Although these projects are interesting, I think specifications in terms of mental states pose a major problem. If we assume that communication standards like KQML are essentially intended to allow agents of various kinds to communicate with each other, then any theory based on mental states compels agents to conform to a specific architecture, which implements this theory precisely. For example, the theory of cognitons which we developed in Chapter 5 does not necessarily agree with that of Cohen and Levesque, which does not utilise the same primitives as that of Y. Shoham. For this reason, any communication standard based on the way an agent behaves restricts the integration options for agents which are really of different kinds. It therefore appears that if communications are no longer specified in terms of mental states but in terms of protocols, this has the advantage that the inherent nature of agents can be set aside, by focusing on the relationships which exist between communications. I may add that this is the approach which has been introduced in this chapter and which will be developed in the following chapters.

# 7 Collaboration and Distribution of Tasks



If cooperation offers advantages in terms of quantitative efficiency and the emergence of quality, as we saw in Chapter 2, it also raises problems concerned with the distribution of tasks between agents. These problems are difficult, for they involve several parameters: agents' cognitive capacities and capacities for commitment; skills of the individuals concerned; nature of tasks; efficiency; cost of sending a message; social structures within which agents move, etc.

For this reason, the distribution of tasks and resources is both one of the major domains of multi-agent systems and one of their principal contributions to computer science in general. By emphasising distributed allocation of tasks and the concepts of contracts and commitments, multi-agent systems pose the problem of activity in terms which are both social and computational. This distinguishes them from the more classical forms used earlier, whether we are dealing with the allocation of machines or with the distribution of processes on processors. As we shall see, task allocation techniques use some of the results obtained from the field of operational research and distributed systems, while adapting them to their needs.

Distributing tasks, data and resources comes down to answering question: *who has to do what, and using what resources, on the basis of the ls and the skills of the agents and the contextual constraints (type and ntity of resources, nature of environment, etc.)*. With the coordination of ons, which we shall examine in Chapter 8, here is one of the main ıes concerning the organisational function of a society of agents Chapter 3).

## 7.1 Modes of task allocation

Task allocation (or distribution) involves the definition of the organisational mechanisms through which agents can combine their skills to perform collective work. So we need to describe how tasks are allocated, knowing that the capacities of an agent depend as much on its intrinsic aptitudes (its available architectures, its cognitive capabilities and the types of communication) and the power resources available to it (computation time and energy level) as on external resources (tools, power sources) and environmental constraints.

Tasks requiring more resources, work or know-how than a single agent is capable of supplying should first be *broken down* into several sub-tasks, then *distributed* (or allocated) among the various agents. These two operations are obviously related, since the breakdown of tasks often has to take account of the skills of the agents, and so make the subsequent distribution easier.

### 7.1.1 Criteria for breaking down tasks

Bond, Gasser and Hill (Bond and Gasser, 1988; Gasser and Hill, 1990) introduced some criteria for breaking down problems. In particular, they indicate that, if problems can be naturally analysed by level of abstraction (working from the most general to the most detailed), it is also necessary to take account of other constraints such as control, data or resources. In fact, it is necessary to make tasks as independent of one another as possible, in such a way as to reduce the coordination required. In particular, we must try to minimise the quantity of data that tasks have to send to each other, and to make sure that tasks can make use of local resources, in order to diminish conflicts in connection with resources.

While breaking down activities are as important as those linked to distribution, the former are generally carried out by human beings. As far as we know, there are no computing systems that can automatically break down tasks. For this reason, research has essentially been concerned with the automatic distribution of tasks, and has rather left aside the problem of breaking them down into sub-tasks.



Nevertheless, we have described the various ways of approaching the breaking down of tasks (using object-oriented or function-oriented approaches in Chapter 3.

### 7.1.2 Roles

An automatic task allocation system should be capable of connecting agents needing information, or task realisation, the *clients*, and agents capable of supplying a service, the *suppliers* or *servers*. The same agents are frequently clients and servers at the same time, their status as clients or suppliers being generally determined in a dynamic fashion during the execution of a multi-agent system. We then say that an agent is taking the *role* of a client or the *role* of a supplier, without this being one of its intrinsic characteristics. Other roles can be defined, such as that of a broker or trader, which brings servers and clients into contact.

### 7.1.3 Forms of allocation

The breakdown of tasks can be managed by centralising the allocation process or by distributing it among all the agents concerned. In a *centralised allocation* mode, two cases may arise:

(1) If the subordination structure is hierarchical, it will be the superior agent that will order a subordinate to carry out the task. We can then speak of rigid or defined allocation. This allocation mode, characteristic of calling up a procedure in classic programming, will not be studied here. This type of distribution is met only in fixed organisations (Chapter 3).

(2) In contrast, if the structure is egalitarian, distribution then involves the definition of special agents – traders or brokers – which manage all the allocation procedures by centralising the requests from the clients and the bids for service from the servers, in order to match up these two categories of agents. This makes it possible to apply centralised modes to variable organisations.

In a *distributed allocation* mode, each agent individually tries to obtain the services it needs from the suppliers. These modes apply only to variable organisations, that is, to those which assume that the relationship between clients and suppliers can evolve over time without calling the general structure of the organisation into question. We can distinguish two distributed allocation mechanisms in predefined organisations:

(1) The *acquaintance network* mode of allocation assumes that clients have a representation of other agents and of their capabilities. We shall see that it is not necessary for agents to know the capabilities of all the other agents to be able to solve their problem, but only that the network formed by the set of contacts should be strongly connected (that is, that there is a path going from any agent to any other agent) and, obviously, coherent (that is, the

representations concerning the skills of an agent's acquaintances actually correspond to its effective skills).

(2) The *request for bids* mode of allocation is better known in distributed artificial intelligence under the name of *contract net*. It has the advantage of being very dynamic, and has proved to be particularly easy to implement. However, we shall see that, when it is implemented, problems inherent in the particularly dynamic nature of this mode have to be borne in mind.

To these modes, which are applied to predefined organisations, we should add the *emergent allocation* mode, less known in general and more characteristic of reactive systems in which communications take place in an incidental manner (cf. Chapter 6) by propagation of stimuli. Figure 7.1 summarises the principal task allocation modes. Obviously, a great many variants of these approaches can exist; in particular, there can be hybrids that try to synthesise the advantages of different approaches.

## 7.2 Centralised allocation of tasks by trader

The simplest case is that where only a single trader exists, with an acquaintance table indicating to it all the agents capable of carrying out a particular task T. This table can be updated directly by suppliers, which tell what their skills are when entering the system.

Figure 7.2 illustrates how the procedure works. When agent A needs to carry out a task T which it cannot (or does not want to) carry out itself, it asks the trader

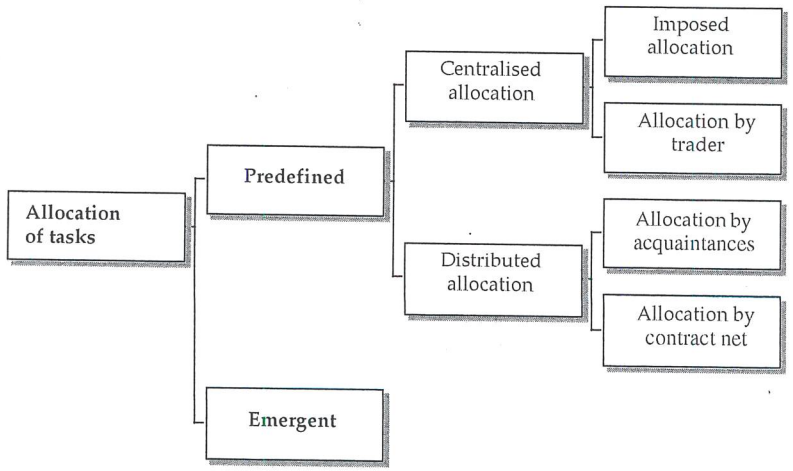


**Figure 7.1** Principal task distribution modes.



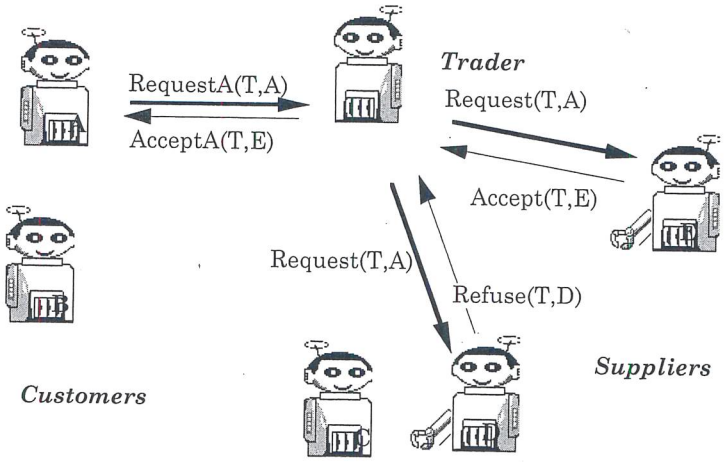


**Figure 7.2** Centralised allocation of tasks.

to find an agent to carry out T. The trader then turns to the agents which are known to it to have the capability to carry out T. If one of them accepts, the trader sends the acknowledgement of acceptance to the client, otherwise it informs the client that it has not found anyone to carry out the task.

We are assuming in this process that a supplier which agrees to carry out a task is effectively committing itself to doing it, and will not back out when the contract is being concluded with a client, or while the task is being carried out. We shall also work on the hypothesis that the trader has a table of skills of the agents which is both complete and coherent, all agents A which have a skill being actually listed in this table, and that all references in this table are correct; in other words, for all pairs <A,C> in the table, agent A actually has skill C.

With these elements, the allocation protocol can be defined in the form of a BRIC element. We shall model only the trader, since the behaviour of the clients and suppliers follows the structures of requests (`Request`) and responses described in Chapter 6 on speech acts. This module, which entirely defines the behaviour of the trader, takes the client's requests and the supplier's responses at the input, and produces at the output requests for suppliers likely to support this request, together with responses for the clients. The communications between the trader, the clients and the suppliers use a set of messages corresponding to the following speech acts:

**RequestA(T,X)** is a request from client X to the trader for task T to be carried out.

**AcceptA(T,Y)** is the response from the trader to the client to indicate to it that agent Y undertakes to carry out task T.

**Impossible(T)** is a response from the trader to the client to indicate to it that there are no agents which agree to carry out task T.

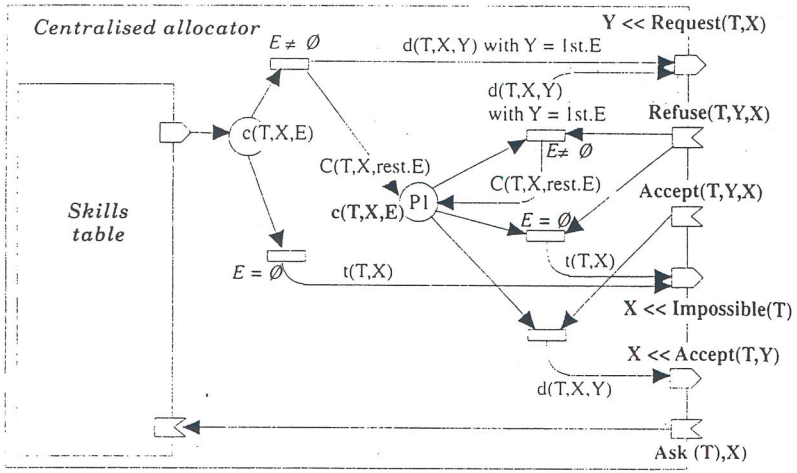


**Figure 7.3** Description of centralised trader in BRIC.

**Request(T,X)** is a request from the trader for a supplier to carry out task P for client X.

**Accept(T,Y,X)** is the positive response of supplier Y to the trader's request. The supplier indicates to the client that it agrees to carry out task T for client X.

**Refuse(T,Y,X)** is the negative response of supplier Y to the request for allocation of task T for client X.

Figure 7.3 shows the representation of the centralised trader in BRIC.

When the trader receives a request on behalf of a client X to do T, it looks in its skills table and makes out a list of agents (which may be sorted in order of preference) capable of carrying out this task. If this set E is empty, then the trader responds with the message `Impossible`. Otherwise, it iteratively asks all the elements of E if it is possible for them to do T, using the location $P_1$ as a memory for all the suppliers still to be examined. If one of them accepts, the location $P_1$ is emptied of this task request, and the trader responds to the client X that supplier Y agrees to carry out T. This module uses the following three data structures:

**s(T,X,E)** gives the list of agents, E, having the skill to carry out the task, T, for the client. `1st.E` sends back the first element in the list (identical to `car` in Lisp) and `rest.E` sends back the list, E, without its first element (identical to `cdr` in Lisp).

**r(T,X,Y)** requests the trader for supplier Y to ask it to carry out task T for X.

**t(T,X)** task T requested by X.

It is possible to improve this mechanism in several ways. The first is to optimise the allocation by priority selection of the most skilled agents, or those that minimise a



cost function. For example, in the case of ore-gathering robots including driller and transporter robots, it is preferable to distribute transport tasks on the basis of the distance separating the transporters from the drillers, in such a way as to minimise the movements to be carried out. We can envisage two cases:

(1) If the trader knows the evaluation functions used by the suppliers, it can sort out the skilled agents on the basis of the results of the evaluation and carry out its requests in this order. The protocol is not modified. It is merely necessary to assume that the trader's acquaintances manager sends back the set of suppliers sorted in order of preference.

(2) If the trader does not know the evaluation functions, it should first ask each agent to formulate its proposition before actually selecting the one that is to be considered the best. This technique then resembles a contract net mechanism (cf. Section 7.3.2).

In our algorithm, the trader runs through the set of agents in a sequential manner, sending a request to the next agent only if the previous one asked has refused to carry out the task. It is also possible to define parallel algorithms. In this case, as soon as a supplier indicates that it agrees to carry out the task, the other suppliers are informed, so that they know it is no longer necessary to respond. This type of management is more precise, for it must be taken into account that agents can send in their acceptances in between times, and that they therefore must be capable of updating their commitments.

The interesting point about centralised allocation is that we can update the set of agents and their respective skills, and therefore favour the coherence of the system. In addition, the requirements for optimisation are more easily satisfied, the trader knowing the set of agents available, and it is easier for it to select the 'best' of the agents in relation to a given task request.

But the major drawback of this type of system is that it creates a bottleneck, and so considerably reduces the performance levels of the system as soon as the number of agents and requests increases. In fact, the number of messages the trader has to manage grows as the square of $N$, where $N$ is the number of agents. If we consider that a rate, $\alpha$, of potential clients exists ($\alpha$ is the number of clients/$N$ ratio) which makes $k$ requests per unit of time, and if we use $\beta$ to refer to the rate of potential suppliers (that is, the number of suppliers/$N$ ratio) for each request, then the number of messages $M$ is equal to:

$$M = \alpha k N(2+2\beta N)$$

that is, to the product of the number of requests times the total number of messages which exist – on the one hand, between the clients and the trader, and on the other hand, between the trader and the suppliers. For example, if $\alpha$ equals 0.5 (half the agents are making requests), $\beta$ equals 0.2 (20% of the agents are potential suppliers for each request, which means that the system is relatively redundant) and $k$ equals 1 (there is one request per unit of time), we get the curve in Figure 7.4.

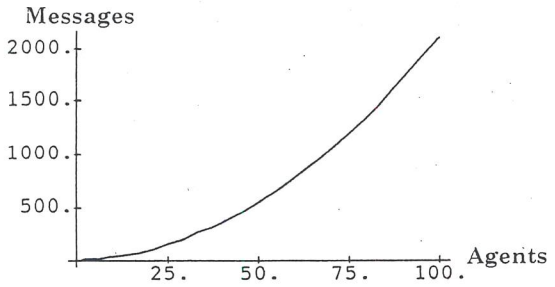


**Figure 7.4** Number of messages processed by the trader, plotted against the number of agents in the system.

It can be observed that if, for 50 agents, the number of messages to be processed by the trader is in the order of 500, it changes to 2000 for 100 agents, which indicates that this method is not to be recommended for a large number of agents. In addition, if the system is distributed, all the messages must go via the trader, without taking account of the structure of the network.

Finally, a centralised system is very sensitive to failures. In effect, if the trader breaks down, the entire system falls to pieces. It is therefore necessary to provide complex mechanisms of assistance to try to avoid these problems, but that goes beyond the scope of this book. It is possible to improve this situation by using several traders. But this raises other problems, such as managing coherence between the different traders, which then becomes very complicated. When it is necessary for the system to distribute its allocation mechanisms, it is preferable to go to a true distributed task allocation mode.

## 7.3 Distributed allocation of tasks

### 7.3.1 Acquaintance network allocation

In systems of acquaintance network allocation, it is assumed that each agent has a skills table for the agents it knows, in the form of a dictionary such as this:

$$\{C_1:[A_{11},...,A_{1k}],...,C_n:[An_1,...,An_1]\}$$

where the $C_i$ are the skills required to carry out a task $T_i$ and the $A_{ij}$ are the agents that know how to carry out this type of task. It is possible to represent this table in the form of a local table for each agent, with the rows showing the skills and the columns its acquaintances, that is, the agents it knows. Each square of the table then displays 1 or 0, depending on whether the agent in the column in question does or does not have the skill for the corresponding line. For example, an agent A which knows that it has skill $C_3$, that skill $C_1$ is available from B, and that D knows how to do $C_3$, has the following skills table:

| | A | B | C | D |
|---|---|---|---|---|
| C1 | 0 | 1 | 1 | 0 |
| C2 | 0 | 0 | 1 | 0 |
| C3 | 1 | 0 | 0 | 1 |

It is not assumed that each agent knows all the skills of the other agents, as this would violate the hypothesis of partial representation for multi-agent systems, and would place much too great a constraint on the creation and updating of such systems. We simply assume that the acquaintances tables are correct, that is, that their indications do correspond to the agents' real skills. We also adopt the hypothesis that the acquaintances tables of each of the agents are given once and for all and are not updated. We shall see, in Section 7.3.1, how to deal with the problem of updating acquaintance networks.

An acquaintance network can be represented in the form of a graph. The agents are the tips of the nodes and the arcs represent the skills which the agents know other agents have. In this way, this graph superimposes the various local acquaintances tables of the agents. For example, Figure 7.5 represents the superimposition of the following tables:

| A | A | B | C | D |
|---|---|---|---|---|
| | C1 | 0 | 1 | 1 | 0 |
| | C2 | 0 | 0 | 1 | 0 |
| | C3 | 1 | 0 | 0 | 1 |

| B | A | B | C | D |
|---|---|---|---|---|
| | C1 | 0 | 1 | 0 | 0 |
| | C2 | 0 | 0 | 0 | 1 |
| | C3 | 1 | 0 | 0 | 0 |

| C | A | B | C | D |
|---|---|---|---|---|
| | C1 | 0 | 1 | 1 | 0 |
| | C2 | 0 | 0 | 1 | 0 |
| | C3 | 0 | 0 | 0 | 0 |

| D | A | B | C | D |
|---|---|---|---|---|
| | C1 | 0 | 0 | 0 | 0 |
| | C2 | 0 | 0 | 1 | 1 |
| | C3 | 0 | 0 | 0 | 1 |

There are essentially two modes of acquaintance network allocation, depending on whether agents are authorised to delegate their requests to other agents or not. So we shall refer to *direct allocation* or *allocation by delegation* modes.

#### *Direct allocation*

In the direct mode, an agent can have a task carried out only by an agent that it knows directly. The distribution mechanism is very simple, and is reminiscent of the behaviour of the single trader in the case of centralised distribution systems. The agent that wants to have a task carried out applies in turn to each of the agents it knows which have the desired skill until one of them accepts. If none of them accept, we reach a special state, which must be dealt with. Either we can consider that there is a fault in the system – but we must then make sure that in all cases there will be at least one agent which will agree to carry out the task – or we take measures tending to bypass the problem, by allocating a task to an agent in an authoritarian manner, or by recommencing operations through a contract net

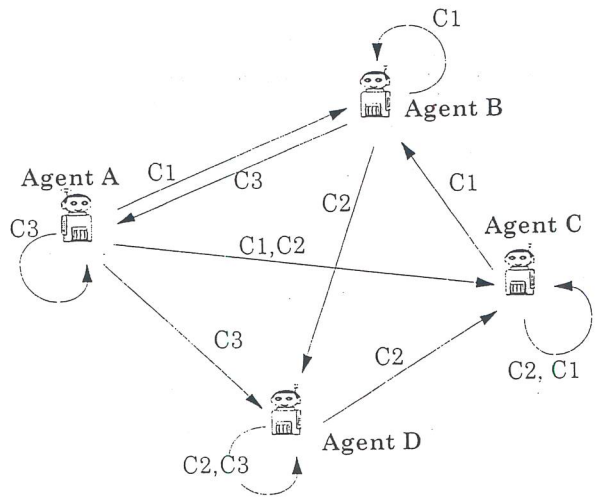


**Figure 7.5** An acquaintance network superimposes all the local acquaintances tables in a single diagram, which represents what each agent knows about the skills of the other agents.

mechanism making it possible to select the agent that has the fewest reasons to refuse this task. If this fails, we can also appeal to a centralised system to allocate the task. The reservations associated with centralised systems are no longer appropriate in this case, as only a small number of requests will be sent to the centralised system, with usual requests going by way of the acquaintance network. In this case, the centralised system, which plays the role of a maintenance and repair agent rather than that of a real trader, proves to be very useful, for it can also serve to update the agents' acquaintances tables and thus reorganise the acquaintance network in order to accelerate the allocation process.

Here this allocation mechanism is not implemented by an agent distinct from the applicants but by a module directly integrated into the organisational system of the potential clients. There are therefore two types of connections: those that connect the allocation module to the motivational and organisational systems, and those that represent the messages which the client exchanges with other agents. The internal connections are defined by the following messages:

**Allocate(T)** is a request from the motivational system to the allocation system for the latter to find someone to carry out task T.

**Impossible(P)** is a response from the allocation manager to the motivational system of the agent which indicates that there is no agent which agrees to carry out task T.

**Committed(Y,T)** is a response from the allocation module to the organisational module indicating that agent Y has undertaken to carry out task T.

External messages are defined by the following speech acts:

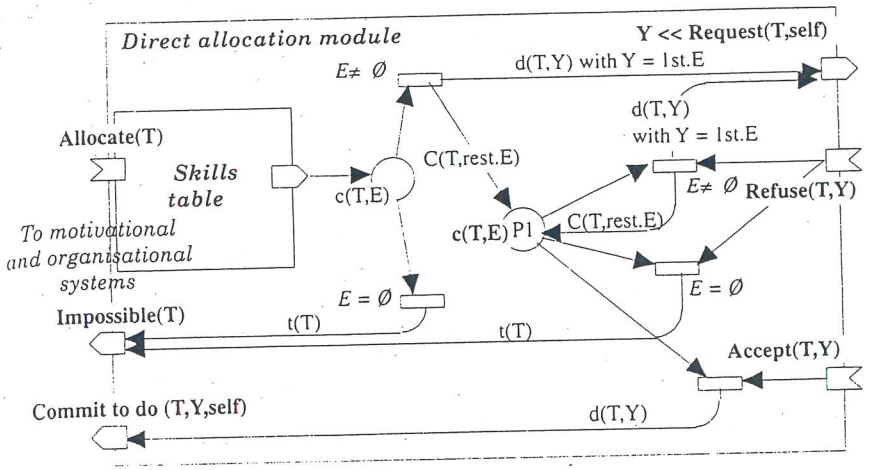


**Figure 7.6** Representation of trader's request module using direct contacts.

**Request(T,self)** is a request from the agent to a potential supplier agent for task T to be carried out.

**Accept(T,Y)** is the positive response of the supplier Y to the client's request.

**Refuse(T,Y)** is the negative response from the supplier Y to the request for allocation of task T.

The structure of the direct allocation module is represented in Figure 7.6. It returns to the main lines of the centralised allocation structure of the preceding section. It differs essentially at two points:

(1) The parameter X, which previously represented the client, has disappeared from the trader's internal data structures.

(2) The results – conveying acceptance or the impossibility of finding an agent – are sent back to the motivational and organisational systems of the client.

As in the case of the centralised allocation system, the order in which the agents are questioned is not necessarily neutral. Several possibilities can be considered, giving priorities to agents or involving a computation taking account of an agent's processing load and the characteristics of the task. The advantage of this allocation mode obviously lies in its simplicity. It simply assumes that clients have the capacity to know enough specialists for the tasks they cannot carry out themselves to be carried out by others. However, it turns out to be much too limited, as it cannot put suppliers into contact which are not direct acquaintances of clients. It is therefore necessary to introduce a mechanism which allows several agents to make indirect contact.

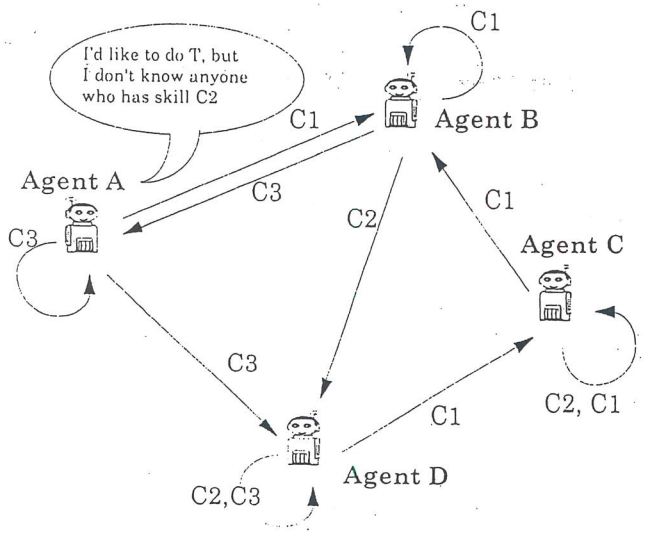


Figure 7.7 The problem of allocation by delegation in an acquaintance network.

*Allocation by delegation*

The technique of allocation by delegation makes it possible to link together clients and suppliers that do not know each other directly. In effect, this allocation mode allows a supplier which is asked to carry out a task to send it to another agent if it is not capable of doing it. For example, in Figure 7.7, agent A knows no one capable of carrying out a task which calls for skill C2. It therefore has to ask all the agents it knows to ask their own acquaintances with skill C2 to carry out the task T requested, this process having to be repeated until an agent is found which agrees to carry out this task, or until the entire network has been covered.

A search procedure of this kind is carried out on the basis of parallel graph search algorithms. There are essentially two approaches: searching in depth and searching across width. We are talking of algorithms which are very well known in distributed algorithmics (we could refer, for example, to Raynal (1987)), but they have to be adapted to task allocation.

Here we shall describe only the technique based on parallel searching, leaving it to the reader to translate its principles to the case of searching in depth. The general technique of parallel searching rests on diffusion algorithms which recursively propagate requests to all known agents except the requesting agent itself. However, attention must be paid to two constraints. It is first necessary, to verify that an agent is asked only once to carry out a task T and, secondly, to make sure that the procedure is completed, so that it may, if necessary, be envisaged that task T can be allocated in an authoritarian manner or a request for bids mode can be used. The first difficulty is resolved by marking the agents during the search, and the second



by using a supplementary message `Acknowledge`, which is used to inform the agents upstream of the process of the fact that all the requests situated downstream of it have actually been carried out.

Each agent Z, when it receives an initial request from an agent Y concerning the carrying out of a task T for an agent X and requiring a skill C, first evaluates the task. If it has the skill and it accepts, it sends agent X an acceptance message, then sends agent Y an acknowledge message, to indicate the end of the process. Otherwise, it marks itself as having already analysed the request, and relaunches the request to all the agents it knows. When all have responded with an acknowledge message, it sends one to itself. When new requests are sent to it for the same task T it then simply sends an acknowledge message.

It is also necessary to manage the option of having several simultaneous acceptances. Owing to parallelism, two agents can actually accept the request and commit themselves to agent X. For this reason, the supplier which accepts does not have to commit itself, but simply has to book its commitment until the contract is concluded with the client. On the other hand, the client has to take account of the fact that it can receive several propositions, and it is assumed that it must accept only the first. Figure 7.8 illustrates the operation of such a system.

When agent A needs to have task T carried out, it asks all its acquaintances, B and C, to do T. As they cannot do it themselves, they delegate the request, to D and E for B, and to B and F for C. For this reason, B receives a new request, and merely sends an `Acknowledge` message to agent C. Since D cannot do it and has no acquaintances, it merely sends back the `Acknowledge` message to B. E, on the other hand, can delegate the request to F. Let us assume that F knows how to carry

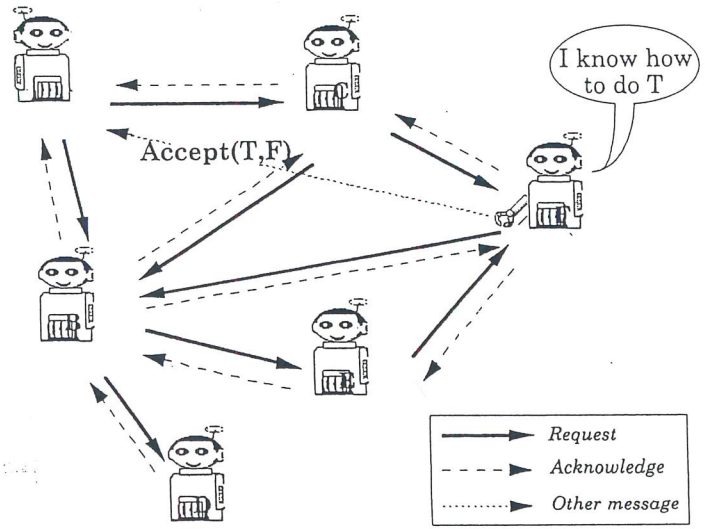


Figure 7.8 A system of allocation by delegation.

out task T and that it receives the first request from B. It sends A its agreement to do T, and it sends B an acknowledge message. When F receives a new request from C, it merely responds by an acknowledge message. If several acceptance messages reach A, it takes account only of the first, by noting that the task it has requested is under contract, that is, that F has committed itself to do it. The speech acts required for communications between agents are as follows:

**Request(P,X,Y)** is a request from client X to the potential supplier for task P to be carried out, this request coming from applicant Y.

**Propose(T,Y)** is the positive response from supplier Y to the client's request. It indicates that it is booking T and is waiting for either an agreement message or a refusal message.

**Acknowledge(T,Y)** indicates either that agent Y agrees to do P or that it has received an acknowledge message from all the agents which it requested to do T.

**OfferAccepted(T,X)** is a positive response from client X to the acceptance proposition from the supplier. The supplier can then commit itself to do P for X.

**OfferRefused(T,X)** is a negative response from client X to the acceptance proposition from the supplier, which can delete P from its booked commitments.

To these messages we should add, as always, the connections between the agent's organisational and motivational systems. Here we shall find once more the internal messages between modules described in the preceding section, namely:

**Allocate(T)** is a request from the motivational system to try to find someone to whom to allocate task T.

**Impossible(T)** is a response to the motivational system indicating that it has not been possible to allocate task T.

**Committed(T,Y,X)** is a commitment from agent X to carry out task T on behalf of Y. Each time an agreement is reached between a client and a supplier, the two agents memorise this commitment in a symmetrical manner.

The allocation by delegation mechanism can be written in the form of a set of four modules (Figure 7.9), which manage both the client and supplier parts of each agent:

(1) A *request evaluation* module, which either proposes an offer or sends the request to the delegation module. If the agent has already received a request, it merely sends an acknowledge message to the agent sending the request.

(2) A *delegation* module, which undertakes to send a request message to all its acquaintances and to receive all the acknowledgements. This module is activated either by a command from the preceding module or by a task allocation request coming from the agent's motivational system.



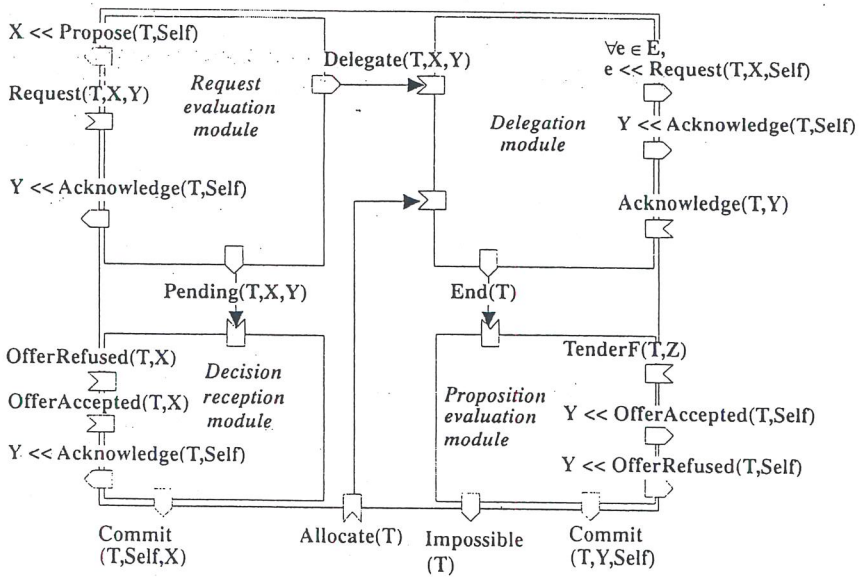


**Figure 7.9** System of allocation by delegation, broken down into four modules.

(3) A *proposition evaluation* module, which merely accepts the first offer received from a supplier and refuses all the others.

(4) A *decision reception* module, which commits the agent to carry out a task if the offer has been accepted and to put the pending offers refused in the trash.

The delegation module (Figure 7.10) merely sends the message `RequestD` to all its contacts and waits for their receipts. It counts the receipts received, thanks to location $P_1$. When it has received as many receipts as it has sent out requests, it sends an acknowledge message in its turn, unless it is the client, that is, the agent initiating the broadcasting. In that case, it sends back the message `End` to the proposition evaluation module.

The request evaluation module (Figure 7.11) undertakes to process a request. If it can carry out task T (and wants to), it proposes its services and then waits. Otherwise, it delegates this request to all its contacts, unless it has already received a similar request, in which case it simply sends back an acknowledge message. Location $P_2$ simply serves to memorise the fact that it has already responded to a request to do T for a client X.

The last two modules deal respectively with the reception of decisions coming from the client (Figure 7.12(a)) and the evaluation of propositions (Figure 7.12(b)). The latter, upon receiving an offer, indicates to the organisational system that the supplier is definitely going to commit itself, memorises the offer, with the

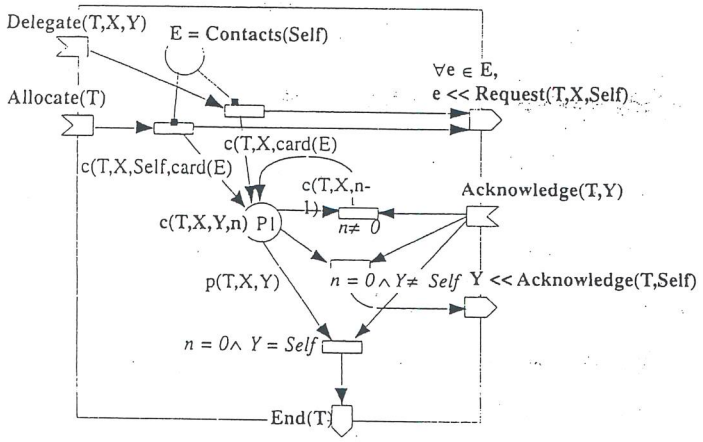


Figure 7.10 Delegation module.

help of location $P_5$, and refuses all the others. The former receives these acceptances or refusals, and sends the requesting agent an acknowledgement for having received a request. Finally, if the offer is accepted, it commits itself to its client. Location $P_4$ memorises propositions waiting to be accepted or refused, and location $P_3$, which serves as a sink, consumes the marks if the offer is refused.

However, this algorithm has some imperfections.

(1) In the first place, no account has been taken of data relating to the acquaintances' skills to optimise the search. Each agent sends requests to all its acquaintances, whereas it could send priority requests to the agents it knows have this skill. But this optimisation, which seems insignificant,

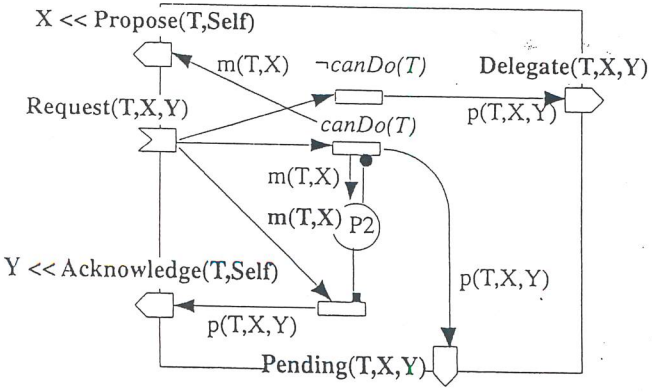


Figure 7.11 Request evaluation module.

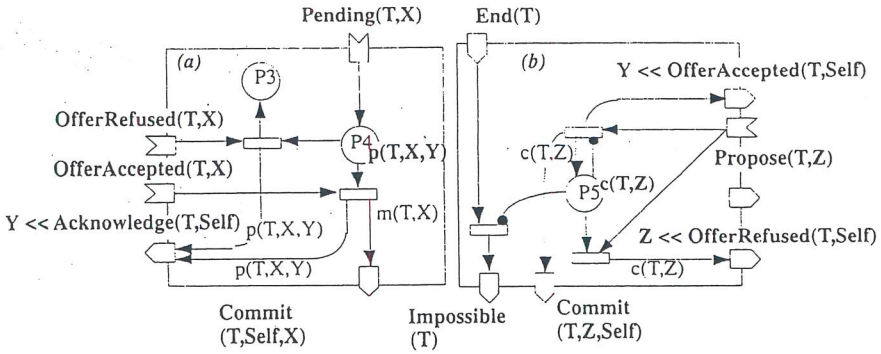


Figure 7.12 Decision reception module (a) and proposition evaluation module (b).

nonetheless conceals several little difficulties. First of all, the requests have to be differentiated. A direct request will initially be sent only to acquaintances having the skill, to ask them if they agree to carry out P, which will be followed by a general request which will be limited to delegating messages. For this algorithm to function, we must be sure that the representations of the agents are complete, that is, that if an agent B forms part of A's acquaintances, then A does actually know what B's skills are.

(2) When an agent indicates that it accepts a task, it would be tempting to want to bring the whole process to a halt and tell all the other branches which are in the process of scouring the network that they can stop looking for a candidate. All that need happen then is for the agent which accepts to send an `AcknowledgeSuccess` message to the applicant, and for this applicant to send a `Stop` message to all the agents to which it sent a request and which have not yet returned a response. This modification, which can increase the number of messages to be sent between agents, is effective only if the `Stop` messages are faster, and have greater priority, than the others. Otherwise, this will merely complicate the mechanism.

(3) This algorithm, like the preceding ones, takes no account of all the various skill levels of agents, in that it considers only the first acceptance. The agent does not ask for an 'estimate', on the basis of which it would be able to express a choice. However, the introduction of such a mechanism is relatively easy. It is sufficient to memorise the list of agents that agree to do the work, to conclude a contract only with the one that 'sounds best', and to indicate to the others that they have not been accepted. This technique then resembles those for requests for bids which we shall be examining in the following section.

(4) Finally, attention must be paid to the way a task request is worded. If several requests corresponding to identical tasks are sent one after another, there is

a risk of confusion, with an agent believing it has already responded to the request relating to this task when it is actually a new one. In fact, the place in the request evaluation module systematically preserves all the requests it has received. To avoid these confusions, it is sufficient to word task requests by stringing together the description of the task, the initial applicant agent, and an order number. Requests are then all systematically differentiated.

### *Reorganisation of an acquaintance network*

An agent's acquaintances can be considered as a kind of cache memory through which an agent can be in a position to have direct knowledge of the characteristics of other agents it may need. For this reason, acquaintance systems have the classic problems of cache memories and, in particular, the difficulty of maintaining coherence between the acquaintances and the state of the agents they represent. This is particularly true when an agent changes its capabilities, when a new agent appears in the system, or when one of them comes to leave it. In that case, the agent's acquaintance base has to be updated in such a way that it is coherent once again, which poses the difficult problem of reorganising an acquaintance network.

For example, let us assume that an agent A knows that an agent B has the necessary skills to carry out a task T and that, for some reason or another (a module of B fails, B has specialised in another task, etc.), B's skills are modified, and it no longer has the capacity to do T. The next time A asks B to do T, it will receive a refusal, whereas so far A has taken it for granted that B would do T every time A asked it to do so. This problem of distributed coherence can be solved in two ways:

- One way is to arrange for B to alert all the agents that know B to its change of state. But this raises two difficulties: (1) B would have to know that A knew B's capabilities, which would mean managing bilateral contacts between all the acquaintances, and (2) attention would have to be paid to problems of parallelism, which could lead to another request's coming to B before it had time to warn all the agents of which it is a contact.
- Alternatively, A's acquaintances table could be modified. When A asks B to do T and this request fails, A memorises the fact that B is no longer able to do T and restarts the procedure of looking for an agent.

There can obviously be many different kinds of acquaintance network. For example, we may wish to reorganise the network in a dynamic manner so that agents that do not know each other initially are put into contact. We can also improve the performance levels of the system by statistically managing the agents' acquaintances in such a way as to optimise the requests. Agents with which an agent is frequently contracted can be in its acquaintances table, while it discards acquaintances that are never used. It is then possible to arrange for the network to 'learn' to allocate the tasks it has to accomplish as it works. 'Good' marks can be given to agents that give satisfaction, and 'bad' marks to those that respond less well to requests. The Cascade system created by H. V. D. Parunak (1990), which controls a storehouse management system, works according to this principle.



What is to be done when new agents are brought into service in a multi-agent universe? Several techniques are available, and almost all of them stem from the technology of nets (for example, the Internet) or of distributed systems. With the check-in technique, in particular, new agents introduce themselves to a 'manager', but that means the manager has to know all the agents. This difficulty can be eliminated almost completely by defining a hierarchical administration system, as in computer networks, with each manager being able to control its region and its set of agents. We can also arrange that all new agents contact only some of the agents in the system – then, by using the acquaintance network (and in particular by using its system for reorganising acquaintances, if it has one), the data is propagated along the network. If the acquaintances graph does, in fact, have a great many connections, all the agents in the system can benefit from the characteristics of the new entrants.

## 7.3.2 Allocation by the contract net

The contract net is a task allocation mechanism based on a market-like protocol. Introduced by R. G. Smith (1979), this technique received an enthusiastic welcome from the community of researchers into distributed artificial intelligence, and numerous systems have used it. It is certainly a control structure which is very easy to understand and to use. However, we shall see that its simplicity conceals some difficulties which must be resolved if the contract net is to be used in a truly distributed context. The contract net is one of the computing techniques in which the conceptual aspect is associated with the implementation to introduce usable mechanisms which are easy to learn and to program, with results that correspond relatively well to expectations.

The contract net makes use of the protocol for the drawing up of contracts in public markets. The relationship between the client, or the *manager*, and the suppliers, or *bidders*, is chanelled through a request for bids and an evaluation of the proposals submitted by the bidders. The request for bids has four stages:

(1) The first stage is devoted to the request for bids itself. The manager sends a description of the task to perform to all those it considers able to respond or to all agents of the system (Figure 7.13(a)).

(2) On the basis of this description, in a second stage, the bidders draw up proposals which they submit to the manager (Figure 7.13(b)).

(3) The manager receives and evaluates proposals and awards the contract to the best bidder in a third stage (Figure 7.13(c)).

(4) Finally, in the fourth and last stage, the bidder which has been awarded the contract and which therefore becomes the *contractor* sends a message to the

manager, to signify that it is still prepared to carry out the required task (Figure 7.13(d)) and that it is therefore committing itself to do it, or else that it cannot carry out the contract, which triggers a re-evaluation of the bids and the awarding of the contract to another agent (taking us back to stage 3).

### *Algorithm with a single manager*

We shall first analyse the protocols of the contract net when only one manager exists at a given moment, and then we shall look at the problems raised by generalising to

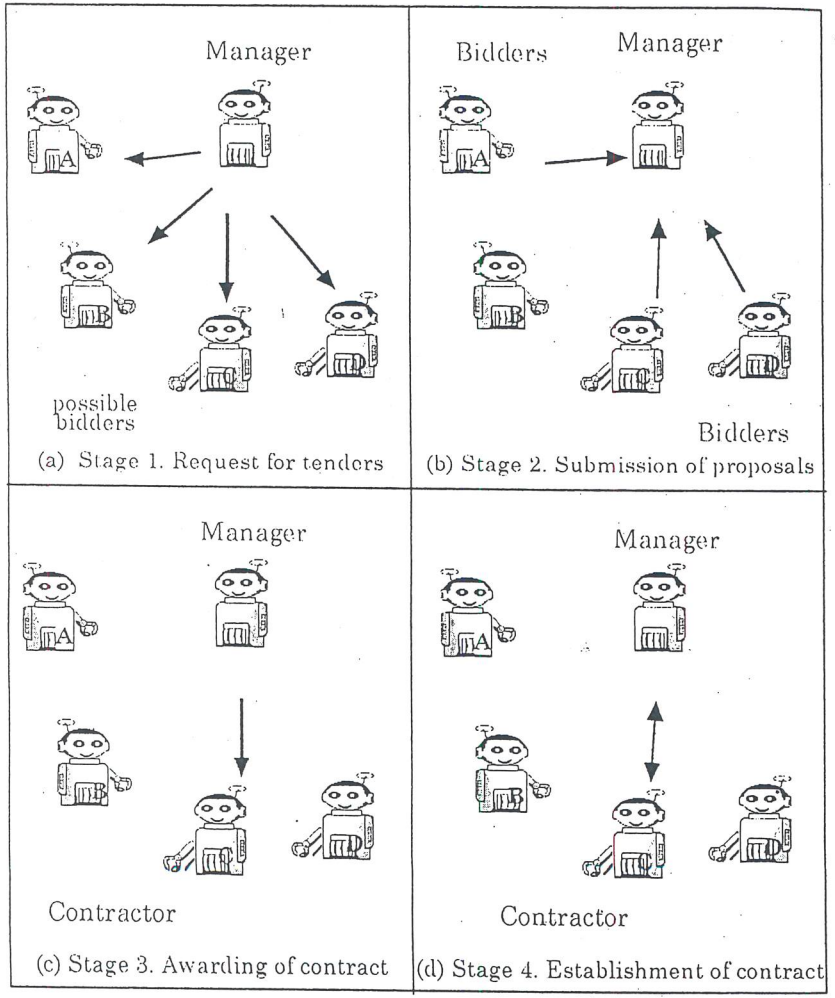


**Figure 7.13** The four stages of the contract net.

any number of simultaneous managers. The types of message necessary for the realisation of this allocation system are as follows:

**RequestForBids(T,X)** is a message from the manager X to a potential bidder, to ask it to submit a proposition for the carrying out of task T if it is interested.

**Propose(T,O)** is the positive response from a bidder Y to a request for bids concerning task T. It then sends back a bid O which, among other data, includes the bidder's name.

**NotInterested(T,Y)** is the negative response of a bidder to the administrator, indicating to the latter that Y is not interested in doing T.

**Award(T,C,X)** is the message from manager X to a bidder to inform it that it has been awarded contract C relating to task T.

**Accept(T,Y)** is a positive response from bidder Y to the awarding of the contract by the manager.

**Refuse(T,Y)** is a negative response from bidder Y to the awarding of the contract by the manager, which re-activates evaluation and the awarding process.

Moreover, as in the other distributed allocation systems looked at earlier, the manager (which is also the client) displays classic connections with its motivational and organisational system, namely the request for allocation `Allocate`, the responses (`Impossible`, `Commit`) and the bidder's commitment (`Commit`).

An agent can be a manager and a bidder at the same time. These are two different roles, but the same agent can as easily take on one as the other. However, to make things clearer, I have found it preferable to break down this algorithm by making a distinction between the behaviour of the manager and that of the bidder. The requests for bids will be recognised in a single manner in the net, using a key made up of the description of the task, the manager responsible for issuing the request, and an order number local to this agent. For reasons of simplification, we shall identify the task with its number. Figure 7.14 shows the general architecture of the modules required to establish a contract net protocol.

The manager comprises two modules (Figure 7.15): the module (A) for the definition of the request for bids and for the reception of proposals being module A. The proposals are then selected by a second module (B) which is responsible for awarding contracts and receiving acceptances or refusals from the bidders. There are three locations in module A. The first, $P_1$, simply contains the identifier for all the agents of the system. The second, $P_2$, serves to count the responses to the request for bids to find out when it can go to the following stage, and location $P_3$ memorises all the proposals received. Module B contains only one location, which is used for defining the iterative awarding of contracts to the bidders. In this implementation of the contract net protocol, it has been assumed that bidders could refuse the contract sent by the manager. In numerous cases, it can be assumed that the bidders always accept contracts, which makes this module easier to write, since we can dispense with the acceptance and refusal responses from the bidders, as well as the

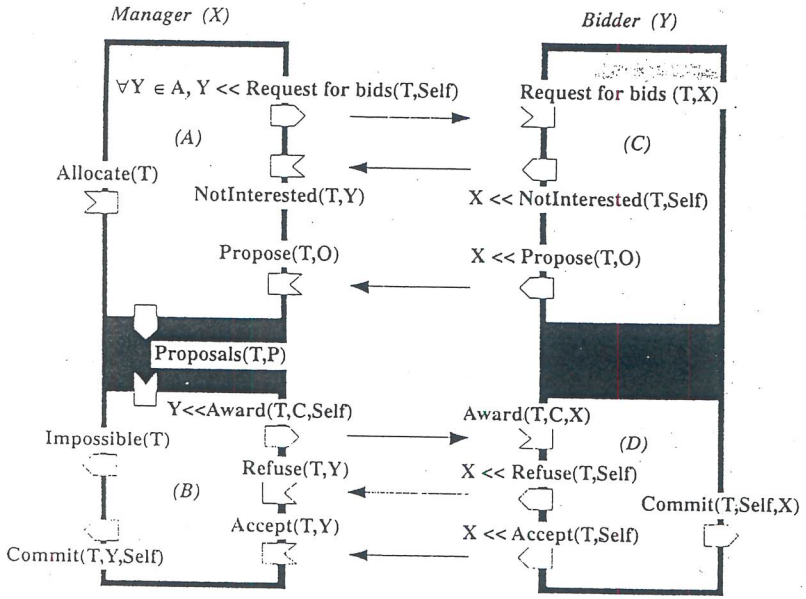


Figure 7.14 Flow chart of a contract net protocol. The manager is responsible for the task announcements and the reception of proposals (module A), then selects the proposals and awards a contract to the bidder that 'sounds best' (module B). The bidder draws up a proposal on the basis of the request for bids (module C) and decides to accept – or not to accept – the contract drawn up by the manager (module D).

iterative search of the list of proposals retained. The bidders also have two modules (Figure 7.16). The first (C) is intended for evaluating the request for bids and for drawing up a proposal, the second (D) for evaluating the contracts awarded by the manager.

### *Contract preparation language*

The communication primitives we have shown above are rather elementary, but adequate to implement the essentials of the contract net. However, R. Smith has put forward a more complete language, which is on a higher level of abstraction (Smith, 1980), in which, as well as the description of the task and the contract number, the request for bids statement includes a *definition of the qualities required* for the task, the *form of proposal* and an *expiry date*. The definition of the required qualities means that agents that do not have these qualities can be eliminated rapidly, without it being necessary to carry out a meticulous analysis of the description of this task.

The form of the proposal indicates to the bidder what the criteria are that will be taken into account during evaluation, and so to facilitate the subsequent



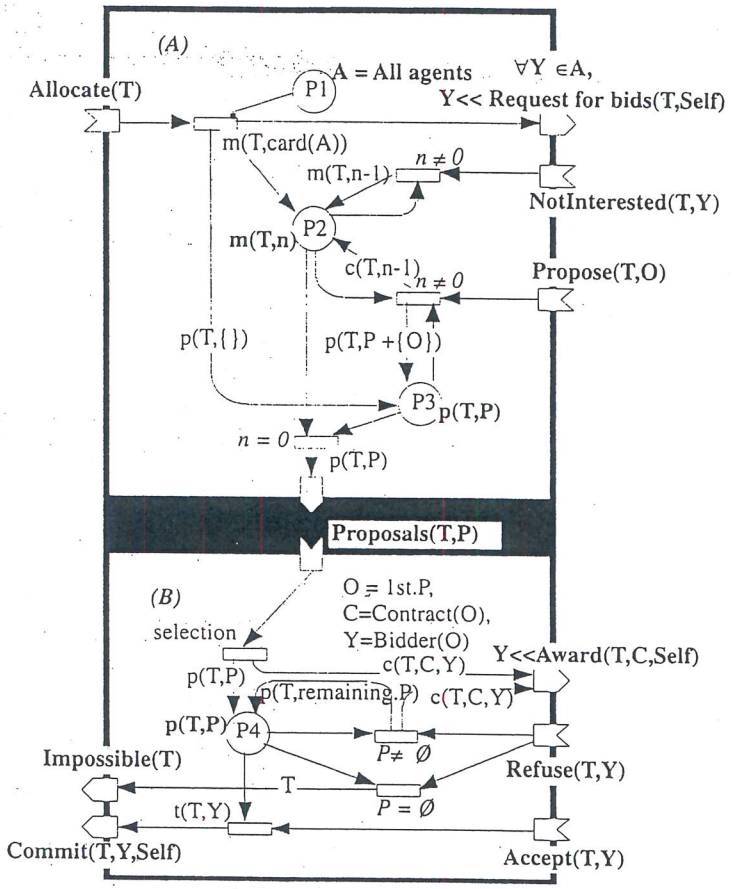


Figure 7.15 Diagram of the behaviour of the manager of a request for bids. Module A is used to issue requests for bids and to receive proposals, which are then sent to module B for the selection of proposals and the awarding of contracts.

evaluation work of the manager. Finally, the expiry date gives the limiting date after which proposals will no longer be retained.

Let us take the example of the ore-gathering robots. Having found a seam, an explorer robot plays the role of a manager and makes a request to all the robots capable of digging into loose ground to drill at its location. It asks them for their positions, so that it can select the driller nearest to it, but also asks for the list of their performance levels for different types of ground, knowing that only robots capable of digging in loose ground will be selected. Using R. Smith's notation to define messages between managers and bidders, here is how this contract net can be issued:

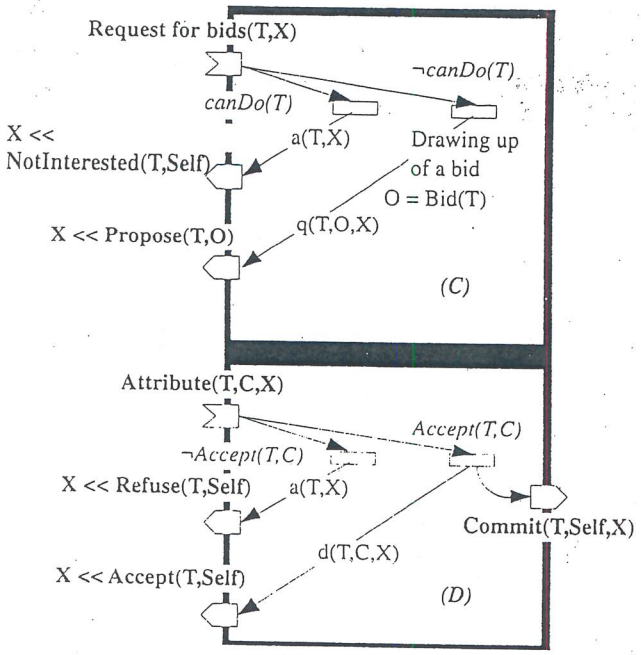


Figure 7.16 Diagram of the behaviour of a bidder. Module C is used to evaluate proposals and for drawing up a bid. Module D is responsible for the acceptance – or not – of a contract as it has been drawn up by the manager.

```
Message: RequestForBids
  To : *    // indicates broadcast message
  From : Explorer-25
  Contract : Drill-Explore-25-234
  DescriptionOfTask :
    TypeOfTask : Drill
  QualitiesRequired :
    MustHave : DrillingSystem
    MustBeAble : DrillLooseGround
  FormOfProposal :
    Position : <Lat, Long>
    DrillingQualities : { TypeOfGround : Performance}
  DateExpiry :
    20 July 2193 13 : 06 : 46
   End RequestForBids
```

The explorer robots will respond in their proposals by giving the characteristics requested by the manager:



```
Message : Propose
  To : Explorer-25
  From : Driller-18
  Contract :Drill-Explorer-25-234
  DescriptionProposal :
    Position: <47N,17W>
    QualitiesDrilling :
      { Sand : 0.5 Clay : 0.3 Shale : 0.8}
End Propose
```

The explorer will award the 'drilling contract' to what it considers to be the best proposal. The apparent simplicity of the algorithm introduced above should not mask a certain number of difficulties. These obscure points relate essentially to the evaluation of the proposal, to the access to the bidders, to taking multiple managers and personal commitments into account, to breaking down tasks into sub-tasks, and to the quality of the allocation obtained.

### *Evaluation of proposal*

A bidder interested in a task submits a proposal containing an evaluation of its capacity to carry out this task. This evaluation can be expressed in the form of a simple numerical measurement or of a description including several criteria evaluated separately. In the application of this mechanism to load balancing in a multi-processor system, it is possible to measure the interest in a processor for a task by means of a simple numerical value that is proportional to the workload of the processor. The greater this is, the more the value of the proposal is reduced, and conversely. In other situations, it appears more difficult to characterise how interesting an agent is in relation to a task, and to measure its capacities to offer a 'good' service. In that case, it is up to the designer of the system to select his/her evaluation function well, together with the criteria which will be recognised as relevant by the system, for the performance of the contract net is closely connected to it.

### *Contacting bidders*

In general, the contract net considers that all agents situated in the net are accessible to any of the net's agents, and that obtaining a list of them is not a problem. In particular, in the algorithm given above, we have intentionally indicated that the manager contacted all the agents. This was nevertheless, at the least, an idyllic concept, which does not deal with the problems of implementation in really distributed contexts, and which does not respond to the question: how can we access agents likely to submit a proposal?

Algorithms for searching the net in parallel can be used to access all the net's agents. Essentially, they resemble the algorithm used for allocation by delegation in an acquaintance network, but instead of asking agents whether they agree to carry

out a task we ask them to draw up a bid and to submit this proposal to the manager. When the process is completed and the manager has received all the acknowledgements, whether they carry a proposal or indicate lack of interest, the manager knows that it has all the proposals and that it can start its evaluation.

This process is obviously very time-consuming and requires a great many messages, since it assumes that all the agents in the system have to be contacted individually. It also poses the problem of creating an acquaintance network, even if this net is simpler than in the case of allocation by delegation, since the agents are not expected to know their neighbours' skills but only to know that they have neighbours. Another solution consists in having all the agents memorised in an overall data structure maintained by a single agent, but then we are once again back with the problems of bottlenecks and sensitivity to breakdowns typical of centralised systems.

The solution most generally adopted by designers of contract nets consists in placing the agents on a 'bus' (Figure 7.17), that is, on a ring-shaped organisation, the functioning of which is very similar to that of the *token ring*.

The messages are then always carried in the same direction, and circulate from neighbour to neighbour, whether they are requests or proposals. One solution then is to consider that the requests for bids are placed on a tray. The bidders each read the requests for bids in turn, evaluate it and place their proposals on the tray. When the tray returns to the manager responsible for this request, it knows that everyone has responded, and it can choose from the set of proposals. This implementation is not very fast either, since evaluation is carried out sequentially and no attention is paid to the potential parallelism of the contract net to accelerate the process. Nevertheless, we shall see later that it has the advantage

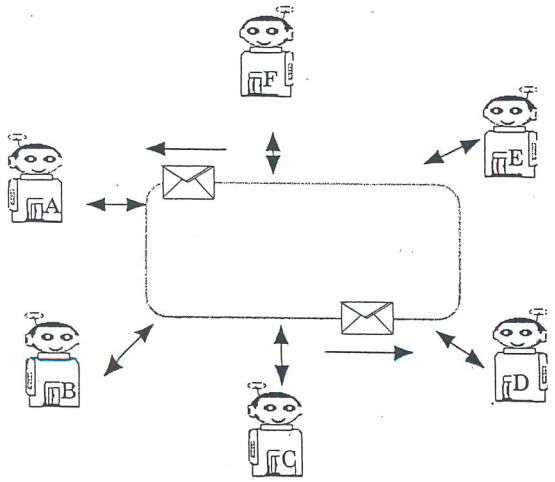


Figure 7.17 Implementation of the contract net by implementing the agents on a ring.

that we can define better allocations than those mentioned above. In order to increase the speed of computation and limit the number of messages, several partial improvements have been made which take account of the nature of the application. In particular, when the net is not very 'volatile', that is, the identity of the agents, their number and their skills show little variation – which is generally the case – it is possible to apply cache memory techniques, which come down to local management of a table of acquaintances by the managers, and indexing agents according to the tasks to which they respond. This technique was developed by Van Parunak (1990) to manage the automatic allocation of tasks in flexible workshops.

A type is associated with each task, and it is assumed that all the agents that know how to carry out tasks of a certain type will necessarily respond to the request by submitting a proposal. When a manager has to issue a request for bids for a new type of task, it broadcasts this request throughout the network, and memorises in its acquaintances table all the agents that respond to this type of task. When faced with a task of the same type, all it has to do then is to search this reduced list to attain the same efficiency. The acquaintances table then plays the part of a cache memory which optimises access. Sometimes the list amounts to a single individual, and the manager can then set up contracts directly, without going through a contract net protocol. If agents enter or leave the network, or if their skills are modified, it is assumed that they will inform the net of this by broadcasting the information to all the managers. Such a technique improves the response time and the number of messages sent, but its advantages are clearly perceptible only if the modifications are few in number compared to the number of requests for bids.

### *Limit date*

Up until now, we have assumed that the manager waits for all the proposals before evaluating them. This obviously makes it possible to take more account of all the potentialities of the system by not forgetting any bid, but at the expense of two major drawbacks:

(1) All the agents have to respond, even those that *a priori* are not interested in the request for bids. For this reason, the network could become saturated by too many messages which contain only very small amounts of information and carry only the message 'not interested'.

(2) An agent that has broken down or is not functioning properly can block the system totally. If it does not send a response, the manager is kept waiting indefinitely.

All that need be done here is that agents should not send messages back if they are not interested or, obviously, if they cannot do what is asked. Secondly, each request will have a deadline for the receipt of proposals. Once this date has passed, the manager evaluates the proposals it has received, and all those arriving after this date are directly rejected.

Obviously, this solution raises other problems. This reaction time does have to be calibrated in such a way that the majority of the proposals have the time to return to the manager. If the deadline is set too early then numerous proposals will not be taken into account. If the deadline is too distant then the manager will be waiting pointlessly before beginning its evaluation. It is nevertheless possible to draw up an adaptative system. If too many proposals arrive after the deadline, the response time is increased, and if there is too long a wait after the final responses have been received, this time is reduced. In this way, agents will all respond with the same response time, so that the system will converge towards a well-adapted state.

### *Multiple managers and optimality*

We have been considering the case of a single manager most of the time, but this is actually a degenerate case of the contract net, for several managers can be working at the same time and can issue their requests for bids simultaneously. It is therefore necessary to manage the fact that several managers can send two requests for bids to the same agent at the same time. They are liable to interfere with each other and to have a negative effect on the overall behaviour of the system. In the contract net, tasks are generally attributed only on the basis of the local knowledge of managers, without taking into account either what other managers want or the other proposals which may arrive later. Van Parunak (1987) in fact considers that, because this is a local process, the agents are ignorant in three ways: their ignorance relates to the temporal and spatial aspects and to the workloads of the bidders.

(1) *Temporal ignorance* comes from the fact that the bidders are aware only of the requests for bids already received, and not of those that are on the point of arriving. An agent may commit itself to carry out a task for which it is not really very well suited, and therefore miss contracts that would correspond better to its true qualities.

(2) *Spatial ignorance* comes from the fact that the managers cannot know about other requests for bids in progress, and the bidders cannot know about other proposals. For example, let us assume that two managers, A and B, send two potential bidders, X and Y, two requests for bids for two tasks, TA and TB, and that the proposals which result from this take the form below.

| | TA | TB |
|---|---|---|
| X | 90 | 80 |
| Y | 80 | 20 |

When these proposals are received, manager A will normally select X, for its proposal looks more interesting than Y's, and likewise B will choose X, because its proposal in this case is also much better than Y's. It follows from this that X will have a great deal to do while Y will remain inactive, whereas it would be preferable to award task TB to X and leave task TA to Y. If X knew about Y's proposal, it could reduce the quality of its proposal for TA, and if Y knew about X's proposal, it could improve its proposal for task TB. There are therefore algorithms in which all the bidders send their proposals to all the other agents, which makes it possible to obtain more fine-tuned proposals, which take account of all proposals from agents. However, these techniques are very time-consuming and require a great many messages, their complexity growing in an exponential manner with the number of agents in the system.



(3) *Ignorance of workloads* is due to the fact that the bidders have no information on other agents' workloads. For this reason, an agent that has work may be motivated to beat another that has practically nothing to do. To remedy this problem, Van Parunak proposes that agents should have information available on one another's workloads, so that they can modify their proposals and, if need be, be more 'aggressive' in order to win the contract.

### *Commitments, reservations and rejection of contracts*

Parallelism raises another problem. Bidders can receive new requests between the time when they submit a proposal and the time they are awarded the contract or are rejected by the manager. The bidder is then faced with a difficult choice. It can either reserve space for all the tasks for which it submits proposals, and so risk finding itself without anything to do, or, on the contrary, it can submit new proposals without taking the previous ones into account, even if it may end up with too much work and be unable to meet its commitments. Each of these possibilities has advantages and drawbacks which it is expedient to analyse in the light of the type of application being considered.

(1) In booking tasks, an agent considers that the probability that it will be awarded a task is great. For this reason, it reserves the resources (in terms of time and space) required to carry out this task. When it receives a new request for bids, it takes account of the capacity reserved for tasks as if it had actually committed itself to doing them, and, should the need arise, envisages submitting proposals which will not be very interesting if the number of booked tasks is too great. However, it is taking the risk of seeing its proposals rejected, and thus of losing contracts, if it reserves too much capacity for tasks and does not become sufficiently 'competitive' in its proposals to acquire new markets. As a result, in this scenario, the risk is to end up without an adequate number of tasks to carry out. However, this method is not without advantages, in particular the simplicity with which it can be implemented. As soon as a proposal is submitted, the agent merely reserves the resources required to carry out this task, and computes its proposals taking into account the capacities reserved for tasks as if they represented actual commitments. Incidentally, this is the algorithm we