In Agents Breaking Away, MAAMAW'96, W. Van de Velde and J. Perram (eds.), Lecture Notes in Artificial Intelligence, 1038, 72-85, Springer-Verlag, Heidelberg, 1996

Formalising the Contract Net as a Goal-Directed System

Mark d'Inverno[1] and Michael Luck[2]

[1] School of Computer Science, University of Westminster, London, W1M 8JS, UK.
Email: dinverm@westminster.ac.uk
[2] Department of Computer Science, University of Warwick, Coventry, CV4 7AL, UK.
Email: mikeluck@dcs.warwick.ac.uk

Abstract. In response to the problems that have arisen regarding the terminology and concepts of agent-oriented systems, previous work has described a formal framework for understanding agency and autonomy. In this paper we outline the framework and refine it by adding further levels of detail to develop a formal model of the *Contract Net Protocol*. The model serves to make precise both the operations of nodes in the contract net, and the state of the net at various points during the protocol. In particular, the nature of the dependencies between the nodes in the net is explicated. Finally, we generalise the relationships that can be found in the contract net which are brought out by the formalisation, and introduce more general concepts such as *cooperation* and *engagement*.

1 Introduction

There are many threads of research in Distributed Artificial Intelligence but, to a greater or lesser extent, they can be grouped under the banner either of experimental work, or of formal, theoretical work. Recently, however, some efforts have been made to provide a greater harmony between these two camps, and to integrate the complementary aspects (e.g. [19]). The current work is one such effort. Previously, we have developed a principled theory of agency and autonomy through the provision of a formal framework which defines and characterises these concepts, and specifies the relationship between them [9]. In this paper, we refine that framework so that it may be applied to the Contract Net Protocol which is very firmly situated in the experimental camp. In so doing, we seek to provide a bridge between the formality on the one hand and the practical work on the other.

The current work uses the Z specification language [15] which is increasingly being used for specifying frameworks and systems in AI (e.g. [6, 11]). Z provides the modularity, abstraction and expressiveness that allows a structured account of a computer system and its associated operations to be given at different levels of detail, with system complexity being added at successively lower levels. In addition, Z schemas are particularly suitable in squaring the demands of formal modelling with the need for implementation, by providing clear and unambiguous definitions of state and operations on state which provide a basis for program development. These qualities satisfy our needs for preciseness

through formality but do not detract from our desire to remain connected to issues of implementation.

The paper begins with a brief review of the agent hierarchy framework specified previously. The next section introduces the Contract Net Protocol, and extends the formal specification of the framework to cover the components of, and relationships within, the Contract Net. Then we examine these relationships to provide more general definitions of *cooperation* and *engagement*, and finally we very briefly review related work.

2 The Agent Hierarchy

In this section, we briefly review the agent hierarchy framework for agency and autonomy. Our treatment will be sketchy and will lack many examples due to space constraints. For full details, the reader is referred to [9]. In short, we propose a three-tiered hierarchy of entities comprising *objects*, *agents* and *autonomous agents*. The basic idea underlying this hierarchy is that all known entities are objects. Of this set of objects, some are agents, and of these agents, some are autonomous agents. Below, we define what is required for each of the entities in the hierarchy. First we must define some primitives.

An *action* is a discrete event which changes the state of the environment. An *attribute* is a perceivable feature. It is the only characteristic of the world which is manifest.

$$[Action, Attribute]$$

A *goal* is a state of affairs to be achieved in the environment. It is just a set of attributes that describe a state of affairs in the world:

$$Goal == \mathbb{P}\ Attribute$$

A *motivation* is any desire or preference that can lead to the generation and adoption of goals and which affects the outcome of the reasoning or behavioural task intended to satisfy those goals. (This draws on the definition used by Kunda [8].)

$$[Motivation]$$

Now we can provide a *template* for all the entities in the world and use it to construct formal definitions and specifications of agency and autonomy.

An *entity* is something that comprises a set of attributes, a set of actions, a set of goals and a set of motivations. This is defined in Z using the *state schema* below that has a *declarative* part which contains four variables and their types. First, *attributes* is the set of features of the entity. Second, *capableof* is the set of actions of the entity, and is sometimes referred to as the *competence* of the entity. Next, *goals* and *motivations* are the sets of goals and motivations of the entity respectively.

Entity

$attributes : \mathbb{P}\ Attribute$
$capableof : \mathbb{P}\ Action$
$goals : \mathbb{P}\ Goal$
$motivations : \mathbb{P}\ Motivation$

The type of any schema can be considered as the cartesian product of the types of each of its variables, without any notion of order, but constrained by predicates.

An *object* is an entity with non-empty sets of actions and attributes, and no further defining characteristics. The *Object* schema below has a declarative part that simply includes the previously defined template schema, *Entity*. In addition, it has a *predicate* part which relates and constrains those variables. This specifies that an object must have non-empty sets of attributes and actions. Objects are therefore defined by their ability in terms of their actions, and their configuration in terms of their attributes. The configuration of an object includes references to the body of the object and its position, similar to the notion of Goodwin [6].

Object

Entity

$attributes \neq \{\}$
$capableof \neq \{\}$

Agents are just objects with certain dispositions. An object is an agent if it serves a useful purpose either to a different agent, or to itself, in which case the agent is *autonomous*. Specifically, an agent is something that 'adopts' or satisfies a goal or set of goals (often of another). Thus if I want a robot to make me a cup of coffee, then the robot is my agent for making coffee since it has *adopted* my goal to make coffee. An *agent* is thus defined in relation to its goals, and is an instantiation of an object together with an associated goal or set of goals.

Agent

Object

$goals \neq \{\}$

An agent has or is *ascribed* a set of goals which it retains over any instantiation. One object may give rise to different instantiations of agents; an agent is instantiated from an object in response to another agent. Agency is *transient*, and an object which becomes an agent at some time may subsequently revert to being an object.

In order to ground this goal adoption, there must be some agents which can generate their own goals. These are *autonomous* agents since they are not dependent on the goals of others. Instead of adopting goals from other agents, autonomous agents possess goals which they *generate* themselves from *motivations*

which are higher-level non-derivative internal components characterising the nature of the agent. However, since they are not describable states of affairs in the environment, motivations are distinct from goals. For example, the motivation *survival* does not specify a state of affairs to be achieved, nor is it describable in terms of the environment, but it may (if other motivations permit) give rise to the generation of a goal to flee from danger. The difference between the motivation of survival and the goal of fleeing is clear, with the former providing a reason for doing the latter.

Thus, a *motivated agent* is one which pursues its own agenda for reasoning and behaviour in accordance with its internal motivation. It is this that is the critical factor in achieving autonomy and, consequently, an *autonomous agent* must necessarily be a *motivated* agent.

An *autonomous agent* is any agent which has its own set of motivations. In other words, the behaviour of the agent is determined by both external and internal factors. This is qualitatively different from an agent with goals because motivations are non-derivative and governed by internal inaccessible rules, while goals are derivative and relate directly to motivations.

$$
\begin{array}{|l}
\hline
AutonomousAgent \\
Agent \\
\hline
motivations \neq \{\,\} \\
\hline
\end{array}
$$

3 Formalising the Contract Net Protocol

Now we consider the Contract Net as described by Smith [13, 14, 3], which can be distilled to the basic components described here. Essentially, a *contract net* is a collection of nodes that cooperate in achieving goals which, together, satisfy some high-level goal or task. Each node may be either a *manager*, who monitors task execution and processes the results, or a *contractor*, who performs the actual execution of the task.

Negotiation to undertake and satisfy tasks arises when new tasks are generated. These tasks are decomposed into sub-tasks and, when there may be inadequate knowledge or data to undertake these sub-tasks directly, they are offered for bidding by other agents. A *task announcement* message is broadcast, detailing the task requirements. In response to a task announcement, agents can evaluate their interest using *task evaluation procedures* specific to the problem at hand. If there is sufficient interest, then that agent will submit a bid to undertake to perform the task. The *manager* selects nodes using *bid evaluation procedures* based on the information supplied in the bid. It sends *award* messages to successful bidders who then become *contractors* to the manager, and who may in turn subcontract parts of their task. The manager terminates a contract with a *termination* message.

We can refine the framework described above to arrive at a formal specification of the Contract Net Protocol which retains the structure of the framework.

First, we specify the different kinds of entity from which a contract net is constructed, and which participate in it. A node in a contract net is just an object.

$$\begin{array}{|l}
\textit{CNode} \\
\hline
\textit{Object} \\
\hline
\end{array}$$

A *ContractAgent* is any node currently involved in some task.

$$\begin{array}{|l}
\textit{ContractAgent} \\
\hline
\textit{CNode} \\
\textit{Agent} \\
\hline
\end{array}$$

All nodes in the net are therefore either doing nothing, or doing something, in which case they are agents. The collection of such nodes is given in the following schema.

$$\begin{array}{|l}
\textit{AllNodes} \\
\hline
\textit{nodes} : \mathbb{P}\ \textit{CNode} \\
\textit{contractagents} : \mathbb{P}\ \textit{ContractAgent} \\
\hline
\textit{contractagents} \subseteq \textit{nodes} \\
\hline
\end{array}$$

This completes the definition of the nodes in the net and we now need to consider the function of the net. A manager engages contractors to perform certain tasks. A task is defined to be the same as a goal, as it just specifies a state of affairs to be achieved.

$$\textit{Task} == \textit{Goal}$$

In the next schema, we define a contract to comprise a task, a manager and a contractor. The contractor and manager must be different, and the task must be a goal of both the manager and the contractor.

$$\begin{array}{|l}
\textit{Contract} \\
\hline
\textit{task} : \textit{Task} \\
\textit{manager} : \textit{ContractAgent} \\
\textit{contractor} : \textit{ContractAgent} \\
\hline
\textit{manager} \neq \textit{contractor} \\
\textit{task} \in (\textit{manager.goals} \cap \textit{contractor.goals}) \\
\hline
\end{array}$$

Now we can define the set of all contracts currently in operation in the contract net. The schema below includes *AllNodes*, and defines *contracts* to be the set of all contracts currently in the net. The managers are the set of nodes which are managing a contract and the contractors are the set of nodes which are contracted. The union of the contractors and the managers gives the set of contract agents.

$$
\begin{array}{|l}
\hline
\textit{AllContracts} \\
\hline
\textit{AllNodes} \\
\textit{contracts} : \mathbb{P}\ \textit{Contract} \\
\\
\textit{managers} : \mathbb{P}\ \textit{ContractAgent} \\
\textit{contractors} : \mathbb{P}\ \textit{ContractAgent} \\
\hline
\textit{managers} = \{ c : \textit{Contract} \mid c \in \textit{contracts} \bullet c.\textit{manager} \} \\
\textit{contractors} = \{ c : \textit{Contract} \mid c \in \textit{contracts} \bullet c.\textit{contractor} \} \\
\textit{managers} \cup \textit{contractors} = \textit{contractagents} \\
\hline
\end{array}
$$

We also need to introduce the notion of *eligibility*. A node is eligible for a task if its actions and attributes satisfy the task requirements. We define *Eligibility* to be a type comprising a set of actions and attributes representing an eligibility specification. This has just the same type as an object.

$$\textit{Eligibility} == \textit{Object}$$

The first step in establishing a contract is to issue a *task announcement*. A *TaskAnnouncement* is issued by a *Sender* to a set of *Recipient*s to request bids for a particular *Task* from agents with a given *Eligibility* specification.

$$
\begin{array}{l}
\textit{Sender} == \textit{CNode} \\
\textit{Recipient} == \textit{CNode}
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{TaskAnnouncement} \\
\hline
\textit{sender} : \textit{Sender} \\
\textit{recs} : \mathbb{P}\ \textit{Recipient} \\
\textit{task} : \textit{Task} \\
\textit{eligibility} : \textit{Eligibility} \\
\hline
\end{array}
$$

Notice that the combination of a task together with an eligibility is, in fact, an *agency* requirement.

A bid is issued from some node who describes a subset of itself in response to an eligibility specification which will be used in evaluating the bid.

$$
\begin{array}{|l}
\hline
\textit{Bid} \\
\hline
\textit{cnode} : \textit{CNode} \\
\textit{eligibility} : \textit{Eligibility} \\
\hline
\textit{eligibility.capableof} \subseteq \textit{cnode.capableof} \\
\textit{eligibility.attributes} \subseteq \textit{cnode.attributes} \\
\hline
\end{array}
$$

The state of the contract net can now be represented as the current set of nodes, contracts, task announcements and bids. Each task announcement will have associated with it some set of bids which are just eligibility specifications as described above. In addition, each node has a means of deciding whether it is

capable of, and interested in, performing certain tasks (and so bidding for them). First, we need to define *bool*.

$$bool ::= True \mid False$$

ContractNet

$$AllContracts$$
$$bids : TaskAnnouncement \pfun \mathbb{P}\ Bid$$
$$interested : CNode \fun Task \fun bool$$
$$taskannouncements : \mathbb{P}\ TaskAnnouncement$$

$$taskannouncements = \mathsf{dom}\ bids$$

The operation of a node making a task announcement is then given in the schema below where there is a change to *ContractNet*, but no change to *AllContracts*. A node that issues a task announcement must be an agent. Note that the variables with a ? suffix indicate *inputs* to the operation. The second part of the schema specifies that the recipients and the sender must be nodes, that the task must be in the sender's goals, and that the sender must not be able to satisfy the eligibility requirements of the task alone. Finally, the task announcement is added to the set of all task announcements, and an empty set of bids is associated with it.

MakeTaskAnnouncement

$$\Delta ContractNet$$
$$\Xi AllContracts$$
$$m? : ContractAgent$$
$$ta? : TaskAnnouncement$$

$$m? \in nodes$$
$$ta?.recs \subseteq nodes$$
$$ta?.sender = m?$$
$$ta?.task \in m?.goals$$
$$\neg\ ((ta?.eligibility.capableof \subseteq m?.capableof) \land (ta?.eligibility.attributes \subseteq m?.attributes))$$
$$taskannouncements' = taskannouncements \cup \{ta?\}$$
$$bids' = bids \cup \{(ta?, \{\})\}$$

In response to a task announcement, a node may make a bid. The schema below specifies that a node making a bid must be one of the receivers of the task announcement, that it must be eligible for the task, that it is interested in performing the task, and that it is not the sender. As a result of a node making a bid, the set of task announcements does not change, but the bids associated with the task announcement are updated to include the new bid.

MakeBid

$\Delta ContractNet$
$con? : CNode$
$bid? : Bid$
$ta? : TaskAnnouncement$

$bid?.cnode = con?$
$con? \in nodes$
$ta? \in taskannouncements$
$con? \in ta?.recs$
$ta?.eligibility.capableof \subseteq bid?.eligibility.capableof$
$ta?.eligibility.attributes \subseteq bid?.eligibility.attributes$
$interested\ con?\ (ta?.task) = True$
$con? \neq ta?.sender$
$taskannouncements' = taskannouncements$
$bids' = bids \oplus \{(ta?, bids\ ta? \cup \{bid?\})\}$

After receiving bids, the issuer of a task announcement awards the contract to the highest rated bid. The node that makes the award must be the node that issued the task announcement, and the bid that is selected must be in the set of bids associated with the task announcement. In order to choose the best bid, the *rating* function is used to provide a natural number as an evaluation of a bid with respect to a task announcement. Thus the bid with the highest rating is selected. After making an award, the set of all contracts is updated to include a new contract for the particular task with the issuer of the task announcement as manager and the awarded bidder as contractor, where the contractor is instantiated from the old node as a new agent with the additional task of the contract. The task announcement is now satisfied and removed from the system, and the set of bids is updated accordingly.

MakeAward

$\Delta ContractNet$
$m? : ContractAgent$
$ta? : TaskAnnouncement$
$bid? : Bid$
$rating : TaskAnnouncement \longrightarrow Bid \longrightarrow \mathbb{N}$

$m? = ta?.sender$
$bid? \in bids\ ta?$
$\forall b : Bid \mid b \in bids\ ta? \bullet rating\ ta?\ bid? \geq rating\ ta?\ b$
$contracts' = contracts$
$\quad \cup \{makecontract\ ta?.task\ m?\ (newagent\ bid?.cnode\ ta?.task)\}$
$contractagents' = contractagents \setminus \{newagent\ bid?.cnode\ ta?.task\}$
$\quad \cup \{newagent\ bid?.cnode\ ta?.task\}$
$taskannouncements' = taskannouncements \setminus \{ta?\}$
$bids' = bids \setminus \{(ta?, bids\ ta?)\}$

The functions *makecontract* and *newagent* are defined as follows.

$$
\begin{array}{l}
makecontract : Task \pfun ContractAgent \pfun CNode \pfun Contract \\
newagent : CNode \fun Task \fun ContractAgent \\
\hline
\forall\, t : Task; c : CNode; a : ContractAgent \bullet newagent\ c\ t = a \Leftrightarrow \\
\quad a.attributes = c.attributes \land a.capableof = c.capableof \land \\
\qquad\qquad a.goals = c.goals \cup \{t\} \\
\forall\, t : Task; m : ContractAgent; c : CNode; con : Contract \bullet \\
\quad makecontract\ t\ c\ m = con \Leftrightarrow t \in (m.goals) \land m \neq c \land \\
\quad con.task = t \land con.manager = m \land con.contractor = newagent\ c\ t
\end{array}
$$

Finally, a manager can terminate a contract as specified below where the contract is removed from the set of all contracts.

Whilst the contractor will remove the task from its set of goals the manager will not, since it will still be a contractor for that task or the monitor of that goal. The goal is therefore removed from the goals of the contractor agent. If this node is still an agent, there will be no change to *contractagents*, but if the node previously had only one goal then it will be removed from *contractagents* since it is no longer an agent.

TerminateContract

$$
\begin{array}{l}
\Delta AllContracts \\
m? : ContractAgent \\
con? : ContractAgent \\
t? : Task \\
\hline
contracts' = contracts \setminus \{makecontract\ t?\ m?\ con?\} \\
oldagent\ con?\ t? \in ContractAgent \Rightarrow \\
\quad contractagents' = contractagents \setminus \{con?\} \cup \{oldagent\ con?\ t?\} \\
oldagent\ con?\ t? \notin ContractAgent \Rightarrow \\
\quad contractagents' = contractagents \setminus \{con?\}
\end{array}
$$

The *oldagent* function makes an agent revert to the node it was before adopting the goal of the contract.

$$
\begin{array}{l}
oldagent : CNode \fun Task \fun ContractAgent \\
\hline
\forall\, t : Task; c : CNode; a : ContractAgent \bullet oldagent\ c\ t = a \Leftrightarrow \\
\quad a.attributes = c.attributes \land \\
\quad a.capableof = c.capableof \land \\
\quad a.goals = c.goals \setminus \{t\}
\end{array}
$$

Davis and Smith[3] also describe a single processor node in a distributed sensing example called a *monitor* node which starts the initialisation as the first step in net operation. If this is just another node which passes on information to another, then it is no different to the manager specified above. If it generated the goal or task to perform by itself, then it is autonomous.

Monitor

$AutonomousAgent$
$ContractAgent$

4 Cooperation and Engagement in the Contract Net

The contract net is a useful and effective example of applying the framework proposed earlier because it is a concrete and well-understood system. In addition, many of the relationships that arise in the contract net can be generalised to other goal-directed systems. In this section, we elaborate the framework described earlier by considering cooperation and engagement, especially in the light of the contract net example. Thus we use the contract net case-study as an exemplar which allows us to analyse these relationships, first in a limited and well-defined way, and then by broadening them to define properties of multi-agent systems in general (fuller details of which can be found in [10]).

We now define a new agent, a *server* agent, which is a *non-autonomous* agent.

ServerAgent

$Agent$

$motivations = \{\}$

Just as a contract net is a collection of *CNodes* and *ContractAgents* we define the world as a collection of objects, agents, and autonomous agents.

World

$objects : \mathbb{P}\ Object$
$agents : \mathbb{P}\ Agent$
$autoagents : \mathbb{P}\ AutonomousAgent$
$serveragents : \mathbb{P}\ ServerAgent$

$autoagents \subseteq agents \subseteq objects$
$autoagents \cup serveragents = agents$

Whenever a node that is not autonomous adopts some goal, it is being *engaged*. This is the normal situation in the contract net, where the nodes that participate in a contract need not be autonomous, and the manager *engages* the contractor. In a direct engagement, an agent with some goal, the *client*, uses another agent, the *server*, to assist in achieving that goal. The *server* agent is never autonomous, but the *client* can be either autonomous or non-autonomous.

We can modify the *Contract* schema so that it applies to more general situations. A *direct engagement* consists of a *client*, a *server* and the *goal* that the *server* is satisfying for the *client*. The server and client cannot be the same and, just as in a contract, both agents must possess the goal. The schema below thus captures a generalised version of the information that the *Contract* schema, which refers specifically to contract nets, contains.

DirectEngagement

$goal : Goal$
$client : Agent$
$server : ServerAgent$

$client \neq server$
$goal \in (client.goals \cap server.goals)$

All of the *direct* engagements in the world are given in the following schema by *dengagement*, analogous to *contracts* in the earlier *AllContracts* schema. The client agents of the world are all those which are the clients for some direct engagement, and the server-agents are those which are the server agent for some direct engagement. All these agents are a subset of the agents in the world. Finally, we can say that an agent, c, *directly engages* another server-agent, s, if, and only if, there is a direct engagement between c and s. The set of all such relationships is given by *dengages*. This schema thus captures the same information about the world as the *AllContracts* schema that refers specifically to contract nets.

WorldEngagements

$World$
$dengagement : \mathbb{P}\ DirectEngagement$
$clientagents : \mathbb{P}\ Agent$
$dengages : Agent \leftrightarrow ServerAgent$

$clientagents = \{d : dengagement \bullet d.client\}$
$serveragents = \{d : dengagement \bullet d.server\}$
$\{d : dengagement \bullet d.server\} \cup \{d : dengagement \bullet d.client\} \subseteq agents$
$dengages = \{e : dengagement \bullet (e.client, e.server)\}$

We can also consider the case of an agent that is contracted to perform a task who subcontracts that task to another agent. This leads to the possibility of an *engagement chain* which is a sequence of *direct engagements*. Thus an *engagement chain* involves a *goal*, the autonomous client-agent that generated the goal, *autoagent*, and a sequence of server-agents, *chain*, where each is directly engaging the next. Note that in a contract, the autonomous agent (or monitor) who originally generated the goal may belong to the contract net or may be some external entity.

EngagementChain

$goal : Goal$
$autoagent : AutonomousAgent$
$chain : \mathsf{seq}_1\ Agent$

$goal \in autoagent.goals$
$goal \in \bigcup \{s : Agent \mid s \in \mathsf{ran}\ chain \bullet s.goals\}$
$\#(\mathsf{ran}\ chain) = \#chain$

This leads to the specifying of all engagement chains in the world by *engchain* in the schema below. In a contract net, engagement chains involve contracts and subcontracts of agents, all for one task. Every engagement chain, *ec*, has a direct engagement between the autonomous agent, $ec.autoagent$, and the first client, $head\ ec.chain$, with respect to the goal, $ec.goal$. There must also be a direct engagement between any two agents which follow each other in the chain with respect to the goal.

WorldEngagementChains

$$
\begin{array}{l}
WorldEngagements \\
engchain : \mathbb{P}\ EngagementChain \\
\hline
\forall\, ec : engchain;\ s_1, s_2 : Agent \bullet \\
\quad (\exists\, d : dengagement \bullet d.goal = ec.goal \land d.client = ec.autoagent \\
\qquad \land\ d.server = head\ (ec.chain)) \land \\
\quad \langle s_1, s_2 \rangle \ \mathsf{in}\ ec.chain \Rightarrow (\exists\, d : dengagement \bullet \\
\qquad d.client = s_1 \land d.server = s_2 \land d.goal = ec.goal)
\end{array}
$$

If an autonomous agent adopts the goal of another autonomous agent, then we say that they are *cooperating* with respect to that goal. The term *cooperation* is reserved for use only when the parties involved are autonomous and potentially capable of resisting. If they are not autonomous (and not capable of resisting), then one simply *engages* the other. We distinguish between these relationships on the basis of the autonomy of the agents involved. Cooperation is a *symmetric* relation between two autonomous agents, in contrast to a normal contract in the contract net which is an engagement, an asymmetric relation between a (client) agent and another server-agent.

Thus, a *cooperation* describes a goal, the autonomous agent that originally generated that goal, and the autonomous agents who have adopted that goal from the original agent. It is a more sophisticated relationship than normally appears in the contract net, because it requires autonomy in all participants, a quality which is not necessary in the simpler engagements prevalent there.

Cooperation

$$
\begin{array}{l}
goal : Goal \\
generatingagent : AutonomousAgent \\
cooperatingagents : \mathbb{P}\ AutonomousAgent \\
\hline
\#cooperatingagents \geq 1 \\
\forall\, aa : cooperatingagents \bullet goal \in aa.goals \\
goal \in generatingagent.goals
\end{array}
$$

The set of all cooperations is given in the schema below by *cooperations*. An agent x_1 *cooperates* with agent x_2 if, and only if, both x_1 and x_2 are autonomous, one of them is the generating agent and the other is one of the cooperating agents. In the following schema, *cooperates* describes the set of all such relationships. Since the relationship is symmetric, it is equal to its own inverse.

$$
\begin{array}{|l}
\textit{WorldCooperations} \\
\hline
\textit{World} \\
\textit{cooperations} : \mathbb{P}\ \textit{Cooperation} \\
\textit{cooperates} : \textit{AutonomousAgent} \leftrightarrow \textit{AutonomousAgent} \\
\hline
\textit{cooperates} = \bigcup \{ a1, a2 : \textit{AutonomousAgent} \mid \\
\qquad (\exists\, c : \textit{cooperations} \bullet a1 = c.\textit{generatingagent} \land \\
\qquad\quad a2 \in c.\textit{cooperatingagents}) \bullet \{(a1, a2), (a2, a1)\}\} \\
\textit{cooperates}^{\sim} = \textit{cooperates} \\
\hline
\end{array}
$$

5 Discussion

The contract net is important both because it was a significant effort to tackle cooperative problem solving, and because it is very definitely situated in the practical and experimental camp. Moreover, it is relatively well-defined and understood, and hence very suitable to be used as an exemplar for the kind of work described here. As a result, there have been several extensions proposed to the basic contract net such as [12, 16], and there have been other attempts at formalisation, by Werner [17], and by Wooldridge [18, 5], for example. However, our approach differs markedly: first, we use a well-known generic specification language which ties in closely with implementation issues, and which has a very large user base; and second, we situate our formalisation in the broader context of a general framework for agency and autonomy. We are not concerned with the development of the formalism, but with its application in a succinct way to the abstract framework proposed, the relationships defined within that framework, and to the specification of concrete systems using the framework to provide structure. Although alternative specification languages such as DESIRE [1], for example, would also have been possible, Z's qualities of encapsulation and abstraction within a formalism used extensively in industry and academia, for both small and large-scale systems (e.g. [2, 11, 7]) and for more theoretical approaches to multi-agent systems (e.g. [4]), provide a more *accessible* method.

In this paper we have outlined previous work on constructing a formal framework for autonomous agent systems, within which particular models and systems can be specified by adding further levels of detail. In that vein, we have described the contract net protocol and specified it formally, making use of the entities defined and described within the framework. The contract net protocol provides exemplars of certain commonly occurring inter-agent relationships such as cooperation and engagement. By using the example of the contract net and generalising the relationships found therein, we have been able to elaborate the formal framework to include definitions of these general relationships, building up a common and general language with which to discuss multi-agent systems and models. A key feature of the work that is illustrated in this paper is the ease with which a particular system, described in detail, can be accommodated by the framework and used to focus further development of the theoretical underpinnings of multi-agent systems.

References

1. F. Brazier, B. Dunin Keplicz, N. Jennings, and J. Treur. Formal specification of multi-agent systems: A real-world case. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 25–32. AAAI Press / MIT Press, 1995.
2. I. Craig. *Formal Specification of Advanced AI Architectures*. Ellis Horwood, 1991.
3. R. Davis and R. G. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20(1):63–109, 1983.
4. M. d'Inverno and M. Luck. A formal view of social dependence networks. In *Proceedings of the First Australian DAI Workshop*. Springer Verlag, To appear, 1996.
5. M. Fisher and M. Wooldridge. Specifying and executing protocols for cooperative action. In S. Deen, ed., *CKBS-94: Proceedings of the Second International Working Conference on Cooperating Knowledge-Based Systems*. Springer-Verlag, 1994.
6. R. Goodwin. Formalizing properties of agents. Technical Report CMU-CS-93-159, Carnegie-Mellon University, 1993.
7. M. G. Hinchey and J. P. Bowen, editors. *Applications of Formal Methods*. Prentice Hall International Series in Computer Science, 1995.
8. Z. Kunda. The case for motivated reasoning. *Psychological Bulletin*, 108(3):480–498, 1990.
9. M. Luck and M. d'Inverno. A formal framework for agency and autonomy. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 254–260. AAAI Press / MIT Press, 1995.
10. M. Luck and M. d'Inverno. Engagement and cooperation in motivated agent modelling. In *Proceedings of the First Australian DAI Workshop*. Springer Verlag, To appear, 1996.
11. B. G. Milnes. A specification of the Soar architecture in Z. Technical Report CMU-CS-92-169, School of Computer Science, Carnegie Mellon University, 1992.
12. T. Sandholm. An implementation of the contract net protocol based on marginal cost calculations. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pages 256–262. AAAI Press / MIT Press, 1993.
13. R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, 29(12), 1980.
14. R. G. Smith and R. Davis. Frameworks for cooperation in distributed problem solving. *IEEE Transactions on Systems, Man and Cybernetics*, 11(1):61–70, 1981.
15. J. M. Spivey. *The Z Notation*. Prentice Hall, Hemel Hempstead, 2nd edition, 1992.
16. H. Van Dyke Parunak. Manufacturing experience with the contract net. In M. Huhns, editor, *Distributed Artificial Intelligence*, pages 285–310. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA, 1987.
17. E. Werner. Cooperating agents: A unified theory of communication and social structure. In L. Gasser and M. Huhns, editors, *Distributed Artificial Intelligence Volume II*, pages 3–36. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA, 1989.
18. M. Wooldridge. *The Logical Modelling of Computational Multi-Agent Systems*. PhD thesis, Department of Computation, UMIST, Manchester, UK, October 1992.
19. M. J. Wooldridge and N. R. Jennings. Formalizing the cooperative problem solving process. In *Proceedings of the Thirteenth International Workshop on Distributed Artificial Intelligence*, 1994.

This article was processed using the LaTeX macro package with LLNCS style