9 Working Together

In the three preceding chapters, we have looked at the basic theoretical principles of multiagent encounters and the properties of such encounters. We have also seen how agents might reach agreements in encounters with other agents, and looked at languages that agents might use to communicate with one another. So far, however, we have seen nothing of how agents can *work together.* In this chapter, we rectify this. We will see how agents can be designed so that they can work together effectively. As I noted in Chapter 1, the idea of computer systems working together may not initially appear to be very novel: the term 'cooperation' is frequently used in the concurrent systems literature, to describe systems that must interact with one another in order to carry out their assigned tasks. There are two main distinctions between multiagent systems and 'traditional' distributed systems as follows.

- Agents in a multiagent system may have been designed and implemented by different individuals, with different goals. They therefore may not share common goals, and so the encounters between agents in a multiagent system more closely resemble *games,* where agents must act strategically in order to achieve the outcome they most prefer.
- Because agents are assumed to be acting autonomously (and so making decisions about what to do *at run time,* rather than having all decisions hardwired in at design time), they must be capable of *dynamically* coordinating their activities and cooperating with others. In traditional distributed and concurrent systems, coordination and cooperation are typically hardwired in at design time.

Working together involves several different kinds of activities, that we will investigate in much more detail throughout this chapter, in particular, the sharing both of tasks and of information, and the dynamic (i.e. run-time) coordination of multiagent activities.

9.1 Cooperative Distributed Problem Solving

Work on *cooperative distributed problem solving* began with the work of Lesser and colleagues on systems that contained agent-like entities, each of which with distinct (but interrelated) expertise that they could bring to bear on problems that the entire system is required to solve:

> CDPS studies how a loosely-coupled network of problem solvers can work together to solve problems that are beyond their individual capabilities. Each problem-solving node in the network is capable of sophis~~ticated problem-solving~~ and can work independently, but the problems faced by the nodes cannot be completed without cooperation. Cooperation is necessary because no single node has sufficient expertise, resources, and information to solve a problem, and different nodes might have expertise for solving different parts of the problem.
>
> (Durfee *et at.*, 1989b, p. G3)

Historically, most work on cooperative problem solving has made the *benevolence* assumption: that the agents in a system implicitly share a common goal, and thus that there is no potential for conflict between them. This assumption implies that agents can be designed so as to help out whenever needed, even if it means that one or more agents must suffer in order to do so: intuitively, all that matters is the *overall* system objectives, not those of the individual agents within it. The benevolence assumption is generally acceptable if all the agents in a system are designed or 'owned' by the same organization or individual. It is important to emphasize that the ability to assume benevolence *greatly* simplifies the designer's task. If we can assume that all the agents need to worry about is the overall utility of the system, then we can design the overall system so as to optimize this.

In contrast to work on distributed problem solving, the more general area of multiagent systems has focused on the issues associated with societies of *self-interested* agents. Thus agents in a multiagent system (unlike those in typical distributed problem-solving systems), cannot be assumed to share a common goal, as they will often be designed by different individuals or organizations in order to represent their interests. One agent's interests may therefore conflict with those of others, just as in human societies. Despite the potential for conflicts of interest, the agents in a multiagent system will ultimately need to cooperate in order to achieve their goals; again, just as in human societies.

Multiagent systems research is therefore concerned with the wider problems of designing societies of autonomous agents, such as why and how agents cooperate (Wooldridge and Jennings, 1994); how agents can recognize and resolve conflicts (Adler *et al.*, 1989; Galliers, 1988b; Galliers, 1990; Klein and Baskin, 1991; Lander *et al.*, 1991); how agents can negotiate or compromise in situations where they are apparently at loggerheads (Ephrati and Rosenschein, 1993; Rosenschein and Zlotkin, 1994); and so on.

It is also important to distinguish CDPS from *parallel* problem solving (Bond and Gasser, 1988, p. 3). Parallel problem solving simply involves the exploitation of parallelism in solving problems. Typically, in parallel problem solving, the computational components are simply processors; a single node will be responsible for *decomposing* the overall problem into sub-components, allocating these to processors, and subsequently assembling the solution. The nodes are frequently assumed to be homogeneous in the sense that they do not have distinct expertise - they are simply processors to be exploited in solving the problem. Although parallel problem solving was synonymous with CDPS in the early days of multiagent systems, the two fields are now regarded as quite separate. (However, it goes without saying that a multiagent system will employ parallel architectures and languages: the point is that the concerns of the two areas are rather different.)

Coherence and coordination

Having implemented an artificial agent society in order to solve some problem, how does one assess the success (or otherwise) of the implementation? What criteria can be used? The multiagent systems literature has proposed two types of issues that need to be considered.

Coherence. Refers to 'how well the [multiagent] system behaves as a unit, along some dimension of evaluation' (Bond and Gasser, 1988, p. 19). Coherence may be measured in terms of solution quality, efficiency of resource usage, conceptual clarity of operation, or how well system performance degrades in the presence of uncertainty or failure; a discussion on the subject of when multiple agents can be said to be acting coherently appears as (Wooldridge, 1994).

Coordination. In contrast, is 'the degree...to which [the agents]...can avoid 'extraneous' activity [such as]...synchronizing and aligning their activities' (Bond and Gasser, 1988, p. 19); in a perfectly coordinated system, agents will not accidentally clobber each other's sub-goals while attempting to achieve a common goal; they will not need to explicitly communicate, as they will be mutually predictable, perhaps by maintaining good internal models of each other. The presence of conflict between agents, in the sense of agents destructively interfering with one another (which requires time and effort to resolve), is an indicator of poor coordination.

It is probably true to say that these problems have been the focus of more attention in multiagent systems research than any other issues (Durfee and Lesser,

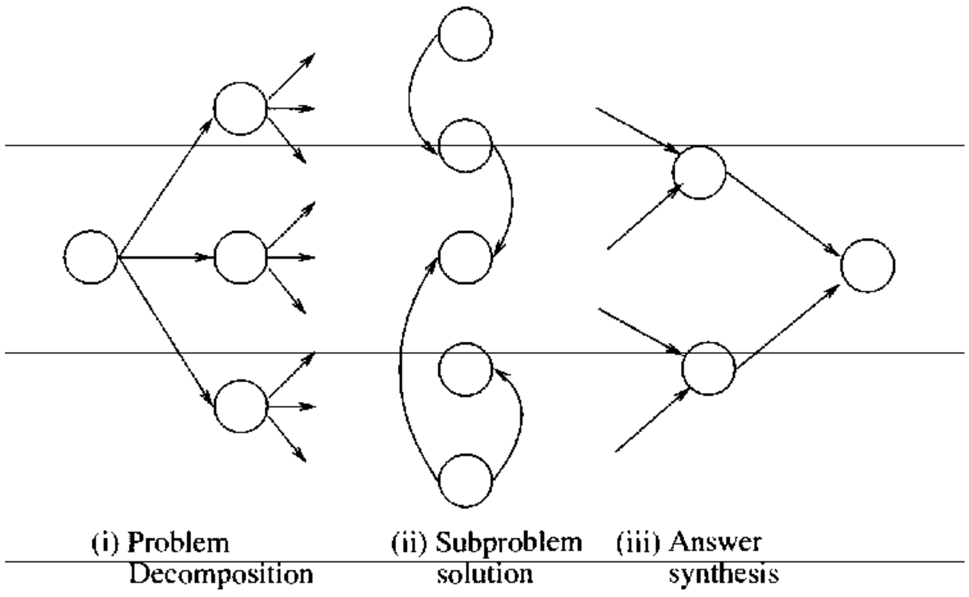

Figure 9.1 The three stages of CDPS.

1987; Durfee, 1988; Gasser and Hill, 1990; Goldman and Rosenschein, 1993; Jennings, 1993a; Weiß, 1993).

The main issues to be addressed in CDPS include the following.

- How can a problem be divided into smaller tasks for distribution among agents?
- How can a problem solution be effectively synthesized from sub-problem results?
- How can the overall problem-solving activities of the agents be optimized so as to produce a solution that maximizes the coherence metric?
- What techniques can be used to coordinate the activity of the agents, so avoiding destructive (and thus unhelpful) interactions, and maximizing effectiveness (by exploiting any positive interactions)?

In the remainder of this chapter, we shall see some techniques developed by the multiagent systems community for addressing these concerns.

9.2 Task Sharing and Result Sharing

How do a group of agents work together to solve problems? Smith and Davis (1980) suggested that the CDPS process can canonically be viewed as a three-stage activity (see Figure 9.1) as follows.

(1) Problem decomposition. In this stage, the overall problem to be solved is decomposed into smaller sub-problems. The decomposition will typically be hierarchical, so that sub-problems are then further decomposed into smaller sub-problems, and so on, until the sub-problems are of an appropriate granularity to be solved by individual agents. The different levels of decomposition will often represent different levels of problem abstraction. For example, consider a (real-world) example of cooperative problem solving, which occurs when a government body asks whether a new hospital is needed in a particular region. In order to answer this question, a number of smaller sub-problems need to be solved, such as whether the existing hospitals can cope, what the likely demand is for hospital beds in the future, and so on. The smallest level of abstraction might involve asking individuals about their day-to-day experiences of the current hospital provision. Each of these different levels in the problem-solving hierarchy represents the problem at a progressively lower level of abstraction. Notice that the grain size of sub-problems is important: one extreme view of CDPS is that a decomposition continues until the sub-problems represent 'atomic' actions, which cannot be decomposed any further. This is essentially what happens in the ACTOR paradigm, with new agents - ACTORs being spawned for every sub-problem, until ACTORs embody individual program instructions such as addition, subtraction, and so on (Agha, 1986). But this approach introduces a number of problems. In particular, the overheads involved in managing the interactions between the (typically very many) sub-problems outweigh the benefits of a cooperative solution.

Another issue is how to perform the decomposition. One possibility is that the problem is decomposed by one individual agent. However, this assumes that this agent must have the appropriate expertise to do this - it must have knowledge of the *task structure*, that is, how the task is 'put together'. If other agents have knowledge pertaining to the task structure, then they may be able to assist in identifying a better decomposition. The decomposition itself may therefore be better treated as a cooperative activity.

Yet another issue is that task decomposition cannot in general be done without some knowledge of the agents that will eventually solve problems. There is no point in arriving at a particular decomposition that is impossible for a particular collection of agents to solve.

(2) **Sub-problem solution.** In this stage, the sub-problems identified during problem decomposition are individually solved. This stage typically involves sharing of information between agents: one agent can help another out if it has information that may be useful to the other.

(3) **Solution synthesis.** In this stage, solutions to individual sub-problems are integrated into an overall solution. As in problem decomposition, this stage may be hierarchical, with partial solutions assembled at different levels of abstraction.

Note that the extent to which these stages are explicitly carried out in a particular problem domain will depend very heavily on the domain itself; in some domains, some of the stages may not be present at all.

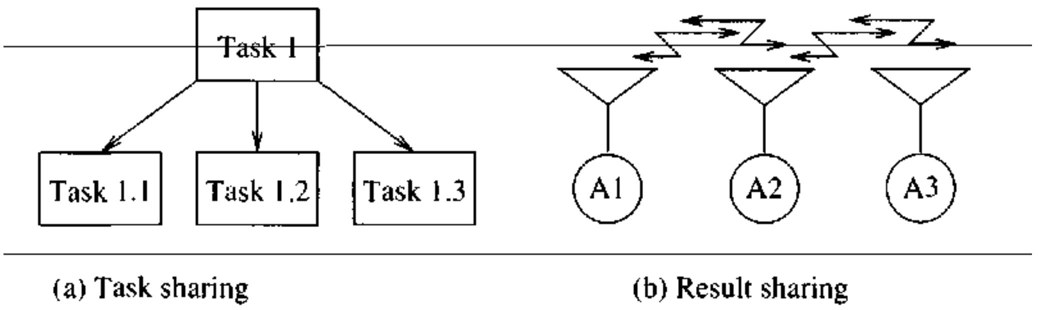

Figure 9.2 (a) Task sharing and (b) result sharing. In task sharing, a task is decomposed into sub-problems that are allocated to agents, while in result sharing, agents supply each other with relevant information, either proactively or on demand.

Given this general framework for CDPS, there are two specific cooperative problem-solving activities that are likely to be present: *task sharing* and *result sharing* (Smith and Davis, 1980) (see Figure 9.2).

Task sharing. Task sharing takes place when a problem is decomposed to smaller sub-problems and allocated to different agents. Perhaps the key problem to be solved in a task-sharing system is that of how tasks are to be *allocated* to individual agents. If all agents are homogeneous in terms of their capabilities (cf. the discussion on parallel problem solving, above), then task sharing is straightforward: any task can be allocated to any agent. However, in all but the most trivial of cases, agents have very different capabilities. In cases where the agents arc really autonomous - and can hence decline to carry out tasks (in systems that do not enjoy the *benevolence* assumption described above), then task allocation will involve agents *reaching agreements* with others, perhaps by using the techniques described in Chapter 7.

Result sharing. Result sharing involves agents sharing information relevant to their sub-problems. This information may be shared *proactively* (one agent sends another agent some information because it believes the other will be interested in it), or *reactively* (an agent sends another information in response to a request that was previously sent - cf. the `subscribe` performatives in the agent communication languages discussed earlier).

In the sections that follow, I shall discuss task sharing and result sharing in more detail.

9.2.1 Task sharing in the Contract Net

The Contract Net (CNET) protocol is a high-level protocol for achieving efficient cooperation through task sharing in networks of communicating problem solvers (Smith, 1977, 1980a,b; Smith and Davis, 1980). The basic metaphor used in the CNET is, as the name of the protocol suggests, contracting - Smith took his inspiration from the way that companies organize the process of putting contracts out to tender (see Figure 9.3).

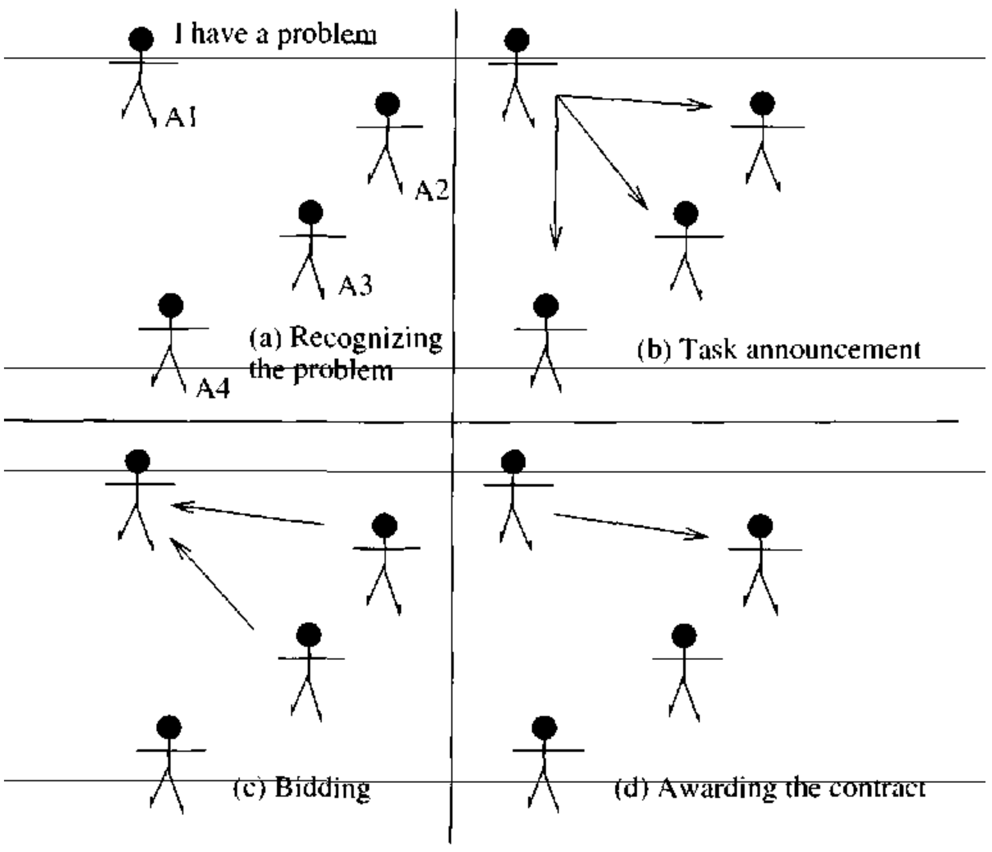

Figure 9.3 The Contract Net (CNET) protocol.

[A] node that generates a task advertises existence of that task to other nodes in the net with a *task announcement,* then acts as the *manager* of that task for its duration. In the absence of any information about the specific capabilities of the other nodes in the net, the manager is forced to issue a *general broadcast* to all other nodes. If, however, the manager possesses some knowledge about which of the other nodes in the net are likely candidates, then it can issue a *limited broadcast* to just those candidates. Finally, if the manager knows exactly which of the other nodes in the net is appropriate, then it can issue a *point-to-point* announcement. As work on the problem progresses, many such task announcements will be made by various managers.

Nodes in the net listen to the task announcements and evaluate them with respect to their own specialized hardware and software resources. When a task to which a node is suited is found, it submits a *bid.* A bid indicates the capabilities of the bidder that are relevant to the execution of the announced task. A manager may receive several such bids in response to a single task announcement; based on the information in the bids, it selects the most appropriate nodes to execute the task. The selection is communicated to the successful bidders through an *award* message. These selected nodes assume responsibil-

> ity for execution of the task, and each is called a *contractor* for that task.
>
> After the task has been completed, the contractor sends a *report* to the manager. (Smith, 1980b, pp. 60, 61)
>
> [This] normal contract negotiation process can be simplified in some instances, with a resulting enhancement in the efficiency of the protocol. If a manager knows exactly which node is appropriate for the execution of a task, a *directed contract* can be awarded. This differs from the *announced contract* in that no announcement is made and no bids are submitted. Instead, an award is made directly. In such cases, nodes awarded contracts must acknowledge receipt, and have the option of refusal.
>
> Finally, for tasks that amount to simple requests for information, a contract may not be appropriate. In such cases, a request-response sequence can be used without further embellishment. Such messages (that aid in the distribution of data as opposed to control) are implemented as *request* and *information* messages. The request message is used to encode straightforward requests for information when contracting is unnecessary. The information message is used both as a response to a request message and a general data transfer message.
>
> (Smith, 1980b, pp. 62, 63)

In addition to describing the various messages that agents may send, Smith describes the procedures to be carried out on receipt of a message. Briefly, these procedures are as follows (see Smith (1980b, pp. 96-102) for more details).

(1) Task announcement processing. On receipt of a task announcement, an agent decides if it is *eligible* for the task. It does this by looking at the *eligibility specification* contained in the announcement. If it is eligible, then details of the task are stored, and the agent will subsequently bid for the task.

(2) Bid processing. Details of bids from would-be contractors are stored by (would-be) managers until some deadline is reached. The manager then awards the task to a single bidder.

(3) **Award processing.** Agents that bid for a task, but fail to be awarded it, simply delete details of the task. The successful bidder must attempt to expedite the task (which may mean generating new sub-tasks).

(4) Request and inform processing. These messages are the simplest to handle. A request simply causes an inform message to be sent to the requestor, containing the required information, but only if that information is immediately available. (Otherwise, the requestee informs the requestor that the information is unknown.) An inform message causes its content to be added to the recipient's database. It is assumed that at the conclusion of a task, a contractor will send an information message to the manager, detailing the results of the expedited task[1].

Despite (or perhaps because of) its simplicity, the Contract Net has become the most implemented and best-studied framework for distributed problem solving.

)3 Result Sharing

In result sharing, problem solving proceeds by agents cooperatively exchanging information as a solution is developed. Typically, these results will progress from being the solution to small problems, which are progressively refined into larger, more abstract solutions. Durfee (1999, p. 131) suggests that problem solvers can improve group performance in result sharing in the following ways.

Confidence: independently derived solutions can be cross-checked, highlighting possible errors, and increasing confidence in the overall solution.

Completeness: agents can share their *local* views to achieve a better overall *global* view.

Precision: agents can share results to ensure that the precision of the overall solution is increased.

Timeliness: even if one agent could solve a problem on its own, by sharing a solution, the result could be derived more quickly.

)A Combining Task and Result Sharing

In the everyday cooperative working that we all engage in, we frequently *combine* task sharing and result sharing. In this section, I will briefly give an overview of how this was achieved in the FELINE system (Wooldridge *et al.*, 1991). FELINE was a *cooperating expert system.* The idea was to build an overall problem-solving system as a collection of cooperating experts, each of which had expertise in distinct but related areas. The system worked by these agents cooperating to both *share knowledge* and *distribute subtasks.* Each agent in FELINE was in fact an idependent rule-based system: it had a working memory, or database, containing information about the current state of problem solving; in addition, each agent had a collection of rules, which encoded its domain knowledge.

Each agent in FELINE also maintained a data structure representing its beliefs about itself and its environment. This data structure is called the *environment model* (cf. the agents with symbolic representations discussed in Chapter 3). It

[1]This is done via a special *report* message type in the original CNET framework.

contained an entry for the modelling agent and each agent that the modelling agent might communicate with (its *acquaintances).* Each entry contained two important attributes as follows.

Skills. This attribute is a set of identifiers denoting hypotheses which the agent has the expertise to establish or deny. The skills of an agent will correspond roughly to root nodes of the inference networks representing the agent's domain expertise.

Interests. This attribute is a set of identifiers denoting hypotheses for which the agent requires the truth value. It may be that an agent actually has the expertise to establish the truth value of its interests, but is nevertheless 'interested' in them. The interests of an agent will correspond roughly to leaf nodes of the inference networks representing the agent's domain expertise.

Messages in FELINE were triples, consisting of a *sender, receiver,* and *contents.* The contents field was also a triple, containing *message type, attribute,* and *value.* Agents in FELINE communicated using three message types as follows (the system predated the KQML and FIPA languages discussed in Chapter 8).

Request. If an agent sends a request, then the attribute field will contain an identifier denoting a hypothesis. It is assumed that the hypothesis is one which lies within the domain of the intended recipient. A request is assumed to mean that the sender wants the receiver to derive a truth value for the hypothesis.

Response. If an agent receives a request and manages to successfully derive a truth value for the hypothesis, then it will send a response to the originator of the request. The attribute field will contain the identifier denoting the hypothesis; the value field ~~will~~ contain the associated truth value.

Inform. The attribute field of an inform message will contain an identifier denoting a hypothesis. The value field will contain an associated truth value. An inform message will be unsolicited; an agent sends one if it thinks the recipient will be 'interested' in the hypothesis.

To understand how problem solving in FELINE worked, consider goal-driven problem solving in a conventional rule-based system. Typically, goal-driven reasoning proceeds by attempting to establish the truth value of some hypothesis. If the truth value is not known, then a recursive descent of the inference network associated with the hypothesis is performed. Leaf nodes in the inference network typically correspond to questions which are asked of the user, or data that is acquired in some other way. Within FELINE, this scheme was augmented by the following principle. When evaluating a leaf node, if it is not a question, then the environment model was checked to see if any other agent has the node as a 'skill'. If there was some agent that listed the node as a skill, then a request was sent to that agent, requesting the hypothesis. The sender of the request then waited until a response was received; the response indicates the truth value of the node.

Typically, data-driven problem solving proceeds by taking a database of facts (hypotheses and associated truth values), and a set of rules, and repeatedly generating a set of new facts. These new facts are then added to the database, and the process begins again. If a hypothesis follows from a set of facts and a set of rules, then this style of problem solving will eventually generate a result. In FELINE, this scheme was augmented as follows. Whenever a new fact was generated by an agent, the environment model was consulted to see if any agent has the hypothesis as an 'interest'. If it did, then an 'inform' message was sent to the appropriate agent, containing the hypothesis and truth value. Upon receipt of an 'inform' message, the recipient agent added the fact to its database and entered a forward chaining cycle, to determine whether any further information could be derived; this could lead to yet more information being sent to other agents. Similar schemes were implemented in (for example) the CoOpera system (Sommaruga *et al.*, 1989).

).5 Handling Inconsistency

One of the major problems that arises in cooperative activity is that of *inconsistencies* between different agents in the system. Agents may have inconsistencies with respect to both their *beliefs* (the information they hold about the world), and their *goals/intentions* (the things they want to achieve). As I indicated earlier, inconsistencies between goals generally arise because agents are assumed to be autonomous, and thus not share common objectives. Inconsistencies between the beliefs that agents have can arise from several sources. First, the viewpoint that agents have will typically be limited - no agent will ever be able to obtain a *complete* picture of their environment. Also, the sensors that agents have may be faulty, or the information sources that the agent has access to may in turn be faulty.

In a system of moderate size, inconsistencies are inevitable: the question is how to deal with them. Durfee *et al.* (1989a) suggest a number of possible approaches to the problem as follows.

- Do not allow it to occur - or at least ignore it. This is essentially the approach of the Contract Net: task sharing is always driven by a manager agent, who has the only view of the problem that matters.
- Resolve inconsistencies through negotiation (see Chapter 7). While this may be desirable in theory, the communication and computational overheads incurred suggest that it will rarely be possible in practice.
- Build systems that degrade gracefully in the presence of inconsistency.

The third approach is clearly the most desirable. Lesser and Corkill (1981) refer to systems that can behave robustly in the presence of inconsistency as *functionally accurate/cooperative* (FA/C):

> [In FA/C systems].. .nodes cooperatively exchange and integrate partial, tentative, high-level results to construct a consistent and complete solution. [An agent's] problem-solving is structured so that its local knowledge bases need not be complete, consistent, and up-to-date in order to make progress on its problem-solving tasks. Nodes do the best they can with their current information, but their solutions to their local sub-problems may be only partial, tentative, and incorrect.
>
> (Durfee *et al.*, 1989a, pp. 117, 118)

Lesser and Corkill (1981) suggested the following characteristics of FA/C systems that tolerate inconsistent/incorrect information.

- Problem solving is not tightly constrained to a particular sequence of events - it progresses opportunistically (i.e. not in a strict predetermined order, but taking advantage of whatever opportunities arise) and incrementally (i.e. by gradually piecing together solutions to sub-problems).
- Agents communicate by exchanging high-level intermediate results, rather than by exchanging raw data.
- Uncertainty and inconsistency is implicitly resolved when partial results are exchanged and compared with other partial solutions. Thus inconsistency and uncertainty is resolved as problem solving progresses, rather than at the beginning or end of problem solving.
- The solution is not constrained to a single solution route: there are many possible ways of arriving at a solution, so that if one fails, there are other ways of achieving the same end. This makes the system robust against localized failures and bottlenecks in problem solving.

9.6 Coordination

Perhaps the defining problem in cooperative working is that of *coordination.* The coordination problem is that of *managing inter-dependencies between the activities of agents:* some coordination mechanism is essential if the activities that agents can engage in can interact in any way. How might two activities interact? Consider the following real-world examples.

- You and I both want to leave the room, and so we independently walk towards the door, which can only fit one of us. I graciously permit you to leave first.

  In this example, our activities need to be coordinated because there is a resource (the door) which we both wish to use, but which can only be used by one person at a time.

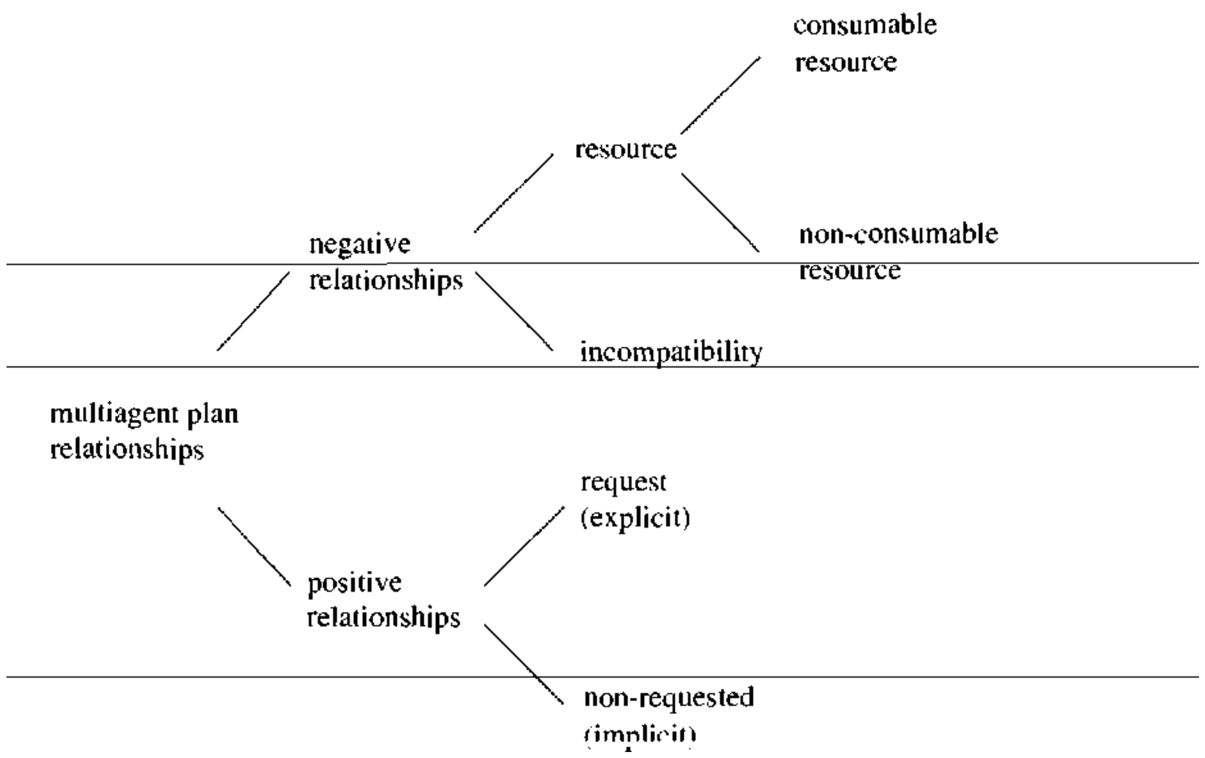

Figure 9.4 Von Martial's typology of coordination relationships.

- I intend to submit a grant proposal, but in order to do this, I need your signature.

  In this case, my activity of sending a grant proposal depends upon your activity of signing it off - I cannot carry out my activity until yours is completed. In other words, my activity *depends* upon yours.

- I obtain a soft copy of a paper from a Web page. I know that this report will be of interest to you as well. Knowing this, I proactively photocopy the report, and give you a copy.

  In this case, our activities do not strictly need to be coordinated - since the report is freely available on a Web page, you could download and print your own copy. But by proactively printing a copy, I save you time and hence, intuitively, increase your utility.

von Martial (1990) suggested a typology for coordination relationships (see Figure 9.4). He suggested that, broadly, relationships between activities could be either *positive* or *negative.*

Positive relationships 'are all those relationships between two plans from which some benefit can be derived, for one or both of the agents plans, by combining them' (von Martial, 1990, p. 111). Such relationships may be *requested* (I *explicitly* ask you for help with my activities) or *non-requested* (it so happens that by working together we can achieve a solution that is better for at least one of us, without making the other any worse off, cf. discussions of pareto optimality in the preceding chapters). von Martial (1990, p. 112) distinguishes three types of non-requested relationship as follows.

The action equality relationship. We both plan to perform an identical action, and by recognizing this, one of us can perform the action alone and so save the other effort.

The consequence relationship. The actions in my plan have the side-effect of achieving one of your goals, thus relieving you of the need to explicitly achieve it.

The favour relationship. Some part of my plan has the side effect of contributing to the achievement of one of your goals, perhaps by making it easier (e.g. by achieving a precondition of one of the actions in it).

Coordination in multiagent systems is assumed to happen *at run time,* that is, the agents themselves must be capable of recognizing these relationships and, where necessary, managing them as part of their activities (von Martial, 1992). This contrasts with the more conventional situation in computer science, where a designer explicitly attempts to anticipate possible interactions in advance, and designs the system so as to avoid negative interactions and exploit potential positive interactions.

In the sections that follow, I present some of the main approaches that have been developed tor dynamically coordinating activities.

9.6.1 Coordination through partial global planning

The Distributed Vehicle Monitoring Testbed (DVMT) was one of the earliest and best-known testbeds for multiagent systems. The DVMT was a fully instrumented testbed for developing distributed problem-solving networks (Lesser and Erman, 1980; Lesser and Corkill, 1988). The testbed was based around the domain of distributed vehicle sensing and monitoring: the aim was to successfully track a number of vehicles that pass within the range of a set of distributed sensors. The main purpose of the testbed was to support experimentation into different problem-solving strategies.

The distributed sensing domain is inherently data driven: new data about vehicle movements appears and must be processed by the system. The main problem with the domain was to process information as rapidly as possible, so that the system could come to conclusions about the paths of vehicles in time for them to be useful. To coordinate the activities of agents in the DVMT, Durfee developed an approach known as *partial global planning* (Durfee and Lesser, 1987; Durfee, 1988, 1996).

The main principle of partial global planning is that cooperating agents exchange information in order to reach common conclusions about the problem-solving process. Planning is *partial* because the system does not (indeed *cannot)* generate a plan for the entire problem. It is *global* because agents form non-local plans by exchanging local plans and cooperating to achieve a non-local view of problem solving.

Partial global planning involves three iterated stages.

(1) Each agent decides what its own goals are, and generates short-term plans in order to achieve them.

(2) Agents exchange information to determine where plans and goals interact.

(3) Agents alter local plans in order to better coordinate their own activities.

In order to prevent incoherence during these processes, Durfee proposed the use of a *meta-level structure,* which guided the cooperation process within the system. The meta-level structure dictated which agents an agent should exchange information with, and under what conditions it ought to do so.

The actions and interactions of a group of agents were incorporated into a data structure known as a *partial global plan.* This data structure will be generated cooperatively by agents exchanging information. It contained the following principle attributes.

e. The objective is the larger goal that the system is working towards.

Activity maps. An activity map is a representation of what agents are actually doing, and what results will be generated by their activities.

Solution construction graph. A solution construction graph is a representation of how agents ought to interact, what information ought to be exchanged, and when, in order for the system to successfully generate a result.

Keith Decker extended and refined the PGP coordination mechanisms in his TÆMS testbed (Decker, 1996); this led to what he called *generalized partial global planning* (GPGP - pronounced 'gee pee gee pee') (Decker and Lesser, 1995). GPGP makes use of five techniques for coordinating activities as follows.

Updating non-local viewpoints. Agents have only local views of activity, and so sharing information can help them achieve broader views. In his TÆMS system, Decker uses three variations of this policy: communicate no local information, communicate all information, or an intermediate level.

Communicate results. Agents may communicate results in three different ways. A minimal approach is where agents only communicate results that are essential to satisfy obligations. Another approach involves sending all results. A third is to send results to those with an interest in them.

Handling simple redundancy. Redundancy occurs when efforts are duplicated. This may be deliberate - an agent may get more than one agent to work on a task because it wants to ensure the task gets done. However, in general, redundancies indicate wasted resources, and are therefore to be avoided. The solution adopted in GPGP is as follows. When redundancy is detected, in the form of multiple agents working on identical tasks, one agent is selected at random to carry out the task. The results are then broadcast to other interested agents.

Handling hard coordination relationships. 'Hard' coordination relationships are essentially the 'negative' relationships of von Martial, as discussed above. Hard coordination relationships are thus those that threaten to prevent activities being successfully completed. Thus a hard relationship occurs when there is a danger of the agents' actions destructively interfering with one another, or preventing each others actions being carried out. When such relationships are encountered, the activities of agents are rescheduled to resolve the problem.

Handling soft coordination relationships. 'Soft' coordination relationships in-~~clude the 'positive' relationships of von Martial~~. Thus these relationships include those that are not 'mission critical', but which may improve overall performance. When these are encountered, then rescheduling takes place, but with a high degree of 'negotiability': if rescheduling is not found possible, then the system does not worry about it too much.

9.6.2 Coordination through joint intentions

The second approach to coordination that I shall discuss is the use of *human teamwork models.* We saw in Chapter 4 how some researchers have built agents around the concept of practical reasoning, and how central intentions are in this practical reasoning process. Intentions also play a critical role in coordination: they provide both the stability and predictability that is necessary for social interaction, and the flexibility and reactivity that is necessary to cope with a changing environment. If you know that I am planning to write a book, for example, then this gives you information that you can use to coordinate your activities with mine. For example, it allows you to rule out the possibility of going on holiday with me, or partying with me all night, because you know I will be working hard on the book.

When humans work together as a team, mental states that are closely related to intentions appear to play a similarly important role (Levesque *et ah,* 1990; Cohen and Levesque, 1991). It is important to be able to distinguish coordinated action that is not cooperative from coordinated cooperative action. As an illustration of this point, consider the following scenario (Searle, 1990).

> A group of people are sitting in a park. As a result of a sudden downpour all of them run to a tree in the middle of the park because it is the only available source of shelter. This may be coordinated behaviour, but it is not cooperative action, as each person has the intention of stopping themselves from becoming wet, and even if they are aware of what others are doing and what their goals arc, it does not affect their intended action. This contrasts with the situation in which the people are dancers, and the choreography calls for them to converge

on a common point (the tree). In this case, the individuals are performing exactly the same actions as before, but because they each have the aim of meeting at the central point as a consequence of the overall aim of executing the dance, this is cooperative action.

How does having an individual intention towards a particular goal differ from being part of a team, with some sort of collective intention towards the goal? The distinction was first studied in Levesque *et al.* (1990), where it was observed that being part of a team implies some sort of *responsibility* towards the other members of the team. To illustrate this, suppose that you and I are together lifting a heavy object as part of a team activity. Then clearly we both individually have the intention to lift the object - but is there more to teamwork than this? Well, suppose I come to believe that it is not going to be possible to lift it for some reason. If I just have an *individual* goal to lift the object, then the rational thing for me to do is simply drop the intention (and thus perhaps also the object). But you would hardly be inclined to say I was cooperating with you if I did so. Being part of a team implies that I show some responsibility towards you: that if I discover the team effort is not going to work, then I should at least attempt to make you aware of this.

Building on the work of Levesque *et al.* (1990), Jennings distinguished between the *commitment* that underpins an intention and the associated *convention* ~~(Jennings, 1993a).~~ A *commitment* is a pledge or a promise (for example, to have lifted the object); a *convention* in contrast is a means of monitoring a commitment - it specifies under what circumstances a commitment can be abandoned and how an agent should behave both locally and towards others when one of these conditions arises.

In more detail, one may commit ~~either~~ to a particular course of action, or, more generally, to a state of affairs. Here, we are concerned only with commitments that are *future directed* towards a state of affairs. Commitments have a number of important properties (see Jennings (1993a) and Cohen and Levesque (1990a, pp. 217-219) for a discussion), but the most important is that *commitments per-~~sist~~*: having adopted a ~~commitment,~~ we do not expect an agent to drop it until, for some reason, it becomes redundant. The conditions under which a commitment can become redundant are specified in the associated convention - examples include the motivation for the goal no longer being present, the goal being achieved, and the realization that the goal will never be achieved (Cohen and ~~1990a).~~

When a group of agents are engaged in a cooperative activity they must have a joint commitment to the overall aim, as well as their individual commitments to the specific tasks that they have been assigned. This joint commitment shares the persistence property of the individual commitment; however, it differs in that its is distributed amongst the team members. An appropriate social convention must also be in place. This social convention identifies the conditions under which the joint commitment can be dropped, and also describes how an agent should behave towards its fellow team members. For example, if an agent drops its joint commitment because it believes that the goal will never be attained, then it is part of the notion of 'cooperativeness' that is inherent in joint action that it informs all of its fellow team members of its change of stale. In this context, social conventions provide general guidelines, and a common frame of reference in which agents can work. By adopting a convention, every agent knows what is expected both of it, and of every other agent, as part of the collective working towards the goal, and knows that every other agent has a similar set of expectations.

We can begin to define this kind of cooperation in the notion of a *joint persistent goal* (JPG), as defined in Levesque *et al* (1990). In a JPG, a group of agents have a collective commitment to bringing about some goal *qp;* the *motivation* for this goal, i.e. the reason that the group has the commitment, is represented by ψ. Thus *qp* might be 'move the heavy object', while ψ might be 'Michael wants the heavy object moved'. The mental state of the team of agents with this JPG might be described as follows:

- initially, every agent does not believe that the goal *qp* is satisfied, but believes *qp* is possible;
- every agent *i* then has a goal of *qp* until the termination condition is satisfied (see below);
- until the termination condition is satisfied, then
  - if any agent *i* believes that the goal is achieved, then it will have a goal that this becomes a mutual belief, and will retain this goal until the termination condition is satisfied;
  - if any agent *i* believes that the goal is impossible, then it will have a goal that this becomes a mutual belief, and will retain this goal until the termination condition is satisfied;
  - if any agent *i* believes that the motivation ψ for the goal is no longer present, then it will have a goal that this becomes a mutual belief, and will retain this goal until the termination condition is satisfied;
- the termination condition is that it is mutually believed that either
  - the goal φ is satisfied;
  - the goal *qp* is impossible to achieve;
  - the motivation/justification ψ for the goal is no longer present.

Commitments and conventions in ARCHON

Jennings (1993a, 1995) investigated the use of commitments and such as JPGs in the coordination of an industrial control system called ARCHON (Wittig, 1992; Jennings *et al,* 1996a; Perriolat *et al,* 1996). He noted that commitments and conventions could be encoded as *rules* in a rule-based system. This makes it possible to explicitly encode coordination structures in the reasoning mechanism of an agent.

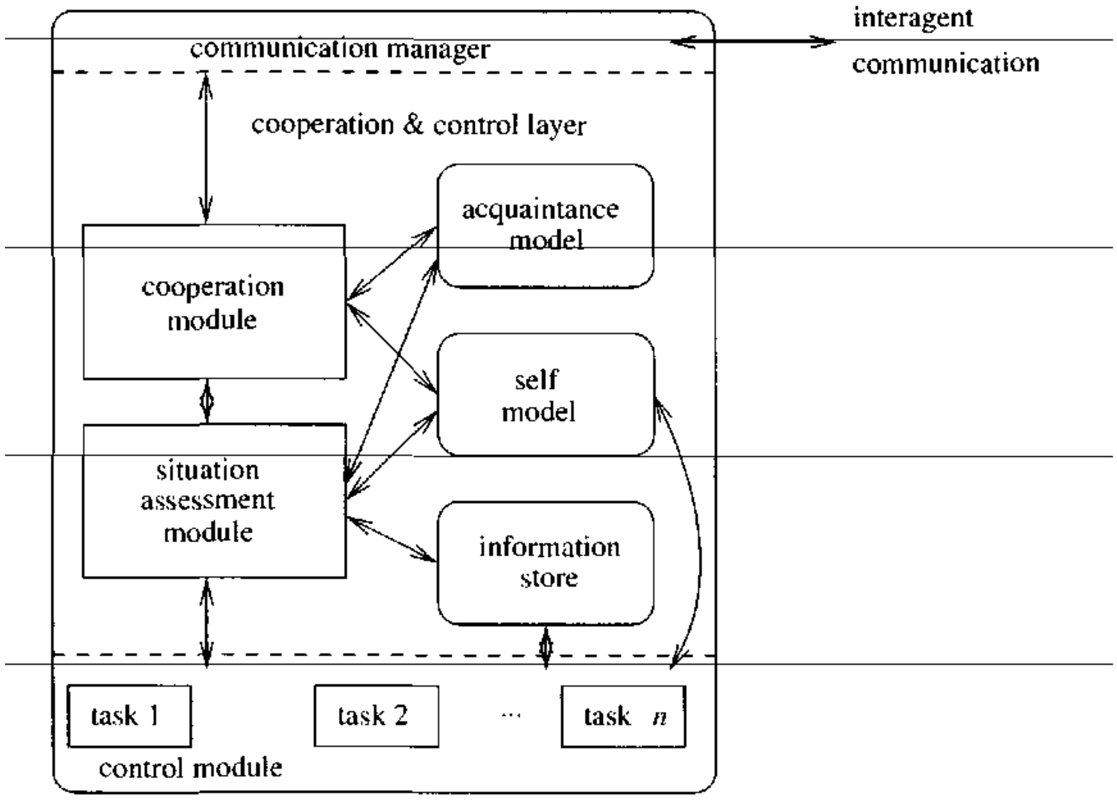

Figure 9.5 ARCHON agent architecture.

The overall architecture of agents in ARCHON is illustrated in Figure 9.5. Agents have three layers. The lowest layer is the *control* layer. This layer contains domain-specific agent capabilities. The idea is that agents in ARCHON *wrap* legacy software in agent-like capabilities. In the ARCHON case, these legacy systems were stand-alone expert systems. The legacy systems were embedded within a control module, which provided access to them via an API. ARCHON agents maintained three different types of information, in the forms of an *acquaintance model* (cf. the acquaintance models of the MACE system described later in this chapter), a *self model* (which contains information about the agent's own skills and interests), and, finally, a general-purpose information store, which contains other information about the agent's environment. The behaviour of the agent was determined by three main control modules: the *cooperation module,* which was responsible for the agent's social ability; the *situation assessment module,* which was responsible for determining when the need for new teamwork arose; and, finally, the *communication manager,* which was responsible for sending/receiving messages.

Some of the rules used for reassessing joint commitments and selecting actions to repair failed teamwork are shown in Figure 9.6 (from Jennings, 1995). The first four rules capture the conditions where the joint goal has been successfully achieved, where the motivation for the joint goal is no longer present, and where the current plan to achieve the joint goal has been invalidated in some way. The following 'select' rules are used to determine a repair action.

Match rules::

```
R1:   if   task t has finished executing and
           t has produced desired outcome of joint action
      then joint goal is satisfied.

R2:   if   receive information i and
           i is related to triggering conditions
             for joint goal G and
           i invalidates beliefs for wanting G
      then motivation for G is no longer present.

R3:   if   delay task t1 and
           t1 is a component of common recipe R and
           t1 must be synchronized with t2 in R
      then R is violated.

R4:   if   finished executing common recipe R and
           expected results of R not produced and
           alternative recipe exists
      then R is invalid.
```

Select rules:

```
R1:   if   joint goal is satisfied
      then abandon all associated local activities and
           inform cooperation module

R2:   if   motivation for joint goal no longer present
      then abandon all associated local activities and
           inform cooperation module

R3:   if   common recipe R is violated and
           R can be rescheduled
      then suspend local activities associated with R and
           reset timings and descriptions associated with R and
           inform cooperation module

R4:   if   common recipe R1 is invalid and
           alternative recipe R2 exists
      then abandon all local activities with R1 and
           inform cooperation module that R1 is invalid and
           propose R2 to cooperation module
```

Figure 9.6 Joint commitment rules in ARCHON.

Milind Tambe developed a similar framework for teamwork called Steam (Tambe, 1997). Agents in Steam are programmed using the Soar rule-based architecture (Newell *et al.*, 1989, 1990). The cooperation component of Steam is encoded in about 300 domain-independent rules, somewhat similar in principle to Jennings's teamwork rules, as shown above. However, the cooperation rules of Steam are far more complex, allowing for sophisticated hierarchical team structures.

The Steam framework was used in a number of application domains, including military mission simulations, as well as the RoboCup simulated robotic soccer domain.

A teamwork-based model of CDPS

Building on Jennings's teamwork-based coordination model (Jennings, 1995), a four-stage model of CDPS was presented in Wooldridge and Jennings (1994, 1999). The four stages of the model are as follows.

(1) **Recognition.** CUPS begins when some agent in a multiagent community has a goal, and recognizes the potential for cooperative action with respect to that goal. Recognition may occur for several reasons. The paradigm case is that in which the agent is unable to achieve the goal in isolation, but believes that cooperative action can achieve it. For example, an agent may have a goal which, to achieve, requires information that is only accessible to another agent. Without the cooperation of this other agent, the goal cannot be achieved. More prosaically, an agent with a goal to move a heavy object might simply not have the strength to do this alone.

Alternatively, an agent may be able to achieve the goal on its own, but may not want to. There may be several reasons for this. First, it may believe that in working alone, it will clobber one of its other goals. For example, suppose I have a goal of lifting a heavy object. I may have the capability of lifting the object, but I might believe that in so doing, I would injure my back, thereby clobbering my goal of being healthy. In this case, a cooperative solution - involving no injury to my back - is preferable. More generally, an agent may believe that a cooperative solution will in some way be better than a solution achieved by action in isolation. For example, a solution might be obtained more quickly, or may be more accurate as a result of cooperative action.

Relieving that you either cannot achieve your goal in isolation, or that (for whatever reason) you would prefer not to work alone, is part of the potential for cooperation. But it is not enough in itself to initiate the social process. For there to be potential for cooperation with respect to an agent's goal, the agent must also believe there is some group of agents that can actually achieve the goal.

Team **formation.** During this stage, the agent that recognized the potential for cooperative action at stage (1) solicits assistance. If this stage is successful, then it will end with a group of agents having some kind of nominal commitment to collective action. This stage is essentially a collective deliberation stage (see the discussion on deliberation in Walton and Krabbe's dialogue types, discussed in

Chapter 7). At the conclusion of this stage, the team will have agreed to the *ends* to be achieved (i.e. to the principle of joint action), but not to the means (i.e. the way in which this end will be achieved). Note that the agents are assumed to be rational, in the sense that they will not form a team unless they implicitly believe that the goal is achievable.

(3) **Plan formation.** We saw above that a group will not form a collective unless they believe they can actually achieve the desired goal. This, in turn, implies there is at least one action known to the group that will take them 'closer' to the goal. However, it is possible that there are many agents that know of actions the group can perform in order to take them closer to the goal. Moreover, some members of the collective may have objections to one or more of these actions. It is therefore necessary for the collective to come to some agreement about exactly which course of action they will follow. Such an agreement is reached via negotiation or argumentation, of exactly the kind discussed in Chapter 7.

(4) Team action. During this stage, the newly agreed plan of joint action is executed by the agents, which maintain a close-knit relationship throughout. This relationship is defined by a convention, which every agent follows. The JPG described above might be one possible convention.

9.6.3 Coordination by mutual modelling

Another approach to coordination, closely related to the models of human teamwork I discussed above, is that of *coordination by mutual modelling.* The idea is as follows. Recall the simple coordination example I gave earlier: you and I are both walking to the door, and there is not enough room for both of us - a collision is imminent. What should we do? One option is for both of us to simply stop walking. This possibility guarantees that no collision will occur, but it is in some sense sub-optimal: while we stand and wait, there is an unused resource (the door), which could fruitfully have been exploited by one of us. Another possibility is for both of us to *put ourselves in the place of the other:* to build a model of other agents - their beliefs, intentions, and the like - and to coordinate our activities around the predictions that this model makes. In this case, you might believe that I am eager to please you, and therefore that I will likely allow you to pass through the door first; on this basis, you can continue to walk to the door.

This approach to coordination was first explicitly articulated in Genesereth *et al.* (1986), where the approach was dubbed 'cooperation without communication'. The models that were proposed were essentially the game-theoretic models that I discussed in Chapter 6. The idea was that if you assume that both you and the other agents with which you interact share a common view of the scenario (in game-theory terms, you all know the payoff matrix), then you can do a game-theoretic analysis to determine what is the rational thing for each player to do.

Note that - as the name of the approach suggests - explicit communication is not necessary in this scenario.

MACE

Les Gasser's MACE system, developed in the mid-1980s, can, with some justification, claim to be the first general experimental testbed for multiagent systems (Gasser *et al.*,1987a,b). MACE is noteworthy for several reasons, but perhaps most importantly because it brought together most of the components that have subsequently become common in testbeds for developing multiagent systems. I mention it in this section because of one critical component: the *acquaintance models,* which are discussed in more detail below. Acquaintance models are representations of other agents: their abilities, interests, capabilities, and the like.

A MACE system contains five components:

- a collection of *application agents,* which are the basic computational units in a MACE system (see below);
- a collection of predefined *system agents,* which provide service to users (e.g. user interfaces);
- a collection of facilities, available to all agents (e.g. a pattern matcher);
- a *description database,* which maintains agent descriptions, and produces executable agents from those descriptions; and
- a set of *kernels,* one per physical machine, which handle communication and message routing, etc.

Gasser *et al.* identified three aspects of agents: they contain knowledge, they sense their environment, and they perform actions (Gasser *et al.,* 1987b, p. 124). Agents have two kinds of knowledge: specialized, local, domain knowledge, and *acquaintance knowledge* - knowledge about other agents. An agent maintains the following information about its acquaintances (Gasser *et al.*,1987b, pp. 126, 127).

Class. Agents are organized in structured groups called *classes*, which are identified by a class name.

Name. Each agent is assigned a name, unique to its class - an agent's address is a $\langle class, name \rangle$ pair.

Roles. A role describes the part an agent plays in a class.

Skills. Skills are what an agent knows are the capabilities of the modelled agent.

Goals. Goals are what the agent knows the modelled agent wants to achieve.

Plans. Plans are an agent's view of the way a modelled agent will achieve its goals.

Agents sense their environment primarily through receiving messages. An agent's ability to act is encoded in its *engine*. An engine is a LISP function, evaluated by default once on every scheduling cycle. The only externally visible signs of

```
((NAME plus-ks)
    (IMPORT ENGINE FROM dbb-def)
    (ACQUAINTANCES
        (plus-ks
            ... model for plus-ks ...
        )
        (de-exp
            [ROLE (ORG-MEMBER)]
            [GOALS ( ... goal list ... )]
            [SKILLS ( ... skill list ... )]
            [PLANS ( ... plan list ...)]
        )
        (simple-plus
            ... acquaintance model for simple-plus ...
        )
    )
    (INIT-CODE ( ... LISP code ...))
) ; end of plus-ks
```

Figure 9.7 Structure of MACE agents.

an agent's activity are the messages it sends to other agents. Messages may be directed to a single agent, a group of agents, or all agents; the interpretation of messages is left to the programmer to define.

An example MACE agent is shown in Figure 9.7. The agent modelled in this example is part of a simple calculator system implemented using the blackboard model. The agent being modelled here is called PLUS-KS. It is a knowledge source which knows about how to perform the addition operation. The PLUS-KS knowledge source is the 'parent' of two other agents; DE-EXP, an agent which knows how to decompose simple expressions into their primitive components, and SIMPLE-PLUS, an agent which knows how to add two numbers.

The definition frame for the PLUS KS agent consists of a name for the agent - in this case PLUS-KS - the engine, which defines what actions the agent may perform (in this case the engine is imported, or inherited, from an agent called DBB-DEF), and the acquaintances of the agent.

The acquaintances slot for PLUS-KS defines models for three agents. Firstly, the agent models itself. This defines how the rest of the world will sec PLUS-KS. Next, the agents DE-EXP and SIMPLE-PLUS are modelled. Consider the model for the agent DE-EXP. The role slot defines the relationship of the modelled agent to the modeller. In this case, both DE-EXP and SIMPLE-PLUS are members of the class denned by PLUS-KS. The GOALS slot defines what the modelling agent believes the modelled agent wants to achieve. The SKILLS slot defines what resources the modeller believes the modelled agent can provide. The PLANS slot defines how the modeller believes the modelled agent will achieve its goals. The PLANS slot consists of a list of skills, or operations, which the modelled agent will perform in order to achieve its goals.

Gasser *et al.* described how MACE was used to construct blackboard systems, a Contract Net system, and a number of other experimental systems (see Gasser *et al.*, 1987b, 1989, pp. 138-140).

i4 Coordination by norms and social laws

In our everyday lives, we use a range of techniques for coordinating activities. One of the most important is the use of *norms* and *social laws* (Lewis, 1969). A norm is simply an established, expected pattern of behaviour; the term social law carries essentially the same meaning, but it is usually implied that social laws carry with them some authority. Examples of norms in human society abound. For example, in the UK, it is a norm to form a queue when waiting for a bus, and to allow those who arrived first to enter the bus first. This norm is not *enforced* in any way: it is simply expected behaviour: diverging from this norm will (usually) cause nothing more than icy looks from others on the bus. Nevertheless, this norm provides a template that can be used by all those around to regulate their own behaviour.

Conventions play a key role in the social process. They provide agents with a template upon which to structure their action repertoire. They represent a behavioural constraint, striking a balance between individual freedom on the one hand, and the goal of the agent society on the other. As such, they also simplify an agent's decision-making process, by dictating courses of action to be followed in certain situations. It is important to emphasize what a key role conventions play in our everyday lives. As well as formalized conventions, which we all recognize as such (an example being driving on the left- or right-hand side of the road), almost every aspect of our social nature is dependent on convention. After all, language itself is nothing more than a convention, which we use in order to coordinate our activities with others.

One key issue in the understanding of conventions is to decide on the most effective method by which they can come to exist within an agent society. There are two main approaches as follows.

Offline design. In this approach, social laws are designed offline, and hardwired into agents. Examples in the multiagent systems literature include Shoham and Tennenholtz (1992b), Goldman and Rosenschein (1993) and Conte and Castelfranchi(1993).

Emergence from within the system. This possibility is investigated in Shoham and ~~Tennenholtz~~ (1992a), ~~Kittock~~ (1993) and Walker and ~~Wooldridge~~ (1995), who experiment with a number of techniques by which a convention can 'emerge' from within a group of agents.

The first approach will often be simpler to implement, and might present the system designer with a greater degree of control over system functionality. However, there are a number of disadvantages with this approach. First, it is not always the case that *all* the characteristics of a system are known at design time. (This is most obviously true of open systems such as the Internet.) In such systems, the ability of agents to organize themselves would be advantageous. Secondly, in complex systems, the goals of agents (or groups of agents) might be constantly changing. To keep reprogramming agents in such circumstances would be costly and inefficient. Finally, the more complex a system becomes, the less likely it is that system designers will be able to design effective norms or social laws: the dynamics of the system - the possible 'trajectories' that it can take - will be too hard to predict. Here, flexibility within the agent society might result in greater coherence.

Emergent norms and social laws

A key issue, then, is how a norm or social law can emerge in a society of agents. In particular, the question of how agents can come to reach a *global* agreement on the use of social conventions by using only *locally available* information is of critical importance. The convention must be *global* in the sense that all agents use it. But each agent must decide on which convention to adopt based solely on its own experiences, as recorded in its internal state: predefined inter-agent power structures or authority relationships are not allowed.

This problem was perhaps first investigated in Shoham and Tennenholtz (1992a), who considered the following scenario, which I will call the *tee shirt game*.

> Consider a group of agents, each of which has two lee shirts: one red and one blue. The agents - who have never met previously, and who have no prior knowledge of each other - play a game, the goal of which is for *all* the agents to end up wearing the same coloured tee shirt. Initially, each agent wears a red or blue tee shirt selected randomly. The game is played in a series of rounds. On each round, every agent is paired up with exactly one other agent; pairs are selected at random. Each pair gets to see the colour of the tee shirt the other is wearing - no other information or communication between the agents is allowed. After a round is complete, every agent is allowed to either stay wearing the same coloured tee shirt, or to swap to the other colour.

Notice that no global view is possible in this game: an agent can never 'climb the wall' to see what every other agent is wearing. An agent must therefore base its decision about whether to change tee shirts or stick with the one it is currently wearing using only its memory of the agents it has encountered on previous rounds. The key problem is this: to design what Shoham and Tennenholtz (1992b) refer to as a *strategy update function,* which represents an agent's decision-making process. A strategy update function is a function from the history that the agent has observed so far, to a colour (red or blue). Note that the term 'strategy' here may be a bit misleading - it simply refers to the colour of the tee shirt. The goal is to develop a strategy update function such that, when it is used by every agent in the society, will bring the society to a global agreement as efficiently as possible.

In Shoham and Tennenholtz (1992b, 1997) and Walker and Wooldridge (1995), a number of different strategy update functions were evaluated as follows.

Simple majority. This is the simplest form of update function. Agents will change to an alternative strategy if so far they have observed more instances of it in other agents than their present strategy. If more than one strategy has been observed more than that currently adopted, the agent will choose the strategy observed most often.

Simple majority with agent types. As simple majority, except that agents are divided into two types. As well as observing each other's strategies, agents in these experiments can communicate with others whom they can 'see', and who are of the same type. When they communicate, they exchange memories, and each agent treats the other agent's memory as if it were his own, thus being able to take advantage of another agent's experiences. In other words, agents are particular about whom they confide in.

Simple majority with communication on success. This strategy updates a form ot communication based on a success threshold. When an individual agent has reached a certain level of success with a particular strategy, he communicates his memory of experiences with this successful strategy to all other agents that he can 'see'. Note, only the memory relating to the successful strategy is broadcast, not the whole memory. The intuition behind this update function is that an agent will only communicate with another agent when it has something *meaningful* to say. This prevents 'noise' communication.

Highest cumulative reward. For this update to work, an agent must be able to see that using a particular strategy gives a particular payoff (cf. the discussion in Chapter 6). The highest cumulative reward update rule then says that an agent uses the strategy that it sees has resulted in the highest cumulative payoff to date.

In addition, the impact of *memory restarts* on these strategies was investigated. Intuitively, a memory restart means that an agent periodically 'forgets' everything it has seen to date - its memory is emptied, and it starts as if from scratch again. The intuition behind memory restarts is that it allows an agent to avoid being over-committed to a particular strategy as a result of history: memory restarts thus make an agent more 'open to new ideas'.

The *efficiency of convergence* was measured by Shoham and Tennenholtz (1992b) primarily by the *time taken to convergence:* how many rounds of the tee shirt game need to be played before all agents converge on a particular strategy. However, it was noted in Walker and Wooldridge (1995) that *changing* from one strategy to another can be expensive. Consider a strategy such as using a particular kind of computer operating system. Changing from one to another has an associated cost, in terms of the time spent to learn it, and so we do not wish to change too frequently. Another issue is that of *stability.* We do not usually want our society to reach agreement on a particular strategy, only for it then to immediately fall apart, with agents reverting to different strategies.

When evaluated in a series of experiments, all of the strategy update functions described above led to the emergence of particular conventions within an agent society. However, the most important results were associated with the highest cumulative reward update function (Shoham and Tennenholtz, 1997, pp. 150, 151). It was shown that, fur any value t such that $0 < t \leqslant 1$, there exists some bounded value n such that a collection of agents using the highest cumulative reward update function will reach agreement on a strategy in n rounds with probability 1 - e. Furthermore, it was shown that this strategy update function is stable in the sense that, once reached, the agents would not diverge from the norm. Finally, it was shown that the strategy on which agents reached agreement was 'efficient', in the sense that it guarantees agents a payoff no worse than that they would have received had they stuck with the strategy they initially chose.

Offline design of norms and social laws

The alternative to allowing conventions to emerge within a society is to design them offline, before the multiagent system begins to execute. The offline design of social laws is closely related to that of mechanism design, which I discussed in Chapter 7 in the context of protocols for multiagent systems, and much of the discussion from that chapter applies to the design of social laws.

There have been several studies of offline design of social laws, particularly with respect to the computational complexity of the social law design problem (Shoham and Tennenholtz, 1992b, 1996). To understand the way these problems are formulated, recall the way in which agents were defined in Chapter 2, as functions from runs (which end in environment states) to actions:

$$Ag : \mathcal{R}^E \to Ac.$$

A constraint is then a pair

$$\langle E', \alpha \rangle,$$

where

- $E' \subseteq E$ is a set of environment states; and
- α e Ac is an action.

The reading of a constraint $\langle E', \alpha \rangle$ is that, if the environment is in some state e $\in E'$, then the action α is forbidden. A *social law* is then defined to be a set sl of such constraints. An agent - or plan, in the terminology of Shoham and Tennenholtz (1992b, p. 279) - is then said to be legal with respect to a social law sl *if* it never attempts to perform an action that is forbidden by some constraint *in* sl.

The next question is to define what is meant by a *useful* social law. The answer is to define a set $F \subseteq E$ of *focal states*. The intuition here is that these are the states that are *always legal,* in that an agent should always be able to 'visit' the focal states. To put it another way, whenever the environment is in some focal state $e \in F$, it should be possible for the agent to act so as to be able to guarantee that any other state $e' \in F$ is brought about. A useful social law is then one that does not constrain the actions of agents so as to make this impossible.

The *useful social law problem* can then be understood as follows.

> Given an environment $Env = \langle E, \tau, e_0 \rangle$ and a set of focal states $F \subseteq E$, find a useful social law if one exists, or else announce that none exists.

In Shoham and Tennenholtz (1992b, 1996), it is proved that this problem is NP-complete, and so is unlikely to be soluble by 'normal' computing techniques in reasonable time. Some variations of the problem are discussed in Shoham and Tennenholtz (1992b, 1996), and some cases where the problem becomes tractable are examined. However, these tractable instances do not appear to correspond to useful real-world cases.

Social laws in practice

Before leaving the subject of social laws, I will briefly discuss some examples of social laws that have been evaluated both in theory and practice. These are *traffic laws* (Shoham and Tennenholtz, 1996).

Imagine a two-dimensional grid world - rather like the Tileworld introduced in Chapter 2 - populated by mobile robots. Only one robot is allowed to occupy a grid point at any one time - more than one is a collision. The robots must collect and transport items from one grid point to another. The goal is then to design a social law that prevents collisions. However, to be useful in this setting, the social law must not impede the movement of the robots to such an extent that they are unable to get from a location where they collect an item to the delivery location. As a first cut, consider a law which completely constrains the movements of robots, so that they must all follow a single, completely predetermined path, leaving no possibility of collision. Here is an example of such a social law, from Shoham and Tennenholtz (1996, p. 602).

> Each robot is required to move constantly. The direction of motion is fixed as follows. On even rows each robot must move left, while in odd rows it must move right. It is required to move up when it is in the rightmost column. Finally, it is required to move down when it is on either the leftmost column of even rows or on the second rightmost column of odd rows. The movement is therefore in a 'snake-like' structure, and defines a Hamiltonian cycle on the grid.

It should be clear that, using this social law,

- the next move of an agent is *uniquely* determined: the law does not leave any doubt about the next state to move to;
- an agent will always be able to get from its current location to its desired location,
- to get from the current location to the desired location will require at most $O(n^2)$moves, where n is the size of the grid (to see this, simply consider the dimensions of the grid).

Although it is effective, this social law is obviously not very efficient: surely there are more 'direct' social laws which do not involve an agent moving around all the points of the grid? Shoham and Tennenholtz (1996) give an example of one, which superimposes a 'road network' on the grid structure, allowing robots to change direction as they enter a road. They show that this social law guarantees to avoid collisions, while permitting agents to achieve their goals much more efficiently than the naive social law described above.

9.7 Multiagent Planning and Synchronization

An obvious issue in multiagent problem solving is that of *planning* the activities of a group of agents. In Chapter 4, we saw how planning could be incorporated as a component of a practical reasoning agent: what extensions or changes might be needed to plan for a team of agents? Although it is broadly similar in nature to 'conventional' planning, of the type seen in Chapter 4, multiagent planning must take into consideration the fact that the activities of agents can interfere with one another - their activities must therefore be coordinated. There are broadly two possibilities for multiagent planning as follows (Durfee, 1999, p. 139).

Centralized planning for distributed plans: a centralized planning system develops a plan for a group of agents, in which the division and ordering of labour is defined. This 'master' agent then distributes the plan to the 'slaves', who then execute their part of the plan.

Distributed planning: a group of agents cooperate to form a centralized plan. Typically, the component agents will be 'specialists' in different aspects of the overall plan, and will contribute to a part of it. However, the agents that form the plan will not be the ones to execute it; their role is merely to generate the plan.

Distributed planning for distributed plans: a group of agents *cooperate* to form individual plans of action, dynamically coordinating their activities along the way. The agents may be self-interested, and so, when potential coordination problems arise, they may need to be resolved by negotiation of the type discussed in Chapter 7.

In general, centralized planning will be simpler than decentralized planning, because the 'master' can take an overall view, and can dictate coordination relationships as required. The most difficult case to consider is the third. In this case, there may never be a 'global' plan. Individual agents may only ever have pieces of the plan which they are interested in.

Plan merging

Georgeff (1983) proposed an algorithm which allows a planner to take a set a plans generated by single agents, and from them generate a conflict free (but not necessarily optimal) multiagent plan. Actions are specified by using a generalization of the STRIPS notation (Chapter 4). In addition to the usual precondition-delete-add lists for actions, Georgeff proposes using a *during* list. This list contains a set of conditions which must hold *while* the action is being carried out. A plan is seen as a set of states; an action is seen as a function which maps the set onto itself. The precondition of an action specifies the domain of the action; the add and delete lists specify the range.

Given a set of single agent plans specified using the modified STRIPS notation, generating a synchronized multiagent plan consists of three stages.

(1) Interaction analysis. Interaction analysis involves generating a description of how single agent plans interact with one another. Some of these interactions will be harmless; others will not. Georgeff used the notions of *satisfiability, commutativity,* and *precedence* to describe goal interactions. Two actions are said to be *satisfiable* if there is some sequence in which they may be executed without invalidating the preconditions of one or both. *Commutativity* is a restricted case of satisfiability: if two actions may be executed in parallel, then they are said to be commutative. It follows that if two actions are commutative, then either they do not interact, or any interactions are harmless. Precedence describes the sequence in which actions may be executed; if action α_1 has precedence over action α_2, then the preconditions of α_2 are met by the postconditions of α_1. That is not to say that α_1 *must* be executed before α_2; it is possible for two actions to have precedence over each other.

Interaction analysis involves searching the plans of the agents to detect any interactions between them.

(2) **Safety analysis.** Having determined the possible interactions between plans, it now remains to see which of these interactions are *unsafe.* Georgeff defines safeness for pairs of actions in terms of the precedence and commutativity of the pair. Safety analysis involves two stages. First, all actions which are harmless (i.e. where there is no interaction, or the actions commute) are removed from the plan. This is known as simplification. Georgeff shows that the validity of the final plan is not affected by this process, as it is only *boundary* regions that need to be considered. Secondly, the set of all harmful interactions is generated. This stage also involves searching; a rule known as the *commutativity theorem*

is applied to reduce the search space. All harmful interactions have then been identified.

(3) **Interaction resolution.** In order to resolve conflicts in the simplified plan, Georgeff treats unsafe plan interactions as *critical sections;* to resolve the conflicts, mutual exclusion of the critical sections must be guaranteed. To do this, Georgeff used ideas from Hoare's CSP paradigm to enforce mutual exclusion, although simpler mechanisms (e.g. semaphores) may be used to achieve precisely the same result (Ben-Ari, 1993).

Stuart (1985) describes an implemented system which bears a superficial resem-~~blance~~ to ~~Georgeff's~~ algorithm. It takes a set of ~~unsynchronized~~ single agent plans and from them generates a synchronized multiagent plan. Like Georgeff's algorithm, Stuart's system also guarantees a synchronized solution if one exists. Also, the final plan is represented as a sequence of actions interspersed with CSP primitives to guarantee mutual exclusion of critical sections (Hoare, 1978). Actions are also represented using an extended STRIPS notation. There, however, the ~~resem-~~blance ends. The process of determining which interactions are possibly harmful and resolving conflicts is done not by searching the plans, but by representing the plan as a set of formulae of temporal logic, and attempting to derive a synchronized plan using a temporal logic theorem prover. The idea is that temporal logic is a language for describing sequences of states. As a plan is just a description of exactly such a sequence of states, temporal logic could be used to describe plans. Suppose two plans, π_1 and π_2, were represented by temporal logic formulae φ_1 and φ_2, respectively. Then if the conjunction of these two plans is satisfiable - if there is some sequence of events that is compatible with the conjunction of the formulae - ~~then~~ there is some way that the two plans could be concurrently executed. The temporal logic used was very similar to that used in the Concurrent MetateM language discussed in Chapter 3.

The algorithm to generate a synchronized plan consists of three stages.

(1) A set of single agent plans are given as input. They are then translated into a set of formulae in a propositional linear temporal logic (LTL) (Manna and Pnueli, 1992, 1995).

(2) The formulae are conjoined and fed into an LTL theorem prover. If the conjoined formula is satisfiable, then the theorem prover will generate a set of sequences of actions which satisfy these formulae. These sequences are encoded in a graph structure. If the formula is not satisfiable, then the theorem prover will report this.

(3) The graph generated as output encodes all the possible synchronized executions of the plans. A synchronized plan is then 'read off' from the graph structure.

In general, this approach to multiagent synchronization is computationally expensive, because the temporal theorem prover has to solve a PSPACE-complete problem.

Notes and Further Reading

Published in 1988, Bond and Gasser's *Readings in Distributed Artificial Intelligence* brings together most of the early classic papers on CDPS (Bond and Gasser, 1988). Although some of the papers that make up this collection are perhaps now rather dated, the survey article written by the editors as a preface to this collection remains one of the most articulate and insightful introductions to the problems and issues of CDPS to date. Victor Lesser and his group at the University of Massachusetts are credited with more or less inventing the field of CDPS, and most innovations in this field to date have originated from members of this group over the years. Two survey articles that originated from the work of Lesser's group provide overviews of the field: Durfee *et al.* (1989a,b). Another useful survey is Decker *et al.* (1989).

The Contract Net has been hugely influential in the multiagent systems literature. It originally formed the basis of Smith's doctoral thesis (published as Smith (1980b)), and was further described in Smith (1980a) and Smith and Davis (1980). Many variations on the Contract Net theme have been described, including the effectiveness of a Contract Net with 'consultants', which have expertise about the abilities of agents (Tidhar and Rosenschein, 1992), and a sophisticated variation involving marginal cost calculations (Sandholm and Lesser, 1995). Several formal specifications of the Contract Net have been developed, using basic set theoretic/first-order logic constructs (Werner, 1989), temporal belief logics (Wooldridge, 1992), and the Z specification language (d'Inverno and Luck, 1996).

In addition to the model of cooperative action discussed above, a number of other similar formal models of cooperative action have also been developed, the best known of which is probably the Shared Plans model of Barbara Grosz and Sarit Kraus (Grosz and Kraus, 1993, 1999); also worth mentioning is the work of Tuomela and colleagues (Tuomela and Miller, 1988;Tuomela, 1991), Power (1984), and Rao and colleagues (Rao *et al,* 1992; Kinny *et al,* 1992).

A number of researchers have considered the development and exploitation of norms and social laws in multiagent systems. Examples of the issues investigated include the control of aggression (Conte and Castelfranchi, 1993), the role of social structure in the emergence of conventions (Kittock, 1993), group behaviour (Findler and Malyankar, 1993), and the reconsideration of commitments (Jennings, 1993a). In addition, researchers working in philosophy, sociology, and economics have considered similar issues. A good example is the work of Lewis (1969), who made some progress towards a (non-formal) theory of normative behaviour.

One issue that I have been forced to omit from this chapter due to space and time iimitations is the use of *normative specifications* in multiagent systems, and, in particular, the use of *deontic logic* (Meyer and Wieringa, 1993). Deontic logic is the logic of obligations and permissions. Originally developed within formal philosophy, deontic logic was been taken up by researchers in computer science in order to express the desirable properties of computer systems. Dignum (1999) gives an overview of the use of deontic logic in multiagent systems, and also discusses the general issue of norms and social laws.

Class reading: Durfee (1999). A detailed and precise introduction to distributed problem solving and distributed planning, with many useful pointers into the literature.