related to changes in network elements† on various

† Elements that are monitored include breakers (devices which allow the connection or disconnection of energised lines, transformers and busbars between them, and which may be open or closed from the DCR), fault recorders (mechanisms that record the voltage and intensity during the disturbance) and protective relays (mechanisms which trip breakers after sensing a fault).

screens. One of the software systems in the DCR (the CSI) has to distinguish between messages related to disturbances and those related to preplanned maintenance operations. The latter are not usually relevant from the point of view of diagnosis and restoration; the former are. The CSI also has to organise the data into a form which is understandable by other agents.

Once a disturbance has been identified, it must be ascertained whether it is transient or permanent, where it originated from and its extent. Two systems have been constructed to carry out these tasks. The BAI calculates the group of elements that are out of service and also monitors the evolution of the network (detecting whether the situation is progressing or if it is getting worse) in order to advise whether a new restoration plan is needed. The AAA locates the individual element at fault, analyses the permanence of the fault and indicates any damaged automatism in the network. While these systems are engaged in their analysis, the auto-reclosing mechanisms§ operates, provoking a reaction in the network. This reaction has to be gathered by the CSI, and passed onto the AAA and BAI in order to confirm, refine or even change their prevous analysis.

After a while, the network reaches a steady state*, and it is time to assess the situation and start planning the actions required to restore the network. The SRA agent is responsible for planning the restoration process, and once a plan which complies with all the known constraints has been developed, its execution and monitoring starts. Monitoring is necessary because the plan may have been constructed incorrectly or is based on invalid assumptions about the status of the network, and is carried out by the BAI.

The following describes how GRATE has been applied to one of the co-operative scenarios in this domain. The social interaction described involves the CSI, AAA and BAI agents in detecting and locating a fault in the network. Owing to space limitations, the restoration process is not covered. The co-operative scenario illustrates how both task and result-sharing forms of co-operation can be instantiated. A more extensive list of co-operative scenarios for this application can be found in Reference 44.

4.1 Co-operatively detecting and locating faults

If the network is stable, alarm messages corresponding to changes of network state arrive at the CSI in relatively small numbers. Their arrival triggers a CSI recipe which analyses them in order to determine whether they correspond to planned maintenance operations or whether they represent a disturbance in the network. In most cases, they correspond to planned operations and, after a fixed time delay of ten seconds, the messages are grouped together and the plan finishes. The outcome of this plan (i.e. the block of messages and the fact that there is no disturbance) is then passed to the CSI co-operation module to see if the results are of use to any other agents in the community. The CSI model of the AAA and BAI contains the following information:

```
MODEL OF AAA/BAI: INTERESTS:
  {. . . ⟨BLOCK-ALARM-MESSAGES, T⟩,
       ⟨DISTURBANCE-DETECTION-MESSAGE,
            HAS-VALUE(DISTURBANCE)⟩ . . .}
```

On receipt of this information, the co-operation module invokes rule 5 (see Section 3.1). The block of alarm messages is sent to the AAA and the BAI as unrequested information (since 'T' trivially evaluates to true), but the disturbance-detection message information is not passed on since it does not have value 'disturbance'. If, however, the alarms correspond to a disturbance, the disturbance-detection message is also sent to the other agents.

On receipt of the notification of a disturbance, the situation assessment module of the AAA determines that the recipe 'DIAGNOSE-FAULT' (see Section 2.2.1) should be started; this process is illustrated in Fig. 3*a*. The plan is retrieved from the self model and the situation asssessment module tells the control module to start the first task, hypothesis generaton (HYP-GEN) (action 1). Hypothesis generation is a fast but inaccurate task and typically produces a large number of hypotheses for the network element at fault.

The situation assessment module also notes (from the second action in the recipe) that information concerned with the black out area (BOA) is useful. The reason for this being that the BOA identifies a list of network elements in which the fault must be situated. Therefore, in order to be consistent with the BAI findings, the individual network element pinpointed by the AAA should appear in the BOA. This constraint means that any faults which the AAA proposes in its first approximation that are not within the BOA can be disregarded. Based on its self model, the AAA is able to deduce that the BOA cannot be produced locally, and so, using situation assessment rule 14 (see section 3.1), a request for aid with this goal is sent to the co-operation module (action 2). The co-operation module then examines its acquaintance models and sees that the BAI is the only agent in the community capable of generating it. It selects the BAI as the agent with which to interact, the client-server as the appropriate protocol and sends a co-operation initiation request (action 3).

The BAI receives the co-operation initiation request, which its co-operation module confirms as an activity that the BAI is capable of, through the identify black out area (IBOA) task. The request is then passed to the situation assessment module to determine whether the request should be honoured (action 4). The situation assessment module confirms the request will be met and informs the co-operation module of this fact (action 5). The co-operation module then confirms with the AAA that its initiation has been accepted (action 6). In parallel

§ The breakers are provided with a mechanism for automatic reclosing, which is triggered when sufficient voltage returns or if sufficient time has elapsed for a non-permanent fault to disappear.

* This means the network has reached a state (configuration, voltages, energy flows) in which it remains until an action is performed by the CEs. This contrasts with the transient state, generated initially by a fault, in which the state is constantly and automatically changing. A steady state is signified by the fact that, during a certain period of time (e.g. one minute), there are no alarm messages.

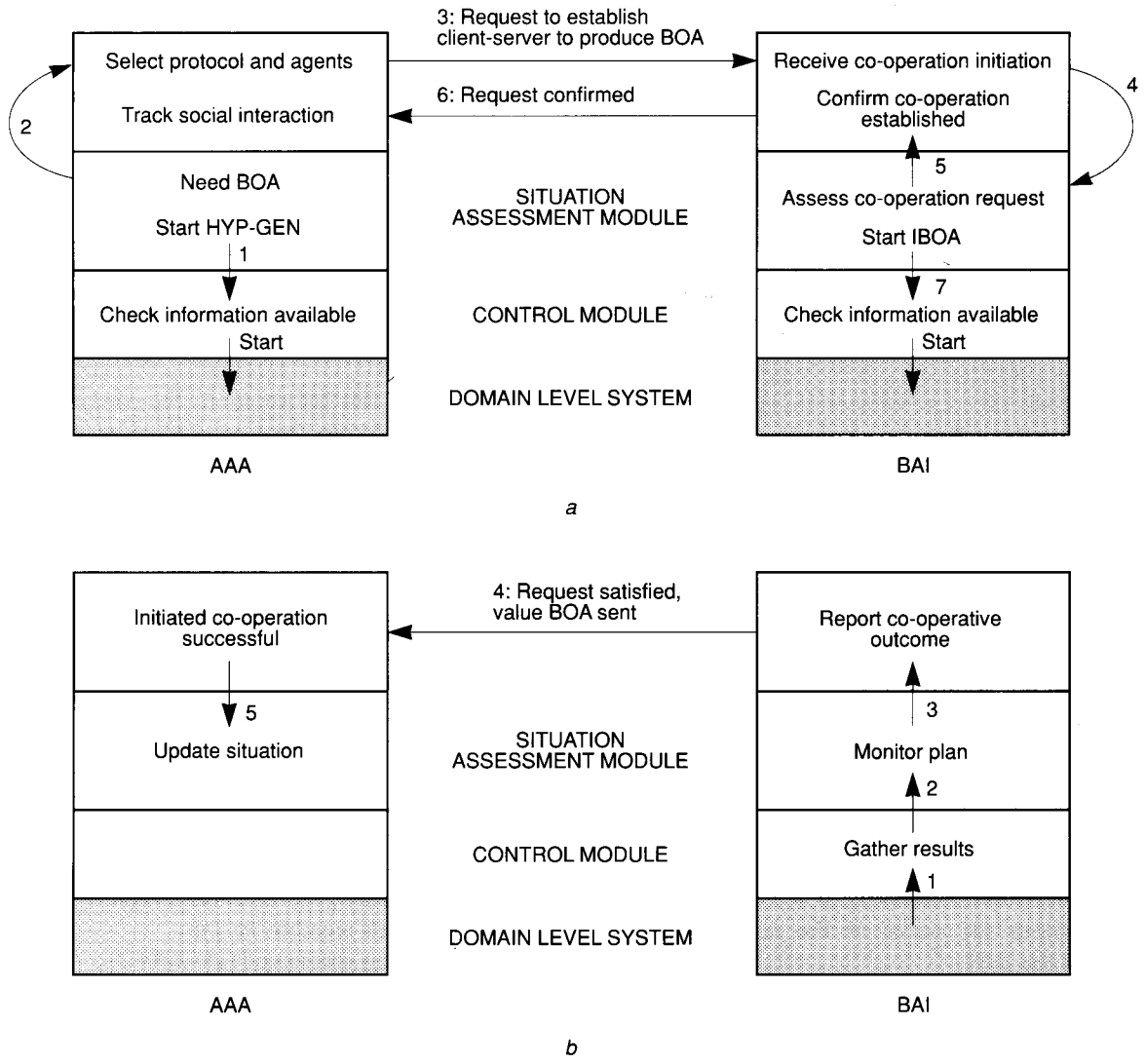

Fig. 3*a* **AAA initiating social activity *b* Terminating social interaction**

with the confirmation process, the BAIs situation assessment module tells the control module to execute the IBOA task (action 7).

When the IBOA task finishes (see Fig. 3*b*), the control module is informed, and it gathers the information produced (action 1) and passes it onto the situation assessment module (action 2). The situation assessment module monitoring the plan execution realises that the plan has been successfully completed (i.e. rule 20 (see Section 3.2) is not applicable). It then passes the results onto the co-operation module (action 3). The co-operation module tracking the social interaction progress is informed that the request has been satisfied and sends an activity complete message back to the originating agent (using rule 19 (see Section 3.1)), along with the desired piece of information (action 4). On receipt of the BOA, the AAA terminates the social action and passes the desired result onto its situation assessment module (action 5).

When the AAA situation assessment module receives the BOA it must determine what action to take. The third action (i.e. WHILE-WHEN) of the 'DIAGNOSE-FAULT' recipe is depicted in Fig. 4 and shows that the next step depends on the current problem-solving context. If it has not yet started validation, the refinement task is executed before validation is undertaken. If it has started validation, it will be suspended, hypothesis refinement executed and then validation restarted.

The rationale behind this is that validation is thorough and time-consuming (verifying a single fault may take a significant amount of time [45]), and in the worst case, it may have to work on the large number of approximate hypotheses produced during generation. The refinement task is very fast, and involves scanning the list of hypothesis which have yet to be validated and removing any which are not in the black out area; thus, enabling a substantial speed-up on the validation process to be achieved because fewer hypothesis have to be examined.

The evaluation of the next action to be carried out is performed by the situation assessment module (making use of the current action component of the recipe), and then the appropriate commands (e.g. SUSPEND, START) are issued to the control module. When the list of validated fault locations have been produced, they are sent to the operator and the SRA so fault repair can commence.

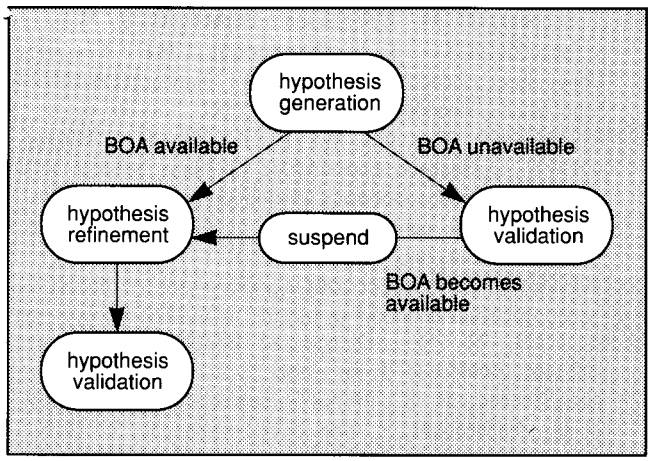

Fig. 4 AAA locate fault plan

5 Conclusions

The aim of this work was to investigate the feasibility of constructing a new and more powerful development environment for multi-agent systems. This environment contains significant amounts of inbuilt knowledge which the builder can utilise, rather than coding the whole system afresh. Such environments offer a new paradigm for building systems, one in which a significant proportion of the time is spent configuring existing repositories of knowledge and then augmenting them with any domain-specific information that is required. The applicability of this approach has been validated by the fact that such a framework has been constructed and applied to a real-world problem in the field of electricity transport management. Further evidence to support the generality claim has emerged from preliminary investigations into applying GRATE to the domain of co-operative fault diagnosis for a particle accelerator beam controller [42].

The logical extension of GRATE is to present the application builder with a framework in which all the necessary control knowledge is inbuilt, and all that has to be done is to instantiate each agent's self model. Community members would then be able to automatically build up models of other agents and the participate in co-operative problem solving based on this general knowledge. Although this appears attractive, it is unlikely that such an approach would have the necessary expressive power for any sufficiently large application nor be able to handle more complex interactions requiring substantial amounts of domain-specific reasoning. The problem is then to find a sufficiently general approach, which retains the expressiveness necessary for real-size applications, and to find mechanisms for incorporating domain-specific reasoning.

The problem of devising a system powerful enough to handle complex interactions in real-size domains, while retaining the necessary performance characteristics and generality, is at present an open research problem in DAI. Based on our experiences with GRATE, we feel that generic knowledge should be an integral part of the solution, but that it needs to be augmented with more powerful mechanisms. Many of the GRATE generic rules encode sequences of actions, and so well established patterns of rule firings can be observed. To enhance the system's power, well defined sequences could be compiled down and activated as a single unit when the appropriate stimulus is received. Such units of activity are similar in nature to reactive planning systems [46, 47], in which agents act on objects and react to events in a contingent, immediate, and therefore in the traditional AI sense of the word, unplanned manner. As a result of these insights, a hybrid approach, in which both general rules and reactive mechanisms are combined, is now being followed within the ARCHON project [25].

An unexpected benefit of the work in developing generic control and co-operation knowledge was that a substantial part of a theory emerged for describing how groups of agents should work together. This theory defines the necessary preconditions for group activity to commence and prescribes how individuals should behave when engaged in joint problem solving [48, 49].

6 Acknowledgments

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, Labein, QMW, IRIDIA, Iberdrola, EA Technology, Amber, Technical University of Athens, University of Amsterdam, Volmac, CERN and University of Porto.

The authors would like to thank Thies Wittig (Atlas) and Erick Gaussens (FTC) for their contribution within the project on issues related to architecture and separation of concerns, and Daniel Gureghian and Jean-Marc Loingtier (Framentec) for numerous discussions on the benefits and use of reactive mechanisms within a general architecture.

7 References

[1] FEIGENBAUM, E. A., McCORDUCK, P., and NII, H. P.: 'The rise of the expert company' (Times Books, 1988)

[2] LEVESON, N. G.: 'The challenge of building process control software', *IEEE Softw.*, 1990, pp. 55-62

[3] PARTRIDGE, D.: 'The scope and limitations of first generation expert systems', *Future Gen. Comput. Syst.*, 1987, **3**, (1), pp. 1-10

[4] STEELS, L.: 'Second generation expert systems', *Future Gen. Comput. Syst.*, 1985, **1**, (4), pp. 213-221

[5] GUHA, R. V., and LENAT, D. B.: 'CYC: a mid term report', *AI Mag.*, 1990, **11**, (3), pp. 32-59

[6] BOND, A. H., and GASSER, L.: 'Readings in distributed artifical intelligence' (Morgan Kaufmann, 19988)

[7] GASSER, L., and HUHNS, M. N.: 'Distributed artificial intelligence' (Pitman, 1990) Vol. 2

[8] HUHNS, M. N.: 'Distributed artificial intelligence' (Pitman, 1989)

[9] McCLELLAND, J. L., and RUMELHART, D. E.: 'Parallel distributed processing' (MIT Press, 1986)

[10] LESSER, V. R., and ERMAN, L. D.: 'An experiment in distributed interpretation', *IEEE Trans.*, 1980, **C-29**, (12), pp. 1144-1163

[11] SMITH, R. G., and DAVIS, R.: 'Frameworks for cooperation in distributed problem solving,' *IEEE Trans.*, 1981, **SMC-11**, (1), pp. 61-70

[12] LESSER, V. R., and CORKILL, D.: 'The distributed vehicle monitoring testbed: a tool for investigating distributed problem solving networks', *AI Mag.*, 1983, pp. 15-33

[13] CAMMARATA, S., MCARTHUR, D., and STEEB, R.: 'Strategies of cooperation in distributed problem solving'. Proc. Int. Joint Conf. on Artificial Intelligence, Karlsruhe, Germany, 1983, pp. 767-770
[14] ERMAN, L. D. and LESSER, V. R.: 'A multi-level organisation for problem solving using many diverse cooperating sources of knowledge'. Proc. Int. Joint Conf. Artificial Intelligence, Tbilisi, Georgia, USSR, 1975, pp. 483-490
[15] PARUNAK, H. V. D.: 'Manufacturing experience with the contract net' *in* HUHNS, M. N. (Ed.): 'Distributed artificial intelligence' (Pitman, 1989), pp. 285-310
[16] GASSER, L., BRAGANZA, C., and HERMAN, N.: 'MACE: a flexible testbed for distributed AI research', *ibid.*, pp. 119-153
[17] GASSER, L., BRAGANZA, C., and HERMAN, N.: 'Implementing distributed artificial intelligence systems using MACE'. Proc. Third IEEE Conf. on Artificial Intelligence Applications, Washington DC, 1987, pp. 315-320
[18] HAYES-ROTH, F. A., ERMAN, L. D., FOUSE, S., LARK, J. S., and DAVIDSON, J.: 'ABE: a cooperative operating system and development environment' *in* RICHER, M. (Ed.): 'AI tools and techniques' (ABLEX, 1988)
[19] BISIANI, R., ALLEVA, F., FORIN, A., LERNER, R., and BAUER, M.: 'The architecture of the AGORA environment', *in* HUHNS, M. N. (Ed.): 'Distributed artificial intelligence' (Pitman, 1989) pp. 99-118
[20] AGHA, G.: 'A model of concurrent computation in distributed systems' (MIT Press, 1986)
[21] HEWITT, C. E.: 'Viewing control structures as patterns of message passing', *Artif. Intell.*, 1977, **8**, (3), pp. 323-364
[22] CHANDRASEKARAN, B.: 'Generic tasks in knowledge based reasoning: high level building blocks for expert system design', *IEEE Expert*, 1983, **1**, (3) pp. 23-30
[23] STEELS, L.: 'Components of expertise', *AI Mag.*, 1990, **11**, (2), pp. 29-49
[24] JENNINGS, N. R.: 'Cooperation in industrial systems'. Proc. ESPRIT Conf., Brussels, 1991, pp. 253-263
[25] JENNINGS, N. R., and WITTIG, T.: 'ARCHON: theory and practice' *in* GASSER, L., and AVOURIS, N. (Eds.): 'Distributed artificial intelligence: theory & Praxis' (Kluwer Academic Press, 1992)
[26] SIMON, H. A.: 'Models of man' (Wiley, 1957)
[27] DEMAZEAU, Y., and MULLER, J. P. (Eds.): 'Decentralized AI' (Elsevier, 1990)
[28] DEMAZEAU, Y., and MULLER, J. P. (Eds.): 'Decentralized AI' (Elsevier, 1991) Vol. 2
[29] BENYON, D., INNOCENT, P., and MURRAY, D.: 'System adaptivity and the modelling of stereotypes'. Human Computer Interaction–INTERACT 87 (BULLINGER, H. J., and SHACKEL, P. (Eds.), Elsevier (B.V) North-Holland) pp. 245-253
[30] CHIN, D. N.: 'User modelling in the UNIX consultant' *in* MANTEI, M., and ORBITON, P. (Eds.): 'Human factors in computing systems' 1986
[31] RICH, E.: 'User modelling via stereotypes', *Cognit. Sci.*, 1979, **3**, pp. 329-354
[32] SUEYOSHI, T. and TOKORO, M.: 'Dynamic modelling of agents for coordination' *in* DEMAZEAU, Y., and MULLER, J. P. (Eds.): 'Decentralized AI' (Elsevier, 1990)
[33] RODA, C., JENNINGS, N. R., and MAMDANI, E. H.: 'The impact of heterogeneity on cooperating agents', Proc. AAAI Workshop on Cooperation among Heterogeneous Intelligent Systems, Anaheim, Los Angeles, 1991
[34] AVOURIS, N. M., LIEDEKEKERKE, M. H. V., and SOMMARUGA, L.: 'Evaluating the CooperA Experiment'. Proc. 9th Workshop on Distibuted Artificial Intelligence, Seattle, Washington, 1989
[35] BRANDAU, R., and WEIHMAYER, R.: 'Heterogeneous multi-agent cooperative problem solving in a telecommunication network management domain'. Proc. 9th Workshop on Distributed Artificial Intelligence, Seattle, Washington, 1989, pp. 41-57
[36] WERNER, E.: 'Cooperating agents; a unified theory of communication & social structure' *in* GASSER, L., and HUHNS, M. N.: 'Distributed artificial intelligence' (Pitman, 1990) Vol. 2, pp. 3-36
[37] BRATMAN, M. E.: 'What is intention?', *in* COHEN, P. R., MORGAN, J., and POLLACK, M. E. (Eds.): 'Intentions in communication' (MIT Press, 1990) pp. 15-33
[38] COHEN, P. R., and LEVESQUE, H. J.: 'Intention is choice with commitment', *Artif. Intell.*, 1990, **42**, pp. 213-261
[39] POLLACK, M. E.: 'Plans as complex mental attitudes' *in* COHEN, P. R., MORGAN, J., and POLLACK, M. E. (Eds.): 'Intentions in communication' (MIT Press, 1990) pp. 77-105
[40] HOROWITZ, E., and MUNSEN, J. B.: 'An expansive view of reusable software', *IEEE Trans.*, 1984, **SE-10**, (5), pp. 477-487
[41] STEFIK, M.: 'The next knowledge medium', *AI Mag.*, 1986, **7**, (1), pp. 34-46
[42] FUCHS, J., SKAREK, P., VARGA, L., and MALANDAIN, E.: 'Distributed cooperative architecture for accelerator operation'. Second Int. Workshop on Software Engineering, Artificial Intelligence and Expert Systems for High Energy and Nuclear Physics, L'Agelonde, La Londe-les-Maures, France, 1992
[43] WITTIG, T. (Ed.); 'ARCHON: an architecture for multi-agent systems' (Eliis Horwood, 1992)
[44] AARNTS. R., CORERA, J., PEREZ, J., GUREGHIAN, D., and JENNINGS, N. R.: 'Examples of cooperative situations and their implementation', *J. Softw. Res.*, 1991, **3**, (4), pp. 74-81
[44] GALLASTEGUI, I., LARESGOITI, I., PEREZ, J., AMANTEGUI, J., and ECHAVARRI, J.: 'Operating experience of an expert system for fault analysis in electrical networks'. Int. Working Conf. on Expert Systems in Electrical Power Systems, Avignon, France, 1990
[46] AGRE, P. E., and CHAPMAN, D.: 'Pengi: an implementation of a theory of activity'. Proc. 6th National Conf. on AI, Philadelphia, Pennsylvania, 1987, pp. 268-272
[47] SUCHMAN, L.: 'Plans and situated actions: the problem of human-machine communication' (Cambridge University Press, 1987)
[48] JENNINGS, N. R.: 'Responsibility in social activity'. Proc. Second Belief Representation and Agent Architectures Workshop (GALLIERS, J. R. (Ed.)) University of Cambridge, 1991, pp. 176-201
[49] JENNINGS, N. R.: 'On being responsible'. Proc. Modelling Autonomous Agents in a Multi-Agent World, Third European Workshop, Kaiserslautern, Germany, 1991

The paper was first received on 20 November 1991 and in revised form on 6 February 1992.

N. R. Jennings and E. H. Mamdani are with the Department of Electronic Engineering, Queen Mary and Westfield College, Mile End Road, London E1 4NS; I. Laresgoiti and J. Perez are with Labein, Bilbao, Spain; J. Corera is with Iberdrola, 48008 Bilbao, Spain.