ELSEVIER Artificial Intelligence 75 (1995) 195-240

Artificial Intelligence

Controlling cooperative problem solving in industrial multi-agent systems using joint intentions

N.R. Jennings*

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

Received November 1992; revised March 1994

Abstract

One reason why Distributed AI (DAI) technology has been deployed in relatively few real-size applications is that it lacks a clear and implementable model of cooperative problem solving which specifies how agents should operate and interact in complex, dynamic and unpredictable environments. As a consequence of the experience gained whilst building a number of DAI systems for industrial applications, a new principled model of cooperation has been developed. This model, called Joint Responsibility, has the notion of joint intentions at its core. It specifies pre-conditions which must be attained before collaboration can commence and prescribes how individuals should behave both when joint activity is progressing satisfactorily and also when it runs into difficulty. The theoretical model has been used to guide the implementation of a general-purpose cooperation framework and the qualitative and quantitative benefits of this implementation have been assessed through a series of comparative experiments in the real-world domain of electricity transportation management. Finally, the success of the approach of building a system with an explicit and grounded representation of cooperative problem solving is used to outline a proposal for the next generation of multi-agent systems.

1. Introduction

Until comparatively recently, computer systems were only used in simple aspects of industrial applications—typically being kept out of critical control loops and replacing exactly the functionality of previous mechanical or analogue devices. However with

* Telephone: +44-171-975-5349. Fax: +44-181-981-0259. E-mail: N.R.Jennings@qmw.ac.uk.

the gradual acceptance of artificial intelligence (AI) techniques, organisations are now attempting to build more sophisticated systems; software is being used to supervise and control manufacturing processes, power networks, chemical plants and so on. As these automated applications become ever more sophisticated, so the knowledge which must be brought to bear increases—this in turn causes a concomitant growth in the size and complexity of the computer system. At this time, software solutions to leading-edge industrial applications are nearing the boundaries of present system engineering methodologies and seemingly insurmountable limitations to current-generation intelligent systems are being observed [48,59,70].

These limitations have caused a fundamental rethink of the paradigms used to tackle large applications [22,72]. Some researchers have advocated the building of systems which embody vast amounts of common sense knowledge [31]; others propose that developing sharable and reusable libraries of problem solving components is the way forward [5,57]; a third group suggest that developing systems which tackle whole classes of problems is the way to progress [14,55,71]; while a final school of thought argues that smaller, more manageable components which can communicate and cooperate should be used [6,29,34]. The work described here concentrates on the cooperating systems paradigm, recounting the insights gained from building a number of real-world industrial applications and offering a novel approach to the construction of multiple agent systems.

In Distributed AI (DAI) systems, problem solving agents cooperate to achieve their local goals and the goals of the community as a whole. Each individual is capable of performing a range of useful activities, has its own aims and objectives, and can communicate with other agents. The recognisable problem solving ability associated with each agent distinguishes components of DAI systems from those of connectionist or neural systems in which individual nodes have very simple states and no real expertise. Within a given DAI community, the agents usually have expertise which is related, but distinct, and which 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 because no one individual has sufficient competence, resources or information to solve the entire problem alone [39].

Interdependence occurs when activities undertaken by individual agents are related—either because local decisions made by one agent have an impact on the decisions of other community members (e.g. when building a house, decisions about the size and location of rooms impacts upon the wiring and plumbing), or because of the possibility of harmful interactions amongst agents (e.g. two mobile robots may collide when attempting to pass through a narrow exit).

Global constraints exist when the solution being developed by a group of agents must satisfy certain conditions if it is to be deemed successful. For instance, a house building team may have a budget of £250,000, a distributed monitoring system may have to react to critical events within 30 seconds, and a distributed air traffic control system may have to control the planes with a fixed communication bandwidth. If individual agents acted in isolation and merely tried to optimise their local performance then such overarching constraints are unlikely to be satisfied. Only through coordinated action will acceptable solutions be developed.

Finally, many problems cannot be solved by individuals working in isolation because they do not possess the necessary expertise, resources or information. Relevant examples include the tasks of lifting a heavy object, driving in a convoy and playing a symphony. It may be impractical or undesirable to permanently synthesize the necessary components into a single entity because of historical, political, physical or social constraints, therefore temporary alliances through cooperative problem solving may be the only way to proceed. Differing expertise may need to be combined to produce a result outside of the scope of any of the individual constituents (e.g. in medical diagnosis, knowledge about heart disease, blood disorders and respiratory problems may need to be combined to diagnose a patient's illness). Different agents may have different resources (e.g. processing power, memory and communications) which all need to be harnessed to solve a complex problem. Different agents may have different information or viewpoints on a problem which need to be combined to give the complete picture (e.g. in concurrent engineering systems, the same product may be viewed from a design, manufacturing and marketing perspective).

In this work, a DAI approach was adopted as the means of coping with the complexity of industrial applications for several reasons. 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 input domain means the complexity of the computation is lower, thus enabling the components to be simpler and more reliable. Secondly, a distributed approach often provides a more natural fit to the problem (e.g. sensor networks [50], air traffic control [12] and telecommunications network management [76]). Indeed, some researchers even go so far as to state that all real systems are distributed [33]. Finally, many organisations have substantial amounts of pre-existing (or legacy [10]) software which could profitably be integrated [42,44].

This paper discusses the insights gained whilst building a number of industrial multi-agent systems under the auspices of the ARCHON project [77]. These insights are predominantly based on the experiences from the real-world applications of electricity transportation management [43] and controlling a high-energy, particle accelerator [44] which were built using an ARCHON prototype system called *GRATE* (Generic Rules and Agent model Testbed Environment). GRATE is a general-purpose integrative system which contains a corpus of inbuilt generic knowledge about cooperation and situation assessment in DAI systems [37,43]. These experiences have been augmented and reinforced by subsequent work using the full ARCHON system in the domains of electricity distribution management [75] and cement factory control [69].

One of the most time-consuming and difficult activities when building the aforementioned multi-agent systems was ensuring that the community acted in a coherent manner even when unplanned events occurred. Examples of such events and the types of problems they cause include: new information becoming available (agents continue to pursue an activity even though one community member knows their processing is obsolete); unexpected requests being received (agents abandon requests, without informing the original agent, if a more important query is received); inter-agent synchronisation points being violated (agents needlessly wait for the results of an activity which has been abandoned or significantly delayed). The comparative ease with which such incoherency occurred was attributed to the fact that the participating agents did not embody sufficient knowledge of the cooperative problem solving process to operate in dynamic, uncertain and complex domains (Section 2). To rectify this problem, it was decided that agents should have a well-grounded and explicit model of cooperative problem solving on which their behaviour could be based. This model should not only prescribe how to behave when everything is progressing as planned, but it should also deal with cases in which something goes wrong (as is often the case in industrial applications!). Previous theoretical work had not satisfactorily addressed this issue, therefore a new model, called *Joint Responsibility*, was developed which had the notion of joint intentions at its core (Section 3). The Responsibility Model was then used to guide the implementation of an extended version of GRATE called *GRATE** (Section 4). A series of empirical experiments were performed to assess the level of coherence attained by GRATE* in environments of varying hostility (Section 5). These experiments also offered a means of comparison with the original GRATE system so that the qualitative and quantitative benefits of the new approach could be assessed. Finally, as a consequence of the insights gained in this work, a proposal for the next generation of multi-agent systems is made (Section 6).

2. Initial experiences with industrial multi-agent systems

If all agents had infinite processing power and complete knowledge of the beliefs, goals, actions and interactions of their fellow community members, it would be possible to know exactly what each individual was doing at each instant in time and also what it is intending to do in the future. In such circumstances, systems could be perfectly coordinated and the cost of achieving this state would not be prohibitively expensive [54]. However for the majority of industrial applications such reasoning capabilities and awareness of events are infeasible because:

(i) bandwidth limitations make it impossible for agents to be constantly informed of all developments in the system;

(ii) agents cannot continuously reason about all the ongoing activities within a community and yet still carry out their necessary local processing [67].

An approach which has recently become popular in industrial applications is to provide one agent with a complete picture and have a number of functionally distributed subcomponents. Such systems are usually ordered in a hierarchical fashion and have clear, predefined communication links. Although this approach increases maintainability, it requires the overall control and the inter-component coordination to be centralised. However as the subcomponents become more complex, so the concomitant control bottleneck increases. For example, in applications which consist of a number of distinct supervision and control subcomponents, the activation of tasks and the decision of what data to exchange between them depends on the state of the entire process. Therefore to perform the necessary assessment the controller must take into account the views of all the relevant subcomponents. A further disadvantage of this approach is that if the overall controller fails then the entire system is rendered useless.

To alleviate the control bottleneck, increase the flexibility of coordination amongst the subcomponents, and produce a system whose performance degrades gracefully, it was decided to utilise an approach in which the control, as well as the data, was decentralised. This regime provides the agents with a degree of autonomy to generate new activities and to decide which tasks to do next, but makes it correspondingly more difficult to attain coherent global behaviour (because each individual operates on the basis of local and incomplete information). This difficulty is exacerbated in cases where perceptions and actions are fallible and when the environment evolves dynamically (i.e. typical industrial applications!).

To provide a concrete basis for the subsequent discussion, consider the following scenario which is taken from an electricity transportation management application. Transportation is one of a series of steps which needs to be completed to make electricity available at consumers' sites—it is preceded by the generation phase and followed by the local customer distribution phase. Managing the transportation network involves a number of separate but inter-related activities, such as: distinguishing between disturbances and pre-planned maintenance operations; identifying the type (transient or permanent), origin and extent of faults when they occur; and determining how best to restore the network after a fault so that the time that customers are without electricity is minimised. A detailed specification of all the agents involved in this application is given in [20] and a description of their many and varied interactions is given in [1]; however for illustrative purposes this work concentrates on three agents and the problem of detecting and diagnosing faults. Two of the agents—the Alarm Analysis Agent (AAA) and the Blackout Area Identifier (BAI)—actually perform the diagnosis (to differing levels of precision and based on different information), while a third agent—the Control System Interface (CSI)—detects the disturbance initially and then monitors the network's evolving state. This particular scenario was chosen because it is typical of a class of behaviour, called functionally accurate cooperative (in which the cooperating agents do not possess the necessary information to solve all their subproblems completely and accurately [49]), which consistently appeared in the industrial applications which were studied.

In their original stand-alone state, both the AAA and the BAI had their own very rudimentary CSI which could collate alarm messages but not detect disturbances. This meant the two diagnosis systems were continually invoked unnecessarily by the receipt of routine maintenance operations and also that they could not exchange predictive information about fault hypotheses. Using a multi-agent approach, it became feasible to develop a common and more sophisticated CSI agent (this also improved consistency) and to allow the diagnosis agents to cooperate with one another [43]. This coupling improved the performance of the overall system in the majority of cases, but when anything unexpected happened incoherent behaviour often ensued. For example, if the CSI discovered, as a result of receiving further information, that the fault it had detected was in fact transient then the AAA and the BAI continued trying to locate a non-existent disturbance. Other events which caused problems included: the AAA realising that it was not being supplied with sufficient data to make a diagnosis (desired result could not be attained, but the BAI and the CSI carried on regardlessly); substantial changes occurring in the network topology which invalidated crucial assumptions made during diagnosis (CSI detected such changes but did not inform the AAA or the BAI so that the results they produce were worthless); and distractions from unforeseen events or agents in the community (synchronisation of activities between the AAA and the BAI were affected, causing one agent to needlessly wait for information from the other) [41]. After careful analysis, these problems were attributed to two main factors:

(i) many of the cooperative actions were not explicitly represented (Section 2.1);

(ii) even when explicit social interactions were set up, the underlying model of cooperation was obscured by a myriad of crucial but unstated assumptions (Section 2.2).

2.1. *Non-explicit joint actions*

The exchange of domain level information (e.g. disturbance detected, fault location hypotheses, etc.) which formed the cornerstone of most instances of cooperative behaviour in this application was controlled by the representations which agents maintained about one other. These agent models contain knowledge about the capabilities, recipes, [1] goals and interests of other community members and are instantiated by the designer when the multi-agent system is constructed. In the fault diagnosis scenario, for instance, the CSI represents the fact that the AAA and the BAI are interested in the information that a disturbance has occurred and the AAA's representation of the BAI indicates that it is capable of identifying the approximate location of the fault (the black out area).

When the CSI detects an unplanned disturbance, it informs the AAA and the BAI (based on the information contained in the interest slots of its agent models) and then enters network monitoring mode. In volunteering such information the CSI does not reason about its impact on the other community members—it merely believes they are interested in knowing that a disturbance has been detected. Upon receipt of this notification, the AAA and the BAI start diagnosing the fault. The AAA uses the recipe depicted in Fig. 1; performing a fast approximate phase which generates a large number of potential solutions (hypothesis-generation) and then a detailed and time-consuming validation. If information about the black out area arrives from the BAI in a timely fashion, it is used to prune the number of hypotheses considered in the AAA's validation phase—since the suspect element must be in the black out area, any initial hypotheses which are not in this list can be removed (hypothesis-refinement).

The AAA, BAI and CSI are clearly engaged in a cooperative action—their related activities combine to attain the overall goal of locating the fault. However examination of the agents' internal structures reveals they are working on local activities for local means. For example, the `component-of` slot of the PRODUCE-DIAGNOSIS recipe shows that the AAA is executing its diagnosis because the system designer indicated that it should perform this activity whenever it receives an indication of a disturbance in the network. There is no representation of the fact that this task is part of a larger cooperative activity which involves the CSI monitoring the network and the BAI providing the black out area. Thus, for example, if the AAA decides to abandon its diagnosis (for whatever reason) it does not know that the BAI and the CSI need to be informed, so that they

[1] Recipes are a sequence of steps known by an agent for attaining a particular objective [60]. A *social recipe* is a series of steps undertaken by two or more agents as part of an explicit cooperative action.

```
Recipe Name: (PRODUCE-DIAGNOSIS)
Trigger: (INFO-AVAILABLE DISTURBANCE-DETECTION-MESSAGE)
Actions: (  (PAR (START (HYPOTHESIS-GENERATION
                              ?BLOCK-ALARMS ?FAULT-HYP))
                 (GET-INFO BLACK-OUT-AREA))
            (WHILE (AND (INFO-UNAVAILABLE BLACK-OUT-AREA)
                        (INFO-UNAVAILABLE VALIDATED-HYP)) DO
               (START(HYPOTHESIS-VALIDATION
                        ?FAULT-HYP ?VALIDATED-HYP)))
            (START-IF (INFO-UNAVAILABLE VALIDATED-HYP)
               (SUSPEND (HYPOTHESIS-VALIDATION))
               (START (HYPOTHESIS-REFINEMENT
                         ?FAULT-HYP ?BLACK-OUT-AREA
                         ?REFINE-HYP))
               (REACTIVATE (HYPOTHESIS-VALIDATION
                              ?REFINE-HYP ?VALIDATED-HYP))))
Time to Execute: 64
Priority: 10
Outcome: (VALIDATED-HYP)
Component of: (SATISFY-LOCAL-GOAL (PRODUCE-DIAGNOSIS))
Current Action: (START (HYPOTHESIS-GENERATION
                         ?BLOCK-ALARMS ?FAULT-HYP))
Local Bindings:
   ((FAULT-HYP ?) (BLOCK-ALARMS ("IG SANTURCE OSCL" "IG ALON-
       SOTE OSCL" "IG HERNANI OSCL" "BR HERNANI 7 (CO) LOCAL"
       "BR HERNANI 8 (CO) LOCAL" "PROT HERNANI 7 INST" "PROT
       HERNANI 8 INST" "IG ALI OSCL" "IG BASAURI OSCL" "BR
       ORMAIZTE 3 (CO) LOCAL" "BR ORMAIZTE 4 (CO) LOCAL" "PROT
       ORMAIZTE 3 BACK-UP" "PROT ORMAIZTE 4 BACK-UP" "IG
       SANGUESA OSCL" "IG LOGRONO OSCL" "IG CORDOVIL OSCL" "IG
       ASUA OSCL" "IG ORMAIZTE OSCL" "IG VITORIA OSCL" "IG HER-
       RERA OSCL" "IG GATICA OSCL" "BR S.SEBAST 1 (CO) LOCAL"
       "BR S.SEBAST 2 (CO) LOCAL" "PROT S.SEBAST 1 BACK-UP"
       "PROT S.SEBAST 2 BACK-UP" "BR ELGOIBAR 1 (CO) LOCAL" "BR
       ELGOIBAR 2 (CO) LOCAL")))
```

Fig. 1. AAA's PRODUCE-DIAGNOSIS recipe.

can halt their associated activities, because there is no representation of the fact that they are carrying out any related activities.

This implicit representation of cooperative problem solving is sufficient when everything is progressing according to plan and nothing unexpected happens. However, as the GRATE experience showed, when something goes awry teamwork quickly becomes disorganised. To try and circumvent the lack of an explicit joint action representation more information could be placed in the interest slots of the agent models—for example, the AAA's representation of the BAI and the CSI could be extended to state that they are interested in being informed whenever the AAA gives up on the PRODUCE-DIAGNOSIS recipe. Similarly, the incoherent behaviour which ensues when the CSI mistakenly indicates that a transient fault is permanent could be avoided if its models of the AAA and the BAI are changed to state that they are interested in knowing about transient faults when they have previously been informed that the same fault is permanent. The disadvantage of this approach, however, is that the system designer has to identify all the events which may cause the cooperative activity to falter. This must be carried out for all possible permutations of agent interactions and all perceived threats to cooperation. Furthermore, the system builder does not have a guiding framework which identifies broad classes of problems—knowledge elicitation is completely unstructured. The twin difficulties of complexity and lack of structure mean that the ensuing system will be far from complete in terms of the types of problem which it can successfully deal with.

2.2. *Obscured model of cooperation*

As well as the cooperation which occurs through the volunteering of relevant information, GRATE also provides facilities for establishing explicit social actions. For example the `GET-INFO` component of the PRODUCE-DIAGNOSIS recipe specifies that the AAA should obtain information about the black out area. After examining its agent models, the AAA will realise that this can only be achieved by sending an explicit request to the BAI—the AAA and the other agents in the community do not have a recipe whose outcome is the black out area. Assuming the BAI accepts the request, an explicit social action comes into existence: the `component-of` slot of the recipe used by the BAI will contain the following statement `(SATISFY-INFO-REQUEST AAA BLACK-OUT-AREA)`.

Even with these explicit social actions, the GRATE agents used an implicit model of cooperation to guide their actions and interactions. Consider the following rules which were part of GRATE's situation assessment functionality:

```
R1:  if successfully finish recipe R and
        R was performed because of a request by agent A
     then inform A that R has finished and supply it
        with the results which were produced

R2:  if recipe R has an agreed deadline D and
        R will not be completed by D and
        R was performed because of a request by agent A
     then inform A that R will not be completed by D
```

The principle underlying R1 is that when an explicit cooperation request has been completed, the agent which has undertaken the processing should inform the originator that the request has finished. This rule encodes a fundamental concept of cooperative interaction, namely that requests are usually made to provide information and services for the originator. Therefore when the desired service has been completed, the originating agent wishes to know at the earliest possible opportunity so that it can exploit this fact. R2 represents the principle that when a request for service is given a deadline it usually means that it must be synchronised with some of the originator's other activities. Any delay in service completion may adversely affect the originator and so the agent supplying the service should inform it at the earliest possible opportunity. Undertaking a similar analysis of the incoherency problems highlighted in the previous subsection reveals:

(i) the BAI and the CSI need to be informed that the AAA has abandoned its PRODUCE-DIAGNOSIS recipe because their activities are only meaningful in the context of the joint action and this will never succeed if the AAA's recipe is not completed;

(ii) the CSI needs to inform the AAA and the BAI of its mistaken identification of a transient fault because the motivation for the joint action is no longer present and so all the team members should cease their associated processing.

R1, R2, and the interest descriptors of the agent models exhibit the characteristics of first-generation expert system rules—they embody associational surface knowledge [23] which is sufficient to have the effect of the required inference, but which represents nothing of the underlying domain. In this case the underlying domain is the process of cooperative problem solving, rather than more traditional domains such as fault diagnosis or network restoration. In GRATE there is no explicit model of how to cooperatively solve a problem, rather the rules represent a straight association between an agent's problem solving state and its actions. Drawing upon the experiences of second-generation expert systems, a better solution would be to provide agents with an explicit representation of the model of cooperation which is being employed. Such a model should specify:

- how individual agents should behave when carrying out their local activities related to the joint action,
- how the joint action may come unstuck,
- how problems with the joint action can be repaired,
- how individuals should act towards their fellow team members when problems arise,
- how agents should re-organise their local activity in response to problems with the joint action.

This method of approach is also consistent with organisational science doctrines which state that the best way to tackle problems in the face of task and environmental uncertainty is to introduce explicit rules and procedures. If everybody adopts the appropriate behaviour the resultant aggregate's response is a coherent pattern of activity [27]. GRATE agents did not possess the necessary procedures to enable them to operate in uncertain environments and so their level of performance in such situations was significantly degraded.

3. Explicit models of cooperative problem solving

To fulfill the desiderata for a comprehensive model of cooperation two types of construct are needed: those which are associated with defining individual behaviour in a social context and those which are associated with cooperative behaviour *per se*.

With respect to defining individual behaviour in a social context, a growing band of researchers believe that the central controlling concept is that of *intentions* [8,17,24,64]—i.e. a commitment to act whilst in a certain mental state. Bratman highlights three important facets of individual behaviour which are governed by an agent's intentions [7]. Firstly, when an agent decides to pursue a goal it commits itself to bring about a particular state of affairs. Secondly, intentions are used to organise future actions; once an agent commits itself to a particular goal it must make subsequent decisions within the context that it will perform the necessary actions at the specified times. Finally, intentions pose problems for means-end analysis—this occurs because agents frequently commit themselves to a goal before they have worked out all the low-level details about how it can be realised.

To adequately fulfill these three roles, intentions must possess the following properties. Firstly, an agent's intentions should be both internally consistent and consistent with its beliefs. The former means that an agent's intentions should not conflict with one another; the latter that if an agent's intended actions are executed in a world in which its beliefs are true, the desired state of affairs should ensue [17]. The second important property is that intentions should have a degree of stability, but, on the other hand, they should not be completely inflexible. If intentions were constantly reconsidered and abandoned, the agent would spend all its time deliberating about actions rather than actually performing them. However an agent's intentions should not be irrevocable because circumstances may change and it is not always possible to correctly predict the future (as the experiences with GRATE showed). Achieving the balance between stability and flexibility means that agents need general policies (*conventions* [39]) which specify under what circumstances intentions should be re-examined. An associated property is that agent's need to monitor the execution of their intentions so that they can detect when the conditions specified in the convention pertain.

Given this description of intentions, it is clear that they meet some of the specified criteria of the explicit model of cooperation. In particular they define an individual's local behaviour when everything is progressing satisfactorily (i.e. honour commitments) and, through the definition of conventions, offer some insight into the types of difficulties which may arise during problem solving. As an example, Cohen and Levesque's model of intentions identifies the following situations under which an agent should reconsider its commitments to a particular goal G:

(i) G is already satisfied;
(ii) G will never be satisfied; and
(iii) the motivation for G is no longer present [17].

Given the success and elegance of modelling individual behaviour using intentions, joint intentions were considered the natural means of describing inherently cooperative phenomena. As a first approximation, joint intentions can be defined as a joint commitment to perform a collective action while in a certain shared mental state [18]. A number of formal models of joint intentions were available in the literature, although no one model was sufficient to be used directly in this work. Jennings presents a detailed review and comparison of the prominent models [39] and notes that the following points were made about cooperative problem solving:

- agents must have a joint goal [18,51,52,61,65,74],
- agents must agree they wish to cooperate to achieve their joint goal [18,51,65,74],
- agents must agree a common recipe for attaining their joint goal [52],
- actions performed by different agents, in the context of the joint action, are inter-dependent [52,61],
- agents must have conventions for monitoring the viability of their commitments [51].

Almost all of the formulations encode the fact that joint action requires a common goal which the group wishes to attain and the fact that they must all want to achieve this goal in a cooperative manner. The purpose of the common goal is to provide the glue to bind individuals' actions into a cohesive whole. The stipulation that the participants must want to achieve their goal in a cooperative manner distinguishes between a group of agents which independently have a goal which just happens to be the same and a group of agents which truly share a common aim. This distinction is important because the two relationships imply different consequences in social interaction: the former give rise to competition if resources are scarce, whereas the latter result in cooperation and coordination [19]. Once both the common goal and the desire to cooperate have been ascertained, the group becomes jointly committed to achieving its joint goal.

Many of the accounts fail to recognise that existence of a joint goal is not sufficient to guarantee that cooperative problem solving will ensue. Consider, for example, a country in which both the government and its independent national bank have the goal of lowering inflation, that they wish to achieve this by working together and that they are both mutually aware of this fact (i.e. they have a joint goal). Unless both parties are also capable of agreeing upon a common recipe for achieving their objective, there will never be cooperative action despite the fact that there is a joint goal. The common recipe provides a context for the performance of actions in much the same way as the joint goal guides the objectives of the individuals. Explicitly including the common recipe requirement in the definition places an additional functional role on joint intentions: namely that they will drive the participants' towards agreement of a common course of action.

Thus for a joint intention to be active, there must be a commonly agreed recipe which the agents are working under.[2] This does not imply that the recipe must be developed before joint action can commence nor does it mean that the recipe cannot evolve over time. Rather it reflects the fact that team members must believe that eventually they will be able to agree upon, and work under, a common recipe with respect to their joint goal. At any instant in time the recipe is likely to be partial—either *temporally partial* in that the exact ordering between some elements may not be specified or *structurally partial* in that not all of the actions have been fully elaborated [60]. Developing or refining the recipe is a complex activity because of the inter-dependencies which exist between the agents' actions. Because of this complexity a number of different planning paradigms, each with their own benefits and drawbacks, have been devised [12,21,25,30]—for

[2] As shown later, if an agent drops its commitment to the common recipe then this may no longer be the case, however this situation will be ignored for the present.

example, the recipe may be generated before any action has started or it may be interleaved with one agent or by a group of agents, and so on.

An important facet of the desired cooperation model is that it specifies criteria against which joint intentions can be monitored and also that it prescribes how to behave when things go wrong. However this aspect has been neglected by almost all of the formulations (the exception is Cohen and Levesque's model of joint intentions [51] which is described further in the next subsection). As a typical example of an extant model, Tuomela and Miller state that when things go wrong with one agent's activities, the other group members will help exert pressure and do whatever they think is necessary for the collective to succeed in achieving its objective [74]. This is particularly weak, it does not specify what it means for something to go wrong nor how to behave in such circumstances.

As none of the extant formulations fully met the list of requirements laid down for our cooperation model, a new model of joint intentions was needed. Rather than starting afresh it was decided that the new model would be based on Cohen and Levesque's work on joint intentions [51] (because it provided many of the desirable features). On the theoretical side, their model needed to be refined in order to more fully meet the requirements associated with the common recipe (see Section 3.1.2 for more details); but the major effort was in defining the path from the formal model to an implementation level system. In previous work on formal models of cooperation this pathway is rarely explained, let alone actually traversed!

3.1. Joint Responsibility

Joint Responsibility is specified formally using modal, temporal logics [35,36]. Like several of the extant formal models, Responsibility can be viewed as having two distinct levels. There is a low-level language which defines notions such as goals, recipes, actions, inter-dependencies and so on. This language is then used to build up higher-level concepts such as joint commitment to a joint goal and joint commitment to a common recipe. Here a discursive account of Responsibility is deemed sufficient because the main emphasis of this paper is in showing how the intuitions of the formal model can be used to guide the process of building robust cooperating communities, not in showing precisely how these concepts are formalised. Section 3.1.1 deals with joint goals; it specifies: what it means for a team of agents to be jointly committed to a joint goal, what types of incoherency can occur when executing a joint goal, and how agents should act towards their fellow team members when such problems occur. Section 3.1.2 specifies the same details for the common recipe. Finally, Section 3.1.3 draws these two strands of work together and presents the definition of Joint Responsibility.

3.1.1. Joint goals

Joint Responsibility uses a portion of Cohen and Levesque's model of joint intentions to represent joint commitment to a joint goal [18,51]. Commitment is based on the notion of joint persistent goals which, in turn, are based upon the concept of achievement goals. Achievement goals define the mental state of individuals participating in a team which is working towards a joint goal with a specified motivation. Agent α has a *weak* *achievement goal*, relative to its motivation q, to bring about p if either of the following are true:

(i) α does not yet believe that p is true (achieved) and has p being eventually true as a goal (i.e. α has a *normal achievement goal* to bring about p);

(ii) α believes that p is true, will never be true, or is irrelevant (q is false), but has a goal of making the status of p mutually believed[3] by all team members.

Thus a weak achievement goal has four separate cases:

(i) α has p as a normal achievement goal;

(ii) α thinks that p is true and wants to make this fact known to all;

(iii) α believes that p will never be true and wants to make this fact mutually believed;

(iv) α believes that the motivation for p is no longer valid and wants to make all agents aware of this fact.

A team of agents has a *joint persistent goal*, relative to q, to achieve p if and only if: they all mutually believe that p is currently false; they all mutually believe that they all want p to be eventually true, and until they all come to mutually believe either that p is true, that p will never be true, or that q is false, they will continue to mutually believe that they each have p as a weak achievement goal relative to q.

If a team is jointly committed to achieving p, they mutually believe that they each have p as a normal achievement goal initially. However as time passes team members cannot rely on the fact that they all still have p as a normal achievement goal, they can only assume that they have it as a weak achievement goal. The reason for this weaker statement is that one team member may have discovered that the goal is finished (impossible or irrelevant) and be in the process of making this fact known to its associates. If at some point it is no longer mutually believed that everybody still has the normal achievement goal then there is no longer a joint persistent goal (as not all the agents wish p to be true) and so the team is no longer jointly committed to p. However a weak achievement goal persists and ensures that all team members are informed of the lack of commitment by the doubting individual. This means agents can rely upon the commitment of others: firstly, to the overall objective (normal achievement goal) and then, if necessary, to the mutual belief of the status of the objective (weak achievement goal). Individuals can therefore undertake activities in the knowledge that others are working towards the same overall objective and that if something goes awry then they will be informed. This guarantee is important because in many cases when a problem surfaces it can only be detected by a subset of the team, but the existence of a weak achievement goal means that the agents which can detect the problem must endeavour to inform those which cannot.

In terms of the desiderata of our cooperation model, joint persistent goals: specify that agents should, in general, honour their commitments with respect to the joint goal; detail the convention by which agents should monitor their commitment to the joint goal; and, finally, describe how agents should behave both locally and towards others when commitments are reneged upon.

[3] Mutual belief is the infinite conjunction of beliefs about other agents' beliefs about other agents' beliefs and so on to any depth about some proposition [32].

3.1.2. The common recipe

Joint commitment to the commonly agreed recipe is specified through the notion of joint recipe commitment which, in turn, is based upon the concept of individual recipe commitment. *Individual recipe commitment* specifies that each team member should remain committed and actually perform the actions it has agreed to undertake, within the context of the common recipe, unless any of the following circumstances arise:

- The desired outcome of a recipe step is already available (COMPONENT-OUTCOME-EXISTS), e.g. the outcome may have been produced by a different sequence of actions or it may have been supplied by another agent.
- Following the agreed recipe does not achieve the desired result (RECIPE-INVALID), e.g. there may be a substantial change in the external environment which means that completing the current sequence of actions is pointless because the model upon which they are based is no longer valid.
- One of the specified actions cannot be performed (RECIPE-UNTENABLE), e.g. a sensor which provides a key source of information to an important task may malfunction meaning that the desired activity cannot be performed.
- One of the agreed actions has not been performed properly (RECIPE-VIOLATED), e.g. an agent which agreed to perform an action at a specified time may delay its execution because it has been distracted by an unanticipated and high priority event which requires its immediate attention.

The above conditions define situations in which an agent is able to detect for itself when problems related to the common recipe have occurred. Such awareness will typically be attained either because the agent is involved in the faulty action (e.g. it failed to execute a specified action at the appropriate time) or because it was able to directly observe the behaviour of a fellow community member and from this infer that something was amiss. A second way in which an agent may realise that there are problems with the common recipe is through being informed by a fellow community member. For example, an acquaintance may indicate that it was unable to perform one of its specified actions or that it believes that the recipe is no longer appropriate. Such indirect sensing is important because of the very nature of cooperative problem solving, if one participant stops contributing then the whole activity may be jeopardised. Therefore if an agent comes to believe that a fellow team member is no longer committed to the recipe then it also needs to reassess its position. Thus an agent can drop its commitment to the common recipe if it detects a problem for itself or if it is informed of a problem by a fellow team member.

Individual recipe commitment defines how each agent should behave with respect to its problem solving actions within the context of the joint action. It also defines the convention with which agents should monitor their commitments to the common recipe. However when an agent discovers that one of the conditions specified in this convention holds, it cannot simply abandon the recipe because its accomplices may not have been able to detect the problem for themselves. Therefore *joint recipe commitment* specifies that when an agent drops its commitment to the common recipe it must endeavour to inform all the other team members. This enables the whole team to reassess the recipe and means that if it needs to be abandoned or refined the amount of wasted resource is minimised because futile activities are stopped at the earliest possible opportunity.[4]

3.1.3. *Defining Joint Responsibility*

Joint Responsibility provides an explicit model of cooperation which meets all the desiderata set out in the previous section. It requires that all the team members $\alpha_1, \ldots, \alpha_n$ have a joint persistent goal to attain their joint goal σ, that they all execute the common recipe Σ according to the principles of joint recipe commitment, and moreover, that all the agents mutually know what they are doing whilst they are doing it (Fig. 2). Two distinct types of joint commitment are needed because they necessitate different courses of action when they are dropped. When commitment to the joint goal is dropped the joint action is over—if the goal is achieved, the group has satisfactorily completed its objective; if the motivation is no longer present or the goal will never be attained, there is no point in continuing. However, if the group becomes uncommitted to the common recipe there may still be useful processing to be carried out. For instance, if the recipe is deemed invalid, the agents may try a different sequence of actions which produce the same result. Similarly, if a recipe action is not performed correctly, the agents would typically reschedule their activities and carry on with the same course of action. Thus dropping commitment to the common recipe plays a different functional role to that of dropping a goal.

4. Implementing jointly responsible agents

The Responsibility Model suggests a high-level architecture for cooperating agents in which joint intentions play a central role in coordinating future actions and in controlling the execution of current activities [38]. Adoption of the Responsibility Model therefore places certain constraints on the agent architecture, it is not neutral in terms

[4] Cohen and Levesque's model deals with actions through their definition of joint intention (a joint persistent goal to perform an action while in a certain shared mental state [51]). Using their formalism it is possible to define the behaviour associated with joint recipe commitment; the joint intention to the common recipe Σ, as a way of attaining the joint goal G, is simply made relative to the joint commitment to G. Because joint intention is based upon joint persistent goals, the agent's behaviour when Σ is discovered to be achieved, impossible or irrelevant (note this means that Σ does not result in G, not necessarily that it is impossible, etc.) is analogous to the behaviour which results if similar discoveries are made about G itself. However because the joint commitment to Σ is a joint subgoal to G, even if the former joint commitment is abandoned the joint commitment to G may remain. In such cases, the agents will try and find an alternative common recipe to which they can jointly commit themselves.
Despite providing the necessary re-planning functionality, it was decided to use the notion of joint recipe commitment because it enumerates the specific conditions under which Σ may become inappropriate. Although component outcome exists, recipe invalid, recipe untenable, and recipe violation are all consequences of Cohen and Levesque's formulation they are not identified as separate categories (rather they are all bundled together under the banner of Σ no longer resulting in the attainment of G). Using joint recipe commitment means that the component which tracks the joint action in the implementation architecture is better defined with respect to the common recipe. It also means that the application designer's work of devising the necessary monitoring rules is better structured as the causes of recipe failure are explicitly provided by the cooperation model (more details of both of these processes are given in Section 4).

```
JOINT-RESPONSIBILITY($\{\alpha_1,\ldots,\alpha_n\},\sigma,\Sigma$) $\equiv$
  FORALL $\alpha_i \in \{\alpha_1,\ldots,\alpha_n\}$
    WHILE Normal-Achievement-Goal($\alpha_i,\sigma$)
              AND Individual-Recipe-Commitment($\alpha_i,\sigma,\Sigma$) DO
      PARALLEL
        Honour commitments to joint goal and common recipe
        Monitor rationality of commitment to $\sigma$
        Monitor rationality of commitment to $\Sigma$
      END-PARALLEL
    Suspend processing of local activities related to $\Sigma$
    CASE Reason for non-commitment OF
      NOT Normal-Achievement-Goal($\alpha_i,\sigma$):
        Abandon commitments to $\sigma$ and subsequent actions in $\Sigma$
      NOT Individual-Recipe-Commitment($\alpha_i,\sigma,\Sigma$):
        IF remedial action available
        THEN select possible remedies
        ELSE seek assistance to devise new common recipe
    END-CASE
    Inform other team members of lack of commitment,
      reason for commitment failure and remedial action if it exist
```

Fig. 2. Joint Responsibility Model of cooperative problem solving.

of implementation. However the descriptions it provides are at a suitably high level to ensure there is still significant leeway in how the concepts will actually be realised. For example, the model does not specify how intentions are represented, how commitment is described, what mechanisms are used to obtain agreements, nor how to develop the common recipe. Therefore the GRATE* implementation which will be described in the remainder of this section is but one computational interpretation of the model.

In this context, Responsibility had to be implementable in a computer system which could be used to control real-size industrial applications; therefore it was decided to adopt a rule-based approach rather than, for instance, a modal theorem prover. Such an interpretation consumes less computational resource and leaves the domain level system with more time to operate effectively. Thus GRATE*'s rules provide the operational semantics for Joint Responsibility; the rules are divided into two groups, those which assess the situation of the cooperating group and those which perform cooperative functions *per se*. According to the Responsibility Model definition, and the properties of intentions in general, situation assessment rules are needed to: decide when cooperation on a joint goal is appropriate, develop and obtain agreement on a common recipe, ensure that existing commitments are honoured, ensure that any new commitments are consistent with existing ones, monitor the problem solving state with respect to the convention for dropping commitment, and decide what actions should be taken if a joint commitment is dropped. The rules which control cooperative interactions must ensure that all team members are informed if the local agent reneges upon its joint

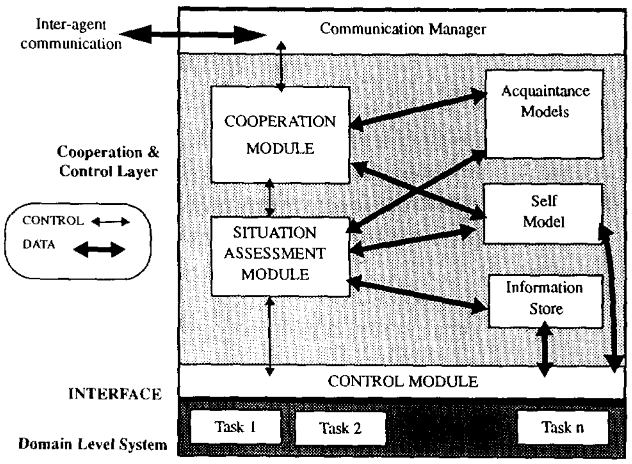

Fig. 3. GRATE* agent architecture.

commitment—in such circumstances they should also be made aware of the reason for this change and the proposed remedial action if one exists. As well as identifying the functionality which must be present, the formal model prescribes the key data structures, these include: goals, recipes, intentions and joint intentions.

Several simplifying assumptions are required to provide a reasonable approximation to the formal description and to make a reasonable implementation feasible. Firstly, it is assumed that communication is fool proof and that the message delay time is known to all agents. Secondly, although mutual belief has many appealing theoretical properties it is not a concept which can easily be attained in practical systems, thus a computational approximation is required—in GRATE* this led to beliefs with one level of nesting (i.e. everybody knows that everybody knows). Thirdly, in order to carry out coordination activities agents share a global clock reference. Finally, agents are able to predict, with a reasonable degree of accuracy, the time taken to execute each of their domain level activities.

4.1. GRATE* agent architecture

GRATE* agents have two clearly identifiable components: a *cooperation and control layer* and a *domain level system* (Fig. 3). The latter solves problems for the organisation, be it in the domain of particle accelerator control, electricity transportation management or cement factory control. Problems are expressed as *tasks*—atomic units of processing when viewed from the cooperation layer. For example, a domain system which diagnoses faults may be composed of two tasks: a heuristic one which produces a quick but approximate solution and a model-based one which uses this information to guide its accurate but lengthy verification. The cooperation layer is a meta-level controller which operates on the domain level system. Its objective is to ensure that the agent's domain level 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.

The cooperation 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 contained in each module are generic, they are applicable for controlling cooperative activities in a broad range of industrial applications [37]. The *control module* is the interface to the domain level system and is responsible for managing all interactions with it. The *situation assessment module* makes decisions which affect both of the other two 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 "get task Y executed by an acquaintance". Requests to the control module are of the form "suspend task T1" and "start task T2". The *cooperation module* is responsible for managing the agent's social activities, the need being detected by the situation assessment module. Three primary objectives related to the agent's social role are supported. Firstly, new social interactions have to be established (e.g. find an agent capable of supplying a desired piece of information). Secondly, ongoing cooperative activity must be tracked (using the Responsibility convention). Finally, cooperative initiations from other agents must be responded to.

The remaining components provide support functions. The *information store* is 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. As described in Section 2, *acquaintance* and *self-models* are representations of other agents and of the local domain level system respectively. The communication manager sends and receives messages to/from other agents in the community.

4.2. Paving the way for cooperation

Before cooperation can proceed a joint action between a group of willing acquaintances must be established. The need for, or desirability of, all new activities is detected by the agent's situation assessment module. After the agent has selected the appropriate recipe to achieve the desired goal, it has to determine whether the activity should be completed locally or whether assistance should be sought from other community members. This decision is based upon the number of recipe components which the agent wants its acquaintances to undertake. There are three potential outcomes of this analysis: joint action is definitely needed, the action can be solved entirely locally, or some assistance is required but not sufficient to warrant a full-scale joint action.

If a full joint action is warranted, the agent which detects the need becomes the *organiser*. The role of the organiser is to: identify other community members who will be willing to participate, get the team members to acknowledge that a common recipe is required and to agree to use the Responsibility cooperation model, and, finally, to actually devise the common recipe. Rather than having a protracted protocol in which each of these pre-requisites is agreed upon separately, a more concise version in which

```
PHASE 1
  Organiser detects need for joint action to achieve goal $G$
    and determines that recipe $R$ is the best means of attaining it.
  Organiser contacts all acquaintances capable of contributing to $R$ to determine if
    they will participate in the joint action using the Responsibility cooperation
    model.
  Let: $\Omega$ = set of willing acqaintances.
PHASE 2
  FORALL actions in $R$
    select agent $A \in \Omega$ to carry out action $\theta \in R$
      (criteria: minimize number group members)
    calculate time $(t_\theta)$ for $\theta$ to be performed based on temporal orderings of $R$
      and the anticipated communication delay
    send $(\theta, t_\theta)$ proposal to $A$
    $A$ evaluates proposal against existing commitments ($C$'s):
      IF no-conflict$(\theta, t_\theta)$ THEN create commitment $C_\theta$ for $A$ to $(\theta, t_\theta)$
      IF conflicts$((\theta, t_\theta), C) \wedge$ priority$(\theta) >$ priority$(C)$
        THEN create commitment $C_\theta$ for $A$ to $(\theta, t_\theta)$ and re-schedule $C$
      IF conflicts$((\theta, t_\theta), C) \wedge$ priority$(\theta) <$ priority$(C)$
        THEN find free time $(t_\theta + \Delta t_\theta)$, note commitment $C_\theta$ and
               return updated time to leader
      Return acceptance or modified time to team organiser
    IF time proposal modified THEN update remaining actions times by $\Delta t_\theta$
  END-FORALL
```

Fig. 4. GRATE* distributed planning protocol.

several conditions are settled in a single message interchange is more appropriate for industrial applications. GRATE* uses a two-phase protocol (Fig. 4). The first phase establishes the agents which will participate and the ground rules which they must adhere to—this phase is similar to the task announcement phase of the contract-net protocol [68] in which focused addressing is used to limit the extent of the broadcast to a subset of the entire community. The second phase of the GRATE* protocol specifies who will perform each action and at what time. Two phases are needed because of the complexities of the application and the agents involved. As the organiser does not have a complete picture of all activities within the community, it is not aware of the existing commitments and desires of all its potential team members. Hence exact timings must be reached through mutual agreement, rather than being dictated by the organiser.

Again the general concepts will be illustrated through the now familiar electricity transportation scenario. After establishing the desirability of the joint action, the organiser (the AAA) instantiates a representation of the joint intention in its self-model (Fig. 5). The `motivation` slot indicates that the reason for carrying out the joint action is that a disturbance has been detected. The `recipe` is a series of actions which need to be performed, together with some temporal partial ordering constraints, which

```
Name: (DIAGNOSE-FAULT)
Motivation: (INFO-AVAILABLE DISTURBANCE-DETECTION-MESSAGE)
Recipe:
  ( (PAR (START-OFF (IDENTIFY-BOA ?BLACK-OUT-AREA))
         (START-OFF (HYPOTHESIS-GENERATION
                       ?BLOCK-ALARMS ?INITIAL-HYPOTHESES))
         (START-OFF (MONITOR-DISTURBANCE)))
    (START-WHEN (INFO-AVAILABLE INITIAL-HYPOTHESES)
         (DETAILED-DIAGNOSIS ?INITIAL-HYPOTHESES
           ?BLACK-OUT-AREA ?VALIDATED-FAULT-HYPOTHESES)))
Start Time: -
Maximum End Time: -
Duration: -
Priority: 20
Status: ESTABLISHING-GROUP
Outcome: (VALIDATED-FAULT-HYPOTHESES)
Participants: ( (SELF ORGANISER AGREED-OBJECTIVE)
                (CSI TEAM-MEMBER AGREEING-OBJECTIVE)
                (BAI TEAM-MEMBER AGREEING-OBJECTIVE))
Bindings: NIL
Proposed Contribution:
  ( (SELF ((HYPOTHESIS-GENERATION YES) (DETAILED-DIAGNOSIS YES)))
    (BAI (IDENTIFY-INITIAL-BOA ?))
    (CSI (MONITOR-DISTURBANCE ?)))
```

Fig. 5. Representation of AAA's DIAGNOSE-FAULT joint intention.

will produce the desired outcome. This recipe uses the same basic constructs as the PRODUCE-DIAGNOSIS recipe of Fig. 1, but in accordance with the philosophy of explicitly representing joint actions it also encodes the related actions and how they combine to achieve the overall goal (i.e. the three main activities of identifying the black out area, of performing hypothesis generation and of monitoring the disturbance are all kicked off in parallel, then the detailed diagnosis[5] starts once the initial list of hypotheses has been produced). The recipe indicates what is to be done, not who is to do it or the time at which it should be done. The detailed timings are dealt with in the second phase of the protocol.

The `priority` slot indicates the importance of the intention and is used as the basis for computing its desirability—the higher the value the more desirable the action. Priorities are a combination of two components: an intrinsic (static) value associated with each recipe and a dynamic component which provides the necessary flexibility by reflecting the current problem solving context (e.g. whether the recipe is part of a local action, a joint action, or a social request). The intrinsic priority of the DIAGNOSE-

[5] Detailed diagnosis covers both hypothesis validation and hypothesis refinement.

FAULT recipe is 10 (see Fig. 1) and because the intention is invoked as a joint action it's context figure is 10, giving an overall priority of 20.

The `status` slot refers to the current phase of the planning protocol and has the value `establishing-group`, `developing-solution` or `executing-joint-action`. The `outcome` slot enumerates the expected results of performing the intention. The `participants` slot indicates the organisational structure of the group and the current stage of each agent's involvement. The diagnose fault action has one team organiser (the AAA) and two other potential team members (the BAI and the CSI). The AAA has agreed to participate in the cooperative act, since it is the originator, and it is in the process of establishing whether the other two wish to join. The `bindings` slot is used exclusively in the protocol's second phase.

`Proposed contribution` records those agents which are adjudged, by the organiser, to be capable of contributing to the joint act and it also specifies whether they have agreed to make that contribution. It can be seen that the AAA has consented to contribute by performing HYPOTHESIS-GENERATION and DETAILED-DIAGNOSIS. The BAI and the CSI have not yet agreed to contribute anything, though the organiser has already defined privately what roles it wishes them to play.

Having identified the need for joint action, the process of establishing a cooperating group can commence. The first phase of the protocol ascertains which acquaintances are willing to participate. As an initial step the organiser's situation assessment module informs its cooperation module that a social act is required (action 1, Fig. 6). The cooperation module identifies all those acquaintances who it believes can contribute to the recipe subcomponents, based on the knowledge of capabilities maintained in its acquaintance models. This approach works well in industrial applications because the community is fairly static and hence such capability knowledge can be assumed to be reasonably accurate and comprehensive. However in more open environments, in which agents enter and leave the community frequently, a more dynamic approach may be appropriate. At this stage the objective of the protocol is to identify the maximum number of agents which are capable of participating so that the organiser has a degree of flexibility in the second phase. After the potential contributors have been identified, they are each sent a message requesting their participation in the joint action (2). This message indicates that the sender wishes to establish a joint action involving the recipient, states the reason why this joint action is needed, the other agents which are being asked to participate, the team organiser's priority for the activity, and the activity's expected outcome.

Each potential participant receives the proposal in its cooperation module and checks that it understands the request. If the request is rejected, a negative response is prepared ($3'$) and passed back to the organiser. Assuming the request is accepted at the cooperation module level, it is then passed to the situation assessment module (3) to see whether the agent has sufficient resources to tackle the problem. This involves analysing existing commitments to ensure that the proposal is consistent and that it can be accommodated successfully. The result of this evaluation, either yes or no, is passed back to the cooperation module (4) which returns the answer to the organiser (5). If the agent wishes to participate, it sets up a joint intention description in its self-model using the information it has been supplied with.

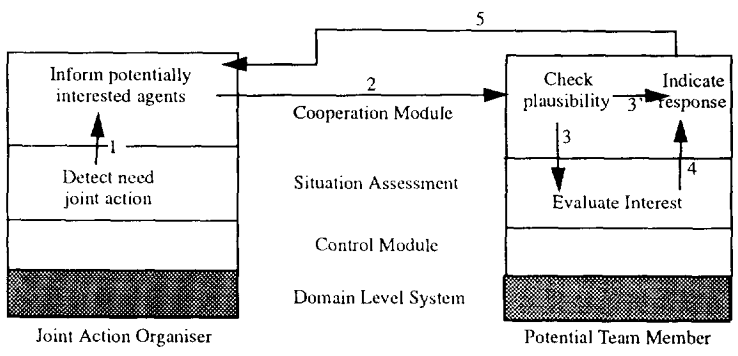

Fig. 6. Establishing a joint action.

By answering affirmative to the group initiation response, the BAI and the CSI acknowledge that: they are interested in participating in the cooperative fault diagnosis, that a common recipe is needed to solve the problem, and that they will use the Responsibility cooperation model for the duration of their collaboration. These criteria are not communicated explicitly because all community members share a common understanding of the semantics of performing a joint action.

Once all the community members who were identified as potential participants have replied, the second phase of the planning protocol begins and the team specifies the exact details of the common recipe. The organiser updates the joint action's `status` slot to `developing-solution` and modifies the `participants` slot to reflect the fact that the team is now in the solution planning phase. From the list of willing agents, the organiser selects those it wishes to be in the team and the contributions it would like them to make. The criterion upon which selection is based, in this instance, is to minimise the number of group members. This strategy was chosen because the fewer agents there are in the team, the lower the likelihood of one of them defaulting (since its individual loss would be greater). This strategy also reduces the coordination overhead because messages need to be sent to fewer agents. In other instances, however, it may be desirable to have the maximum number of agents in the team to balance the load throughout the community or to select the most competent agent to perform each task. The team leader updates its `proposed contribution` slot to indicate those agents which were selected and those which were not:

```
Proposed Contribution:
 ( (SELF ((HYPOTHESIS-GENERATION SELECTED)
          (DETAILED-DIAGNOSIS SELECTED)))
   (BAI (IDENTIFY-BOA SELECTED))
   (CSI (MONITOR-DISTURBANCE SELECTED)))
```

The team leader then makes an initial proposal for the timings of the individual actions, expressed in the `bindings` slot, and fills in the joint action's `duration`, and its `start` and `end times`. These timings take into account the fact that some time is required to obtain agreement on the common recipe—for each action which needs to be

```
Name: (ACHIEVE (IDENTIFY-BOA))
Motivation: (SATISFY-JOINT-ACTION (DIAGNOSE-FAULT))
Recipe: (IDENTIFY-BOA)
Start Time: 19          Maximum End Time: 34
Duration: 15            Priority: 5
Status: PENDING         Outcome: (BLACK-OUT-AREA)
```

Fig. 7. BAI's individual intention for producing the black out area.

performed by an acquaintance at least two messages must be sent to establish its start time and agents also take time to process a message.

```
Bindings:
  ((BAI IDENTIFY-BOA 19)
   (SELF HYPOTHESIS-GENERATION 19)
   (CSI MONITOR-DISTURBANCE 19)
   (SELF DETAILED-DIAGNOSIS 36))
```

The organiser then takes each recipe action in a temporally sequential order and agrees with the agent concerned the time at which it should be performed. The proposal sent to each contributor indicate the action to be performed, the time at which it should be started, and the team leader's current proposal for its context. Upon receipt of the proposal, the team member evaluates it to see whether it is acceptable. If there is no conflict, the agent adopts the intention and sets up an individual intention description in its self-model (see Fig. 7 for an example).[6] The agent then returns a message indicating its acceptance to the team organiser. If the suggested time is unacceptable the prospective team member proposes a time at which the action can be fitted in with its existing commitments, makes a tentative commitment for this time, and returns the suggestion to the organiser. If the modified time is acceptable to the organiser it will make appropriate adjustments to the solution bindings and proceed with the next action. If the modified time proposal is unacceptable then the organiser will look in its list of proposed contributors for a new actor to perform the action.

The process of agreeing a time for each action continues until all of them have been successfully dealt with. At this point, the common recipe is agreed upon and the organiser informs all team members of the final solution. The joint action is now operational. The organiser changes the `status` of the social action to `executing-joint-action` and the `participants` slot is updated to indicate that all team members are now in the process of performing the joint action.

[6] There are two slight differences between this representation and that of joint intentions. Firstly, the `recipe` slot contains the name of a local recipe rather than a list of actions and their relationships. Secondly, the status is simply `executing` or `pending` since there is no distributed planning phase.

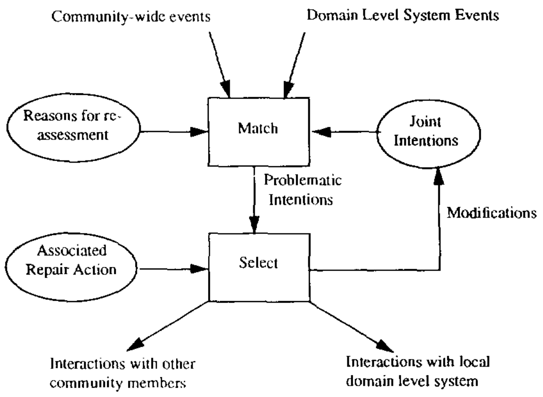
Community-wide events
Domain Level System Events
Reasons for re-assessment
Match
Joint Intentions
Problematic Intentions
Modifications
Associated Repair Action
Select
Interactions with other community members
Interactions with local domain level system

Fig. 8. Responsibility schemata for executing joint actions.

4.3. Using the Responsibility Model to derive generic cooperation rules

Once a joint action has been established, the agent must monitor its execution. It is during this phase that the tracking aspect of the Responsibility Model comes to the fore—specifying the conditions under which the agent should reconsider its commitments and describing how the agent should behave both locally and with respect to its fellow team members if any such problems arise. In terms of the GRATE* implementation, the Responsibility Model provides a specification of how the aforementioned aspects of the situation assessment module should function. The existence of such a specification meant the developer of GRATE* had a structure and organisational framework from which the necessary generic rules could be developed.

Responsibility provides a convention which clearly distinguishes between the situations in which commitment to the joint action needs to be re-examined and the actions which should be taken in such circumstances (Fig. 8). An agent's situation assessment module is continuously monitoring events which occur within the system—these events may originate from the domain level system (e.g. goal finished, information needed) or they may come from other agents within the community (e.g. request for information, return of requested information). In the majority of cases these events will have no effect on the joint action, however in some instances they will impinge upon it and cause the agent to reconsider its commitments. The difficulty in the original GRATE system was in knowing exactly which of the many events were relevant and which were not. However by using Joint Responsibility the relevant events are clearly specified—they are precisely those conditions in which an agent can drop its commitment to the joint goal (i.e. goal has been attained, goal will never be attained and goal motivation no longer present) and those conditions in which the agent can drop its commitment to the common recipe (i.e. component outcome available, recipe invalid, recipe untenable and recipe violated). All of these conditions are represented in the "reasons for re-assessment" component.

It is the task of the "match" process to identify when one of the situations in the reasons for reassessment store becomes true. Thus, for example, all those events which have the potential to cause the recipe to become invalid must be recognised, as must all those events which undermine the motivation for the joint goal, and so on, for all of the other conditions in the Responsibility convention. The formalisation of these events is beyond the scope of the Responsibility Model; therefore the developer of a GRATE* application has to analyse the domain to determine exactly what these events are likely to be. A portion of this analysis for the electricity transportation scenario is shown in Fig. 9. In this application the common recipe remains valid if its initial assumptions are correct, if each task provides the expected results and if all the action inter-relationships are attainable. Similarly, the motivation for the joint goal remains in tact if the beliefs for wanting the goal remain true and if the goal does not conflict with another more highly rated one.

Having assimilated all those events which cause the Responsibility problem situations to happen, the application developer must encoded them into GRATE*'s situation assessment module. The events themselves will form the rule antecedents, whilst the rules' conclusions will be that one of the Responsibility conditions for joint action reassessment has occurred. Examples of four such rules, which respectively encode the fact that the joint goal has been satisfied, that the motivation for the joint goal is no longer present, that the common recipe has been violated, and that the common recipe is invalid are given below:

```
R1_match:
  if task t has finished executing and
     t has produced the desired outcome of the joint action
  then the joint goal is satisfied

R2_match:
  if receive information i and
     i is related to the triggering conditions for joint goal G
     and i invalidates beliefs for wanting G
  then the motivation for G is no longer present

R3_match:
  if delay task t1 and
     t1 is a component of the common recipe R for a joint action
     and t1 must be synchronised with task t2 in R
  then R is violated

R4_match:
  if finished executing common recipe R and
     expected results of R not produced and
     alternative recipe exists
  then R is invalid
```

If any of the above match rules are fired, they identify an intention which has run into difficulty in its present form and which has jeopardised the entire joint action—R1 and R2 mean that the agent no longer has a normal achievement goal to attain the joint action, whilst R3 and R4 mean that the agent's individual recipe commitment has been compromised. According to the schemata of Fig. 8, the next step is to determine

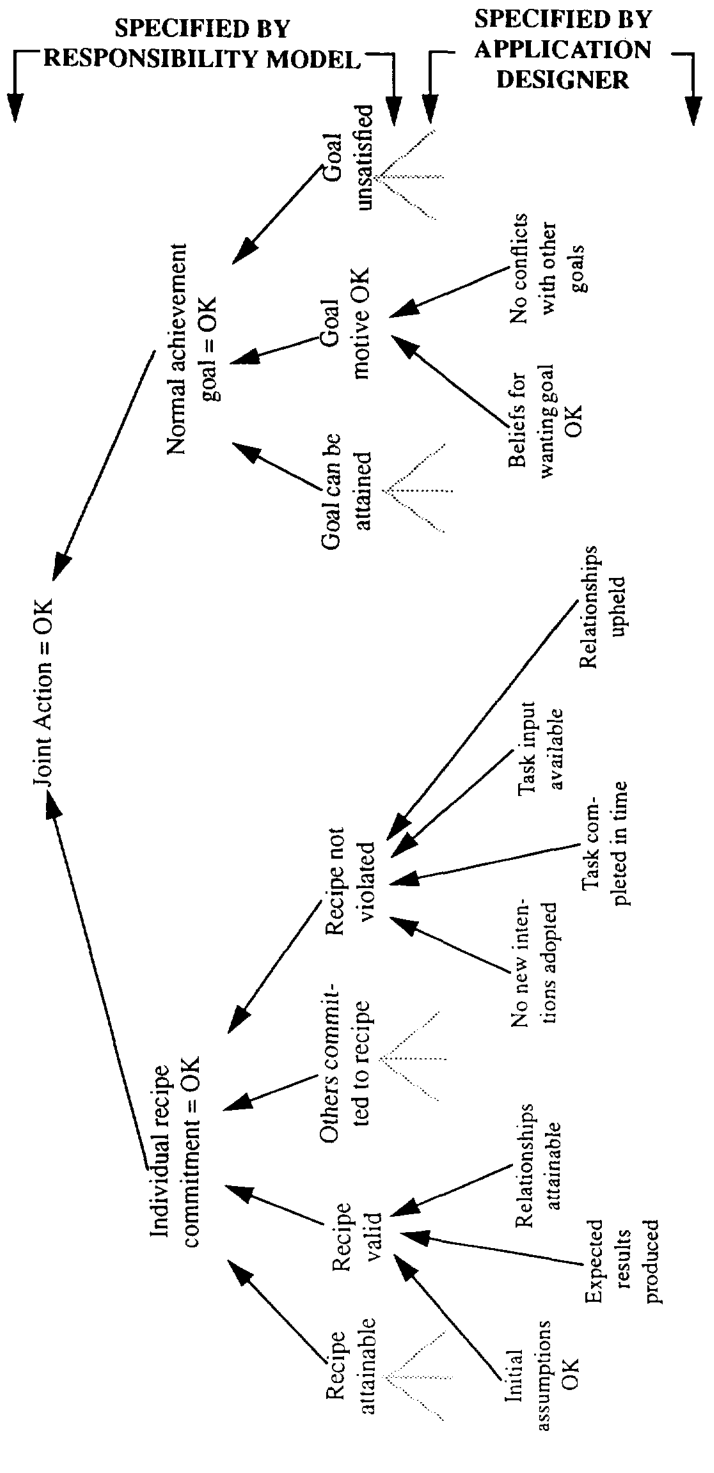
SPECIFIED BY RESPONSIBILITY MODEL
SPECIFIED BY APPLICATION DESIGNER
Joint Action = OK
Normal achievement goal = OK
Individual recipe commitment = OK
Goal unsatisfied
Goal motive OK
Goal can be attained
No conflicts with other goals
Beliefs for wanting goal OK
Recipe not violated
Others committed to recipe
Recipe valid
Recipe attainable
Relationships upheld
Task input available
Task completed in time
No new intentions adopted
Relationships attainable
Expected results produced
Initial assumptions OK

Fig. 9. Responsibility causal network for identifying problematic intentions in electricity transportation application.

which remedial actions, if any, should be taken—this operation is also carried out in an agent's situation assessment module. Available options are represented in the "associated repair action" component and vary from simply rescheduling the existing tasks, through entering a replanning phase, to abandoning the joint action completely. The "select" process takes the intention which is in difficulty, the problem identified by the match process, the list of available repair actions, and determines what the agent should do next. For example, when an agent becomes uncommitted to the common recipe it should suspend processing its associated local actions and enter a replanning phase, whereas if the agent believes the motivation for the joint goal is no longer present then it should abandon all of its associated local actions. Examples of some of the rules which encode the select process for the types of problem used by the exemplar match rules are as follows:

```
R1select :
  if joint goal is satisfied
  then abandon all associated local activities
       inform cooperation module that the joint action
       has successfully finished

R2select :
  if motivation for joint goal is no longer present
  then abandon all associated local activities
       inform cooperation module that the motivation for the
       joint action is no longer present

R3select :
  if common recipe R is violated and
     R can be successfully rescheduled
  then suspend local activities associated with R
       reset the descriptions and timings of R
       inform cooperation module that R has been violated
       and propose new timings

R4select :
  if common recipe R1 is invalid and
     alternative recipe R2 exists
  then abandon local activities associated with R1
       inform cooperation module that R1 is invalid
       and propose R2 as an alternative course of action
```

As well as taking actions locally (such as abandoning, suspending or rescheduling tasks) the Responsibility Model stipulates that the other team members must be informed when an individual's commitment is compromised. This aspect of the model is realised by GRATE*'s cooperation module, based on the information provided by the situation assessment module. In the above cases, the assessment module indicates that the joint action has been satisfied, that the motivation for the joint action is no longer present, that the common recipe has been violated, and that the common recipe is invalid. It is then up to the cooperation module to ensure that the other team members are informed.

```
R1_inform:
  if joint action has successfully finished
  then inform all team members of successful completion
       see if results should be disseminated outside the team

R2_inform:
  if motivation for joint goal G is no longer present
  then inform other members of the team that G needs to be
       abandoned

R3_inform:
  if common recipe has been violated
     and new timings proposal exists
  then inform other team members of violation and also
       propose new timings as a means of fixing the problem

R4_inform:
  if common recipe R1 is invalid
     and alternative proposal R2 exists
  then inform other team members that R1 is invalid
       and offer R2 as an alternative
```

The vast majority of extant DAI systems do not have such an explicit representation of cooperation—their monitoring is *ad hoc* with each and every case having to be discovered and enumerated separately, without the benefits of a clear organising framework. Analysis of the surface level rules depicted in Section 2 reveals that they have a direct link between the matching of environmental conditions (e.g. recipe successfully finished and recipe deadline not met) and associated actions (respectively, inform acquaintance that recipe has finished and inform acquaintance that recipe will not meet deadline). By adopting the Responsibility approach, the many tacit assumptions about the process of cooperative problem solving which underpin these surface level rules were explicitly available to the GRATE* agents. This meant that the agents could base their reasoning on a principled model of cooperation and could therefore respond more flexibly and robustly to the vagaries of complex applications.

5. Experimental results

To assess the utility of the Responsibility Model a series of controlled experiments were undertaken. The objective of these tests was twofold: (i) to verify the predictions made by the formal model; and (ii) to ascertain the computational benefits and drawbacks of Joint Responsibility (particularly with respect to its use in industrial domains). The key performance metrics can be stated by the following set of testable hypotheses:

(1) The Responsibility Model facilitates coherent cooperation between groups of willing agents in complex and dynamic environments.

(2) The Responsibility Model's utility is correspondingly greater in domains in which there is significant scope for foul-ups and inconsistencies during cooperation.
(3) The Responsibility Model can be implemented without compromising the agent's ability to operate in environments where the resources available for coordination are limited.

To provide comparative, as well as quantitative results, three distinct types of cooperating community were constructed: (i) a Responsibility community; (ii) an implicit community; and (iii) a selfish community. The exemplar application was the familiar fault detection and diagnosis in electricity transportation management.

The Responsibility community was represented by the GRATE* implementation which was described in detail in the previous section.

In the implicit community, agents exchange relevant domain level information and cooperation is seen as emerging through their interactions. The agents do not maintain explicit representations of their joint actions, they do not participate in a team planning phase to decide who will do what, nor do they have a principled model of cooperation on which to base their behaviour. This organisation corresponds to a typical GRATE community, it is also similar to several agent architectures based solely on individual intentions [9,11] and to the seminal Distributed Vehicle Monitoring Testbed in which the organisational knowledge guiding the cooperating knowledge sources is specified as a set of "interest areas" [50]. In our implicit community, the agents form individual intentions (related to their own actions) and plan which local tasks to carry out at what times—thus recipes are triggered solely on the basis of events the agent observes or local planning it undertakes. The exchange of relevant information is based upon knowledge contained in acquaintance models, in particular on representations of the domain level data which others are interested in receiving. The acquaintance models used in these experiments were those of the original GRATE multi-agent system—their contents were specified after a lengthy analysis of the application, but before the Responsibility Model existed, and were thought, at the time, to encapsulate the entire information sharing needs of this application.

The final community is composed of *selfish problem solvers*—these are similar to the Responsibility agents in many respects: forming joint goals, tracking their intentions using the Responsibility convention, and agreeing common recipes using the GRATE* distributed planning protocol. Differences only emerge when the joint action runs into difficulty or when it is successfully completed. As before, the agent(s) which detects the problem stops its associated local processing immediately and drops commitment to any subsequent activities related to the joint action; however this agent does *not* inform its fellow team members of its lack of commitment. This behaviour is selfish because the agent which notices the problem and realises that the joint action is doomed to failure does not wish to expend further resources informing others, since doing so brings it no direct benefit with respect to that social action. This community has been included because a number of authors have suggested that self-interest is the basis of robust cooperation [4,26]—our selfish problem solvers represent agents which are purely driven by self-interest, they have no consideration for their fellow team members.

A number of different types of experiment were carried out. Firstly, the behaviour of the three different communities was analysed to identify the main phenomena exhibited during joint action unsustainability and also to analyse the levels of coherence attained in correspondingly more complex and dynamic environments (Section 5.1). Secondly, an assessment was made of the effects of varying the time taken for inter-agent communication on the community's behaviour (in both successful and unsuccessful collaboration) (Section 5.2). Finally, the effect of limiting an agent's reasoning about coordination was examined in order to gain an insight into the resource overheads required to sustain each community (Section 5.3).

In all of these experiments, each agent was given an equal priority (time slice) and could carry out identical amounts of reasoning about the process of coordination. Coherence is defined quantitatively in terms of the cooperating community's wasted effort—*the sum, over all team members, of the number of processing cycles from when the event causing the problem occurs to the time when an agent stops performing its associated domain level actions.* The higher this sum the more incoherently the community is behaving. So, for example, if the CSI detects a problem at time 10, stops its associated processing at time 12, and then informs the AAA and the BAI who respectively stop their associated processing at times 13 and 14 then the community's wasted effort is 9 $(2 + 3 + 4)$. The figure does *not* include the effort spent before the joint action ran into difficulty, up to this point it was perfectly rational to carry out the activity and hence the community was performing coherently. Unless stated to the contrary, agents had infinite processing power to carry out their reasoning about coordination and the delay in inter-agent communication was one processing cycle.

5.1. Coherence in complex dynamic environments

5.1.1. Coherent and incoherent responses to problems

In this set of experiments, two predominant patterns of behaviour occurred:

(i) the implicit community behaved as coherently as the Responsibility one because the agents exchanged the necessary information which allowed some, or all, of the team members to deduce that the joint action should be abandoned;

(ii) the implicit community behaved as incoherently as the selfish one because the agents did not exchange sufficient information for the team members to deduce that the joint action was in difficulty.

In all cases the Responsibility community behaved more coherently than the selfish one.

An illustration of the former case occurs when the CSI realises that the network fault which the group is working on is only transient or never existed at all. As Fig. 10 shows, both the Responsibility and the implicit groups wasted very little effort (always less than 20 units), whereas the selfish community wasted up to 85 units if the problem occurs near the beginning of the joint action (elapsed time 0).

In the Responsibility community it is the CSI's domain level system which actually detects the fault's non-permanence. This information is passed up to the situation assessment module (via the control module) whereupon $R2_{\text{match}}$ (Section 4.3) is fired and the motivation for the joint goal is noted as invalid. An invalid motivation means the CSI no longer has a normal achievement goal to complete the diagnosis and hence the joint action should be terminated. The existence of a weak achievement goal related to the diagnosis activity ensures that the CSI informs the other two agents that it is no longer

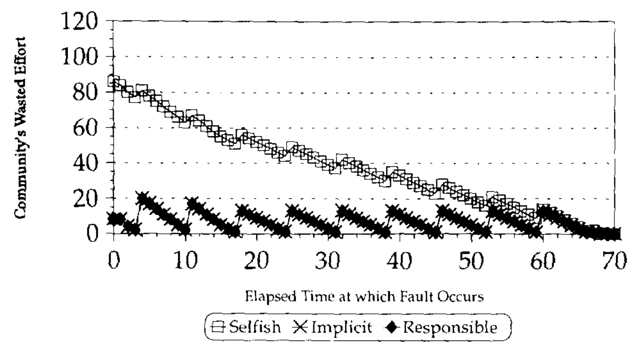

Fig. 10. Goal motivation lacking.

committed and the reason for this change of state. The resulting level of coherence exhibits a periodic pattern because the CSI can only detect that the fault is transient at the end of its network monitoring task—an activity taking 7 units of time and which is repeated continuously until the list of faulty elements is available. Whenever this unexpected event occurs, the AAA wastes more effort than the CSI because it cannot identify the problem for itself—it relies on being informed (indirect sensing). Therefore this difference is directly proportional to the time taken for the message indicating that the fault was non-permanent to be sent from the CSI to the AAA. The BAI's role in the joint action is comparatively small as it is only active for the first 15 time units. If a problem occurs in this time, the BAI also relies on being informed by the CSI. This means the community's level of incoherence is greatest if the problem occurs near the beginning of the joint action—once the BAI is not actively involved the wasted effort levels off.

The implicit community behaves identically to the Responsibility one, from an external perspective, because the CSI represents the fact that the other agents are interested in knowing about a change in the electricity network's state. This interest in state change was included in the CSI's models for two reasons. Firstly, because it is what triggers the diagnosis activities of the AAA and the BAI in the first place. Secondly, because previous experience had highlighted that the CSI's disturbance detection action was prone to inaccuracies. Therefore if the CSI comes to believe that there is not a permanent fault after all, network state change from faulty to normal, this fact is sent to the AAA and the BAI as unrequested information. Within their local individual intention representations, the AAA and the BAI store the reason why they are carrying out their diagnosis activities (i.e. because there is a fault) and so receipt of the CSI's message undermines this motivation. As the AAA and the BAI are rational agents they stop performing their related diagnosis activities.

The selfish community behaves differently from the other two. When the CSI realises the fault is transient it abandons its associated local processing, as before, since it is no longer committed to the joint goal. However as the CSI is selfish, it will not expend additional resources on the joint activity and so it deliberately avoids informing the others that there is a problem. As a consequence, the pattern of coherence is markedly different; the effort wasted by the AAA and the BAI is linear, rather than periodic,

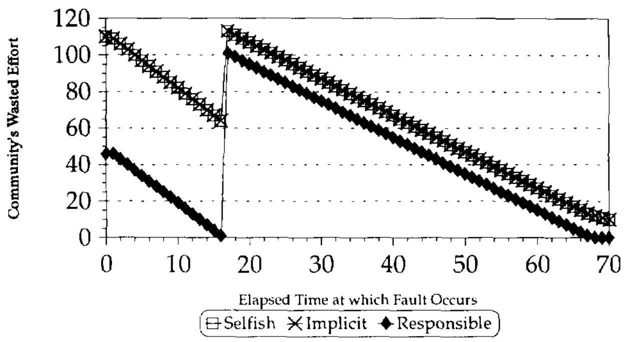

Fig. 11. Goal unattainable.

because they never realise the fault was transient. Both agents just continue with their diagnosis activities until they finish. Therefore problems with the joint action which occur near the beginning have a significantly greater impact on the community's level of coherence than those which occur near the end (the small peaks in the line are due to the periodic nature of the CSI's wasted effort).

An illustration of a case in which the implicit community behaves as incoherently as the selfish one occurs when the joint goal becomes unattainable (Fig. 11). This situation arises if the AAA realises that too much of the data which it needs to perform its diagnosis is missing. As the AAA is the only community member capable of pinpointing the element at fault the joint goal cannot be attained. This realisation may occur either at the end of the AAA's preliminary or detailed diagnosis phase.

In the Responsibility community, the AAA's domain level system indicates that the goal of diagnosing the network fault cannot be met. This information is passed, via the agent's control module, to its situation assessment module where it results in the deduction that the joint goal is unattainable—$R4_{match}$ (Section 4.3) is not fired since there are no alternative recipes available. As the AAA believes that the joint goal is unattainable, it no longer has a normal achievement goal to diagnose faults; however a weak achievement goal persists and so the AAA informs the other two of its change of state. Even with the Responsibility convention in operation, there can still be a substantial amount of wasted effort (100 units in the worst case) due to the time lag in detecting that there is a problem. This phenomena is exacerbated if the realisation occurs in the AAA's very long detailed diagnosis phase. The amount of effort wasted by the CSI is greater than that of the AAA because it relies on being informed of the problem.

In the implicit community, the external behaviour of the AAA is the same as with the Responsibility community, it detects that there is a problem and stops its associated processing. However the AAA does not inform the other two agents which continue with their activities until they terminate, oblivious to the fact that they are pointless. The information that the BAI and the CSI are interested in knowing if the AAA abandons its diagnosis is not contained in the AAA's acquaintance models because the need for this interaction was not uncovered by the initial analysis of the application; also because there is no representation of the joint action the AAA is unaware that its activities are related to those of the BAI and the CSI. The behaviour of the selfish problem solvers is identical to that of the implicit group—in neither case is information exchanged related to the joint goal's unattainability.

All the other types of problem specified by the Responsibility convention result in basically the same two kinds of behaviour—with the implicit community either mirroring the coherence of the Responsibility community (if the relevant information is contained in the acquaintance models and the receiving agents can make the appropriate local deductions), or the coherence of the selfish community (if the necessary information is not contained in the acquaintance models). In some cases there was a middle ground in which the implicit community performed less well than the Responsibility one but better than the selfish one. This happened, for instance, when the common recipe became invalid—if a critical or substantial change occurred in the network, while the AAA and the BAI were performing their diagnosis, the common recipe did not produce the desired (correct) outcome because the answers were based on invalid assumptions. In the Responsibility community, the CSI detects that a substantial change has occurred which means the initial assumptions of the AAA and the BAI are no longer valid. Invalid initial assumptions mean that the common recipe may no longer produce the desired result (Fig. 9) and so the CSI no longer has individual recipe commitment to the agreed course of action. Although its commitment to the common recipe has been dropped, the CSI still has joint recipe commitment which ensures that it informs the other two that it is no longer committed to the recipe. The implicit community behaves worse than the Responsibility one, even though the information that the network topology has substantially changed is sent to both the AAA and the BAI as unrequested information. The CSI's acquaintance model represents the fact that the other two agents are interested in this information because it is a trigger for updating their network model. In the AAA, this recipe has the highest priority and so it displaces the diagnosis activity—its first act is to kill all tasks which are making use of the outdated network model, meaning the AAA's diagnosis activity is abandoned. However in the BAI, the diagnosis continues because the update recipe has a lower priority than producing the black out area. Thus the AAA and the CSI behave the same in this community as they do in the Responsibility one (for different reasons though), but the BAI performs less coherently.

The purpose of this group of experiments is to identify the types of interactions which can happen in the three different types of community—the results that the joint goal becoming unattainable causes greater incoherence than when the goal motivation is lacking is not generalisable, nor is the information that the implicit group performs coherently in the former case but less well in the latter. What these results do show is that neither the implicit nor the selfish organisations can produce behaviour which is more coherent than that obtained in the Responsibility community; however in certain cases the implicit community is able to match the performance of the Responsibility one. The experiments also show that, in the majority of cases, when problems occur with the joint action they can only be detected by a subset of the group—the others rely on being informed.

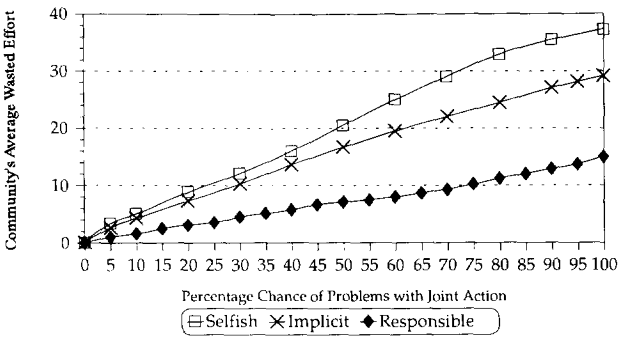

Fig. 12. Varying chance of joint action failure.

5.1.2. *Overall trends*

Having examined the performance of the three communities in terms of the individual types of problem which can cause the joint action to falter, the next step is to assess the typical benefits of the Responsibility Model over a broad range of circumstances. To achieve the necessary diversity a number of environments were tested—ranging from those in which it is guaranteed that the joint action will not falter (0% chance of problem), to those in which a problem is assured (100% chance of problem). An increased chance of unsustainability corresponds to a more dynamic environment or one in which decisions are based on less stable data. In all cases, the cause of the problem is uniformly distributed over the conditions identified by the Responsibility Model, the time at which the problem occurs is uniformly spread over the joint action's duration, and the community's average wasted effort is obtained by repeating the runs 100 times.

Fig. 12 shows that the average wasted effort for the Responsibility community is significantly less than the other two confirming the prediction that the Responsibility Model facilitates robust cooperative behaviour even in the harshest environments. The implicit community performs better than the selfish one because agents exchange information which can lead to the recipient realising that some of its intended actions should be abandoned. Thus an agent which cannot detect a problem with the joint action for itself may inadvertently be supplied with sufficient information to enable it to drop its commitments by an agent which can. In the selfish group, such communication was deliberately eschewed since calculating agents interested in a piece of information requires resources—in this community agents which were unable to detect problems for themselves were simply left to needlessly complete their actions. To show the statistical significance of these results, linear regression lines were computed for each of the three cases—this analysis showed that the slopes are significantly different (probability of such a difference occurring by chance alone is less than 0.005).

These results also show that pure self-interest is *not* a good basis for cooperation; it may be appropriate for deciding whether to participate in a particular activity, but it should not be used to define agent behaviour once the social action has started. Participation in cooperative problem solving requires some element of compromise—self-interest needs to be tempered with consideration for the group as a whole.

5.1.3. Repairing problems with joint actions

Up to now, the experiments have concentrated on detecting problems with the joint action and informing other team members so that the community responds in a coherent manner to unexpected events. However when problems occur with the common recipe, the Responsibility Model stipulates that a new course of action should be devised. This functionality, and its effect on the coherence of the cooperating group, is examined by this set of experiments. The particular problem which is explored is that of common recipe violation. One of the most frequent causes of recipe violation is that unexpected events occur during the course of the joint action which distract the agent from its intended commitments. These distractions may mean that the specified relationships are not upheld, which in turn means that the agent no longer has the necessary individual recipe commitment (see Fig. 9).

In order to simulate recipe violation, varying numbers of unexpected new tasks were assigned to the agent community during the lifetime of the joint action. These additional tasks could all be solved by an individual agent working in isolation and were all of a relatively short duration. Tasks arrived and were spread over the community's members according to a uniform distribution. Task priority was randomised, using the value of the joint action components as a mean, and the results were averaged over 100 test runs. If a domain level task was running when the coordination mechanism decided that it should be rescheduled, it had to be suspended before any new activity could start. When such tasks were restarted, they carried on from the point at which they left off.

In the Responsibility community, if an agent delays an action which is being performed in the context of a joint goal then $R3_{match}$ (Section 4.3) is fired and the agent no longer has the necessary individual recipe commitment. However because of its joint recipe commitment, the agent in question informs its fellow team members of its non-commitment and of the revised deadline ($R4_{inform}$—Section 4.3). When such a message arrives, the recipient evaluates whether the change will affect its processing. If the delay is relevant, the agent reschedules its local activities ensuring that all other action inter-relationships are upheld. The agent then evaluates whether any actions can be moved into the vacuum which has been created so that it does not lie idle during this period. It attempts to bring as many tasks forward as possible—they are selected in priority order until no more tasks will fit into the remaining time. To minimise the number of action inter-relationships which are broken it is predominantly local tasks which are moved. If such rescheduling is impossible, the agent remains idle until its next intention becomes active.

In the implicit organisation, when agents delay activities they do not inform other community members, since it only reflects an internal state change which it is assumed that acquaintances are not interested in. The agent where the delay occurs, undertakes the type of rescheduling described above, but its acquaintances are unable to do so because they are unaware of the changes. This meant that agents were sometimes completely unproductive for substantial lengths of time because they were waiting for synchronisation events to occur, unaware that the relevant task had been put back.

Fig. 13 highlights the superior performance of the Responsibility Model along two important directions. Firstly, the average effort expended on the joint action rises more

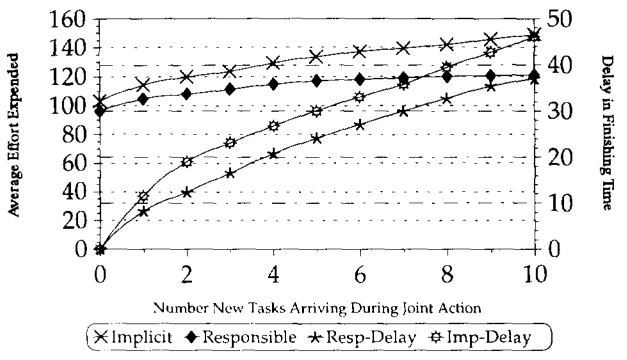

Fig. 13. Effect of recipe violation.

steeply for the implicit community. As the effort to actually achieve the diagnosis is identical in both communities any disparity is solely a function of the amount of time agents needlessly waste waiting for delayed synchronisation events to occur. As the graph shows, the Responsibility community remains virtually impervious to delays and rescheduling activities—this is because the agents send the necessary information to the appropriate agents in a timely manner. As the number of disruptions become larger, the community does expend slightly more effort—this increase in wastage occurs as a consequence of the delay in inter-agent communication and the time the recipients take to respond to the warning messages. In the implicit community, on the other hand, the joint action is more costly because team members are spending significant amounts of time in an unprofitable manner. This time increases linearly as the probability and number of disruptions becomes larger.

The second measure is that of delay in completion time—calculated from the time at which the joint action would have finished if there were no interruptions (time spent executing additional tasks is subtracted from this figure so that the comparisons remain valid). It can be seen that the delay in finishing the joint action rises less steeply and to a lower value for the Responsibility organisation. Therefore by indicating when delays have (or will) occur, team members are able to make better use of their resources. Rather than simply idling until synchronisation events occur, Responsibility agents are able to intertwine other activities into spare moments when they know acquaintances have delayed actions. Again agents in the implicit community are unaware of such delays and so suffer prolonged periods of inactivity.

5.2. Effect of increasing communication delay

This set of experiments evaluate the impact on the cooperating community of varying the communication delay for inter-agent message passing. Both successful and unsuccessful joint actions are considered. In the former case, two organisational forms are compared: the Responsibility community and the implicit community.[7] In the latter

[7] The selfish community was not considered because it is identical to the Responsibility one for successful joint problem solving.

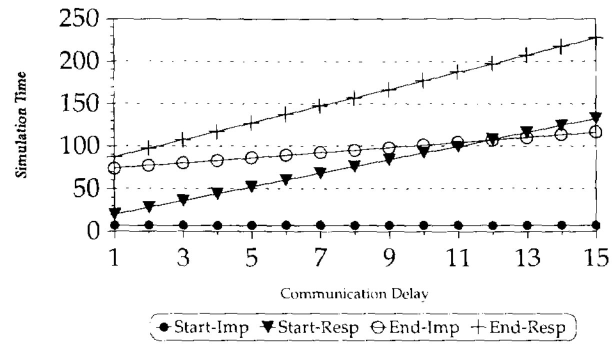

Fig. 14. Effect of communication delay on successful joint actions.

case, only the Responsibility community is considered and the representative problem is that of goal motivation lacking.

In the Responsibility community the start time is agreed by all the participants, hence it is the same for each team member (Fig. 14). As such agreements are deliberately avoided by the implicit community, individuals start activities which are part of the joint action at different times—the time shown in this case is for the first agent to start processing. It can be seen that the Responsibility community always takes longer to start its activities, despite the fact that the events which initiate the diagnosis process occur at precisely the same time in both cases. The reason for this disparity is that the Responsibility community has to construct the cooperating group and the common recipe at run time. This process needs communication and is, therefore, more adversely affected by increased communication delays (indicated by the sharply increasing start time). In the implicit community, on the other hand, much of the coordination is carried out by the application builder at design time—thus the runtime communication overhead is reduced.

The Responsibility Model also requires that there be an explicit wrapping up phase when all team members acknowledge that the joint action has terminated—this needs message interchange and so is affected by the communication delay. In contrast, the implicit community is adjudged to have finished when all agents have stopped their associated processing—no explicit finalisation is needed.

Fig. 15 shows the effect on coherence of varying communication delays in the Responsibility community when something goes awry with the joint action. The exemplar problem is that of goal motivation lacking (GML) which was discussed more thoroughly in Section 5.1.1. The figures in the legend indicate the number of processing cycles for message interchange to occur. This graph shows that as the communication time increases, so the amount of wasted effort becomes larger. This occurs because agents take longer to be informed about problems detected by their fellow team members. This phenomena is especially noticeable near the beginning of the joint action when all three agents are active; as the activity progresses so the number of active agents diminishes and hence the effect of the delays are reduced.

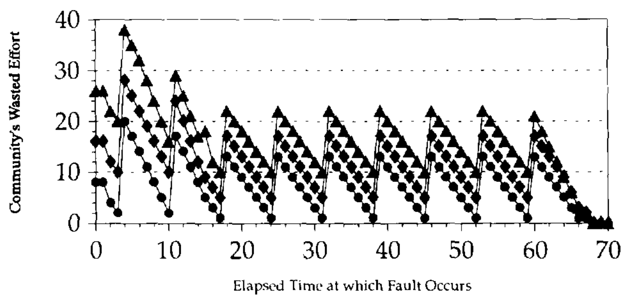

Fig. 15. Effect of communication delay on unsuccessful joint actions.

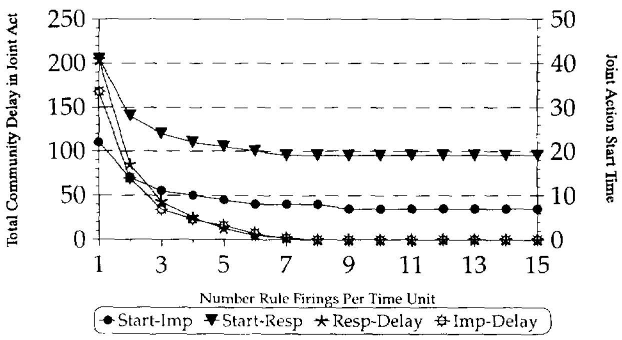

Fig. 16. Effect of limited processing power on level of coherence.

5.3. *Effect of varying processing power available for coordination*

The final set of experiments examine the effects on Responsibility and implicit agents of having to achieve coordinated activity with varying amounts of computational resource at their disposal. This metric is important because in realistic scenarios the resources required by the coordination mechanism should be significantly less than those needed by the domain level system. Resource boundedness was simulated by setting a limit on the number of rules which could be fired by the agent's coordination mechanism on each processing cycle—it was varied from one rule per agent per cycle to an infinite number (which in this experiment turned out to be 15). When an agent's limit was reached, no more rules were fired.

As Fig. 16 shows, only when the processing power is reduced to 8 rules per cycle does it cause the joint action to be delayed for either organisational form. Between 5 and 8 rules per cycle the delay is approximately the same for both communities. However in severely resource bounded situations (less than 5 rules per cycle) the Responsibility community is delayed by up to 23% more than the implicit one—indicating that the former requires more processing resource to sustain it than the latter. The graph also shows that the differences in starting times for the two communities remains virtually constant except for very resource-limited situations, again less than 5 rules per cycle, where it increases by 33% for the Responsibility organisation. These two pieces of information indicate, somewhat surprisingly, that the Responsibility Model does not require substantially more processing power—either to monitor joint action once it has started or to initiate it—than in the first place.

5.4. *Discussion of results*

The aim of this empirical evaluation was twofold: to provide an insight into the computational behaviour of a community of cooperating agents which use the model of Joint Responsibility to guide their actions and to provide a means of verifying the predictions made by the formal model in a realistic scenario. The precise values which appear in the graphs are less important than the relative trends that they indicate. Thus, for example, it cannot be concluded that dropping commitment because the joint goal is no longer attainable will lead to greater incoherence than if the reason for dropping commitment is that the motivation for the joint goal is no longer present. In some cases this will undoubtedly be true, however in another application the situation may be reversed, whilst in a third application there may be no major difference between the two.

The results which are generalisable to other applications and to larger or smaller cooperating groups can be summarised as follows. Firstly, the Responsibility community maintains coherent cooperation even in the most hostile environments. In particular cases, other organisational forms can attain similar levels of coherence, but, on average, they will perform less well than the Responsibility community. Secondly, as well as detecting irreparable problems at an early stage, the Responsibility Model ensures that problems with joint actions can be overcome. It also ensures that during this repair process, all team members are kept informed of the situation so that they can continue to carry out profitable problem solving on other activities. Thirdly, except in situations in which the amount of processing available for the coordination mechanism is severely limited, the implementation of the Responsibility Model requires no more computational resources than other typical organisational structures.

As well as highlighting the benefits of the Responsibility Model, these experiments also indicate a potential drawback. The GRATE* distributed planning protocol is heavy on communication and it takes a significant amount of time to establish a joint action (especially as message communication time increases). Therefore it could be argued that the model is inappropriate for cases in which agents are situated in time-critical environments such as industrial applications. However, this characteristic is not an inherent property of the Responsibility formulation, rather it is indicative of the planning protocol used to implement the model. The GRATE* protocol built up all joint actions from scratch, making no use of knowledge about previous events or interactions—the group, the actions to be performed, the agent which will perform them, and their timings, all had to be agreed upon at run time. However as construction of the implicit community indicated, it is possible for the designer to make use of prior knowledge about the agents and their interactions to considerably shorten this process. Utilising such knowledge would bring about significant reductions in the communication overheads and the time needed to establish joint actions. Although there is scope for enhancements, the level of communication could never realistically be reduced to that of the implicit group.

When deciding whether to employ the Responsibility Model to control cooperative problem solving the system designer is faced with a tradeoff between the ability to ensure robust behaviour even in the most hostile environments and the overhead associated with constructing the group and agreeing a common recipe. As there is no universally best choice, this decision is ultimately dependent on the problem being tackled; if joint actions rarely run into difficulties then the overheads associated with the Responsibility Model may prove prohibitive. However if the application involves dynamic change, unpredictable events, and decision making based on incomplete information, then the model of Joint Responsibility is a good choice.

6. Conclusions

There were two key constraints on this research which distinguished it from previous formal work on cooperative problem solving. Firstly, as it was borne out of the need to solve practical problems the model of Joint Responsibility had to have a clear path to implementation level systems. The importance of this link has been highlighted by several researchers, some even going so far as to state that AI will not advance as a science until the gap between those who construct models and those who build systems is closed [16]. The formal model of Responsibility aids the application (system) designer by: offering a structured framework for knowledge elicitation, providing a domain-independent characterisation of the types of events which can cause problems during cooperative problem solving, specifying the key mental states of the collaborating agents, and defining how agents should behave in nominal and exceptional circumstances both with respect to their local activities and with respect to their fellow team members. By basing the implemented system on a firm footing it was easier to: predict the range of agent responses, verify that the agent had been implemented correctly, and provide a clear boundary on the types of situation in which the agent could be expected to successfully operate.

The second distinguishing characteristic was that this work was targeted specifically at the problem of ensuring robust, coherent behaviour in dynamic and unpredictable environments. The electricity transportation scenario used throughout this work provides a typical example of such an environment: there is a relatively high loading of tasks for each agent, new activities emerge at unpredictable times, interactions are prolonged, and decisions are based on uncertain and incomplete information. Addressing domains with these characteristics is especially important for DAI as they are typical of the situations in which conventional technologies have difficulties—they therefore represent a niche in which DAI could gain a foothold [40]. From a more theoretical perspective, the importance of tackling this type of problem was highlighted by Gasser—he observed that a theory of DAI ought to account for how aggregates of agents can achieve joint actions that are robust and continuable despite intermediate foul-ups and inconsistencies [28]. As the empirical evaluations of Section 5 highlighted, Joint Responsibility offers a step towards this objective—providing procedures for controlling activities in dynamic and unpredictable environments, whilst retaining a degree of generality and predictability.

The philosophy of providing more explicit representations of social phenomena, which was so evident in the Responsibility Model, also surfaces in other recent DAI work. The most prevalent illustration is with respect to communication—speech act theory [2,63], in which the effect of utterances are explicitly represented and can be reasoned about by the sender in order to try and induce specific mental states in the receiver, has been used as the basis of a number of multi-agent systems [53,66]. Other illustrations occur in conflict resolution [46,47] in which resolution strategies are categorised and selected according to the desired objective and the prevailing circumstances; in persuasion and negotiation [13,73] in which agents reason about how to induce greater cooperativeness in other community members; in the definition of hedonic states, likes, goals and values based on physical dynamics [45]; and in game-theoretic approaches to the selection of individual actions based on notions of utility function evaluation [78].

Being explicit about the knowledge which is brought to bear during problem solving and clearly distinguishing it from representational issues is hardly a new idea. Second-generation expert systems, for example, have explicit models of their inference structure [15], their problem solving method [56], and the generic task they are tackling [14]. This analogy is particularly pertinent because, at the time of writing, DAI systems are suffering from many of the problems which beset first-generation expert systems, including: difficulties operating in dynamic environments, weak explanation of (social) problem solving activity, poor software reusability, and lack of a structured framework within which development can proceed. These failings appear to be symptomatic of a more fundamental problem—namely, the dearth of high-level principles about cooperative problem solving as a process. Therefore just as recognition of the existence of a knowledge level [58] allowed high-level principles of the individual problem solving process to be identified, it is believed that a similar set of principles exist, and should be exploited, for multi-agent problem solving. This development process will be enhanced by separating out issues of cooperation from those of individual problem solving and representing them in a distinct computer level. This *cooperation knowledge level* [36] will concentrate on developing principles and explicit models of various social phenomena (such as cooperation, coordination, hostility, competition and conflicts) as well as the reasoning processes which control them. In Newell's taxonomy of computer system levels, the cooperation level will be directly above the knowledge level. Like the others, the cooperation knowledge level can be reduced to the level directly below it—ultimately being expressed in terms of single agents and individual goals, actions, and knowledge states.

The benefits of cooperation knowledge level systems are envisaged to be similar to those experienced when transferring from first to second-generation expert systems. They include:

(i) better explanations of the objectives and means of social interactions;
(ii) greater system generality and flexibility in terms of the mechanisms and knowledge brought to bear during joint problem solving; and
(iii) a more structured knowledge acquisition process.

Explanation will be enhanced because the group's activities can be described at a high level of abstraction. For instance, it will be possible to provide a more meaningful characterisation of a protracted series of message exchanges if there is an explicit representation of the social activity being undertaken. So with cooperation knowledge level systems it would be possible to state that agent A_1 is trying to persuade A_2 to do X, or that A_1 and A_2 are engaged in a conflict about Y which they are trying to resolve by Z (rather than merely indicating that message M_1 has been sent from A_1 to A_2, A_2 has responded with M_2, and so on). These advances are especially important in situations in which the user plays an active problem solving role [3,53].

System generality will be improved by clearly distinguishing between the domain-independent principles on which behaviour is based (e.g. the reasons for re-assessment and the associated repair action components of Fig. 8) and the domain-dependent knowledge which is used as data during the reasoning process (e.g. the match and select processes of Fig. 8). The generic component, which represents the cooperation knowledge level description, can be isolated and coded as an off the shelf software system. It can be applied to many different problems merely by substituting in the appropriate domain knowledge. The feasibility of this approach has been demonstrated by the development of GRATE and GRATE* which embody the high-level domain-independent principles in their generic rules and provide appropriate facilities for incorporating domain specific knowledge where it is needed.

Finally, the multi-agent system developer will be aided by having a focused set of questions, strategies, and options with which to confront the organisation who commissioned the system. This framework allows development to proceed more rapidly because its inherent structure means much of the groundwork has already been carried out. It also helps avoid omissions by providing a comprehensive view of the problem being tackled. Refer to the discussion in Section 4.3 for a specific illustration of this idea with respect to the Responsibility Model.

The cooperation knowledge level differs from the individual knowledge level in at least two important ways: in terms of the *system* and the *laws which govern component behaviour*. Firstly, the cooperation level has societies as the system, whereas the single agent knowledge level has individuals. Thus formalisms are needed to describe the actions of collectives, as well those of the individuals—Joint Responsibility allows collectives to be represented by means of social actions and GRATE* translates this into joint intention representations. Since it is individuals who ultimately have the ability to act, work must be done on the means of mapping descriptions of collective actions into those of the individuals and of combining related individual acts into a single supra-activity.

The second major difference between the individual and cooperation knowledge levels is in the laws which determine how system behaviour depends upon component behaviour and the structure of the system. Within the knowledge level, the behavioural law is that of rationality: if an agent knows an action will lead to the satisfaction of one of its goals, it will select that action [58]. However Joint Responsibility stipulates that even after an agent has discovered that there will be no personal benefit in pursuing the social activity, it must endeavour to inform the other team members of this fact. As this activity requires computational resource, which the agent could be spending on potentially beneficial activities, it is an irrational act when viewed from the position of the individual agent. Therefore notions such as individual rationality or self-interest, even when tempered with notions such as "shadow of the future" [4], are insufficient for defining the behaviour of participants engaged in cooperative problem solving. What is required is *team rationality*, an embodiment of intuitive notions such as "cooperativeness", "team spirit" and "being a good team member."

In addition to extending this preliminary work on the cooperation knowledge level, there are two other avenues in which further research is still needed. Firstly, although Joint Responsibility has a wider scope than the majority of the extant formal cooperation models, it concentrates predominantly on the execution phase of cooperative problem solving. Thus there is a need to formalise the remaining aspects of the cooperation process, issues which still need to be addressed include:

(i) the process by which the need (desirability) for cooperative problem solving is ascertained;

(ii) the process by which a team of cooperating agents is assembled—including deciding what organisational form the team will take (will there be a single controller, a controlling committee or will all members be equal? and will decisions require unanimous or majority support?), determining who should be in the team (is it best to have small teams with each member doing significant amounts of processing or larger teams with less active members?) and deciding how to recruit community members to the team (will individuals join out of benevolence or will they need convincing?, if so how?); and

(iii) the process by which the common recipe is developed.

The second avenue of further investigation is to narrow the gap between formal and computational models of cooperative behaviour. In mapping from Joint Responsibility to GRATE*, the key behavioural definitions had to be taken from a modal logic and manually turned into production rules. This transformation should be made more straightforward and should ideally be automatic. Two good illustrations of how this objective can be achieved are Shoham's agent-oriented programming paradigm [66] and Rosenschein and Kaebling's work on analysing the knowledge states of situated automata [62]. The former has an "agentification" process which takes high-level programming language primitives and converts them into low-level executable objects. The latter uses epistemic temporal logic to specify what a designer would have a machine "know". This intentional description is then compiled down into a gate level description of a digital machine which satisfies the properties expressed in the intentional description.

Acknowledgments

The work described in this paper has been partially supported by the ESPRIT II project ARCHON (P-2256) whose partners are: Atlas Elektronik, Framentec-Cognitech, Labein, QMW, Iberdrola, EA Technology, Amber, University of Athens, Univ. of Amsterdam, CAP-Volmac, CERN and University of Porto. In particular Inaki Laresgoiti, Juan Perez, Jose Corera and David Cockburn helped me to understand the electricity transportation domain; Rob Aarnts, Paul Skarek, Joachim Fuchs and Laszlo Varga provided valuable feedback by applying GRATE at CERN; John Bigham helped with the statistical analysis of my results; Abe Mamdani, Thies Wittig and Erick Gaussens gave general guidance on system design issues. Finally, I would like to thank the anonymous reviewers for their insights and constructive comments.

References

[1] R.P. Aarnts, J. Corera, J. Perez, D. Gureghian and N.R. Jennings, Examples of cooperative situations and their implementation, *J. Softw. Res.* **3** (4) (1991) 74-81.

[2] J.L. Austin, *How to do Things with Words* (Harvard University Press, Cambridge, MA, 1962).

[3] N.M. Avouris, M.H. van Liedekerke, G.P. Lekkas and L.E. Hall, User interface design for cooperating agents in industrial process supervision and control applications, *Int. J. Man Mach. Stud.* **38** (5) (1993) 873-890.

[4] R. Axelrod, *The Evolution of Cooperation* (Basic Books, New York, 1984).

[5] B.I. Blum, The fourth decade of software engineering, *J. Intell. Cooperative Inf. Syst.* **1** (3-4) (1992) 475-514.

[6] A. Bond and L. Gasser, eds., *Readings in Distributed AI* (Morgan Kaufmann, San Mateo, CA, 1988).

[7] M.E. Bratman, Two faces of intention, *Philos. Rev.* **93** (1984) 375-405.

[8] M.E. Bratman, *Intention Plans and Practical Reason* (Harvard Univ. Press, Cambridge, MA, 1987).

[9] M.E. Bratman, D.J. Israel and M.E. Pollack, Plans and resource bounded practical reasoning, *Comput. Intell.* **4** (1988) 349-355.

[10] M.L. Brodie, The promise of distributed computing and the challenge of legacy information systems, in: *Proceedings IFIP Conference on Semantics of Interoperable Database Systems*, Lorne, Australia (1993).

[11] B. Burmeister and K. Sundermeyer, Cooperative problem solving guided by intentions and perception, in: E. Werner and Y. Demazeau, eds., *Decentralised AI* 3 (Elsevier Science, Amsterdam, 1992) 77-92.

[12] S. Cammarata, D. McArthur and R. Steeb, Strategies of cooperation in distributed problem solving, in: *Proceedings IJCAI-83*, Karlsruhe, Germany (1983) 767-770.

[13] C. Castelfranchi, Social power: a point missed in multi-agent, DAI and HCI, in: Y. Demazeau and J.P. Muller, eds., *Decentralised AI* (Elsevier Science, Amsterdam, 1990) 49-62.

[14] B. Chandrasekaran, Generic tasks in knowledge based reasoning: high level building blocks for expert system design, *IEEE Expert* **1** (3) (1983) 23-30.

[15] W.J. Clancy, Heuristic classification, *Artif. Intell.* **27** (1985) 289-350.

[16] P.R. Cohen, A survey of the Eighth National Conference on Artificial Intelligence: pulling together or pulling apart, *AI Mag.* **12** (1) (1991) 16-41.

[17] P.R. Cohen and H.J. Levesque, Intention is choice with commitment, *Artif. Intell.* **42** (1990) 213-261.

[18] P.R. Cohen and H.J. Levesque, Teamwork, *Nous* **25** (1991) 487-512.

[19] R. Conte, M. Miceli and C. Castelfranchi, Limits and levels of cooperation: disentangling various types of prosocial interaction, in: *Proceedings Modelling an Autonomous Agent in a Multi-Agent World*, Saint-Quentin en Yveslines, France (1990) 152-166.

[20] J. Corera, I. Laresgoiti, D. Cockburn and A. Cross, A cooperative approach towards the solution of complex decision problems in energy management and electricity networks, in: *Proceedings 12th International Conference on Electricity Distribution*, Birmingham, England (1993) 4.19.1-4.19.6.

[21] D.D. Corkill, Hierarchical planning in a distributed environment, in: *Proceedings Sixth IJCAI-79*, Cambridge, MA (1979) 168-175.

[22] B.J. Cox, Planning the software industrial revolution, *IEEE Softw.* (November 1990) 25-33.

[23] J. de Kleer, How circuits work, *Artif. Intell.* **24** (1-3) (1984) 205-281.

[24] D.C. Dennett, *The Intentional Stance* (MIT Press, Cambridge, MA, 1987).

[25] E.H. Durfee, *Coordination of Distributed Problem Solvers* (Kluwer Academic Press, Boston, MA, 1988).

[26] E.H. Durfee, V.R. Lesser and D.D. Corkill, Coherent cooperation among communicating problem solvers, *IEEE Trans. Comput.* **36** (1988) 1275-1291.

[27] J. Galbraith, *Designing Complex Organizations* (Addison-Wesley, Reading, MA, 1973).

[28] L. Gasser, Social conceptions of knowledge and action: DAI foundations and open system semantics, *Artif. Intell.* **47** (1991) 107–138.

[29] L. Gasser and M.N. Huhns, eds., *Distributed AI* **II** (Pitman, London, 1989).

[30] M.P. Georgeff, A theory of action for multi-agent planning, in: *Proceedings AAAI-84*, Austin, TX (1984) 125–129.

[31] R.V. Guha and D.B. Lenat, CYC: a mid term report, *AI Mag.* **11** (3) (1990) 32–59.

[32] J.Y. Halpern and Y.O. Moses, Knowledge and common knowledge in a distributed environment, in: J.Y. Halpern, ed., *Theoretical Aspects of Reasoning about Knowledge* (Morgan Kaufmann, San Mateo, CA, 1984) 50–61.

[33] F. Hayes-Roth, Towards a framework for Distributed AI, *SIGART Newsletter* (1980) 51–52.

[34] M.N. Huhns, ed., *Distributed AI* (Pitman, London, 1988).

[35] N.R. Jennings, On being responsible, in: E. Werner and Y. Demazeau, eds., *Decentralized AI* **3** (Elsevier Science, Amsterdam, 1992) 93–102.

[36] N.R. Jennings, Towards a cooperation knowledge level for collaborative problem solving, in: *Proceedings ECAI-92*, Vienna, Austria (1992) 224–228.

[37] N.R. Jennings, Using GRATE to build cooperating agents for industrial control, in: *Proceedings IFAC/IFIP/IMACS Int. Sym. on AI in Real Time Control*, Delft, The Netherlands (1992) 691–696.

[38] N.R. Jennings, Specification and implementation of a belief desire joint intention architecture for collaborative problem solving, *J. Intell. Cooperative Inf. Syst.* **2** (3) (1993) 289–318.

[39] N.R. Jennings, Commitments and conventions: the foundation of coordination in multi-agent systems, *Knowledge Eng. Rev.* **8** (3) (1993) 223–250.

[40] N.R. Jennings, *Cooperation in Industrial Multi-Agent Systems* (World Scientific Press, Singapore, 1994).

[41] N.R. Jennings and E. Mamdani, Using joint responsibility to coordinate collaborative problem solving in dynamic environments, in: *Proceedings AAAI-92*, San Jose, CA (1992) 269–275.

[42] N.R. Jennings and T. Wittig, ARCHON: theory and practice, in: N.M. Avouris and L. Gasser, eds., *Distributed Artificial Intelligence: Theory and Praxis* (Kluwer Academic Press, Boston, MA, 1992) 179–196.

[43] N.R. Jennings, E. Mamdani, I. Laresgoiti, J. Perez and J. Corera, GRATE: a general framework for cooperative problem solving, *J. Intell. Syst. Eng.* **1** (2) (1992) 102–114.

[44] N.R. Jennings, L. Varga, R. Aarnts, J. Fuchs and P. Skarek, Transforming standalone expert systems into a community of cooperating agents, *Eng. Appl. AI* **6** (4) (1993) 317–331.

[45] G. Kiss and H. Reichgelt, Towards a semantics of desires, in: E. Werner and Y. Demazeau, eds., *Decentralised AI* **3** (Elsevier Science, Amsterdam, 1992) 115–128.

[46] M. Klein, Supporting conflict resolution in cooperative design systems, *IEEE Trans. Syst. Man Cybern.* **21** (6) (1991) 1379–1390.

[47] S. Lander, V.R. Lesser and M.E. Connell, Conflict resolution strategies for cooperating expert agents, in: S.M. Deen, ed., *Cooperating Knowledge Based Systems* (Springer Verlag, New York, 1991) 183–200.

[48] D.B. Lenat and E.A. Feigenbaum, On the thresholds of knowledge, *Artif. Intell.* **47** (1991) 185–250.

[49] V.R. Lesser, A retrospective view of FA/C distributed problem solving, *IEEE Trans. Syst. Man Cybern.* **21** (6) (1991) 1347–1362.

[50] V.R. Lesser and D.D. Corkill, The distributed vehicle monitoring testbed: a tool for investigating distributed problem solving networks, *AI Mag.* **4** (3) (1983) 15–33.

[51] H.J. Levesque, P.R. Cohen and J.H. Nunes, On acting together, in: *Proceedings AAAI-90*, Boston, MA (1990) 94–99.

[52] K.E. Lochbaum, B.J. Grosz and C.L. Sidner, Models of plans to support communication, in: *Proceedings AAAI-90*, Boston, MA (1990) 485–490.

[53] A. Lux, P. de Greef, F. Bomarius and D. Steiner, A generic framework for human computer cooperation, in: *Proceedings International Conference on Intelligent and Cooperative Information Systems*, Rotterdam, The Netherlands (1993) 89–97.

[54] T.W. Malone, Modelling coordination in organizations and markets, *Management Sci.* **33** (1987) 1317–1332.

[55] J. McDermott, Developing software is like talking to eskimos about snow, in: *Proceedings AAAI-90*, Boston, MA (1990) 1130–1133.

[56] J. McDermott, A taxonomy of problem solving methods, in: S. Marcus, ed., *Automating Knowledge Acquisition for Expert Systems* (Kluwer Academic Press, Boston, MA, 1988) 225–256.

[57] R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator and W.R. Swartout, Enabling technology for knowledge sharing, *AI Mag.* **12** (3) (1991) 36–56.

[58] A. Newell, The knowledge level, *Artif. Intell.* **18** (1982) 87–127.

[59] D. Partridge, The scope and limitations of future generation expert systems, *Future Gen. Comput. Syst.* **3** (1) (1987) 1–10.

[60] M.E. Pollack, The uses of plans, *Artif. Intell.* **57** (1992) 43–68.

[61] A.S. Rao, M.P. Georgeff and E. Sonenberg, Social plans: a preliminary report, in: E. Werner and Y. Demazeau, eds., *Decentralised AI* **3** (Elsevier Science, Amsterdam, 1992) 57–76.

[62] S.J. Rosenschein and L.P. Kaelbling, The synthesis of digital machines with provable epistemic properties, in: J.Y. Halpern, ed., *Proceedings of Theoretical Aspects of Reasoning about Knowledge* (Morgan Kaufmann, San Mateo, CA, 1985) 83–98.

[63] J. Searle, *Speech Acts: An Essay in the Philosophy of Language* (Cambridge University Press, New York, 1969)

[64] J. Searle, *Intentionality: An Essay in the Philosophy of Mind* (Cambridge University Press, New York, 1983)

[65] J. Searle, Collective intentions and actions, in: P.R. Cohen, J. Morgan and M.E. Pollack, eds., *Intentions in Communication* (MIT Press, Cambridge, MA, 1990) 401–416.

[66] Y. Shoham, Agent oriented programming, *Artif. Intell.* **60** (1993) 51–92.

[67] H.A. Simon, *Models of Man* (Wiley, New York, 1957).

[68] R.G. Smith, The contract-net protocol: high level communication and control in a distributed problem solver, *IEEE Trans. Comput.* **29** (12) (1980) 1104–1113.

[69] G. Stassinopoulos and E. Lembessis, Application of a multi-agent cooperative architecture to process control in the cement factory, ARCHON Technical Report 43/3-93 (1993).

[70] L. Steels, Second generation expert systems, *Future Gen. Comput. Syst.* **1** (4) (1985) 213–221.

[71] L. Steels, Components of expertise, *AI Mag.* **11** (2) (1990) 29–49.

[72] M. Stefik, The next knowledge medium, *AI Mag.* **7** (1) (1986) 34–46.

[73] K.P. Sycara, Argumentation: planning other agents' plans, in: *Proceedings IJCAI-89*, Detroit, MI (1989) 517–523.

[74] R. Tuomela and K. Miller, We-intentions, *Philos. Stud.* **53** (1988) 367–389.

[75] L. Varga, N.R. Jennings and D. Cockburn, Integrating intelligent systems into a cooperating community for electricity distribution management, *Expert Syst. Appl.* **7** (4) (1994) 563–579.

[76] R. Weihmayer and R. Brandau, Cooperative distributed problem solving for communication network management, *Comput. Commun.* **13** (9) (1990) 547–557.

[77] T. Wittig, ed., *ARCHON: An Architecture for Multi-Agent Systems* (Ellis Horwood, Chichester, England, 1992).

[78] G. Zlotkin and J.S. Rosenschein, Negotiation and task sharing among autonomous agents in cooperative domains, in: *Proceedings IJCAI-89*, Detroit, MI (1989) 912–917.