Development and Application of a Formal Agent Framework

Mark d'Inverno
Department of Computer Science
University of Westminster
London, W1M 8JS, UK
dinverm@wmin.ac.uk

Michael Luck
Department of Computer Science
University of Warwick
Coventry, CV4 7AL, UK
mikeluck@dcs.warwick.ac.uk

Abstract

Previous work has addressed the development of a framework to categorise and understand agent-based systems. It described and formalised an agent-hierarchy that included objects, agents and autonomous agents, each with different levels of functionality, and provided a precise vocabulary with which to discuss agent systems. This paper reviews a large variety of further work that has built on that foundation in several ways. First, the framework itself has been refined to detail important aspects of agent functionality such as goal generation and adoption. Second, the structures and relationships between agents have been specified and analysed allowing a more complete understanding of the dynamics of agent systems. Third, existing systems and theories have been formalised within the framework so that they may be evaluated and compared in a coherent and consistent way. Finally, some steps have been taken in attempting to construct a methodology for the development of agent-based systems. Though this work spans a large range of concerns, it is based on a single set of basic concepts providing fundamental structure.

1. Introduction

The problems with existing notions of agency and autonomy are now well-understood, but the importance of these notions remains high, nevertheless. In previous work we have addressed this by constructing a formal specification to identify and characterise those entities called agents and autonomous agents, in a precise yet accessible way. Our taxonomy provides clear definitions for objects, agents and autonomous agents that allow a better understanding of the functionality of different systems. It explicates those factors that are necessary for agency and autonomy, and is sufficiently abstract to cover the gamut of agents, both hardware and software, intelligent and unintelligent. A significant claim of that work was that it would provide a general mathematical framework within which different models, and particular systems, could be defined and contrasted.

In particular, we argued that a formal framework should satisfy three distinct requirements [8, 4] which we summarise below.

1. It must precisely and unambiguously provide meanings for common concepts and terms and do so in a readable and understandable manner.

2. It should enable alternative designs of particular models and systems to be explicitly presented, compared and evaluated.

3. It should be sufficiently well-structured to provide a foundation for subsequent development of new and increasingly more refined concepts.

Though the framework was, of itself, useful in illuminating some key issues in agent-oriented systems and in providing a precise and structured vocabulary for discussing them much of the merit of our work lies in its ability to span a range of levels of abstraction, including application to both existing systems and theories [2, 3], and in allowing further theoretical and practical development. This paper is an attempt to take stock of progress to date in addressing these concerns, and in evaluating how successful we have been with regard to the requirements enumerated above. It brings together much of the work that has previously been done in our research programme. The next section briefly reviews the agent framework and the components within it as a base for the remainder of the paper. Then we describe further development of the framework in mechanisms for goal generation and adoption, the structure of inter-agent relationships of engagement and cooperation, and an analysis of those relationships. After that the application of the framework to existing systems and theories is illustrated with the example of the Contract Net, and finally we consider the implications for practical systems development.

2. Agent Framework

In short, we propose a four-tiered hierarchy comprising *entities*, *objects*, *agents* and *autonomous agents*. The basic idea underlying this hierarchy is that all components of the world are entities. Of these entities, some are objects, of which some, in turn, are agents and of these, some are autonomous agents. In this section, we briefly outline the agent hierarchy. Many details are omitted — a more complete treatment can be found in [7].

Entities can be used to group together attributes into a whole without adding a layer of *functionality*. They serve as a useful abstraction mechanism by which they are regarded as distinct from the remainder of the environment, and which can organise perception. An object is just an entity with abilities which can affect environments in which it is situated. An agent is just an object either that is useful to another agent where this usefulness is defined in terms of satisfying that agent's goals, or that exhibits independent purposeful behaviour. In other words, an agent is an object with an associated set of goals. One object may give rise to different instantiations of agents which are created in response to other agents. This definition of agency relies upon the existence of such other agents which provide the goals that are adopted instantiate an agent. In order to escape an infinite regress of goal adoption, we can define autonomous agents which are just agents that generate their own goals from motivations.

$$
\begin{array}{|l}
\hline
\mathit{Entity} \\
\mathit{attributes} : \mathbb{P}\,\mathit{Attribute} \\
\mathit{capableof} : \mathbb{P}\,\mathit{Action} \\
\mathit{goals} : \mathbb{P}\,\mathit{Goal} \\
\mathit{motivations} : \mathbb{P}\,\mathit{Motivation} \\
\hline
\mathit{attributes} \neq \{\ \} \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\mathit{Object} == [\mathit{Entity} \mid \mathit{capableof} \neq \{\ \}] \\
\mathit{Agent} == [\mathit{Object} \mid \mathit{goals} \neq \{\ \}] \\
\mathit{AutoAgent} == [\mathit{Agent} \mid \mathit{motivations} \neq \{\ \}]
\end{array}
$$

We also distinguish those objects which are not agents, and those agents which are not autonomous and refer to them as *neutral-objects* and *server-agents* respectively. An agent is then either a server-agent or an autonomous agent, and an object is either a neutral-object or an agent.

$$
\begin{array}{l}
\mathit{NeutralObject} == [\mathit{Object} \mid \mathit{goals} = \{\}] \\
\mathit{ServerAgent} == [\mathit{Agent} \mid \mathit{motivations} = \{\}]
\end{array}
$$

With the basic components of the framework in place, we now go on to develop the framework in order to give an account of how goals are initially generated by autonomous agents and subsequently adopted by other entities in the environment.

3. Goal Generation and Adoption

The four-tiered framework described above involves the generation of *goals* from *motivations* in an autonomous agent, and the adoption of goals by, and in order to create, other agents. In this section, we consider issues in goal generation that must occur before goal adoption can take place. Specifically, we describe how an autonomous agent, can construct goals or concrete states of affairs to be achieved in the environment. We extend the framework in this way and add more detail by introducing new schemas that specify the relevant aspects.

An autonomous agent will try to find a way to mitigate motivations, either by selecting an action to achieve an existing goal as above for simple agents, or by retrieving a goal from a repository of known goals. Thus, our model requires a repository of known *goals* which capture knowledge of limited and well-defined aspects of the world. These goals describe particular *states* or *sub-states* of the world with each autonomous agent having its own such repository.

In order to retrieve goals to mitigate motivations, an autonomous agent must have some way of assessing the effects of competing or alternative goals. Clearly, the goals which make the greatest positive contribution to the motivations of the agent should be selected unless a greater motivational effect can be achieved by *destroying* some subset of its goals. The motivational effect of generating or destroying goals is not only dependent on the current motivations but also on the current goals of the agent. For example, an autonomous agent should not generate a goal that it already possesses or that is incompatible with the achievement or satisfaction of its existing goals.

Formally, the ability of autonomous agents to assess goals is given in the next schema, *AssessGoals*. The schema describes how an autonomous agent monitors its motivations for goal generation. First, the *AutoAgent* schema is included and the new variable representing the repository of available known goals, *goalbase* is declared. Then, the motivational effect on an autonomous agent of satisfying a set of new goals is given. The *generate* function returns a numeric value representing the motivational effect of satisfying a set of goals with a particular configuration of motivations and a set of existing goals. Similarly, the *destroy* function returns a numeric value representing the motivational effect of removing some subset of its existing goals with the same configuration. The predicate part specifies that the goal base is non-empty, and that all the current goals must be goals that exist in the goalbase. For ease of expression, we also define a function related to *generate* called *satgen*, which returns the motivational effect of an autonomous agent satisfying an additional set of goals. The function, *satdes*, is analogously related to *destroy*.

$$\begin{array}{|l}
\textit{AssessGoals} \\
\hline
\textit{AutoAgent} \\
\textit{goalbase} : \mathbb{P}\,\textit{Goal} \\
\textit{generate}, \textit{destroy} : \\
\quad \mathbb{P}\,\textit{Motivation} \rightarrow \mathbb{P}\,\textit{Goal} \rightarrow \mathbb{P}\,\textit{Goal} \rightarrow \mathbb{Z} \\
\textit{satgen}, \textit{satdes} : \mathbb{P}\,\textit{Goal} \rightarrow \mathbb{Z} \\
\hline
\textit{goalbase} \neq \{\} \ \wedge \ \textit{goals} \subseteq \textit{goalbase} \\
\forall\, g : \mathbb{P}\,\textit{goalbase} \bullet \\
\ \textit{satgen}\ g = \textit{generate motivations goals}\ g \ \wedge \\
\ \textit{satdes}\ g = \textit{destroy motivations goals}\ g \\
\hline
\end{array}$$

Now we can describe the generation of a new set of goals in the *GenerateGoals* operation schema. This simply states that there is a set of goals in the goalbase that has a greater motivational effect than any other set of goals, and the current goals of the agent are updated to include the new goals.

$$\begin{array}{|l}
\textit{GenerateGoals} \\
\hline
\Delta \textit{AutoAgent} \\
\textit{AssessGoals} \\
\hline
\textit{goalbase} \neq \{\ \} \\
\exists\, \textit{gs} : \mathbb{P}\,\textit{Goal} \mid \textit{gs} \subseteq \textit{goalbase} \bullet \\
\qquad (\forall\, \textit{os} : \mathbb{P}\,\textit{Goal} \mid \textit{os} \in (\mathbb{P}\,\textit{goalbase}) \bullet \\
\qquad\qquad (\textit{satgen gs} \geq \textit{satgen os}) \wedge \\
\qquad\qquad \textit{goals}' = \textit{goals} \cup \textit{gs}) \\
\hline
\end{array}$$

Since we are interested in multi-agent systems, we must consider the world as a whole rather than just individual agents. A multi-agent system contains entities, objects, neutral objects, agents, server agents and autonomous agents.

$$\begin{array}{|l}
\textit{MultiAgentSysComponents} \\
\hline
\textit{entities} : \mathbb{P}\,\textit{Entity} \\
\textit{objects} : \mathbb{P}\,\textit{Object} \\
\textit{agents} : \mathbb{P}\,\textit{Agent} \\
\textit{autoagents} : \mathbb{P}\,\textit{AutoAgent} \\
\textit{neutralobjects} : \mathbb{P}\,\textit{NeutralObject} \\
\textit{serveragents} : \mathbb{P}\,\textit{ServerAgent} \\
\hline
\textit{autoagents} \subseteq \textit{agents} \subseteq \textit{objects} \\
\textit{agents} = \textit{autoagents} \cup \textit{serveragents} \\
\textit{objects} = \textit{neutralobjects} \cup \textit{agents} \\
\hline
\end{array}$$

In multi-agent systems, agents may wish, or need, to use the capabilities of other entities. They can make use of the capabilities of these others by *adopting* their goals. For example, if agent *A* needs to move a table and requires the help of another agent, *B*, to do so, then *B* must first adopt the goal to move the table. This notion of goal adoption underlies social behaviour, and an understanding of the ways in which it can be achieved is fundamental for effective modelling and simulation. In general, entities may serve the purposes of others by adopting their goals. However, the ways in which they adopt goals depends on the kind of entity. They may be either neutral-objects, server-agents or autonomous agents, and each requires a separate analysis.

In the description given in the previous section, goals may be generated only by autonomous agents. Both non-autonomous (server) and autonomous agents, however, can adopt goals. With autonomous agents, goal adoption amounts to a problem of *negotiation* or *persuasion*, requiring an analysis of the *target* autonomous agent. With non-autonomous agents, goal adoption requires an analysis of both the agent intended to adopt the goal, and any other agent *engaging* that agent. With objects, no analysis is required, since agents are *created* from objects with the relevant associated goals.

There are three fundamental cases of goal adoption which we consider in detail. In the simplest case, goal adoption by non-autonomous agents occurs by instantiating an agent from a neutral object with the goals to be adopted. In this case, no *agent* exists before the goals are adopted, but the act of goal transfer causes an agent to be created from a neutral object using those particular goals. Thus, for example, the table in my office, which is just an object, becomes an agent when I use it for supporting my computer, when it *adopts* or *is ascribed* my goal of supporting the computer. It is only possible to create the agent from the object because the table is not being used by anyone else — it is not *engaged* by another agent. An entity can only be a neutral object if it is not *engaged*.

We now specify how a non-autonomous disengaged object, or neutral-object, is instantiated as a server-agent. A neutral-object and a set of goals are input, the entities in the world change and the sets of objects and agents are updated accordingly. First, the set of neutral objects no longer includes the originally disengaged object. Second, the set of server agents now includes the newly created server-agent. Finally, there is no change to the set of autonomous agents. In addition, the variables, *entities*, *objects* and *agents*, are updated by removing the neutral-object and adding the newly instantiated server-agent. The auxiliary function *EntityAdoptGoals* creates a new entity by ascribing a set of goals to an existing entity.

$$\begin{array}{|l}
\textit{NeutralObjectAdoptGoals} \\
\hline
o? : \textit{NeutralObject};\ \textit{gs}? : \mathbb{P}\,\textit{Goal} \\
\Delta \textit{MultiAgentSysComponents} \\
\hline
o? \in \textit{neutralobjects} \\
\textit{neutralobjects}' = \textit{neutralobjects} \setminus \{o?\} \\
\textit{serveragents}' = \textit{serveragents} \cup \\
\qquad\qquad \{\textit{EntityAdoptGoals}\ (o?, \textit{gs}?)\} \\
\textit{autoagents}' = \textit{autoagents} \\
\hline
\end{array}$$

If the table was *engaged* by another (possibly non-autonomous) agent, then it is itself an agent, and the protocol for goal adoption changes. In this case, there are alternative ways for me to *engage* the table. The first of these involves me trying to persuade the engaging agent to release the table so that I may then subsequently engage it for my purposes. This relates to the issues of goal adoption for autonomous agents which are considered later. The second involves supplying the agent with more goals, so that the agent is shared between different engaging agents. The third possibility involves *displacing* the engaging agent so that I become the engaging agent and ascribe to the table my own goals. For example, the table may currently be used as a door-stop for my office-mate, and is therefore her agent with her goal of holding open the door. I can displace the goal ascribed to the table by removing the table and placing my computer on it. Now the table is ascribed my goal of supporting my computer, and it has switched from one agent to another. In fact, this is equivalent to the agent reverting to an object and then being re-instantiated as a new agent. This method may not be an appropriate strategy, however, because in destroying the agency of the table as a door-stop, I risk a conflict with the existing engaging agent, my office-mate. It would be better for me to negotiate first, to obtain permission to destroy the original agency. Our notion of agency thus contributes to a better understanding of the world, regardless of whether we are concerned with tables or robots, since the only important difference between them is their functionality through agency.

Below, we specify a server-agent being ascribed an additional set of goals. It describes the table serving as a door-stop subsequently being given the goal of supporting the computer. The adopting agent must be a server-agent in the system and the new goals are distinct from the existing goals.

$$
\begin{array}{|l}
\textit{ServerAgentAdoptGoals} \\
\hline
a? : \textit{ServerAgent};\ gs? : \mathbb{P}\,\textit{Goal} \\
\Delta \textit{MultiAgentSysComponents} \\
\hline
a? \in \textit{serveragents} \\
gs? \cap a?.\textit{goals} = \{\} \\
\textit{neutralobjects}' = \textit{neutralobjects} \\
\textit{serveragents}' = \textit{serveragents} \setminus \{a?\} \\
\qquad \cup \{\textit{EntityAdoptGoals}\,(a?, gs?)\} \\
\textit{autoagents}' = \textit{autoagents} \\
\hline
\end{array}
$$

With autonomous agents, goals must explicitly be adopted, as opposed to an implicit ascription of goals for non-autonomous agents. This may be more difficult than the previous case, since it requires some form of negotiation. Autonomous agents are motivated agents and will only participate in an activity and assist another agent if it is to their motivational advantage to do so. They create their own agendas and for them, goal adoption is a *voluntary* process as opposed to *obligatory* adoption for non-autonomous agents. The schema below merely states that the set of new goals which the agent adopts are the best that it can find in its goalbase at that time. First, an autonomous agent and set of goals are input, and there is a change to the *MultiAgentSysComponents*. The predicate part of the schema then simply describes a change to the set of autonomous agents by which the old autonomous agent agent is removed and the new one, formed from the old agent with new goals, is added, giving the required change. The last line of the schema states that there is no other set of goals in the goalbase which could have a better motivational effect than the set of goals adopted. Note the use of the *motiveffect* function originally defined for goal generation, so that autonomous goal adoption requires goal generation.

$$
\begin{array}{|l}
\textit{AutoAgentAdoptGoals} \\
\hline
\textit{AssessGoals} \\
aa? : \textit{AutoAgent};\ gs? : \mathbb{P}\,\textit{Goal} \\
\Delta \textit{MultiAgentSysComponents} \\
\hline
aa? \in \textit{autoagents} \\
\textit{autoagents}' = \textit{autoagents} \setminus \{aa?\} \cup \\
\qquad \{\textit{EntityAdoptGoals}\,(aa?, gs?)\} \\
\textit{agents}' = \textit{agents} \ \wedge\ \textit{objects}' = \textit{objects} \\
\neg\,(\exists\, hs : \mathbb{P}\,\textit{Goal} \mid hs \subseteq \textit{goalbase} \wedge \\
\qquad hs \neq gs? \bullet \textit{satgen}\ hs > \textit{satgen}\ gs?) \\
\hline
\end{array}
$$

4. Inter-Agent Relationships

Now, a direct engagement occurs when a neutral-object or a server-agent adopts some goals. In a direct engagement, a *client*-agent with some goals uses another *server*-agent to assist them in the achievement of those goals. A server-agent either exists already as a result of some other engagement, or is instantiated from a neutral-object for the current engagement. No restriction is placed on a client-agent. We define a *direct engagement* below to consist of a client agent, *client*, a server agent, *server*, and the goal that *server* is satisfying for *client*. An agent cannot engage itself, and both agents must have the goal of the engagement.

$$
\begin{array}{|l}
\textit{DirectEngagement} \\
\hline
\textit{client} : \textit{Agent};\ \textit{server} : \textit{ServerAgent} \\
\textit{goal} : \textit{Goal} \\
\hline
\textit{client} \neq \textit{server} \\
\textit{goal} \in (\textit{client}.\textit{goals} \cap \textit{server}.\textit{goals}) \\
\hline
\end{array}
$$

The set of all *direct* engagements in a system is given by *direngs* in the following schema. For any direct engagement in *direngs*, there can be no intermediate *direct* engagements of the goal, so there is no other agent, *y*, where *client* engages *y* for *goal*, and *y* engages *server* for *goal*.

$$\begin{array}{l}
\textit{SysEngagements} \\
\hline
\textit{MultiAgentSysComponents} \\
\textit{direngs} : \mathbb{P}\, \textit{DirectEngagement} \\
\hline
\forall\, en : direngs \bullet \\
\quad \neg\, (\exists\, y : Agent;\ c, d : direngs \mid \\
\qquad c.goal = d.goal = en.goal \bullet \\
\qquad c.server = en.server \land \\
\qquad d.client = en.client \land \\
\qquad d.client = d.server = y)
\end{array}$$

An *engagement chain* represents a sequence of *direct engagements*. Specifically, an *engagement chain* comprises a goal, *goal*, the autonomous client that generated the goal, *auto*, and a sequence of server-agents, *chain*, where each agent in the sequence directly engages the next. For any engagement chain, there must be at least one server-agent, all the agents involved must share *goal*, and each agent can only be involved once.

$$\begin{array}{l}
\textit{EngChain} \\
\hline
goal : Goal;\ auto : AutoAgent \\
chain : \mathrm{seq}_1\, Agent \\
\hline
goal \in auto.goals \\
goal \in \bigcup \{ s : Agent \mid \langle s \rangle \text{ in } chain \bullet s.goals \}
\end{array}$$

The set of all engagement chains in a system is given in the schema below by *engchains*. For every engagement chain, *ec*, there must be a direct engagement between the autonomous agent, *ec.auto*, and the first client of *ec*, *head ec.chain*, with respect to the goal of *ec*, *ec.goal*. Further, there must be a direct engagement between any two agents following each other in *ec.chain* with respect to *ec.goal*. In general, an agent *engages* another agent if there is some engagement chain in which it precedes the server agent.

$$\begin{array}{l}
\textit{SysEngChains} \\
\hline
\textit{SysEngagements} \\
engchains : \mathbb{P}\, EngChain \\
\hline
\forall\, ec : engchains;\ s_1, s_2 : Agent \bullet \\
\quad (\exists\, d : direngs \bullet \\
\qquad d.goal = ec.goal \land \\
\qquad d.client = ec.auto \land \\
\qquad d.server = head\ ec.chain) \land \\
\qquad \langle s_1, s_2 \rangle \text{ in } ec.chain \Rightarrow \\
\qquad (\exists\, d : direngs \bullet \\
\qquad\quad d.client = s_1 \land \\
\qquad\quad d.server = s_2 \land d.goal = ec.goal)
\end{array}$$

Two autonomous agents are said to be *cooperating* with respect to some goal if one of the agents has adopted goals of the other. This notion of autonomous goal acquisition applies both to the *origination* of goals by an autonomous agent for its own purposes, and the *adoption* of goals from others, since in each case the goal must have a positive motivational effect. For autonomous agents, the goal of another can only be adopted if it has such an effect, and this is also exactly why and how goals are originated. Thus goal adoption and origination are related forms of goal generation. Thus the term *cooperation* can be used only when those involved are autonomous and, at least potentially, capable of resisting. If they are not autonomous, nor capable of resisting, then one simply *engages* the other.

A *cooperation* describes a goal, the autonomous agent that generated the goal, and those autonomous agents that have adopted that goal from the generating agent. In addition, all the agents involved have the goal of the cooperation, an agent cannot cooperate with itself, and the set of cooperating agents must be non-empty. Cooperation cannot, therefore, occur unwittingly between agents, but must arise as a result of the motivations of an agent and the agent recognising that goal in another.

$$\begin{array}{l}
\textit{Cooperation} \\
\hline
goal : Goal;\ genagent : AutoAgent \\
coopagents : \mathbb{P}\, AutoAgent \\
\hline
goal \in genagent.goals \\
\forall\, aa : coopagents \bullet goal \in aa.goals \\
genagent \notin coopagents \\
coopagents \neq \{\ \}
\end{array}$$

The set of cooperations in a multi-agent system is given by the variable, *coops*, in the schema, *SysCoops*. The predicate part of the schema states that for any cooperation, the union of the cooperating agents and the generating agent is a *subset* of the set of all autonomous agents which have that goal. As a consequence, two agents sharing a goal are not necessarily cooperating. In addition, the set of all cooperating agents is a subset of all autonomous agents since not all are necessarily participating in cooperations.

$$\begin{array}{l}
\textit{SysCoops} \\
\hline
\textit{MultiAgentSysComponents} \\
coops : \mathbb{P}\, Cooperation \\
\hline
\forall\, c : coops \bullet c.coopagents \cup \{c.genagent\} \\
\quad \subseteq \{ a : autoagents \mid c.goal \in a.goals \bullet a \} \\
\bigcup \{ c : coops \bullet c.coopagents \} \subseteq autoagents
\end{array}$$

We define the structure of a multi-agent system in terms of the set of entities in it, and the inter-agent cooperation and engagement relationships between them.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{MultiAgentSysStructure}} \\
\hline
\textit{MultiAgentSysComponents} \\
\textit{SysEngChains} \\
\textit{SysCoops} \\
\hline
\end{array}
$$

Considering the set of engagements and cooperations between agents provides precise information about the relationships between them. This allows a richer understanding of the social configuration of agents, suggesting different possibilities for interaction. For example, if I am currently engaging an entity, and no other agent is doing so, then I can interact with that entity without concern for the potential effects of the interaction on others. This is because the engagement is independent of the existing social configuration of the entire system.

We have provided a full taxonomy of these relations but consider some specific definitions of relations which hold between two agents. These are: *dengs*, *engages*, *owns*, *downs* and *cooperates*. In what follows, we give an initial description followed by the formal definition. The basic relationships between agents are either when one is directly engaging another or when one is cooperating with another.

Definition An agent, *c*, *directly engages* another server-agent, *s*, if, and only if, there is a direct engagement between *c* and *s*.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Dengages}} \\
\hline
\textit{MultiAgentSysStructure} \\
\textit{dengs} : \textit{Agent} \leftrightarrow \textit{ServerAgent} \\
\hline
\textit{dengs} = \{ e : \textit{direngs} \bullet (e.\textit{client}, e.\textit{server}) \} \\
\hline
\end{array}
$$

Definition An agent, *c*, *engages* another server-agent, *s*, if there is some engagement chain in which *c* precedes *s*.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Engages}} \\
\hline
\textit{MultiAgentSysStructure} \\
\textit{engages} : \textit{Agent} \leftrightarrow \textit{ServerAgent} \\
\hline
\textit{engages} = \\
\quad \{ \textit{ec} : \textit{engchains} \bullet (\textit{ec}.\textit{auto}, \textit{head}\ \textit{ec}.\textit{chain}) \} \\
\quad \cup \{ \textit{ec} : \textit{engchains};\ c, s : \textit{Agent} \mid \\
\qquad ((c, s), \textit{ec}.\textit{chain}) \in \textit{follows} \bullet (c, s) \} \\
\hline
\end{array}
$$

We have made use here of the generic relation, *follows* as defined below. It holds between a pair of elements and a sequence of elements if the first element of the pair precedes the second element in the sequence.

$$
\begin{array}{|l}
\multicolumn{1}{l}{[X]} \\
\hline\hline
\textit{follows} : (X \times X) \leftrightarrow \operatorname{seq} X \\
\hline
\forall a, b : X;\ s : \operatorname{seq} X \bullet ((a, b), s) \in \textit{follows} \Leftrightarrow \\
\quad \exists t, u, v : \operatorname{seq} X \bullet s = t \frown \langle a \rangle \frown u \frown \langle b \rangle \frown v \\
\hline
\end{array}
$$

If many agents are directly engaging the same entity, then no single agent has complete control over that entity. Any actions that an agent takes affecting the entity may destroy or hinder the engagements of other engaging agents, and this, in turn, may have a future deleterious effect on any agents or autonomous agents engaging it. It is thus important multi-agent systems analysis to understand and specify exactly *when* the behaviour of an engaged entity can be modified without any such deleterious effect. This can occur between an agent and an entity precisely when there is no other agent using the entity for a *different* purpose.

Definition An agent, *c*, owns another agent, *s*, if, for every sequence of server-agents in an engagement chain in which *s* appears, *c* precedes it, or *c* is the autonomous client-agent that initiates the chain.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Owns}} \\
\hline
\textit{Dengages} \\
\textit{owns} : \textit{Agent} \leftrightarrow \textit{ServerAgent} \\
\hline
\forall c : \textit{Agent};\ s : \textit{ServerAgent} \bullet \\
\qquad (c, s) \in \textit{owns} \Leftrightarrow \\
\qquad (\forall \textit{ec} : \textit{engchains} \mid s \in \operatorname{ran} \textit{ec}.\textit{chain} \bullet \\
\qquad\qquad (c, s) \in \textit{owns} \cap \textit{dengs}) \\
\hline
\end{array}
$$

Definition An agent, *A*, *cooperates* with agent, *B*, if and only if both agents are autonomous, and there is some cooperation in which *A* is the generating agent, and *B* is in the set of cooperating agents.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Cooperates}} \\
\hline
\textit{MultiAgentSysStructure} \\
\textit{cooperates} : \textit{AutoAgent} \leftrightarrow \textit{AutoAgent} \\
\hline
\textit{cooperates} = \bigcup \{ a, b : \textit{AutoAgent} \mid \\
\qquad (\exists c : \textit{coops} \bullet a = c.\textit{genagent} \wedge \\
\qquad\qquad b \in c.\textit{coopagents}) \bullet \{(a, b)\} \} \\
\hline
\end{array}
$$

5. Application to Systems and Theories

We have refined the agent framework described above to arrive at formal specifications of existing multi-agent systems and in this section we review the specification of the Contract Net Protocol which retains the structure of the framework. The Contract Net is by far the most successful multi-agent system technique and has been used for many applications as well as a means of relating new agent theories. As described by Smith [12], it 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.

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. Similarly, a *CAgent* is any node currently involved in some task.

$$\mathit{CNode} == \mathit{Object};\ \ \mathit{CAgent} == \mathit{Agent}$$

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.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{AllNodes}}\\
\hline
\mathit{nodes} : \mathbb{P}\,\mathit{CNode};\ \ \mathit{conagents} : \mathbb{P}\,\mathit{CAgent}\\
\hline
\mathit{conagents} \subseteq \mathit{nodes}\\
\hline
\end{array}$$

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.

$$\mathit{Task} == \mathit{Goal}$$

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.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{Contract}}\\
\hline
\mathit{task} : \mathit{Task};\ \ \mathit{man} : \mathit{CAgent};\ \ \mathit{con} : \mathit{CAgent}\\
\hline
\mathit{man} \neq \mathit{con}\\
\mathit{task} \in (\mathit{man}.\mathit{goals} \cap \mathit{con}.\mathit{goals})\\
\hline
\end{array}$$

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.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{AllContracts}}\\
\hline
\mathit{AllNodes}\\
\mathit{contracts} : \mathbb{P}\,\mathit{Contract}\\
\mathit{mans}, \mathit{cons} : \mathbb{P}\,\mathit{CAgent}\\
\hline
\mathit{mans} = \{c : \mathit{Contract} \mid\\
\qquad\qquad c \in \mathit{contracts} \bullet c.\mathit{man}\}\\
\mathit{cons} = \{c : \mathit{Contract} \mid\\
\qquad\qquad c \in \mathit{contracts} \bullet c.\mathit{con}\}\\
\mathit{mans} \cup \mathit{cons} = \mathit{conagents}\\
\hline
\end{array}$$

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.

$$\mathit{Eligibility} == \mathit{Object}$$

The first step in establishing a contract is to issue a *task announcement*. A *TaskAnn* 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.

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

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{TaskAnn}}\\
\hline
\mathit{sender} : \mathit{Sender};\ \ \mathit{recs} : \mathbb{P}\,\mathit{Recipient}\\
\mathit{task} : \mathit{Task};\ \ \mathit{elig} : \mathit{Eligibility}\\
\hline
\end{array}$$

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.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{Bid}}\\
\hline
\mathit{cnode} : \mathit{CNode};\ \ \mathit{elig} : \mathit{Eligibility}\\
\hline
\mathit{elig}.\mathit{capableof} \subseteq \mathit{cnode}.\mathit{capableof}\\
\mathit{elig}.\mathit{attributes} \subseteq \mathit{cnode}.\mathit{attributes}\\
\hline
\end{array}$$

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).

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{ContractNet}} \\ \hline
\mathit{AllContracts} \\
\mathit{bids} : \mathit{TaskAnn} \pfun \mathbb{P}\,\mathit{Bid} \\
\mathsf{interested}\ \_ : \mathbb{P}(\mathit{CNode} \times \mathit{Task}) \\
\mathit{taskanns} : \mathbb{P}\,\mathit{TaskAnn} \\ \hline
\mathit{taskanns} = \operatorname{dom} \mathit{bids} \\ \hline
\end{array}$$

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. The second part of the schema specifies that the recipients and the sender must be nodes, that the task must be in the senders 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.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{MakeTaskAnn}} \\ \hline
\Delta \mathit{ContractNet} \\
\Xi \mathit{AllContracts} \\
m? : \mathit{CAgent};\ ta? : \mathit{TaskAnn} \\ \hline
m? \in \mathit{nodes}\ \wedge\ ta?.\mathit{recs} \subseteq \mathit{nodes} \\
ta?.\mathit{sender} = m?\ \wedge\ ta?.\mathit{task} \in m?.\mathit{goals} \\
\neg\ ((ta?.\mathit{elig}.\mathit{capableof} \subseteq m?.\mathit{capableof}) \wedge \\
\qquad (ta?.\mathit{elig}.\mathit{attributes} \subseteq m?.\mathit{attributes})) \\
\mathit{taskanns}' = \mathit{taskanns} \cup \{ta?\} \\
\mathit{bids}' = \mathit{bids} \cup \{(ta?, \{\})\} \\ \hline
\end{array}$$

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.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{MakeBid}} \\ \hline
\Delta \mathit{ContractNet} \\
con? : \mathit{CNode} \\
bid? : \mathit{Bid} \\
ta? : \mathit{TaskAnn} \\ \hline
bid?.\mathit{cnode} = con? \wedge con? \in \mathit{nodes} \\
ta? \in \mathit{taskanns} \wedge con? \in ta?.\mathit{recs} \\
ta?.\mathit{elig}.\mathit{capableof} \subseteq bid?.\mathit{elig}.\mathit{capableof} \\
ta?.\mathit{elig}.\mathit{attributes} \subseteq bid?.\mathit{elig}.\mathit{attributes} \\
\mathsf{interested}\ (con?, (ta?.\mathit{task})) \\
con? \neq ta?.\mathit{sender} \\
\mathit{taskanns}' = \mathit{taskanns} \\
\mathit{bids}' = \mathit{bids} \oplus \{(ta?, \mathit{bids}\ ta? \cup \{bid?\})\} \\ \hline
\end{array}$$

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 (*rating*) 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. The auxiliary function, *makeC* forms an contract from its constituent parts and the auxiliary function *addT* takes an agent and a task and instantiates a new agent which has the additional task in its set of goals.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{MakeAward}} \\ \hline
\Delta \mathit{ContractNet} \\
m? : \mathit{CAgent} \\
ta? : \mathit{TaskAnn} \\
b? : \mathit{Bid} \\
\mathit{rating} : \mathit{TaskAnn} \rightarrow \mathit{Bid} \rightarrow \mathbb{N} \\
\mathit{newcon} : \mathit{CAgent} \\ \hline
m? = ta?.\mathit{sender} \wedge b? \in \mathit{bids}\ ta? \\
\forall\, b : \mathit{Bid} \mid b \in \mathit{bids}\ ta? \bullet \\
\qquad \mathit{rating}\ ta?\ b? \geq \mathit{rating}\ ta?\ b \\
\mathit{taskanns}' = \mathit{taskanns} \setminus \{ta?\} \\
\mathit{bids}' = \mathit{bids} \setminus \{(ta?, \mathit{bids}\ ta?)\} \\
\mathit{newcon} = \mathit{addT}\ b?.\mathit{cnode}\ ta?.\mathit{task} \\
\mathit{contracts}' = \mathit{contracts} \cup \\
\qquad \{\mathit{makeC}\ ta?.\mathit{task}\ m?\ \mathit{newcon}\} \\
\mathit{conagents}' = \mathit{conagents} \setminus \{b?.\mathit{cnode}\} \cup \\
\qquad\qquad \{\mathit{newcon}\} \\ \hline
\end{array}$$

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 *conagents*, but if the node previously had only one goal then it will be removed from *conagents* since it is no longer an agent. The auxiliary function, *remT*, reverts an agent to the node it was before adopting the goal of the contract.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{TerminateContract}} \\
\Delta \mathit{AllContracts} \\
m?, con? : \mathit{CAgent};\ t? : \mathit{Task} \\
\hline
\mathit{contracts}' = \mathit{contracts} \setminus \{\mathit{makeC}\ t?\ m?\ con?\} \\
\mathit{remT}\ con?\ t? \in \mathit{CAgent} \Rightarrow \\
\qquad \mathit{conagents}' = \mathit{conagents} \setminus \{con?\} \\
\qquad\qquad\qquad\qquad \cup \{\mathit{remT}\ con?\ t?\} \\
\mathit{remT}\ con?\ t? \notin \mathit{CAgent} \Rightarrow \\
\qquad \mathit{conagents}' = \mathit{conagents} \setminus \{con?\} \\
\hline
\end{array}$$

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.

6. Discussion

The field of agent-oriented systems is growing dramatically in many directions. Coupled with its relative youth, however, this has given rise to the concentration of research in distinct niches so that there are very different approaches to essentially similar problem areas with, in some cases, little or no interrelation. Such a fragmentation leads to a lack of consensus regarding such fundamental notions as agents and autonomous agents as discussed above, but also impedes progress towards integrated approaches to agent theory and agent construction. As the field matures, the broader acceptance of agent-oriented systems will become increasingly tied to the availability and accessibility of well-founded techniques and methodologies for system development.

A major criticism of much formal or theoretical work is that while it is important and contributes to a solid underlying foundation for practical systems, no direction is provided as to how it may be used in the development of these systems. Recently, however, some efforts have been made to provide a greater harmony between these two camps, and to integrate the complementary aspects. Wooldridge and Jennings have developed a model of cooperative problem solving (CPS) [13] which attempts to capture relevant properties of CPS in a mathematical framework while serving as a top-level specification of a CPS system. Rao has attempted to unite theory and practice in two ways. First, he provided an abstract agent architecture that serves as an idealization of an implemented system and as a means for investigating theoretical properties [11]. A second effort developed an alternative formalization by starting with an implemented system and then formalizing the semantics in an agent language which can be viewed as an abstraction of the implemented system, and which allows agent programs to be written and interpreted [10]. Goodwin has also attempted to bridge the gap by providing a formal description in Z of agents, tasks and environments, and then defining agent properties in these terms [5].

More recent work in our research program [9] has aimed to provide an environment which allows the development and investigation of a variety of agent systems, within the confines of the framework. In particular, the framework specifies certain constraints on the design of agents and describes inheritance of properties between different classes of entity, from object to agent and from agent to autonomous agent. The system is implemented using object-oriented methods in C++, based on the formal framework outlined earlier. It both relies upon the structure of the framework, and reflects it, so that they are very strongly related. The formal definitions of agents and autonomous agents rely on inheriting the properties of lower-level components. In Z, this is achieved through schema inclusion and is easily modelled in C++ by deriving one class from another. Thus, just as the agents are defined in terms of objects, and autonomous agents in terms of agents in the framework, the implemented agent classes are derived from object classes, and autonomous agent classes are derived from agent classes. At each point in the design and implementation, the process of refinement to code forces the clarification of assumptions in the design of agents.

This is not just an elegant means of relating agent architecture and design. It provides increasingly more sophisticated building blocks with which to construct more sophisticated agents incrementally in a rigorous and structured fashion. One question that arises from such a transition between theory and practice is to what extent this can be used as a basis for providing a methodology of agent-based systems. Kinny [6] argues that “a clear conceptual framework that enables the complexity of the system to be managed by decomposition and abstraction,” is vital in such a methodology. This is our starting point, and indeed our formal framework plays exactly this role, using the standard properties of the Z specification language to satisfy these requirements. In particular, we can construct a model of the computational system by refining the abstract definitions provided in the framework to include the relevant system constraints.

The first task is to identify each of the distinct entities in the application domain through an analysis of their functionality in terms of behaviour. That is to say that the result of this first step is an enumeration of all entities together with their purpose. Each of these entities can then be considered in terms of control, both with regard to themselves, and of others. This involves the examination of the dependencies that exist between entities, which rely on others to determine current behaviour and which are independent of others. At this point we should be able to classify each entity as an object, agent or autonomous agent, and then to use the analysis of functionality to design the necessary behaviours and methods for their control (essentially, the action-selection functions) for each. Finally, the hierarchical relationships between entities must be considered in more detail so that any structural similarities which can be exploited are revealed.

Object oriented approaches provide an ideal paradigm for the implementation of the agents designed as a result of such a process. The structural relationships inherent in the multi-agent system can be readily captured by the abstraction provided by object classes, and the inheritance that is available within class hierarchies. Perhaps more importantly, object-oriented methods provide a means by which the model given in a formal specification can be easily transformed into an executable program with minimal effort, and making use of existing object or agent class libraries.

Thus we move from principled but abstract theoretical framework through a more detailed, yet still formal, model of the system, down to an object-oriented implementation, preserving the hierarchical structure at each stage.

7. Conclusions

As the fields of intelligent agents and multi-agent systems move relentlessly forwards, it is becoming increasingly more important to maintain a coherent world view that both structures existing work and provides a base on which to keep pace with the latest advances. Our framework has allowed us to do just that. By elaborating the agent hierarchy in different ways, we have been able to detail both individual agent functionality and develop models of evolving social relationships between agents with, for example, our analyses of goal generation and adoption, and our treatment of engagement and cooperation. Not only does this provide a clear conceptual foundation, it also allows us to refine our level of description to particular systems and theories. For example, we have shown how the Contract Net Protocol can be specified within the framework so that it reflects the more general structure of inter-agent relationships.

Moreover, the move to use the framework as a base for development, though still early, indicates much promise. Indeed, one of the key challenges facing agent systems is to construct development methodologies relevant to the specific needs of the field. As a result of our previous work, there has been a great deal of interest in the use of Z in the specification and development of agent-based systems, both in academic and industrial circles, and our future work must seek both to cement this interest and to capitalise on it. Further, we would aim to formalise any proposed methodologies within the framework using Z such as those developed for the development of high-performance systems [1].

Certainly, much remains to be done, but serious and extensive application of agent technology will only progress with such well-founded and coherent approaches which provide accessible frameworks with which system analysis and development can take place.

References

[1] M. d'Inverno, G. R. Justo, and P. Howells. A formal framework for specifying design methodologies. *Software Process: Improvement and Practice*, 2(3):181–195, September, 1996.

[2] M. d'Inverno and M. Luck. A formal view of social dependence networks. In *Proceedings of the First Australian DAI Workshop - Proceedings of the First Australian Workshop on Distributed Artificial Intelligence*, pages 115–129. Springer Verlag, 1996.

[3] M. d'Inverno and M. Luck. Formalising the contract net as a goal directed system. In W. Van de Velde and J. Perram, editors, *Agents Breaking Away - Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi Agent World*, pages 72–85. Springer Verlag, 1996.

[4] M. d'Inverno and M. Priestley. Structuring a Z specification to provide a unifying framework for hypertext systems. In J. P. Bowen and M. G. Hinchey, editors, *ZUM'95: 9th International Conference of Z Users, Lecture Notes in Computer Science*, pages 81–102, Heidelberg, 1995. Springer-Verlag.

[5] R. Goodwin. A formal specification of agent properties. *Journal of Logic and Computation*, 5(6), 1995.

[6] D. Kinny, M. Georgeff, and A. Rao. A methodology and modelling technique for systems of BDI agents. In W. Van de Velde and J. W. Perram, editors, *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World, LNAI 1038*, pages 56–71. Springer-Verlag: Heidelberg, Germany, 1996.

[7] 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.

[8] M. Luck and M. d'Inverno. Structuring a Z specification to provide a formal framework for autonomous agent systems. In J. P. Bowen and M. G. Hinchey, editors, *ZUM'95: 9th International Conference of Z Users, Lecture Notes in Computer Science*, pages 47–62, Heidelberg, 1995. Springer-Verlag.

[9] M. Luck and M. d'Inverno. From agent theory to agent construction: A case study. In *Intelligent Agents III: ATAL'96*, pages 215–230. Springer Verlag, 1997.

[10] A. S. Rao. Agentspeak(l): BDI agents speak out in a logical computable language. In W. Van de Velde and J. W. Perram, editors, *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents*

in a Multi-Agent World, LNAI 1038, pages 42–55. Springer-Verlag: Heidelberg, Germany, 1996.

[11] A. S. Rao and M. P. Georgeff. An abstract architecture for rational agents. In C. Rich, W. Swartout, and B. Nebel, editors, *Proceedings of Knowledge Representation and Reasoning*, pages 439–449, 1992.

[12] 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.

[13] 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.