Contract Net Protocol

A common problem in multiagent systems is deciding how to re-allocate a set of tasks among a set of agents. This is known as the *task allocation problem***.** In this problem there is a set of tasks *T*, a set of agents, and a cost function $c_i : s \to \mathbb{R}$ which tells us the cost that agent *i* incurs in carrying out tasks $s \subseteq T$. In some simplified versions of the problem we assume that all agents share the same cost function. Each agent starts out with a set of tasks such that all tasks are distributed amongst all agents. We can think of this initial allocation as δ^- since, if negotiations break down then each agent is responsible for the tasks it was originally assigned. Similarly, every allocation of tasks to agents is a possible deal δ where $s_i(\delta)$ is the set of tasks allocated to *i* under deal δ. The problem we then face is how to design a negotiation protocol such that the agents can negotiate task re-allocations and arrive at a final re-allocation of the tasks that is one of the axiomatic solution concepts, such as the utilitarian deal.

An example of this type of problem is the *postman problem* in which a set of postmen are placed around a city each with a bag of letters that must be delivered. A postman prefers to have all his letters with delivery addresses that are very close to each other so as to minimize his delivery costs. The postmen can negotiate with each other in order to make this happen. That is, a postman can trade some of its letters with another postman. In this example the tasks are the letters to be delivered and the cost function is the distance travelled to deliver a set of letters. Different cost functions arise when the postmen have different final destinations, for example, if they are required to go to their respective homes after delivering all the letters, or if they prefer certain areas of town, etc.

Once we are given a task allocation problem, we must then decide how the agents will exchange tasks. The simplest method is for agents to pair up and exchange a single task. This means that not all deals are directly accessible from other deals since, for example, we can't go from the deal where *j* does 2 tasks and *i* does nothing to the deal where *i* does 2 tasks and *j* does nothing in one step. We can represent this constraint graphically by drawing an edge between every pair of deals that are reachable from one

another by the exchange of a single task. An example of such a graph is shown in figure 1.

δ	$s_i(\delta)$	$s_j(\delta)$	$c_i(\delta)$	$c_j(\delta)$	$u_i(\delta) = 8 - c_i(\delta)$	$u_j(\delta) = 8 - c_j(\delta)$
δ^1	$\emptyset$	$\{t_1, t_2, t_3\}$	0	8	8	0
δ^2	$\{t_1\}$	$\{t_2, t_3\}$	1	4	7	4
δ^3	$\{t_2\}$	$\{t_1, t_3\}$	2	5	6	3
δ^4	$\{t_3\}$	$\{t_2, t_3\}$	4	7	4	1
δ^5	$\{t_2, t_3\}$	$\{t_1\}$	6	4	2	4
δ^6	$\{t_1, t_3\}$	$\{t_2\}$	5	3	3	5
δ^7	$\{t_1, t_2\}$	$\{t_3\}$	3	1	5	7
δ^8	$\{t_1, t_2, t_3\}$	$\emptyset$	7	0	1	8

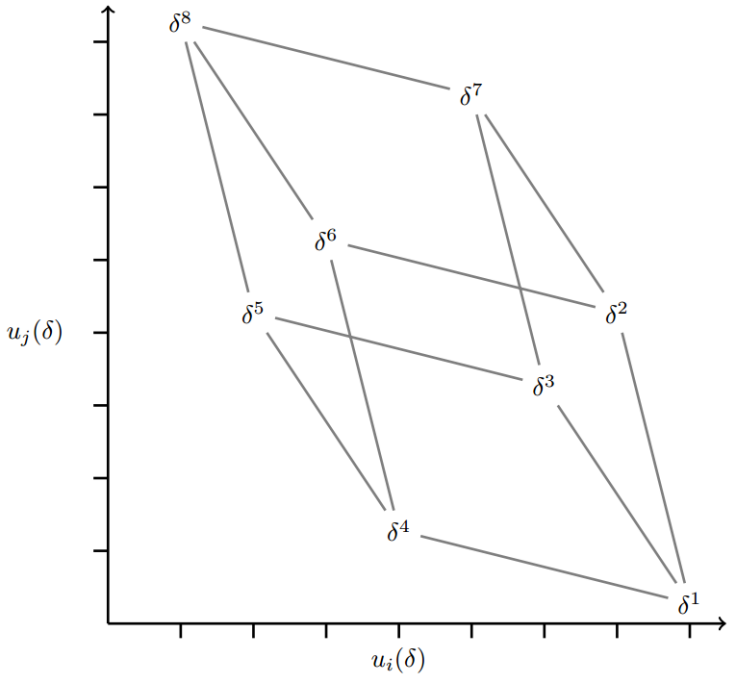

Figure 1. *An example task allocation problem is shown in the table and its graphical representation as a bargaining problem is shown in the graph. The edges on the graph connect deals that can be reached by moving a single task between the agents. $s_i(\delta)$ is the set of tasks i has under deal δ*

The table in this figure shows the tasks assignments that each deal represents and the costs that each one of the agents incurs for carrying out various subsets of tasks. We let the utility to the agent be 8 minus the cost of carrying out the tasks in order to have positive utility numbers. Utility is often simply the negative of the cost so these two are used interchangeably. The graph shows all the possible deals as points and the edges connect deals that can be reached from one another by moving a single task between agents.

Note that the example in figure 1 has a Pareto frontier that consists of deals $\delta^1, \delta^2, \delta^7$ and δ^8. Thus, if we were to apply our simple hill-climbing search strategy where we always move to a neighbouring dominant deal we are guaranteed to always end up stuck at one of these four deals. In terms of the task allocation problem this greedy search corresponds to the agents exchanging a task only when both of them are better off from the exchange. In other words, if your solution to a task allocation problem is to have the agents exchange a single task at a time only when both agents have a utility gain from the transaction then, eventually, the system will converge to an allocation that is on the Pareto frontier. Of course, this means that, for example, you might end up at deal δ^8 from figure 2.1 which might not be so desirable. A more desirable deal might be δ^7 which is the utilitarian solution for this problem.

One possible way to allow the agents to find deals that are closer to the utilitarian deal is by allowing them to use *monetary payments*. For example, agent *i* might convince *j* to take a deal that gives *j* less utility if *i* can sweeten the deal by giving *j* some money to take that deal. This basic idea was implemented in the *contract net protocol*. In contract net each agent takes the roles of either a contractor or contractee. The contractor has a task that it wants someone else to perform. The contractor is also willing to pay to get that task done. In the protocol, the contractor first announces that it has a task available by broadcasting a call for bids. All agents receive this call for bids and, if they want, reply to the contractor telling him how much they charge for performing the task. The contractor then chooses one of these bids assigns the task to the appropriate agent and pays him the requested amount.

For example, given the current task allocation δ agent *i* might find that one of its current tasks $t \in s_i(\delta)$ costs it a lot. That is, $c_i(s_i(\delta)) - c_i(s_i(\delta) - t)$ is a large number. In this case *i* will be willing to pay up to $c_i(s_i(\delta)) - c_i(s_i(\delta) - t)$ in order to have some other agent perform *t* – the agent will pay up to the utility it will gain from losing the task, any more than that does not make sense as the agent can simply do the task itself. Similarly, any other agent *j* will be willing to perform *t* as long as it gets paid an amount that is at least equal to any cost increase it will endure by performing the task, that is, the payment must be at least $c_j(s_j(\delta)) - c_j(s_j(\delta) + t)$. Note that in the general case some of these numbers could be negative, for example, an agent could actually have lower costs from performing an extra task. However, these sub-additive cost functions are rare in practice.

The use of monetary payments has the effect of turning one deal into an infinite number of deals: the original deal and the infinite number of payments, from $-\infty$ to ∞ that can be made between the agents. Figure 2 shows a graphical representation of this process. Here we see deal δ^1 transformed into the set of deals represented by the line going through δ^1. The figure also shows us how this transformation creates new deals that dominate another existing deal δ^0. Thus, if the system was in δ^0 and we were doing hill-climbing then we would be stuck there as δ^1 does not dominate δ^0. But, if we used the contract net then agent *j* could make a payment to *i*, along with the task transfer, which would give *i* a higher total utility. This new task allocation plus payment is a deal that lies within the thick area of the line that intersects δ^1. Thus, contract net allows us to reach a task allocation that is the utilitarian solution. Unfortunately, the addition of payments does not guarantee that we will always reach the utilitarian solution; this depends on the particular characteristics of the cost function.

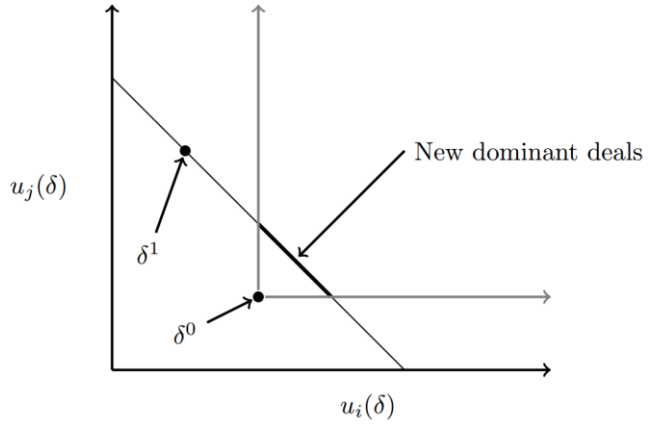

Figure 2. *Deal δ^1 is turned into an infinite number of deals, represented by the line that intersects δ^1, with the use of payments. Some of those new deals Pareto dominate δ^0, as shown by the thicker part of the line*

One type of cost functions that have been found to be easy to search over are additive cost functions where the cost of doing a set of tasks is always equal to the sum of the costs of doing each task individually.

A function $c(s)$ is an *additive cost function* if $\forall s \subseteq T$ it is true that $c(s) = \sum_{t \in s} c(t)$.

In these scenarios it has been shown that the contract net protocol, or any other protocol that uses payments and always moves to dominant deals, will eventually converge to the utilitarian social welfare solution.

In a task allocation problem with an additive cost function where we only allow exchanges of one task at a time, any protocol that allows payments and always moves to dominant deals will eventually converge to the utilitarian solution.

The contract net protocol is an interaction protocol for cooperative problem solving among agents. It is modelled on the contracting mechanism used by businesses to govern the exchange of goods and services. The contract net provides a solution for the so-called *connection problem:* finding an appropriate agent to work on a given task. Figure 3 illustrates the basic steps in this protocol. An agent wanting a task solved is called the *manager*; agents that might be able to solve the task are called potential *contractors.*

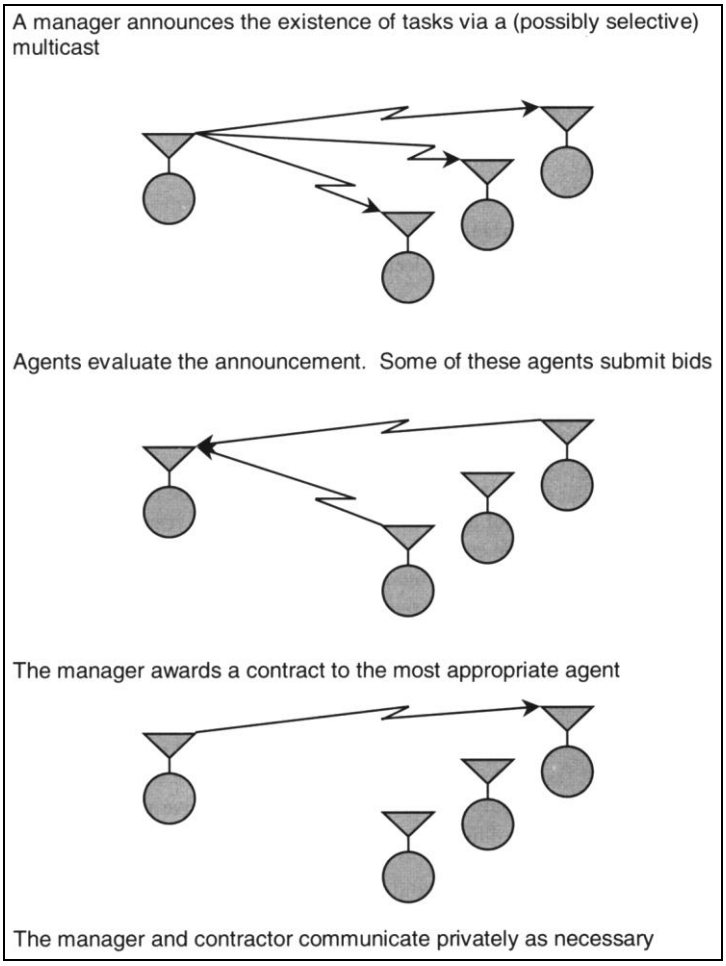

Figure 3. *The basic steps in the contract net, an important generic protocol for interactions among cooperative agents*

From a manager's perspective, the process is:

- Announce a task that needs to be performed;
- Receive and evaluate bids from potential contractors;
- Award a contract to a suitable contractor;
- Receive and synthesize results.

From a contractor's perspective, the process is:

- Receive task announcements;
- Evaluate my capability to respond;
- Respond (decline, bid);
- Perform the task if my bid is accepted;
- Report my results.

The roles of agents are not specified in advance. Any agent can act as a manager by making task announcements; any agent can act as a contractor by responding to task announcements. This flexibility allows for further task decomposition: a contractor for a specific task may act as a manager by soliciting the help of other agents in solving parts of that task. The resulting manager-contractor links form a control hierarchy for task sharing and result synthesis.

The contract net offers the advantages of graceful performance degradation. If a contractor is unable to provide a satisfactory solution, the manager can seek other potential contractors for the task.

The structure of a task announcement includes slots for *addressee, eligibility specification, task abstraction, bid specification* and *expiration time.* The tasks may be addressed to one or more potential contractors who must meet the criteria of the eligibility specification. The task abstraction, a brief description of the task, is used by contractors to rank tasks from several task announcements. The bid specification tells potential contractors what information must be provided with the bid; returned bid specifications give the manager a basis for comparing bids from different potential contractors. The expiration time is a deadline for receiving bids.

Each potential contractor evaluates unexpired task announcements to determine if it is eligible to offer a bid. The contractor then chooses the most attractive task (based on some criteria) and offers a bid to the corresponding manager.

A manager receives and evaluates bids for each task announcement. Any bid deemed satisfactory may be accepted before the expiration time of the task announcement. The manager notifies the contractor of bid acceptance with an *announced award* message.

A limitation of the contract net protocol is that a task might be awarded to a contractor with limited capability if a better qualified contractor is busy at award time. Another limitation is that a manager is under no obligation to inform potential contractors that an award has already been made.

A manager may not receive bids for several reasons:

- All potential contractors are busy with other tasks;
- A potential contractor is idle but ranks the proposed task below other tasks under consideration;
- No contractors, even if idle, are capable of working on the task.

To handle these cases, a manager may request immediate response bids to which contractors respond with messages such as *eligible but busy, ineligible* or *not interested* (task ranked too low for contractor to bid). The manager can then make adjustments in its task plan. For example, the manager can wait until a busy potential contractor is free.

The contract net provides for *directed contracts* to be issued without negotiation. The selected contractor responds with an *acceptance* or *refusal.* This capability can simplify the protocol and improve efficiency for certain tasks.

The Contract Net Protocol has been standardized by FIPA (Foundation for Intelligent Physical Agents).

The FIPA Contract Net Interaction Protocol (IP) is a minor modification of the original contract net IP pattern1 in that it adds rejection and confirmation communicative acts. In the contract net IP, one agent (the Initiator) takes the role of manager which wishes to have some task performed by one or more other agents (the Participants) and further wishes to optimise a function that characterizes the task. This characteristic is commonly expressed as the price, in some domain specific way, but could also be soonest time to completion, fair distribution of tasks, etc. For a given task, any number of the Participants may respond with a proposal; the rest must refuse. Negotiations then continue with the Participants that proposed.

The representation of this IP is given in figure 4.

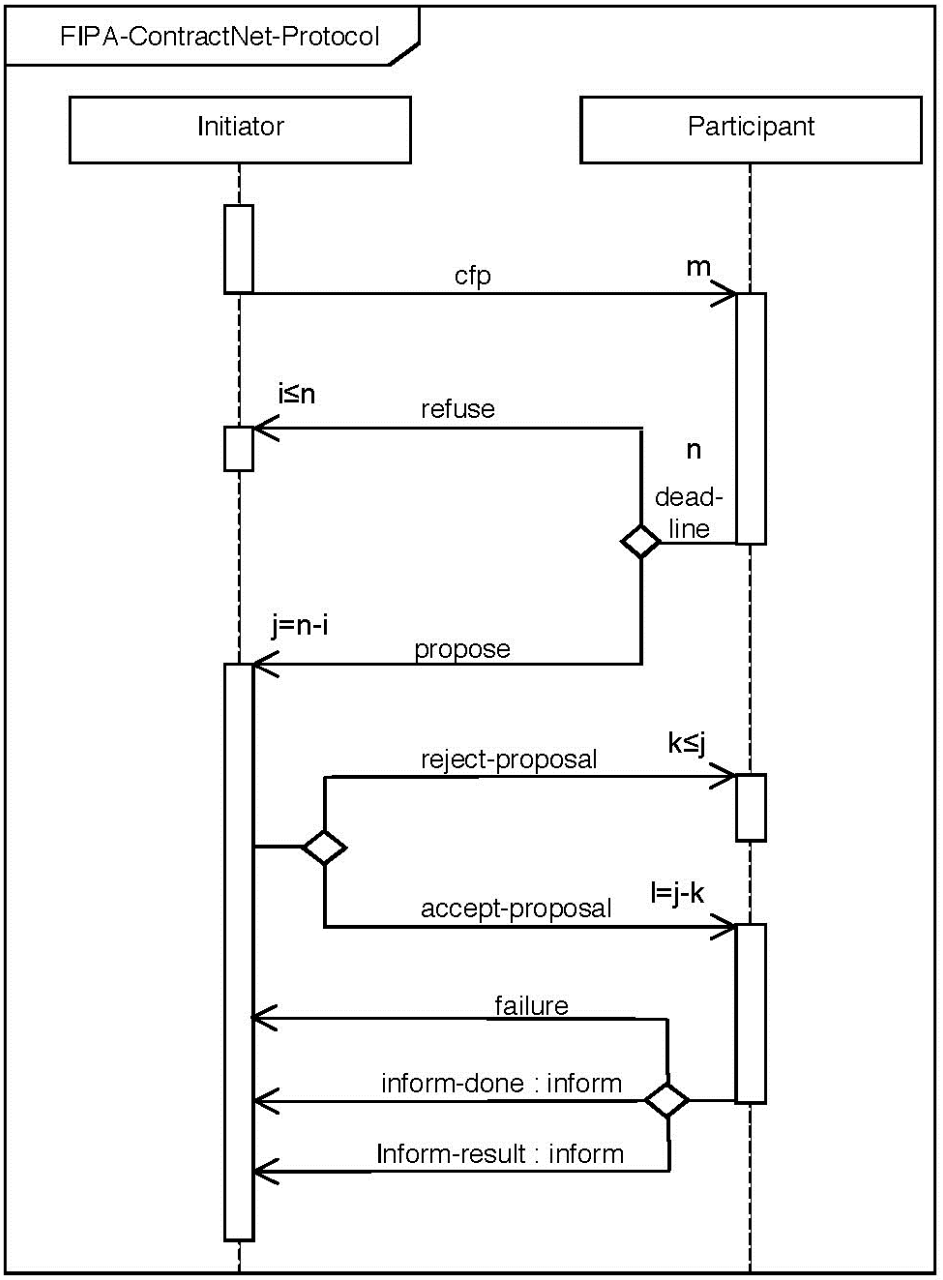

Figure 4. *FIPA Contract Net Interaction Protocol*

The Initiator solicits m proposals from other agents by issuing a call for proposals (cfp) act, which specifies the task, as well any conditions the Initiator is placing upon the execution of the task. Agents (Participants) receiving the call for proposals are viewed as potential contractors and are able to generate *n* responses. Of these, *j* are proposals to perform the task, specified as propose acts.

The Participant's proposal includes the preconditions that the Participant is setting out for the task, which may be the price, time when the task will be done, etc. Alternatively, the $i = n - j$ Participants may refuse to propose. Once the deadline passes, the Initiator evaluates the received; proposals and selects agents to perform the task; one, several or no agents may be chosen. The *l* agents of the selected proposal(s) will be sent an accept-proposal act and the remaining *k* agents will receive a reject-proposal act. The proposals are binding on the Participant, so that once the Initiator accepts the proposal, the Participant acquires a commitment to perform the task. Once the Participant has completed the task, it sends a completion message to the Initiator in the form of an inform-done or a more explanatory version in the form of an inform-result. However, if the Participant fails to complete the task, a failure message is sent.

Note that this IP requires the Initiator to know when it has received all replies. In the case that a Participant fails to reply with either a propose or a refuse act, the Initiator may potentially be left waiting indefinitely. To guard against this, the cfp includes a deadline by which replies should be received by the Initiator. Proposals received after the deadline are automatically rejected with the given reason that the proposal was late. The deadline is specified by the reply-by parameter in the ACL message.

Any interaction using this interaction protocol is identified by a globally unique, non-null conversation-id parameter, assigned by the Initiator. The agents involved in the interaction must tag all of its ACL messages with this conversation identifier. This enables each agent to manage its communication strategies and activities, for example, it allows an agent to identify individual conversations and to reason across historical records of conversations.

In the case of 1:*N* interaction protocols or sub-protocols the Initiator is free to decide if the same conversation-id parameter should be used or a new one should be issued. Additionally, the messages may specify other interaction-related information such as a timeout in the reply-by slot that denotes the latest time by which the sending agent would like to have received the next message in the protocol flow.

At any point in the IP, the receiver of a communication can inform the sender that it did not understand what was communicated. This is accomplished by returning a not-understood message. As such, figure 4 above does not depict a not-understood communication as it can occur at any point in the IP. The communication of a not-understood within an interaction protocol may terminate the entire IP and termination of the interaction may imply that any commitments made during the interaction are null and void. However, since this IP broadcasts to more than one Participant, multiple responses are also possible. Each response, then, must be evaluated separately – and some of these responses might be not-understood. However, terminating the entire IP in this case might not be appropriate, as other Participants may be continuing with their sub-protocols.

At any point in the IP, the initiator of the IP may cancel the interaction protocol by initiating the meta-protocol shown in figure 5. The conversation-id parameter of the cancel interaction is identical to the conversation-id parameter of the interaction that the Initiator intends to cancel. The semantics of cancel should roughly be interpreted as meaning that the initiator is no longer interested in continuing the interaction, and that it should be terminated in a manner acceptable to both the Initiator and the Participant. The Participant either informs the Initiator that the interaction is done using an inform-done, or indicates the failure of the cancellation using a failure.

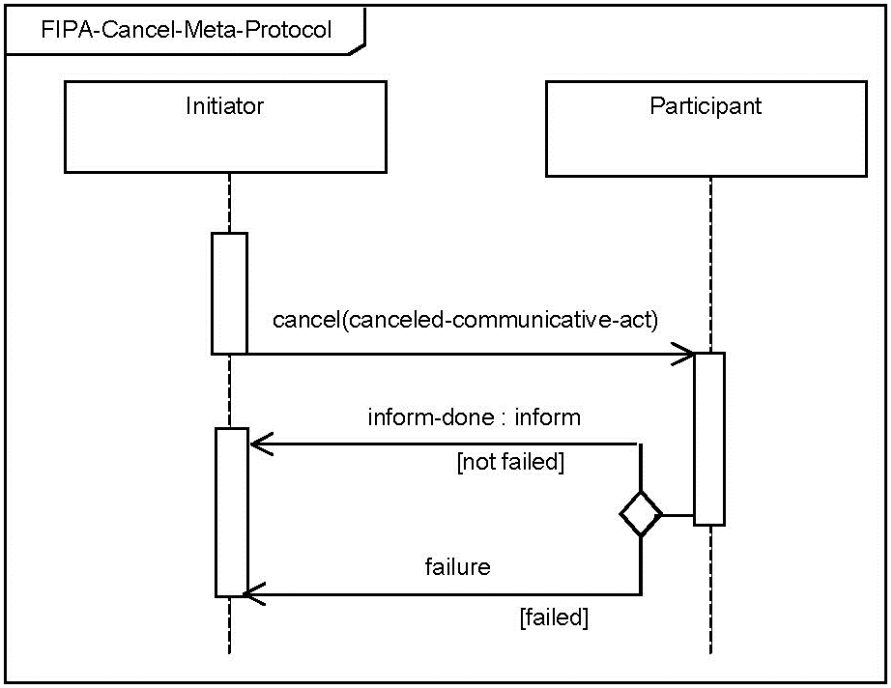

Figure 5. *FIPA Cancel Meta-Protocol*

References

J. M. Vidal, *Fundamentals of Multiagent Systems with NetLogo examples*, http://jmvidal.cse.sc.edu/papers/mas.pdf, 2010.

M. N. Huhns, L. M. Stephens, *Multiagent Systems and Societies of Agents*, in G. Weiss (ed.), Multiagent systems, The MIT Press, 1999.

Foundation for Intelligent Physical Agents, *FIPA Contract Net Interaction Protocol Specification*, http://www.fipa.org/specs/fipa00029/SC00029H.pdf, 2002.