10-1993

Declarative Representations of Multiagent Systems

Munindar P. Singh

Michael N. Huhns
University of South Carolina - Columbia, huhns@sc.edu

Larry M. Stevens
University of South Carolina - Columbia, stephens@cec.sc.edu

Publication Info

Published in *IEEE Transactions on Knowledge and Data Engineering*, Volume 5, Issue 5, 1993, pages 721-739.
http://ieeexplore.ieee.org/servlet/opac?punumber=69

Declarative Representations of Multiagent Systems

Munindar P. Singh, *Student Member, IEEE*, Michael N. Huhns, *Member, IEEE*, and Larry M. Stephens, *Senior Member, IEEE*

***Abstract*—This paper explores the specification and semantics of multiagent problem-solving systems, focusing on the representations that agents have of each other. It provides a declarative representation for such systems. Several procedural solutions to a well-known test-bed problem are considered, and the requirements they impose on different agents are identified. A study of these requirements yields a representational scheme based on temporal logic for specifying the acting, perceiving, communicating, and reasoning abilities of computational agents. A formal semantics is provided for this scheme. The resulting representation is highly declarative, and useful for describing systems of agents solving problems reactively.**

***Index Terms*—Declarative Representations, Distributed Artificial Intelligence, Formal Specifications, Knowledge Representation, Multiagent Systems, Problem-Solving Systems**

I. Introduction

DISTRIBUTED artificial intelligence (DAI) is concerned with how a group of intelligent computational agents should coordinate their activities to achieve their goals. When pursuing common or overlapping goals, they should act cooperatively so as to accomplish more as a group than individually; when pursuing conflicting goals, they should compete intelligently. Interconnecting computational agents and expert systems enables them to cooperate in solving problems, to share expertise, to work in parallel on common problems, to be developed and implemented modularly, to be fault-tolerant through redundancy, to represent multiple viewpoints and the knowledge of multiple human experts, and to be reusable. DAI is the appropriate technology for applications where

1) expertise is distributed, as in design;
2) information is distributed, as in office automation;
3) data are distributed, as in distributed sensing;
4) decisions are distributed, as in manufacturing control; and
5) knowledge bases are developed independently but must be interconnected or reused, as in next-generation knowledge engineering.

DAI has progressed much in recent years and has been garnering an increasing amount of attention lately. There have been several successful implementations of DAI systems, notably the distributed vehicle monitoring testbed (DVMT) for distributed sensing [8], the Pilot's Associate for control of jet fighters [24], the MINDS system for information retrieval [17], and the RAD platform for multiagent system development [3]. Most of these implementations were designed to solve particular domain problems or to demonstrate the feasibility of some DAI features and architectures.

However, principles for the systematic design of DAI systems are still hard to find. For the most part, the capabilities and features of the above DAI systems were represented procedurally. Although useful, these procedural representations are difficult to extend to novel domains or to characterize formally. They also make it difficult to compare different kinds of agents. By contrast, the declarative representation espoused herein makes explicit the knowledge and other capabilities that agents must possess in order to interact successfully. It also permits a formal model of the agents to be developed. A formal model is useful because it is possible to prove its properties and specify its predictions. It yields representations that are concise, yet clear and uniform across several domains. The development of a formal model is the first step in the development of design rules and then of tools for the design and validation of DAI systems—a formal model provides the basis for verifying that a given system meets its specification and that only the desired properties hold of it.

Declarative representations can be difficult to develop, however. Our methodology is to develop a declarative model for a simple problem, and then to extend the model to cover increasingly more general and more interesting versions of that problem. The model specifies what the agents know and what capabilities they have. The problem we consider is the pursuit problem of Benda, *et al.* [6], which is a well-known test problem for distributed systems [9], [12], [13], [25]. This problem is simple to describe and understand, but allows a large number of interesting variations. Our version is taken from [25].

A finite square grid of locations is given (see Fig. 1), each of which may be occupied by either an entity called "Red" (the "enemy") *or* any of a given number of "Blue agents" (who try to capture Red). At each step of the game, each entity can stay in its location or move one square up, down, left, or right. The pursuit starts in some arbitrary configuration and ends in either a win or a loss. The Blue agents win when they occupy the four locations surrounding Red; they lose if Red gets to the edge of the grid.

A problem such as this is solved by a *problem-solving system*, which consists of a problem, some agents, and an agent organization. These are specified externally. However, each agent has an internal representation for the problem

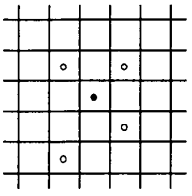
Manuscript received July 1, 1990; revised May 22, 1991.

M. P. Singh is with Microelectronics and Computer Technology, Austin, TX 78759, and the Department of Computer Sciences, University of Texas, Austin TX 78712.

M. N. Huhns is with the Object-Oriented Database Laboratory, Microelectronics and Computer Technology Corporation, Austin, TX 78759.

L. M. Stephens is with the Department of Electrical and Computer Engineering, University of South Carolina, Columbia, SC 29208.

IEEE Log Number 9211315.



(• = Red, ○ = Blue)

Fig. 1. An example configuration of the pursuit problem.

he is solving and the agents he interacts with. An agent's representations may differ from those of other agents, and those given in the system. The system describes the problem, the agents, and their organization as they *really* are; an agent's representations, however, may be partial or incorrect. The global problem may be solved successfully even if agents represent only parts of the original problem. For example, if different Blue agents try only to occupy different locations surrounding Red (e.g., one on the left, one on the right, and so on) they do *not* represent the original pursuit problem; however, when their individual problems are solved, the given problem is solved as well. And if one agent is the controller and the other Blue agents his slaves, only the controller needs to represent the given problem; the other agents just follow orders and need not represent anything at all.

We employed the following methodology to develop declarative representations of distributed agents.

- Start with a procedural description of a solution for the simple case of a centrally controlled system in which the central controller is omniscient (but not able to predict the moves of the Red agent).
- Abstract out the parts of the problem and problem-solving method that depend only on the peculiarities of the test-bed problem.
- Identify the assumptions made tacitly in the problem-solving system, especially about the agents' knowledge and capabilities.
- Remove the assumptions one at a time to obtain increasingly more realistic versions of the distributed system and to identify the impact of relaxing each assumption.
- Represent the uncovered parameters of the method explicitly.
- Define a formal language with a formal semantics for writing the declarations, so that the representations are genuinely explicit representations of the problem, the agents' abilities, and the state of the solution.
- Using this notation, write declarative representations of the pursuit problem and the various possible systems that may be used to solve it.
- Identify the connections among different parts of the problem and the nature of the agents solving it. Express these connections in a frame system so that specifications of new problems and problem-solving systems may be made more compact.

We develop the following in this paper:

- a scheme for declaratively specifying the acting, perceiving, communicating, and reasoning abilities of agents in a distributed system;
- a scheme for specifying the different kinds of protocols (namely, information, command, request, permission, prohibition, and explanation protocols) used for communicating at the problem-solving level;
- a formal semantics for the above schemes in terms of a simple language based on *temporal logic*. A novel feature is the specification of protocols in terms of constraints among the agents.

II. Systems for Solving the DAI Pursuit Problem

A variety of problem-solving systems can be designed for the pursuit problem: each could incorporate different assumptions about the agents' abilities and organizations. We start with the simplest case in which one agent controls the others, and proceed to an organization in which the agents are autonomous. Often, we refer to the Red agent as being the environment. At no point do we assume that the environment can be perfectly modeled by the Blue agents, either singly or jointly.

2.1. Omniscient Central Controller

In this variation, one of the four Blue agents, B^1, is made the controller. He is omniscient about the state of the entire system and issues commands to the other three Blue agents (who must accept all his orders).

2.1.1. The Algorithm: The basic solution scheme follows.

1) B^1 perceives the location of Red and of the B^k's, $1 \leq k \leq 4$.
2) B^1 computes "quadrants" using Red's current location as the origin. Quadrants [13] are defined as the partitions of the grid induced by the two diagonal lines passing through Red's location. Using the locations of the B^k's, B^1 assigns different quadrants to different B^k's.
3) B^1 decides the moves the agents should make in order to enter the quadrants that were assigned to them and commands them accordingly.
4) The slave Blue agents move as commanded, thus changing their locations on the grid. Red may also move at the same time.
5) Red's movement changes the exact description of each quadrant, possibly leading to a reassignment of quadrants to the agents. This process continues until either Red escapes, is captured, or each quadrant determined by Red's location is occupied by some Blue agent.
6) If each quadrant is occupied by Blue agents, Red cannot escape unless the Blue agents fail, or make a mistake. Now the moves of the Blue agents are determined in the following manner:
   a. If Red does not move, they should all draw closer to Red.
   b. If Red moves in some direction (i.e., into some quadrant) then:
      - If Red moves next to B^k, B^k should not move at all.

- The Blue agent in the quadrant into which Red moves should move so as to stay within the quadrant assigned to it.
- The agent in the opposite quadrant should simply follow Red.
- If the other agents are still within their assigned quadrants, they should move perpendicular to Red (toward him); otherwise, they should move to stay in their quadrants.

The procedure terminates either when Red escapes—i.e., gets to the edge of the grid—or when Red is captured—i.e., is surrounded by the Blue agents.

2.1.2. Knowledge and Capability Requirements: For the Blue agents to execute this method, they must possess certain abilities and knowledge.

1) B^1 must know where Red is at all times.
2) B^1 must know where each Blue agent is at all times.
3) B^1 must be able to compute appropriate quadrant assignments.
4) B^1 must know what commands each Blue agent can execute.
5) B^1 must be able to compute appropriate moves for the Blue agents.
6) B^1 must be able to communicate commands to its slaves.
7) The slave Blue agents must be able to receive commands from B^1. They do not need to send any acknowledgment, since B^1 can perceive their location at all times.
8) The Blue agents must be able to move up, down, left, and right, or simply maintain their position.

Slave agents need not know their own location, or that of any other agent. They need not be able to transmit anything to anyone, or to reason.

2.2. Central Controller, Agents Perceive Own Location

In the previous case, the central controller was responsible not only for making optimal decisions, but also for gathering all factual knowledge about the locations of the agents and the state of the environment. Now, the agents perceive their own location. The rest of the algorithm proceeds as before.

2.2.1. Knowledge and Capability Requirements: The knowledge requirements are unchanged because the central controller still makes all the decisions. The capability requirements change since

1) The B^k's should be able to perceive their own location.
2) The slave B^k's should be able to communicate their location to B^1 from anywhere on the grid.

This method requires that the controller "combine" location information from the other agents. Therefore, B^1 should be able to express the locations of the different agents and to relate the coordinate systems used by each. Later we suggest that the appropriate translations be made by the protocols that exist between B^1 and the other agents.

2.3. Central Controller, Agents Search for Red

We now relax the assumption that the controller is always able to detect Red's location by making all the Blue agents capable of perceiving Red within a limited range. The Blue agents patrol the grid until one of them detects Red. Then the pursuit begins. Since we have assumed that the Blue agents are all as fast as Red, the one who detects Red need never lose him. Since we are not interested in the pursuit problem for its own sake, we simply assume that the viewing ranges of the Blue agents are defined appropriately.

2.3.1. The Algorithm: The revised algorithm is as follows.

1) Each B^k informs B^1 of his location.
2) B^1 partitions the grid into regions and assigns each to a Blue agent.
3) B^1 commands the slaves to make them enter their assigned regions.
4) B^1 repeatedly commands the slaves to "patrol" those regions.
5) Each B^k looks out for Red and on detecting him informs B^1.

The rest of the algorithm is as before, with the restriction that the agent who detects Red be assigned his current quadrant (so that, once detected, Red will not be lost).

2.3.2. Knowledge and Capability Requirements:

1) B^1 knows the viewing range for each B^k and can compute an appropriate patrolling region for him.
2) The Blue agents are equally capable of perceiving Red.
3) The slave agents can transmit Red's location to the controller.

2.4. Central Control by Abstract Commands

The agents can already perceive their own location as well as that of Red. Now we make them smart enough to make some decisions by themselves. The simplest way of incorporating local decision making, while allowing a single global goal, is to have the agents execute commands more complex than the directly physical ones like *move-left*, *move-right*, and so on.

2.4.1. The Algorithm:

1) Each B^k reports his location to B^1.
2) B^1 assigns a region to each.
3) B^1 commands the agents to *go* to the assigned regions.
4) As each agent enters his assigned region, B^1 commands him to *patrol* that region. When some B^k detects Red, he informs B^1, as before.
5) Now B^1 makes quadrant assignments, as before, and commands each agent to *go* to his assigned quadrant.
6) As each B^k enters the quadrant, he is commanded to *approach* Red; i.e., to get closer to Red if possible and to maintain distance otherwise.

2.4.2. Knowledge and Capability Requirements:

1) The B^k's must be capable of executing the abstract commands *go*, *patrol*, and *approach* as used above, by computing the optimal physical move; i.e., they must have a limited ability to plan.
2) B^1 need no longer know which physical actions the slaves can perform.
3) B^1 has to compute the appropriate high level commands for the slaves, and the physical commands, just for himself.

4) B^1 must be able to communicate those high level commands.

The basic structure of the algorithm stays the same. A modification is that a slave need transmit his location to the controller only when commanded to do so or on moving to a new region (so that the controller always knows where he is).

2.5. Control Distributed among Altruistic Peers

Already the slaves can perceive their environment, communicate, and choose their actions for abstract commands. Now we make the Blue agents peers of each other, so they all participate in making global decisions (e.g., deciding which agent should be assigned what task). Such global decision making can be difficult in the general case, so we let each agent broadcast all his information to other agents, and require that they all use the same globally optimal method in deciding which tasks each should do.

2.5.1. The Algorithm:

1) The Blue agents broadcast their locations to each other.
2) Each agent computes the globally optimal assignment of agents to the regions to be patrolled and takes on the right task for himself.
3) The agent who detects Red broadcasts his location to the other agents.
4) The agents broadcast their own locations to each other.
5) Each agent computes the globally optimal assignment of agents to quadrants and takes on the right assignment for himself.

2.5.2. Knowledge and Capability Requirements:

1) Each agent must now be able to broadcast to other agents.
2) Each agent must compute the globally optimal assignment whenever it is needed.

This algorithm relies on the agents being "altruistic" in the sense that each of them takes on an assignment depending on global optimality (i.e., for the good of all), rather than on local optimality (i.e., for the good of just himself). Each agent decides locally, but always comes to a conclusion that coheres with the conclusions of the others.

2.6. Control Distributed among Self-Interested Agents

While we can often design our systems so that the agents in them are aware of the global goal and are altruistic, the requirement of making the same information available to each agent can prove troublesome. Now agents estimate their costs locally, but optimize over these estimates globally. An agent may use local information (e.g., his physical condition and his other pending tasks) for estimating the costs of achieving different goals. We need to assume that the agents are "honest" to the system; i.e., they do not give improper estimates just to avoid work!

2.6.1. The Algorithm:

1) Each agent estimates his costs of occupying different regions of the grid.
2) Each agent broadcasts his estimates to other agents.
3) Each agent makes the globally optimal assignment and takes on the task of getting to the appropriate region.
4) As before, the agents search for Red. The agent who detects Red broadcasts Red's location to the other agents.
5) Each agent estimates the cost of his occupying each quadrant and broadcasts his table of estimated costs to the other agents.
6) Each agent then chooses his own task assignment on the basis of global optimization, using the local information supplied by the other agents.

2.6.2. Knowledge and Capability Requirements:

1) Each agent is able to estimate his cost for achieving each goal.
2) Each agent is able to determine the globally optimal assignment given some cost estimates.

III. Formal Specifications

Our specification language must be usable for a variety of domains. Therefore, it must be able to express at least the following about any domain:

- The state of the environment and the important parts of the states of the agents in the system.
- The legal state transitions of the environment.
- Constraints on the state of the environment that must be respected.
- The legal moves for the agents, i.e., the actions that the agents can perform, or be requested, permitted, or commanded to perform.
- The important parts of the reasoning abilities of the agents, i.e., how their beliefs at one time lead to their beliefs at a later time.
- The perceiving abilities of the agents, i.e., how the environment and some parts of their internal state are related.
- The communicating abilities of the agents.

One way of specifying problems and actions declaratively is to use the situation calculus [19]. However, the situation calculus, while quite simple, is not very usable in practice. It requires that one specify for each possible action what its preconditions and effects are, and how it affects different parts of the world state. These problems, respectively known as the qualification, ramification, and frame problems [16], [23], are themselves the objects of considerable research; no good solution that works for all of them is available at present. The situation calculus also assumes that there is only one agent acting in the world, and the world changes only because of his actions—this is rather restrictive in DAI.

The option that seems best is to define a formal representational scheme based on a simple version of Propositional Linear Temporal Logic (PLTL) [10]. PLTL provides an abstract language for characterizing time and events. Procedural knowledge can be characterized in PLTL, yielding formal specifications that can then be used for explicit reasoning. PLTL provides a simple mechanism for abstractly specifying the actions performed and the choices taken by different agents. It is abstract in that the actions taken do not need to be mentioned at any stage. By contrast, approaches based on

the situation calculus require that all knowledge be expressed declaratively, and that all reasoning be theorem proving using low-level declarative knowledge. This turns out to be prohibitively inefficient in practice [2], [22].

3.1. The Formal Language

A PLTL formula may be defined by the following grammar:

1) ⟨cond⟩ ::= ⟨atomic-cond⟩
2) ⟨cond⟩ ::= ¬ ⟨cond⟩
3) ⟨cond⟩ ::= ⟨cond⟩ ∨ ⟨cond⟩
4) ⟨cond⟩ ::= ⟨cond⟩ ∧ ⟨cond⟩
5) ⟨cond⟩ ::= ⟨cond⟩ → ⟨cond⟩
6) ⟨cond⟩ ::= X ⟨cond⟩
7) ⟨cond⟩ ::= F ⟨cond⟩
8) ⟨cond⟩ ::= G ⟨cond⟩
9) ⟨cond⟩ ::= ⟨cond⟩ U ⟨cond⟩

The temporal operators (X, F, G, U) are described in Section 3.2.

An action description describes an action that may be done by an agent, i.e., action descriptions are *types* of actions. Some actions have names (e.g., *move-left*, *move-right*) of their own and can be described directly. Other actions are described in terms of the conditions with which they are related. Using PLTL as inspiration, actions may be written as

1) ⟨action-desc⟩ ::= ⟨action-name⟩
2) ⟨action-desc⟩ ::= *Achieve*(⟨cond⟩)
3) ⟨action-desc⟩ ::= *Maintain*(⟨cond⟩).

3.2. Intuitions about the Semantics

The above language includes just the operators required for reasoning about all multiagent systems; for each domain, an extension of this language would be required that included the operators required to represent the specific aspects of that domain; e.g., we assume that the language is augmented to allow a certain amount of arithmetic to compute distances over the grid.

The semantics for the above language is given relative to a formal model composed of *temporal structures*. A linear temporal structure is a sequence of world states, each following the previous one in time. We use linear temporal structures, with each structure taken to be a "run" of the problem-solving system, i.e., a possible way in which the system may evolve over time. An atomic condition is true at some time in a structure iff it has been stipulated to be true there. Boolean combinations of conditions are determined in the obvious manner. The temporal condition "X cond" is true at some time in a structure iff the condition "cond" is true at the next time in that structure. That is, X stands for "next time" and makes sense only in discrete structures. "F cond" is true at some time in a structure iff "cond" is true at some later time in that structure. That is, F is to be read as "sometimes in the future." Similarly, G stands for "always in the future" and U stands for "until." $c_1 U c_2$ is true at a time t_1, iff c_2 holds at some time t_2 in the future of t_1, and c_1 holds continuously from t_1 to t_2.

The semantics for conditions are quite standard, as are the applications of PLTL to the specifications of procedures. The novel part of the semantics concerns the different kinds of protocols that we have introduced. Roughly, these correspond to one agent making utterances to another agent: the different protocol types discussed in Section VI correspond to different "illocutionary forces," as described by J. L. Austin [4]. Philosophers, such as Hamblin [15], recognize at least two different levels of semantics: "extensional" satisfaction, and "wholehearted" satisfaction. Extensional satisfaction is more basic and, in giving a semantics based on PLTL, we analyze protocols at just that level. A theory of wholehearted satisfaction is the ultimate goal, but extensional satisfaction is a reasonable approximation for our DAI needs.

Action descriptions are either action names or constructed from conditions by *Achieve* and *Maintain*. The semantics of an action name is the set of structures and times where its defining condition is satisfied. The semantics of "*Achieve*(cond)" is the set of structures and times where "F cond" holds; i.e., from where "cond" can be achieved in the future. Similarly the semantics for "*Maintain*(cond)" is the set of structures and times where "G cond" holds; i.e., where "cond" always holds in the future. This is what is meant by "extensional satisfaction": we do not care whether the condition "cond" is achieved intentionally by the agent, or due to the actions of other agents, or through events in the environment, or even because of mistakes on the part of the agent. (By contrast, wholehearted satisfaction involves the agent's doing the action by means of other intentional actions, in a manner he intends—this is a far more complex notion and not easily usable in simple kinds of reactive systems.) We emphasize, however, that *Achieve* and *Maintain* are not merely nicer syntax for F and G, respectively. While F and G yield conditions, *Achieve* and *Maintain* yield action descriptions. These descriptions may be used in the abstract plans of agents, and issued as commands or requests to other agents. Their final pragmatic impact depends on how they are finally interpreted, e.g., whether the agent begins to work for them immediately and whether he succeeds with them.

The semantic intuition formalized here is that several linear temporal structures are *a priori* possible, but only one of them will be actualized. A statement, which expresses a condition or describes an action, is evaluated relative to a structure and a time in that structure. A statement thus excludes the structures and times where it is not satisfied. For example, a constraint is a statement of the form "always p." When evaluated at the "start of time," it excludes all structures where p does not hold throughout.

A linear temporal structure may be discrete, dense, continuous, or even arbitrary [27]; only the discrete case is computationally tractable. We assume that time is discrete and point-based, but allow concurrent events. Only variables that correspond to parts of the environment are used in its specification. The agents' internal variables are considered only for the specification of their internal states and capabilities. The flow of time corresponds to the happening of an external or internal event, i.e., time flows as the environment changes, or as agents act, communicate, or reason. Thus the same "clock" applies to reasoning as does to acting. The granularity of the model corresponds to the shortest event (action, or step of

reasoning)—if none of our specifications uses the "next-time" operator, this is not required. Each action name has a defining temporal condition, called the *action-condition* of the action.

Using the semantics for action descriptions, we can obtain a semantics of the protocols. A transmitted action description must be in the language of the protocol. In a command protocol, a receiver must accept the action description (i.e., agree to perform it) when it is received, which depends on the time delay of the communication channels being used for the protocol. The structure and time must then be such that the action can be performed successfully. Receivers in request protocols can choose whether or not to perform a requested action, depending on their internal states when they receive the request. We assume that the request itself includes details of how much delay can be tolerated by the requesting agent, so that the receiver does not have to perform the requested action immediately and can drop it after it has lapsed.

The effect of a message, sent according to some protocol, depends on not only the static relationship between the agents involved, but also the state of the receiver. The effects of the messages may be predicted in cases where other constraints on the states of the agents are known; e.g., if in a system a request for an action always occurs after the receiver has agreed to be helpful and has the requisite know-how, it would certainly be carried out. It is known that there are no monotonic theories that may be used to predict the effects of simple actions, let alone those that (like message transmissions and receptions) involve more than one agent over a period of time. So instead of striving for such a theory, we propose that protocols, and the actions they result in, be represented in terms of constraints on structures. While, for simplicity, constraints are treated as simple conditionals, they could be refined to be defeasible conditionals, or statements of probability or certainty. This idea is elaborated later in this section. Treating constraints in this manner as an objective part of the world has proven quite effective in linguistics and philosophy [5], [26].

3.3. Formal Semantics

We define a model, M, as a set of structures and an assignment of atomic formulae to time indexes in those structures; i.e., $M = \langle \mathbf{S}, \mathbf{P} \rangle$. Each structure, $S \in \mathbf{S}$, is a linear sequence of time indexes, $\langle t_0, \ldots \rangle$. The function $\mathbf{P}$ assigns to each time index the set of atomic formulae that are true in it (the indexes in different structures are distinct). When the model is understood from the context, it is not mentioned explicitly. We thus write "the formula p is satisfied in structure S at index t" as $S, t \models p$. A formula, p, for which such a structure and index can be found is called *satisfiable*. A formula that is satisfied at all structures and time indexes is called *valid*. Given the motivation of the previous subsection, the following definitions result:

- $S, t \models p$, where p is an atomic formula, iff $p \in \mathbf{P}(t)$
- $S, t \models \neg p$ iff $S, t \not\models p$
- $S, t \models p \vee q$ iff $S, t \models p$ or $S, t \models q$
- $S, t \models p \wedge q$ iff $S, t \models p$ and $S, t \models q$
- $S, t \models p \rightarrow q$ iff $S, t \not\models p$ or $S, t \models q$
- $S, t \models Xp$ iff $S, t' \models p$, where t' is the successor of t in structure S
- $S, t \models Fp$ iff for some t': $S, t' \models p$, where $t' > t$
- $S, t \models Gp$ iff for all t': $S, t' \models p$, where $t' > t$
- $S, t \models pUq$ iff for some t'': $S, t'' \models q$, where $t' > t$, and for all $t' : t < t' < t''$: $S, t' \models p$.

Using the intuitions expressed earlier about what we mean by actions in the model, we can provide the following satisfaction conditions for actions:

- $S, t \models A$, where A is an action name, iff $S, t \models$ *action-condition*(A)
- $S, t \models$ *Achieve*(p) iff $S, t \models Fp$
- $S, t \models$ *Maintain*(p) iff $S, t \models Gp$.

A linear temporal structure satisfies the sending of a message on a protocol at a particular time iff that message is actually sent on that protocol then. Whether and when that message actually *arrives* depends on the conditions under which it was sent and the protocol used. The relevant conditions and properties of the protocol are captured by a constraint on the sending and potential reception of that message. The message would be received if this constraint is satisfied. The reception of a message, in turn, imposes another constraint: one which relates the reception of that message to the (eventual) invocation of some processing routine on it; e.g., it would relate the reception of a request to the receiving agent's adding it to his agenda. Another kind of constraint is needed to involve the capabilities of an agent. Thus we might have a constraint that if a certain action is on the agent's agenda, then it is eventually acted upon.

Constraints, applied in the manner described above, have several important consequences: 1) the capabilities of agents, the behaviors of protocols, the modes of reasoning and acting of agents, and properties of the environment can all be uniformly described; 2) it is possible to consider both the internal (representational) view, and the external (objective) view simultaneously—we can go from an agent's decision to send a request, to its actual transmission through the physical world, to its reception and eventual incorporation in the receiver's agenda, to action on it by the receiver (in the real world), and so on, with ease; 3) while our original aim was to take care of just extensional satisfaction, we actually do better, because constraints allow us to model behavior in classes of situations, rather than the individual situations that are realized in the real world. As we show below, many of the constraints that would be useful from a DAI point of view can be expressed simply as formulae in the logic as defined so far. Nothing more is required for them in the formalism. However, it is another problem to express these constraints—we expect that the condition-result form of the constraints will greatly simplify this task.

IV. The Problem-solving System

A problem-solving system consists of a problem, agents, and an agent organization. A simulation of such a system includes the environment of the agents as well (here it contains the process that decides Red's moves). The Blue agents have, in general, inaccurate *models* of the environment, based on

those parts of it that we explicitly declare they can perceive. However, the simulation maintains the global state of the problem, consisting of the real state of the environment and of each agent.

4.1. The Parameters of the Problem-Solving System

Several parameters can be used to distinguish among multiagent problem-solving systems. We consider *reactive* problem-solving systems only. Reactive problems, a generalization of the pursuit problem that we have been discussing, have the following features:

- The environment changes rapidly and unpredictably.
- The agents are limited reasoners.
- The agents are able to perceive only a small part of the environment.
- The agents may act concurrently with each other and with events in the environment.

A reactive problem-solving *system* can be specified by the following information.

- The initial state of the problem.
- The way in which the environment may change.
- The legal moves for the agents.
- The organization of the agents.
- The abilities of the agents (these determine, in part, the organizations they can participate in, and in what roles). Important abilities are the ones of *reasoning, perceiving, communicating,* and *acting.*
- The resources available to the agents, and how the agents try to optimize their usage of these resources.

The specifications of the above parameters can be simplified if we specify separately the properties of the protocols that the agents would use in their communications and the different kinds of agents they would have to deal with. That is, we need specifications for

- The protocols that are used by the agents in different organizations; e.g., in a master-slave organization, only command protocols are needed.
- The abilities and dispositions (e.g., helpfulness) of the different kinds of agents in an organization.

4.2. Description of the Agents' Internal States

Just as in a single-agent system, an agent in a multiagent system has to manage all his tasks using the resources available to him. The primary differences are 1) a multiagent system may achieve much more than any of its agents could individually, and 2) the resources of an agent in a multiagent system might themselves be "intelligent" and the tasks an agent has to do could be simplified by other agents. Agents who provide services to other agents (e.g., by accepting commands or requests, or by giving permissions) are resources of the other agents. Such resources can be assigned high level tasks and do not need to be actively controlled by the assigning agent. The resource interfaces capture the expertise, knowledge, and availability of the other agents, and do so uniformly for both intelligent agents and "dumb" resources. Agents with whom the agent has no useful interaction are best either ignored or treated as a part of the environment.

An important point is that, while the world is complex, agents are limited. As a result, they might act inappropriately: they act on the basis of their possibly inaccurate representations. We consider the agents' internal representations to be important, but make no claim about the form they must take. The accuracy and detail of these representations depends on the agents' capabilities. Potentially, we could have the following aspects (relative to a particular problem instance):

- The state of the environment according to the agent.
- The problem that the agent is solving.
- The abstract plan adopted for execution.
- An agenda of tasks to do.
- Current tasks; i.e., tasks selected from the agenda for working on *now,* or those derived directly from some unprecedented change in the environment. The latter option is important in reactive systems—events can occur that demand immediate action. Once an agent has the situation under control locally, he can always resume working on the long-term or global task. Such reactive behavior could be accounted for by the presence on the agenda, at all times, of a basic task like "stay alive," which could, e.g., yield a subtask "run" if a fire starts nearby. Of course, the agent may use an interrupt mechanism to invoke it.
- The resources available to the agent. For each resource, the agent needs to represent the following information:
  a. Static information, e.g., about the protocol for accessing it.
  b. A cost metric for the resource.
  c. A model for the usage of the resource, e.g., the other resources that have to be assigned to the resource in order to get a particular performance out of it. In the pursuit problem example, the controller can use a Blue agent to occupy a quadrant, but an agent can occupy only one quadrant at a time. Also, it costs fuel and time to get an agent to the quadrant assigned to him.
  d. The current state of the resource: 1) how much of it is available, and 2) what operations are needed to be able to use it, e.g., whether the agent is facing in the wrong direction, and whether the agent is in sensing or moving mode.
- An assignment of the resources to the current tasks. Factors such as cost metrics, which tacitly represent the agent's self interest, need to be considered in making these assignments.
- The actions to be done *now.* These depend on the resources assigned to different tasks and the protocols for using them.

4.3. The Simulation Process

The simulation proceeds as follows. At any given moment, the real world is in a certain state. The Blue agents are able to access parts of the global state depending on their perceptual

abilities, such as range of perception and accuracy. Depending on the state of the world as he perceives it, each Blue agent computes the optimal resource assignments for himself and decides what actions he must take. The resources available to an agent include the other agents whom he might command, or somehow get to act for himself. Agents may make predictions about the effects of their actions, but only the actions that they perform affect the state of the world. Any action may fail, or have unexpected consequences, or have consequences that are not perceptible until much later. However, the world "knows" its actual state immediately; the agents only know what they can perceive or infer. Thus the agents perceive, decide, and act, the world changes, and the simulation continues.

4.4. Specifications of the Pursuit Problem

This section contains an example specification for our main example, the pursuit problem. We define a notation in order to simplify the specification. The two coordinates of the problem grid are X and Y. and the variables i and j range over these coordinates, respectively. We refer to the Red agent as R and to the Blue agents as B^k's. B is used for any B^k and A is used for both R and the B^k's. B' denotes the agent to whom the specification applies. The coordinates of A are written as A_x and A_y.

1) The initial state of the problem, i.e., the environment and the agent locations: $R_x = 9, R_y = 9, B_x^1 = 1$, and so on.
2) Legal transitions for the environment: $[(R_x, R_y) = (i, j)] \rightarrow [X((R_x, R_y) = (i', j'))]$, where $(i', j') \in \{(i, j), (i+1, j), (i-1, j), (i, j+1), (i, j-1)\}$.
3) Constraints on the system: $(1 \leq A_x, A_y \leq 30) \wedge ((R_x, R_y) \neq (B_x, B_y))$.
4) Legal moves for the agents at the physical level: $[(B'_x, B'_y) = (i, j)] \rightarrow [X((B'_x, B'_y) = (i', j'))]$, where $(i', j') \in \{(i, j), (i+1, j), (i-1, j), (i, j+1), (i, j-1)\}$.
5) Win condition: $(\exists k : (B_x^k, B_y^k) = (R_x - 1, R_y)) \wedge (\exists k : (B_x^k, B_y^k) = (R_x + 1, R_y)) \wedge (\exists k : (B_x^k, B_y^k) = (R_x, R_y - 1)) \wedge (\exists k : (B_x^k, B_y^k) = (R_x, R_y + 1))$. This complex condition is referred to as "Win" in what follows.
6) Loss condition: $(R_x = 1) \vee (R_x = 30) \vee (R_y = 1) \vee (R_y = 30)$.

V. Agent Abilities

As stated in Section 4.1, agents have at least four classes of abilities: acting, perceiving, communicating, and reasoning. These abilities must be specified declaratively 1) to define a particular problem-solving system, and 2) to define an agent's representations of other agents. Each agent tries to enter into protocols with other agents and predicts and explains their actions, on the basis of the representations he has of them. These representations also determine whether he should aid or hinder others. A controlling agent must have the expertise to guide other agents. Thus he must represent the abilities, availability, and work-load of the agents he controls, as representations of his resources.

5.1. Acting Abilities

The acting abilities of an agent are the actions he can perform. These may be simple actions that can be done in a single step, or more complex ones that require that a condition be achieved or maintained. The latter kind of actions may call for an arbitrary number of steps; in particular, a maintenance action may never terminate. Clearly, the actions an agent can do depend on his perceiving and reasoning abilities and his physical attributes. As described below, actions may be specified both by name (e.g., *move-left*, *move-right*), and by applying the operators, *Achieve* and *Maintain*, to specifications of conditions (e.g., *Achieve*(*good-state*), *Achieve*(*good-state*$_1 \vee$ *good-state*$_2$), *Maintain*($\neg$*bad-state*)). All agents can do the following actions:

1) *move-left*: $[(B'_x, B'_y) = (i, j)] \rightarrow [X((B'_x, B'_y) = (i-1, j))]$
2) *move-right*: $[(B'_x, B'_y) = (i, j)] \rightarrow [X((B'_x, B'_y) = (i+1, j))]$
3) *move-down*: $[(B'_x, B'_y) = (i, j)] \rightarrow [X((B'_x, B'_y) = (i, j-1))]$
4) *move-up*: $[(B'_x, B'_y) = (i, j)] \rightarrow [X((B'_x, B'_y) = (i, j+1))]$
5) *no-op*: $[(B'_x, B'_y) = (i, j)] \rightarrow [X((B'_x, B'_y) = (i, j))]$.

For agents that execute abstract commands, we can specify the acting abilities as a set that includes not only the moves above but also the constructs *Achieve*(right-quadrant), *Maintain*(distance), and so on, where each of the conditions on which the constructors are applied might itself be further written as a complex temporal condition. Other actions, which may take more than one step, could have the intermediate steps undefined, as in the case of actions constructed using *Achieve* and *Maintain*, or partially defined, as in the following three possible specifications for *move-left-up*:

1) *move-left-up* by "first move left, then move up": $(B'_x, B'_y) = (i, j) \rightarrow (X((B'_x, B'_y) = (i-1, j)) \wedge XX((B'_x, B'_y) = (i-1, j+1)))$.
2) *move-left-up* by a partially determined path: $(B'_x, B'_y) = (i, j) \rightarrow (X(((B'_x, B'_y) = (i-1, j)) \vee ((B'_x, B'_y) = (i, j+1))) \wedge XX((B'_x, B'_y) = (i-1, j+1)))$. Now it is possible to move left and then up, or up and then left.
3) *move-left-up* by an undetermined path: If the intermediate states are not important, they need not be specified at all. Just to add further indeterminacy, we will make this specification allow four units of time: $(B'_x, B'_y) = (i, j) \rightarrow XXXX((B'_x, B'_y) = (i-1, j+1))$. Now B' may go from (i, j) to $(i-1, j+1)$ using any path that takes four steps.

A possible strategy for the Blue agents, since they must surround the Red agent in order to win, is for each Blue agent to occupy one of the four quadrants around Red. This strategy is called "Lieb" [13]. It is thus useful to define a quadrant condition, which is true when a Blue agent occupies the appropriate quadrant around Red:

1) left quadrant condition, $q_l : (B'_x < R_x) \wedge (|B'_y - R_y| < R_x - B'_x)$
2) right quadrant condition, $q_r : (B'_x > R_x) \wedge (|B'_y - R_y| < B'_x - R_x)$

3) down quadrant condition, $q_d : (B'_y < R_y) \wedge (|B'_x - R_x| < R_y - B'_y)$
4) up quadrant condition, $q_u : (B'_y > R_y) \wedge (|B'_x - R_x| < B'_y - R_y)$

There may be task descriptions corresponding to these quadrant descriptions, e.g., we may define a new task description, tq_l, corresponding to *Achieve*(q_l). When a task is assigned to an agent, the task specification must take into account the identity of the agent being assigned the task. For example, if B^3 is to be assigned the task of occupying the left quadrant, the condition in the command must involve B^3; we write this as $tq_l, B^3)$. In general, this allows tasks for which the assigned agent must find the appropriate physical actions.

5.2. Perceiving Abilities

The perceiving abilities of an agent depend on both his reasoning abilities and his perceptual hardware. Indeed, perception and reasoning may be distinguished only by looking at an agent's internal architecture. For example, whether an agent "sees" a corner, or just sees edges and "infers" a corner, depends on whether his perception module can, by itself, detect a corner. Perceiving abilities are specified formally as follows: an agent has a particular perceiving ability if, given a state of the world, he can come to a particular internal state. An example specification is "if Red is i steps away from a Blue agent and $i < 5$, then the Blue agent believes that Red is i steps from him." The *accuracy* of an agent's perceptions can also be captured; e.g., "if Red is more than j steps from a Blue agent and $j > 10$, then the Blue agent believes that Red is exactly 10 steps away." These specifications state that the given Blue agent's sensor for Red is accurate for a distance of up to 4 steps, and cannot distinguish among distances greater than 10 steps. Nothing is known about its accuracy for distances between 5 and 10 steps.

An agent might not perceive and represent all of the features of the problem-solving system as they are externally specified; on the other hand, an agent might represent some features that are not in the externally given specifications. We refer to different components of the problem-solving system, as perceived by an agent, by using the agent's name followed by a dot followed by the name of the relevant component.

We specify perceiving abilities by expressions of the form "$G($condition$_1 \rightarrow$ condition$_2)$," where the two conditions are formulas in the temporal specification language, with the following use and meaning intended. "Condition$_1$" expresses the conditions under which the given perceiving ability applies, e.g., whether the object being examined is well-lit, whether the agent is at a suitable location and facing in the right direction, and whether the agent is attentive. "Condition$_2$" expresses a restriction on the agent's internal representations, as a result of his perception using the given perceiving ability, e.g., the agent knows the actual location of Red, or believes it to be within a range of 5 of the true value of the X-coordinate. Thus "condition$_2$" must necessarily involve the agent's internal representations; even "condition$_1$" may involve these representations, e.g., to express that the agent is in a particular state of attentiveness. However, specifications of perceiving abilities must not involve the internal states of other agents. If there are any connections among the internal states of different agents, they must emerge in their interactions through the shared world or through explicit communication. Some interesting examples of perceiving abilities follow. We notate "agent B's representation of a'' as $B.a$.

- $G(B'.B'_x = B'_x)$: "agent B' always knows his x coordinate." Note that in this example, the condition corresponding to "condition$_1$" is identically true.
- $G((0 \leq (R_x - B'_x) \leq 5) \rightarrow (B'.R_x = R_x))$: "agent B' knows the x coordinate of Red, whenever the latter is within 5 steps to his right."
- $G(((R_x - B'_x) \geq 15) \rightarrow (B'.R_x = B'_x + 15))$: "agent B' knows the x coordinate of Red, whenever the latter is exactly 15 steps to his right, but cannot distinguish among locations of Red further to his right."

5.3. Reasoning Abilities

The reasoning abilities of an agent are the hardest to specify formally. We specify them as relationships between an agent's internal representations at one point in time and his representations at a later time. This time difference captures the speed of the agents' reasoning system, in terms of a real-time clock. The time referred to by the agents' representations can also vary: present for beliefs about the current state of the world, future for predictions, and past for explanations. The time delay due to reasoning can be ignored, if one assumes that the reasoning is especially simple relative to the agents' capabilities. For example, if an agent believes that a particular coded message was transmitted and is able to decipher the code, he might come to believe that a command, say *move-left*, was transmitted. Other aspects of reasoning abilities are more domain dependent and involve whether an agent can compute the optimal usage of his resources or not, predict another agent's actions or not, and so on.

We specify reasoning abilities by expressions of the form "$G($condition$_1 \rightarrow$ condition$_2)$." Both "condition$_1$" and "condition$_2$" must involve just the agent's internal representation, the first expressing his state before the corresponding reasoning ability is invoked, and the second expressing his state after the invocation. Some important examples of reasoning abilities that we use later in this paper follow.

1) Reasoning-ability-quadrant-assignment: the ability to compute optimal quadrant assignments. We do not discuss the predicate "optimal-1" and similar domain dependent functions in this paper. The algorithm discussed here assumes global optimality.

$$G(\text{optimal-1}((tq_l, B^{k_1}), (tq_r, B^{k_2}), (tq_d, B^{k_3}), (tq_u, B^{k_4})) \rightarrow B'.\text{resource-assignment} = (tq_l, B^{k_1}), (tq_r, B^{k_2}), (tq_d, B^{k_3}), (tq_u, B^{k_4})).$$

2) Reasoning-ability-optimal-resource-moves-for-all (see top of page): the ability to compute optimal moves, given a resource-assignment, and to decide to act according to those moves. The action *Send* is described in Section VI.

$$G(\text{optimal-2}(B'.\text{resource-assignment}, (\text{move}_1, B'), (\text{move}_2, B^{k_2}), (\text{move}_3, B^{k_3}), (\text{move}_4, B^{k_4}))$$

$$\rightarrow B'.\text{actions} = \text{move}_1, \left(\begin{array}{l} \textit{Send}(B', B^{k_2}, \text{access-protocol}, \text{move}_2), \\ \textit{Send}(B', B^{k_3}, \text{access-protocol}, \text{move}_3), \\ \textit{Send}(B', B^{k_4}, \text{access-protocol}, \text{move}_4))) \end{array}\right.$$

---

3) Reasoning-ability-send-own-location: the ability to send one's own location to the controller, B''.

$$G((B'_x, B'_y) = (i,j)) \rightarrow B'.\text{actions} = \textit{Send}(B', B'', \textit{blue-location}, (i,j))$$

4) Reasoning-ability-send-red-location: the ability to send Red's location to the controller, B''.

$$G((R_x, R_y) = (i,j)) \rightarrow B'.\text{actions} = \textit{Send}(B', B'', \textit{red-location}, (i,j))$$

5) Reasoning-ability-mount-search: the ability to compute optimal moves in order to have all agents search for Red, if his location is not known. This can be specified in greater detail following the specifications for Reasoning-ability-start-patrolling, Reasoning-ability-scan-outwards, and Reasoning-ability-scan-inwards.

$$G(\neg B'.\text{found} \wedge \text{optimal-3}\ ((\text{move}_1, B'), (\text{move}_2, B^{k_2}), (\text{move}_3, B^{k_3}), (\text{move}_4, B^{k_4}))$$

$$\rightarrow B'.\text{actions} = (\text{move}_1, \textit{Send}(B', B^{k_2}, \textit{basic-commands}, \text{move}_2), \textit{Send}(B', B^{k_3}, \textit{basic-commands}, \text{move}_3), \textit{Send}(B', B^{k_4}, \textit{basic-commands}, \text{move}_4)))$$

6) Reasoning-ability-close-in-on-red: the ability to compute optimal moves to have all agents close in on Red. This can be specified in greater detail using the specification for Reasoning-ability-approach-red as a guide.

$$G(\text{optimal-4}\ ((\text{move}_1, B'), (\text{move}_2, B^{k_2}), (\text{move}_3, B^{k_3}), (\text{move}_4, B^{k_4}))$$

$$\rightarrow B'.\text{actions} = (\text{move}_1, \textit{Send}(B', B^{k_2}, \textit{basic-commands}, \text{move}_2), \textit{Send}(B', B^{k_3}, \textit{basic-commands}, \text{move}_3), \textit{Send}(B', B^{k_4}, \textit{basic-commands}, \text{move}_4)))$$

7) Reasoning-ability-start-patrolling: the ability to patrol a quadrant about vertex (i,j). Here only the case of quadrant q_r is given. Note that here and in the sequel, current-tasks just denotes a set of tasks.

$G(B'.\text{current-tasks} = \{\textit{patrol-region}(i,j,q_r)\} \wedge \neg B'.\text{found} \rightarrow [[(B'_x, B'_y) = (i+1, j)] \rightarrow [[B'.\text{actions} = \textit{move-down}] \wedge [B'.\text{current-tasks} = \{\textit{patrol-region}(i,j,q_r), B'.\textit{out-scan}(i,j,q_r)\}]]] \wedge [[(B'_x, B'_y) \neq (i+1, j)] \rightarrow [[B'.\text{actions} = \textit{move-down}] \wedge [B'.\text{current-tasks} = \{\textit{patrol-region}(i,j,q_r), B'.\textit{in-scan}(i,j,q_r)\}]]])$

8) Reasoning-ability-scan-outwards: the ability to scan a quadrant outwards from vertex (i,j). Only the case of quadrant q_r is given here. Note that $B'.\text{found} \equiv (\exists i,j : (B'.R_x, B'.R_y) = (i,j))$, and is made true due to the agent's perceiving abilities. Fig. 2 shows the patrolling path taken by B' when using this reasoning ability.

$G(B'.\textit{out-scan}(i,j,q_r) \in B'.\text{current-tasks} \wedge \neg B'.\text{found} \wedge \text{odd}(B'_x - i) \rightarrow [[(j - B'_y) < (B'_x - i) \rightarrow B'.\text{actions} = \textit{move-down}] \wedge [(j - B'_y) = (B'_x - i) \wedge (B'_x < 30) \rightarrow B'.\text{actions} = \textit{move-right}] \wedge [(j - B'_y) = (B'_x - i) \wedge (B'_x = 30) \rightarrow B'.\text{current-tasks} = \{\textit{patrol-region}(i,j,q_r), B'.\textit{in-scan}(i,j,q_r)]\}]) \wedge G(B'.\textit{out-scan}(i,j,q_r) \in B'.\text{current-tasks} \wedge \neg B'.\text{found} \wedge \neg\text{odd}(B'_x - i) \rightarrow [[(B'_x - i) > (B'_y - j) \rightarrow B'.\text{actions} = \textit{move-up}] \wedge [(B'_y - j - 1) = (B'_x - i) \wedge (B'_x < 30) \rightarrow B'.\text{actions} = \textit{move-right}] \wedge [(B'_y - j - 1) = (B'_x - i) \wedge (B'_x = 30) \rightarrow B'.\text{current-tasks} = \{\textit{patrol-region}(i,j,q_r), B'.\textit{in-scan}(i,j,q_r)]\}])$

9) Reasoning-ability-scan-inwards: the ability to scan a quadrant inwards from vertex (i,j). Only the case of quadrant q_r is given here.

$G(B'.\textit{in-scan}(i,j,q_r) \in B'.\text{current-tasks} \wedge \neg B'.\text{found} \wedge \neg\text{odd}(B'_x - i) \rightarrow [[(j - B'_y) < (B'_x - i - 1) \rightarrow B'.\text{actions} = \textit{move-down}] \wedge [(j - B'_y) = (B'_x - i - 1) \rightarrow B'.\text{actions} = \textit{move-left}]]) \wedge G(B'.\textit{in-scan}(i,j,q_r) \in B'.\text{current-tasks} \wedge \neg B'.\text{found} \wedge \text{odd}(B'_x - i) \rightarrow [[(B'_x - i) > (B'_y - j - 2) \wedge ((B'_x, B'_y) \neq (i+1, j)) \rightarrow B'.\text{actions} = \textit{move-up}] \wedge [(B'_x - i) > (B'_y - j - 2) \wedge ((B'_x, B'_y) = (i+1, j)) \rightarrow B'.\text{actions} = \textit{move-down} \wedge B'.\text{current-tasks} = \{\textit{patrol-region}(i,j,q_r), B'.\textit{out-scan}(i,j,q_r)\}] \wedge [(B'_y - j) = (B'_x - i - 2) \rightarrow B'.\text{actions} = \textit{move-left}]])$

10) Reasoning-ability-approach-red: the ability to approach Red; i.e., to follow Red if he moves, and to get closer, if he does not.

$G(B'.\text{current-tasks} = \{\textit{approach-red}\} \rightarrow [(R_x, R_y) = (i,j)] \wedge [X((R_x, R_y) = (i, j+1))] \rightarrow [B'.\text{actions} = \textit{move-up}] \wedge [(R_x, R_y) = (i,j)] \wedge [X((R_x, R_y) = (i, j-1))] \rightarrow [B'.\text{actions} = \textit{move-down}] \wedge [(R_x, R_y) = (i,j)] \wedge [X((R_x, R_y) = (i+1, j))] \rightarrow [B'.\text{actions} = \textit{move-right}] \wedge [(R_x, R_y) = (i,j)] \wedge [X((R_x, R_y) = (i-1, j))] \rightarrow [B'.\text{actions} = \textit{move-left}] \wedge [(R_x, R_y) = (i,j)] \wedge [X((R_x, R_y) = (i,j))] \rightarrow [(B'_x >$

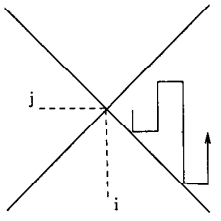


Fig. 2. Path Taken when Scanning Outwards.

$R_x) \rightarrow [B'.\text{actions} = \textit{move-left}] \wedge (B'_x < R_x) \rightarrow [B'.\text{actions} = \textit{move-right}] \wedge ((B'_x = R_x) \wedge (B'_y > R_y)) \rightarrow [B'.\text{actions} = \textit{move-down}] \wedge ((B'_x = R_x) \wedge (B'_y < R_y)) \rightarrow [B'.\text{actions} = \textit{move-up}]])$

The correctness of this can be seen from the fact that *approach-red* is really an abbreviation for $\textit{Maintain}((|B'_x - R_x| + |B'_y - R_y| = m) \rightarrow X(|B'_x - R_x| + |B'_y - R_y| \leq m) \wedge (|B'_x - R_x| + |B'_y - R_y| = m) \wedge (m > 1) \rightarrow F(|B'_x - R_x| + |B'_y - R_y| < m))$

11) Reasoning-ability-optimal-resource-abstract-moves-for-self: the ability to compute optimal moves, given a resource-assignment, and to decide to do only ones own action according to those moves.

$$G(\text{optimal-2}\ (B'.\text{resource-assignment}, (\text{task}_1, B'), (\text{task}_2, B^{k_2}), (\text{task}_3, B^{k_3}), (\text{task}_4, B^{k_4})) \rightarrow B'.\text{current-tasks} = (\text{task}_1))$$

12) Reasoning-ability-send-cost-estimate:

$$G((\text{cost} = c \rightarrow B'.\text{actions} = \textit{Send}(B', B'', \text{cost-estimate-protocol}, c)))$$

the ability to make and send the optimal cost estimate to the controller, B''.

13) Reasoning-ability-abstract-resource-commands: the ability to give abstract commands to subordinates.

$$G(\text{optimal-5}\ (B'.\text{resource-assignment}, (\text{abs}_1, B'), (\text{abs}_2, B^{k_2}), (\text{abs}_3, B^{k_3}), (\text{abs}_4, B^{k_4})) \rightarrow B'.\text{actions} = (\text{abs}_1, \textit{Send}(B', B^{k_2}, \textit{abstract-commands}, \text{abs}_2), \textit{Send}(B', B^{k_3}, \textit{abstract-commands}, \text{abs}_3), \textit{Send}(B', B^{k_4}, \textit{abstract-commands}, \text{abs}_4)))$$

14) Reasoning-ability-achieve-anything: the ability to do an *Achieve* task optimally. In general, this is too powerful a requirement for real-life agents.

$$G(B'.\text{current-tasks} = \{\textit{Achieve}(t)\} \wedge \text{optimal-4}(\textit{Achieve}(t), \text{move}) \rightarrow B'.\text{actions} = \text{move}).$$

5.4. Communicating Abilities

The communicating abilities of an agent tell us what kinds of messages he can send, to which kinds of agents, over what distances, and in what circumstances. These abilities fix the physical restrictions under which one agent is able to communicate with another. We specify communicating abilities in two ways: 1) as conditions on when certain agents can send or receive messages on different protocols, and 2) as conditions on the operations of those protocols themselves. The latter are considered in Section VI along with other conditions on protocols. Communicating abilities are specified in the first way by "$G(\text{condition}_1 \rightarrow \text{condition}_2)$." "$\text{Condition}_1$" may involve conditions about the real world and the agent's internal state, as well as the state of the agent with whom he is to communicate. "Condition_2" must be of the form *Can-send* or *Can-receive* applied to arguments specifying the protocol and both of the agents involved. An example specification expressing "if B' is within 5 steps of B'', then B' and B'' can communicate using the protocol for *basic-commands*" is

$$G((|B'_x - B''_x| < 5) \rightarrow (\textit{Can-send}(B', B'', \textit{basic-commands})))$$
$$G((|B'_x - B''_x| < 5) \rightarrow (\textit{Can-receive}(B'', B', \textit{basic-commands}))).$$

This specification language can also express the reliability of a communication channel. However, the security of a communication channel cannot be expressed by this language, since that depends on the knowledge and perceiving and reasoning abilities of other agents (a communication channel is *secure* if no other agent than the intended receiver can monitor it).

VI. Communication Protocols

Protocols determine how agents communicate. To a large extent, they determine the organization that exists among the agents, and the nature and extent of the interactions allowed among them. Since agents can command, permit, inform, give explanations to, and request other agents, they can participate in the protocols that correspond to these activities. It is sometimes convenient to talk of a "protocol" with the sender and receiver abstracted out, i.e., to have the same protocol exist between different pairs of agents.

We consider synchronous and perfectly reliable protocols only, and focus on their logical aspects, which are the most important for our purposes here—their physical aspects are relatively unimportant since we are not considering performance issues. Some protocols, such as elections, can involve many agents, but we assume in the following that all protocols involve only two agents. A message to be broadcast will thus

have to be transmitted to each agent in turn, and knowledge about synchronization will not be available, even tacitly, to the agents. The following aspects of protocols are relevant.

- The sender: the agent who sends communications under this protocol.
- The receiver: the agent who receives communications sent under the protocol.
- The languages used in the protocol by the senders and the responders, respectively. These languages determine what may be communicated using the protocol.
- The functions that are used first to encode the message of the sender, and second (on its arrival at the receiver's site) to decode the message into a form acceptable to the receiver.
- The actions to be taken by the receiver of the protocol when a message is actually received, so as to incorporate the contents of the message into its local data structures for further processing.

Protocols are specified simply in terms of the language of communication that they allow and the pairs of agents who use them to communicate. However, the more interesting part of their specification concerns their other properties. These include the specification of when they are effective, how much time-delay they introduce in the transmission of particular messages, and what kinds of errors they may introduce in the transmitted message. These properties and the information that is communicated by the use of a protocol between two agents are specified by the use of the predicates *Sends* and *Receives* corresponding to the occurrence of the actions *Send* and *Receive*, respectively. $Sends(B', B'', P, M)$ evaluated at t means that message M is sent by agent B' at time t according to protocol P: if agent B'' can understand protocol P, then B'' receives M' at t'. M' is the message received after the noise of the channel is factored in and $(t' - t)$ is the communication delay. Note that the above specification applies only to B'. An example specification is

$$G((|B'_x - B''_x| < 5) \rightarrow (Sends(B', B'', basic\text{-}commands, M) \rightarrow XX(Receives(B'', B', basic\text{-}commands, M')))).$$

The above are more or less the physical properties of protocols. We would like to clarify that we have not addressed the problem of what agents or systems ought to do to transmit messages successfully or to recover from failure: the first is a problem in networks and distributed computing; the second concerns domain-dependent heuristics. Each is an important research problem that we are abstracting out. We indicated above that different types of protocols correspond to different types of illocutionary force. We now give the illocutionary force semantics of protocols.

6.1. Information Protocols

Information protocols provide the languages for communicating factual knowledge. The messages received on these protocols are interpreted as information about a particular part of the world. However, they are best treated uniformly as information about the sender's internal state, and then processed (depending on how the sender and receiver are related) to cause changes in the receiver's internal state about the world. There is no monotonic theory possible for how the internal states of agents should be updated after the receipt of information from other agents [21], so we rely on a processing function associated with the protocol to provide the operational semantics of the computation required. If P is an *information protocol*, then

$$G(Receive(B', B'', P, message) \rightarrow X([P.\text{function-to-apply}](message, B'.\text{local-state}, P.\text{slot-affected})))$$

P.slot-affected is the slot whose value is affected by the information received on protocol P. Note that P.function-to-apply is treated like a predicate, and is applied to its arguments at the next time index. In practice, P.function-to-apply would be a function used to compute the value of $B'.P$.slot-affected at the next time index. For example, let P be a protocol instance of *blue-location* using which an agent B^2 sends its location, (i, j), to controller B^1. Then $P.\text{function-to-apply}((i, j), B^1.\text{local-state}, B^1.B^2\text{-location})$ is true at the next time index.

6.2. Command Protocols

The messages transmitted on command protocols are action descriptions, interpreted as commands from the sender to the receiver. These protocols specify the receiving agent's acting abilities as they can be directly used by the sender. The receiver might be able to perform other actions, but may not accept those as commands under the given protocol. This could occur if the sender does not know of those abilities at all, or can get them done, not as a matter of right, but only by making proper requests. Even legitimate commands are accepted only conditionally, i.e., the receiver would accept and work on only those commands that do not violate any of several kinds of constraints. As an example of a physical constraint, a Blue agent cannot move off the grid or move into a location occupied by Red. Furthermore, a command may be executed only if some local interrupt (e.g., a fire) does not change the receiver's immediate priorities or render him unable to execute it. Even "social" constraints might come into play, e.g., a subordinate may refuse to follow an order if he believes that doing so would be unethical. As a specific example of using this protocol, if P is a *basic command protocol*, then

$$G(Receives(B', P, B'', message) \rightarrow X(message \in B'.\text{actions})).$$

For abstract command protocols, the slot affected is "current-tasks."

6.3. Request Protocols

A sending agent uses a request protocol to ask a receiving agent to grant a permission, or provide information, or perform an action that he desires. Request protocols are somewhat harder to specify than command protocols, since action descriptions sent over them are not necessarily accepted for doing by the receiver. Even if the receiver initially agrees

to do the requested action, there is no guarantee that it will be done—ordinarily, actions that are done in order to satisfy requests are of a lower priority than actions that are done to satisfy commands or the agent's own plans. Even if priorities are ignored, the semantics of a request protocol cannot be determined statically, because it depends on the situations under which the protocol is used and on the state of the receiving agent at that point.

But as in the specification for information protocols, we may use the function associated with the protocol to decide how a given message affects the receiver's agenda. If P is a *request protocol*, then

$$G(\mathit{Receives}(B', P, B'', \mathit{message}) \rightarrow X([P.\text{function-to-apply}](\mathit{message}, B'.\text{local-state}, B'.\text{agenda}))).$$

6.4. Permission Protocols

A permission protocol is used by a sender to grant permissions to a receiver. These permissions might not have been solicited by the receiver, and could be given just to allow certain options that are not available by default, e.g., the sender may allow the receiver to enter a certain region should the latter ever need to. We analyze permission protocols in terms of the actions they make available to the receiver. This presumes that the sender has the requisite authority to issue such permissions. In many cases, we would want that once an agent is assigned some acting-abilities, it continues to have them for some unspecified period, perhaps until it is issued a counteracting prohibition. How an agent's abilities ought to persist in the face of sequences of permissions and prohibitions is an issue that we leave to future research (see Section IX). For *permission protocol* P,

$$G(\mathit{Receives}(B', P, B'', \mathit{message}) \rightarrow X(\mathit{message} \in B'.\text{acting-abilities})).$$

6.5. Prohibition Protocols

Prohibition protocols are used by a sending agent to limit the choices that are available to the receiver. Prohibitions might explicitly rule out certain actions, or introduce new conditions that the receiver must not allow to be violated. We analyze prohibition protocols in terms of the actions they disallow the receiver that he was allowed before. This presumes that the sender has the requisite authority. For *prohibition protocol* P,

$$G(\mathit{Receives}(B', P, B'', \mathit{message}) \rightarrow X(\mathit{message} \notin B'.\text{acting-abilities})).$$

6.6. Explanation Protocols

Explanation protocols are like information protocols, but more explicitly involve an agent transmitting justifications (or, at least, potential or putative justifications). Such knowledge is useful when an agent who received some information must try to estimate its reliability, or predict the effects of some action he is considering performing. These protocols concern the internal state of the receiver being a particular way (and not another). Accordingly, they directly affect just the receiver's view of the sender's internal state. The justifications or explanations received may also be accepted by the receiver, in which case they may affect his own representations about the world; however, we treat this process as a part of the receiver applying his reasoning abilities on his own internal state. There is no easy specification for this. We use the following (with P an *explanation protocol*):

$$G(\mathit{Receives}(B', P, B'', \mathit{message}) \rightarrow X([P.\text{function-to-apply}](\mathit{message}, B'.B''.\text{local-state}))).$$

VII. Specifications of Example Systems

This section contains specifications in our formal language for several variations of the pursuit problem. The centrally controlled systems involve a master and some slaves, so both kinds of agents must be specified; only one kind of agent is needed for distributively controlled systems. The resource assignments and actions given below are meant only as examples; the actual values would vary with time. The other slot values are defined in Section V. For brevity, the empty slots are not displayed.

7.1. Omniscient Central Controller

1) The problem: as given above
2) The agents: master B^1, and slaves B^2, B^3, B^4

   a. master:

   - acting abilities: *move-left*,*move-right*, *move-down*,*move-up*,*no-op*.
   - perceiving abilities: $G(B'.A_x = A_x)$, and $G(B'.A_y = A_y)$, where A may be Red, or any Blue agent
   - communicating abilities: $G(\mathit{Can\text{-}send}(B', B'', \mathit{basic\text{-}commands}))$, where B'' is any slave
   - reasoning abilities: Reasoning-ability-quadrant-assignment, Reasoning-ability-optimal-resource-moves-for-all
   - problem state: $\langle (R_x, R_y), (B_x^1, B_y^1), (B_x^2, B_y^2), (B_x^3, B_y^3), (B_x^4, B_y^4) \rangle$
   - method: *Achieve*(Lieb), *Achieve*(Win)
   - current tasks: tq_l, tq_r, tq_d, tq_u
   - resources: $B^k, k \in [1 \ldots 4]$
     - augmented access protocol: *basic-commands*
     - cost metric: time
     - constraints: $G(1 \leq B_x^k, B_y^k \leq 30)$ and $G((B_x^k, B_y^k) \neq (R_x, R_y))$
     - state vector: B_x^k, B_y^k
   - resource assignment: $(tq_l, B^1), (tq_r, B^2), (tq_d, B^3), (tq_u, B^4)$
   - actions: *move-left*, *Send*(B', B^2, *basic-commands*, *move-right*), ...

b. slave:

- acting abilities: *move-left*, *move-right*, *move-down*, *move-up*, *no-op*
- communicating abilities: G(*Can-receive*(B', B^1, *basic-commands*))
- actions: *Receive*(B', B^1, *basic-commands*, *message*)

7.2. Central Controller, Agents Perceive Own Location

1) The problem: as given above
2) The agents: master B^1, and slaves B^2, B^3, B^4

a. master:

- acting abilities: *move-left*, *move-right*, *move-down*, *move-up*, *no-op*
- perceiving abilities: $G(B'.R_x = R_x)$, $G(B'.R_y = R_y)$, $G(B'.B'_x = B'_x)$, and $G(B'.B'_y = B'_y)$
- communicating abilities: G(*Can-send*(B', B'',*basic-commands*)) G(*Can-receive*(B', B'',*blue-location*)), where B'' is any slave
- reasoning abilities: Reasoning-ability-quadrant-assignment, Reasoning-ability-optimal-resource-moves-for-all
- problem state: $\langle (R_x, R_y), (B^1_x, B^1_y), (B^2_x, B^2_y), (B^3_x, B^3_y), (B^4_x, B^4_y)\rangle$
- method: *Achieve*(Lieb), *Achieve*(Win)
- current tasks: tq_l, tq_r, tq_d, tq_u
- resources: $B^k, k \in [1 \ldots 4]$
  - augmented access protocol: *basic-commands*
  - cost metric: time
  - constraints: $G(1 \leq B^k_x, B^k_y \leq 30)$ and $G((B^k_x, B^k_y) \neq (R_x, R_y))$
  - state vector: B^k_x, B^k_y
- resource assignment: $(tq_l, B^1), (tq_r, B^2), (tq_d, B^3), (tq_u, B^4)$
- actions: *move-left*, *Send*(B', B^2, *basic-commands move-right*), ...

b. slave:

- acting abilities: *move-left*, *move-right*, *move-down*, *move-up*, *no-op*
- perceiving abilities: $G(B'.B'_x = B'_x)$, and $G(B'.B'_y = B'_y)$
- communicating abilities: G(*Can-receive*(B', B^1, *basic-commands*)), G(*Can-send*(B', B^1, *blue-location*))
- reasoning abilities: Reasoning-ability-send-own-location
- actions: *Receive*(B', B^1, *basic-commands*, *message*)

7.3. Central Controller, Agents Search for Red

1) The problem: as given above
2) The agents: master B^1, and slaves B^2, B^3, B^4

a. master:

- acting abilities: *move-left*, *move-right*,*move-down*, *move-up*, *no-op*
- perceiving abilities: $G(|B'_x - R_x| + |B'_y - R_y| \leq 2) \rightarrow (B'.R_x = R_x) \wedge (B'.R_y = R_y)$, $G(B'.B'_x = B'_x)$, and $G(B'.B'_y = B'_y)$
- communicating abilities: G(*Can-send*(B', B'', *basic-commands*)),G(*Can-receive*(B', B'', *blue-location*)), and G(*Can-receive*(B', B'', *red-location*)), where B'' is any slave
- reasoning abilities: Reasoning-ability-quadrant-assignment, Reasoning-ability-optimal-resource-moves-for-all, Reasoning-ability-mount-search, Reasoning-ability-close-in-on-red
- problem state: $\langle (R_x, R_y), (B^1_x, B^1_y), (B^2_x, B^2_y), (B^3_x, B^3_y), (B^4_x, B^4_y)\rangle$
- method: *Achieve* (B'.found), *Achieve*(Lieb), *Achieve*(Win)
- current tasks: tq_l, tq_r, tq_d, tq_u resources: $B^k, k \in [1 \ldots 4]$
  - augmented access protocol: *basic-commands*
  - cost metric: time
  - constraints: $G(1 \leq B^k_x, B^k_y \leq 30)$ and $G((B^k_x, B^k_y) \neq (R_x, R_y))$
  - state vector: B^k_x, B^k_y
- resource assignment: $(tq_l, B^1), (tq_r, B^2), (tq_d, B^3), (tq_u, B^4)$
- actions: *move-left*, *Send*(B', B^2, *basic-commands*,*move-right*), ...

b. slave:

- acting abilities: *move-left*, *move-right*, *move-down*, *move-up*, *no-op*, as specified previously
- perceiving abilities: $G(|B'_x - R_x| + |B'_y - R_y| \leq 2) \rightarrow (B'.R_x = R_x) \wedge (B'.R_y = R_y)$, $G(B'.B'_x = B'_x)$, and $G(B'.B'_y = B'_y)$
- communicating abilities: G(*Can-receive* (B', B^1, *basic-commands*)),G(*Can-send* (B', B^1,*blue-location*)), and G(*Can-send* (B', B^1, *red-location*))
- reasoning abilities: Reasoning-ability-send-own-location, Reasoning-ability-send-red-location
- actions: *Receive*(B', B^1, *basic-commands*, *message*)

7.4. Central Control by Abstract Commands

1) The problem: as given above

2) The agents: master B^1, and slaves B^2, B^3, B^4

a. master:
- acting abilities: *move-left*, *move-right*, *move-down*, *move-up*, *no-op*
- perceiving abilities: $G(|B'_x - R_x| + |B'_y - R_y| \leq 2) \rightarrow (B'.R_x = R_x) \wedge (B'.R_y = R_y)$, $G(B'.B'_x = B'_x)$, and $G(B'.B'_y = B'_y)$
- communicating abilities: $G($*Can-send*$(B', B''$, *abstract-commands*$))$,$G($*Can-receive*$(B', B''$, *blue-location*$))$, and $G($*Can-receive*$(B', B''$, *red-location*$))$, where B'' is any slave
- reasoning abilities: Reasoning-ability-quadrant-assignment, Reasoning-ability-start-patrolling, Reasoning-ability-scan-outwards, Reasoning-ability-scan-inwards, Reasoning-ability-approach-red, Reasoning-ability-abstract-resource-commands
- problem state: $\langle (R_x, R_y), (B^1_x, B^1_y), (B^2_x, B^2_y), (B^3_x, B^3_y), (B^4_x, B^4_y) \rangle$
- method: *Achieve* $(B'$.found$)$, *Achieve*(Lieb), *Achieve*(Win)
- current tasks: tq_l, tq_r, tq_d, tq_u resources: $B^k, k \in [1 \ldots 4]$
  - augmented access protocol: *abstract-commands*
  - cost metric: time
  - constraints: $G(1 \leq B^k_x, B^k_y \leq 30)$ and $G((B^k_x, B^k_y) \neq (R_x, R_y))$
  - state vector: B^k_x, B^k_y
- resource assignment: $(tq_l, B^1), (tq_r, B^2), (tq_d, B^3), (tq_u, B^4)$
- actions: *move-left*, *Send*$(B', B^2,$ *abstract-commands*$, \text{abs}_2), \ldots$

b. slave:
- acting abilities: *move-left*, *move-right*, *move-down*, *move-up*, *no-op*, as specified previously
- perceiving abilities: $G(|B'_x - R_x| + |B'_y - R_y| \leq 2) \rightarrow (B'.R_x = R_x) \wedge (B'.R_y = R_y)$, $G(B'.B'_x = B'_x)$, and $G(B'.B'_y = B'_y)$
- communicating abilities: $G($*Can-receive*$(B', B^1,$*abstract-commands*$))$, $G(B', B^1,$ *Can-send* *(blue-location)*$)$, and $G($*Can-send*$($*red-location*$))$
- reasoning abilities: Reasoning-ability-send-own-location, Reasoning-ability-start-patrolling, Reasoning-ability-scan-outwards, Reasoning-ability-scan-inwards, Reasoning-ability-approach-red
- actions: *Receive*$(B', B^1, abstract-command, message)$

7.5. Control Distributed among Altruistic Peers

1) The problem: as given above

2) The agents: B^1, B^2, B^3, and B^4

a. acting abilities: *move-left*, *move-right*, *move-down*, *move-up*, *no-op*

b. perceiving abilities: $G(|B'_x - R_x| + |B'_y - R_y| \leq 2) \rightarrow (B'.R_x = R_x) \wedge (B'.R_y = R_y)$, $G(B'.B'_x = B'_x)$, and $G(B'.B'_y = B'_y)$

c. communicating abilities: $G($*Can-receive*$(B', B''$, *blue-location*$))$,$G($*Can-receive*$(B', B''$, *red-location*$))$, $G($*Can-send*$(B', B''$, *blue-location*$))$, and $G($*Can-send*$(B', B''$, *red-location*$))$, where B'' is any other blue agent

d. reasoning abilities: Reasoning-ability-quadrant-assignment, Reasoning-ability-start-patrolling, Reasoning-ability-scan-outwards, Reasoning-ability-scan-inwards, Reasoning-ability-approach-red, Reasoning-ability-optimal-resource-abstract-moves-for-self

e. problem state: $\langle (R_x, R_y), (B^1_x, B^1_y), (B^2_x, B^2_y), (B^3_x, B^3_y), (B^4_x, B^4_y) \rangle$

f. method: *Achieve* $(B^1$.found $\wedge$ B^2.found $\wedge$ B^3.found $\wedge$ B^4.found$)$, *Achieve*(Lieb), *Achieve*(Win)

g. current tasks: tq_l, tq_r, tq_d, tq_u resources: $B^k, k \in [1 \ldots 4]$
  - augmented access protocol: *basic-commands*
  - cost metric: time
  - constraints: $G(1 \leq B^k_x, B^k_y \leq 30)$ and $G((B^k_x, B^k_y) \neq (R_x, R_y))$
  - state vector: B^k_x, B^k_y

h. resource assignment: $(tq_l, B^1), (tq_r, B^2), (tq_d, B^3), (tq_u, B^4)$

i. actions: if assignment-to-self(tq) then tq

7.6. Control Distributed among Self-Interested Agents

1) The problem: as given above

2) The agents: B^1, B^2, B^3, and B^4

a. acting abilities: *move-left*, *move-right*, *move-down*, *move-up*, *no-op*, as specified previously

b. perceiving abilities: $G(|B'_x - R_x| + |B'_y - R_y| \leq 2) \rightarrow (B'.R_x = R_x) \wedge (B'.R_y = R_y)$, $G(B'.B'_x = B'_x)$, and $G(B'.B'_y = B'_y)$

c. communicating abilities: $G($*Can-receive*$(B', B''$, *red-location*$))$, $G($*Can-send* $(B', B''$,*red-location*, $G($*Can-receive*$(B', B''$, *cost-estimate*$))$, and G $($*Can-send*$(B', B''$, *cost-estimate*$))$

d. reasoning abilities: Reasoning-ability-quadrant-assignment, Reasoning-ability-start-patrolling, Reasoning-ability-scan-outwards, Reasoning-ability-scan-inwards, Reasoning-ability-approach-red, Reasoning-ability-optimal-resource-abstract-moves-for-self, Reasoning-ability-send-cost-estimate, Reasoning-ability-abstract-resource-commands

e. problem state: $\langle (R_x, R_y), (B^1_x, B^1_y), (B^2_x, B^2_y), (B^3_x, B^3_y), (B^4_x, B^4_y) \rangle$

f. method: *Achieve* (B^1.found$\wedge B^2$.found$\wedge B^3$.found $\wedge B^4$.found), *Achieve*(Lieb), *Achieve*(Win)

g. current tasks: tq_l, tq_r, tq_d, tq_u resources: $B^k, k \in [1 \ldots 4]$

- augmented access protocol: *basic-commands*
- cost metric: time
- constraints: $G(1 \leq B_x^k, B_y^k \leq 30)$ and $G((B_x^k, B_y^k) \neq (R_x, R_y))$
- state vector: B_x^k, B_y^k

h. resource assignment: $(tq_l, B^1), (tq_r, B^2), (tq_d, B^3), (tq_u, B^4)$

i. actions: if assignment-to-self(tq) then tq

VIII. The Implementation

The representation scheme developed in this paper has been implemented in a small system built on top of the CYC knowledge base [18]. The implementation uses a set of frames (called "units" in CYC terminology) to define the attributes of different parts of the system. We describe only the important slots of some important units to give the reader a feel for the implementation. The core of the implementation is an interpreter that ensures that the appropriate semantics is operationally assigned to the different frames. Thus, when a frame contains, say, "cond1" and "cond2," it may actually stand for a temporal condition of the form: "G(cond1 $\rightarrow$ X(cond2))."

The problem-solving system on which the interpreter is to be executed is defined by defining the environment, the agent organization, and the win and lose conditions for the system. The environment is specified by giving a function that returns its initial state and a bunch of functions that generate its successive states. The win and lose conditions are simple functions on the state of the environment that return T under the appropriate conditions. The agent organization is the most complex part of the specification. It is given by listing the agents who participate in it, and the relationships they have among themselves. These are given in terms of the protocols that are defined between pairs of agents. The agents are further specified by giving the different abilities they have, each ability corresponding to a function in the implementation. The interpreter begins with the initial state of the environment. Then it loops as follows. It checks whether the win or the loss condition holds in the current state of the environment. If either does, it terminate. Otherwise, it invokes the agents' perceiving abilities on the environment. These yield values for some of the slots in the agents. The agents' internal representations are thus modified in virtual parallelism. Next, it invokes the agents' reasoning abilities on their internal representations, also in virtual parallelism. Since each ability has a precondition, the preconditions are tested to see which apply. We assume that when more than one ability is applicable, the order of applying them is not significant. The reasoning abilities lead to some slots of the agents being updated. Some of these slots could describe the agent's actions. The interpreter then looks at these slots and incorporates the effects of the physical actions in the environment. For actions that involve communication, it uses the encoding function of the protocol in composing messages for transmission to other agents. As these messages arrive, and are decoded using the decoding function of the given protocol, they may trigger further reasoning by their receivers, leading to further actions by them. If a zero delay is assumed, these actions too have to be incorporated in the environment. Finally, the interpreter incorporates the changes due to events in the environment itself, and the loop continues.

superclasses	(ProblemSolvingSystem)
canHaveSlots	(environmentOfSystem initialStateOfRPSS usesAgentOrganization ...)

Fig. 3. *ReactiveProblemSolvingSystem* Class

instanceOf	(ReactiveProblemSolvingSystem)
configurationConstraints	(BLUE-NOT-OVER-RED? IN-GRID?)
environmentOfSystem	(PursuitEnvironment-1)
initialStateOfRPSS	(INITIAL-STATE)
involvesAgents	(Blue-4 Blue-3 Blue-2 Blue-1)
lossCondition	(LOSS-CONDITION)
usesAgentOrganization	(BlueOrganization-CC3)
winCondition	(WIN-CONDITION)

Fig. 4. *CentralController-3*.

We emphasize that the temporal language described in previous sections is merely the specification language: it is not interpreted itself. Indeed, as already argued, if we attempted to have the specification language itself be interpreted, we would quickly run into the problems that plague the situation calculus approach. Some of the specifications correspond to assertions in the declarative representation; others are just specifications of different procedures. The general declarative scheme is useful even in those cases, since it "parses" a complex distributed system into simple chunks that can be coded easily and reliably. Also, the implementation is designed as a frame system, so different systems can be easily composed by reusing previously defined agents. This makes it easy to modify parts of a specification to see what the agents must know or be capable of doing in each case.

The functions named in the slot values have been implemented in Common Lisp. Some terminology in what follows has been modified, and some extraneous slot values have been deleted for expository purposes. At the top of the frame hierarchy for our implementation is the unit defining the class of *ReactiveProblemSolvingSystem*, displayed in Fig. 3. One of the instances of this class is *CentralController-3*, displayed in Fig. 4. This system's organization is described by the unit *BlueOrganization-CC3* in Fig. 5. Fig. 6 shows *ReactiveProblemSolvingAgent*, the class of agents who participate in the reactive problem-solving systems.

The perceiving ability of some agents to sense their own location is given by the unit, *SensingSelf*, displayed in Fig. 11. The slot **conditionOfPerception** is filled with the condition

instanceOf	(CentrallyControlledAgentOrg)
usedByRPSS	(CentralController-3)
controllerAgent	(Blue-1)
slaveAgents	(Blue-2 Blue-3 Blue-4)
slotInfoProtocols	(RedLocationInfo-Blue-2-1 RedLocationInfo-Blue-3-1 RedLocationInfo-Blue-4-1 ...)
slotInfoResources	(SlaveLocation-2-reason-1 Slavelocation-2-reason-2 Slavelocation-3-reason-1 ...)
agents	(Blue-2 Blue-3 Blue-4 Blue-1)

Fig. 5. *BlueOrganization-CC3.*

superclasses	(Agent)
subclasses	(ControllerAgent SlaveAgent)
canHaveSlots	(locationOfRed locationOfSelf initialstateofagent ...)

Fig. 6. *ReactiveProblemSolvingAgent* class.

instances	(Blue-1)
superclasses	(ReactiveProblemSolvingAgent)
canHaveSlots	(ControlsAgentOrganization)

Fig. 7. *ControllerAgent* class.

instances	(Blue-4 Blue-3 Blue-2)
superclasses	(ReactiveProblemSolvingAgent)
canHaveSlots	(isSlaveInOrganization)

Fig. 8. *SlaveAgent* class.

under which this ability is operable (in this case it is identically T), and the slot **whatIsPerceived** with what the agent actually perceives (in this case, the agent's own location).

The physical constraints on the participation by agent *Blue-1* in different communication protocols are expressed by the unit *ProtocolForBlue-1* shown in Fig. 12. This communicating ability endows this agent with the ability to send basic commands to the other three slave agents; i.e., to use the appropriate command protocols.

The unit *LocationInfo-Blue-2-1*, shown in Fig. 13, specifies the protocol used by agent *Blue-2* to send information about his own location to agent *Blue-1*. This protocol is described in terms of its encoding and decoding functions, the reasoning abilities of the receiver that it influences, and the way in which it influences them. The unit *SlaveLocation-2-reason-1*, displayed in Fig. 14, specifies the agent who supplies the information carries over the above information protocol as a "resource" of the receiving agent.

The reasoning ability of an agent (in this case *Blue-1*) to compute the moves for all agents to converge on Red is specified by the unit *Reasoning-Converge-On-Red*. This unit,

instanceOf	(ControllerAgent)
actingAbilities	(NoOp MoveDown MoveUp ...)
controlsOrganization	(BlueOrganization-CC3)
communicatingAbilities	(ProtocolForBlue-1)
isInAgentOrganization	(BlueOrganization-CC3)
isInvolvedInRPSS	(CentralController-3 ...)
reasoningAbilities	(Reasoning-ConvergeOnRed Reasoning-OccupyQuadrant Reasoning-SearchForRed)
receivesFromProtocols	(LocationInfo-Blue-2-1 LocationInfo-Blue-3-1 LocationInfo-Blue-4-1)
sensingAbilities	(SensingRed SensingSelf)

Fig. 9. *Blue-1.*

instanceOf	(SlaveAgent)
actingAbilities	(MoveLeft MoveRight MoveUp ...)
communicatingAbilities	(ProtocolForRedLocationInfo-Blue-2 ProtocolForSlaveLocationInfo-Blue-2 ProtocolForBlue-2)
isInAgentOrganization	(BlueOrganization-CC3)
isInvolvedInRPSS	(CentralController-3)
isResourceIn	(ResourceBlue-2)
isSlaveInOrganization	(BlueOrganization-CC3)
reasoningAbilities	(Reasoning-SlaveSendRedLocation Reasoning-SlaveSendSelfLocation)
receivesFromProtocols	(BasicComm-1-2)
sensingAbilities	(SensingRed SensingSelf)

Fig. 10. *Blue-2.*

instanceOf	(PerceptionSpecification)
conditionOfPerception	(SENSING-CONDITION-1)
whatIsPerceived	(SENSED-STATE-1)
whoseSensingAbility	(Blue-1 Blue-2 Blue-3 Blue-4)

Fig. 11. *SensingSelf.*

shown in Fig. 15, specifies the information protocols this ability relies on to supply the required information, the slot whose value this ability changes, the values it assigns, and the condition under which this ability is effective (in this case, always).

IX. Conclusions and Future Work

Starting with some obvious intuitions about the knowledge and capability requirements that even simple distributed algorithms impose upon the agents in a system, we have devised a simple scheme for declaratively representing such problem-solving systems, and the agents who compose them. This scheme simplifies the task of the system designer by encouraging him to think in terms of what the different agents

instanceOf	(Protocol)
conditionSpecification	(PROTOCOL-CONDITION-1)
sendsOnProtocols	(BasicComm-1-4 BasicComm-1-3 BasicComm-1-2)
whoseCommunicatingAbility	(Blue-1)

Fig. 12. *ProtocolForBlue-1.*

instanceOf	(SlotInfoProtocols)
actionsToBeTakenByReceiver	(INTRODUCE-SLAVE-LOCATION-1)
affectsReasoningAbilitiesOfReceiver	(Reasoning-ConvergeOnRed Reasoning-OccupyQuadrant ReasoningBlue-1 ReasoningBlue-2)
affectsInfoResource	(SlaveLocation-2-reason-2 SlaveLocation-2-reason-1)
hasDecodingFunction	(DECODE-SLOT-INFO-1)
hasEncodingFunction	(ENCODE-SLOT-INFO-1)
involvesCommunicatingAbilities	(ProtocolSlaveLocation-2)
isSlotInfoProtocolOf	(BlueOrganization-CC3 ...)
agentsReceivingOnThisProtocol	(Blue-1)
usesSlotsOfSender	(locationOfSelf localState)

Fig. 13. *LocationInfo-Blue-2-1.*

instanceOf	(InfoResource)
isInfoResourceOf	(BlueOrganization-CC3)
affectsReasoningAbilities	(ReasoningBlue-1 Reasoning-OccupyQuadrant)
resourceSuppliedBySlotInfoProtocols	(LocationInfo-Blue-2-1)

Fig. 14. *SlaveLocation-2-reason-1.*

are expected to know and to be able to do, at different stages of their problem-solving activity. The structure created by this representational scheme partitions the tasks to be done by a system designer into simple ones that may be easily and correctly done. The formal semantics that we provide further crystallizes the intuitions behind this scheme. In this way, this scheme may be used in the design of multiagent systems. It may also be used to prove certain properties of systems that have already been designed, e.g., whether requests issued in a certain state would be accepted; whether, given certain behavior by the environment, the system would be able to succeed; and so on. By partitioning a system into simple components, this scheme not only simplifies design, but also implementation. This process will be further improved as appropriate tools, such as temporal logic proof checkers and theorem provers, become available [1], [20]. We leave these further technical developments to future research.

One issue we left unexplored is to relate the scheme presented here with some well-known paradigms in DAI, e.g., the contract net of Davis and Smith [7]. It would also be

instanceOf	(ReasoningSpecification)
conditionOfLocalState	(REASONING-CONDITION-1)
resultantValueOfSlot	(STEPS-TO-CONVERGE)
usesSlotInfoProtocolsForInput	(LocationInfo-Blue-4-1 LocationInfo-Blue-3-1 LocationInfo-Blue-2-1)
usesSlotInfoResources	(SlaveLocation-2-reason-2 SlaveLocation-3-reason-2 SlaveLocation-4-reason-2)
valueForAgentSlot	(currentActions)
whoseReasoningAbility	(Blue-1)

Fig. 15. *Reasoning-Converge-On-Red.*

useful to consider "metaprotocols" that may be used by agents to establish the protocols they would use during a session of problem-solving. We believe that the machinery to simulate metaprotocols already exists (e.g., by using information, prohibition, and permission protocols appropriately), but the spirit of our work has been to state explicitly and declaratively what would otherwise be hidden in hard-coded procedural interactions. We have assumed that protocols can be accurately specified with constraints. However, in complex systems, the kinds of considerations that go into an agent's decision to accept or decline a request would not be monotonically specifiable. Also, it would be useful to have some kind of a normative theory of communication for agents—research into this area has begun only recently [21]; closer connections remain to be explored.

The representational scheme presented here is state-based, i.e., it considers the states of the world explicitly and sees events as transitions between successive states. This makes it simple, but also makes it incapable of representing some information about events naturally, e.g., about the manner in which they are done, their direct ramifications, and their causes. It remains to be seen how advantageous such information would be in DAI domains. Current event-based schemes cannot represent all such information easily either [14]. A feature of our approach is that it uses *linear* temporal logic. A possible extension of our theory is branching time logic in which alternative actions can be expressed easily [11]. The positive consequences of using temporal logic of whatever form are that it is simple and well known, and tools for it are already under development [1], [20]. A potential weakness of temporal logic is that it does not express actions explicitly, even though it can be used to prove general properties of systems; it is still to be seen how significant this limitation is in the context of DAI.

In complex systems, it is important that an agent be able to represent other agents, especially regarding whether they are cooperative, neutral, or antagonistic to him. In this paper, we have referred directly to parts of the agents' internal representations (e.g., as $B'.R_x$). This is satisfactory for only the simplest agents, solving simple problems, as in this paper. In more realistic systems, an agent must reason about other agents explicitly, but their physical states would be too complex to

be reasoned about directly. In such settings, beliefs, know-how, and intentions are useful abstractions to use. It would be interesting to see how the theory of this paper would be extended for such cases.

REFERENCES

[1] M. Abadi, "Temporal-logic theorem proving," Ph.D dissertation, Stanford University, 1987, Department of Computer Science, *Tech. Report STAN-CS-87-1151*.

[2] P. Agre and D. Chapman, "Pengi: An implementation of a theory of activity," in *Proc. AAAI-87*, pp. 268–272, 1987.

[3] N. Arni *et al.*, "Overview of RAD: A hybrid and distributed reasoning tool," *Tech. Rep. ACT-RA-098-90*, Microelectronics and Computer Technology Corporation, Artificial Intelligence Laboratory, Austin, TX, Mar. 1990.

[4] J. L. Austin, *How to do Things with Words*. Clarendon, Oxford, UK, 1962.

[5] J. Barwise and J. Perry, *Situations and Attitudes*. Cambridge, MA: MIT Press, 1983.

[6] M. Benda, V. Jaganathan, and R. Dodhiawala, "On optimal cooperation of knowledge sources," *Tech. Rep.*, Boeing Advanced Technology Center, Boeing Computer Services, Seattle, WA, Sept. 1986.

[7] R. Davis and R. G. Smith, "Negotiation as a metaphor for distributed problem solving,"*Artificial Intelligence*, vol. 20, p. 63–109, 1983.

[8] E. H. Durfee, V. R. Lesser, and D. D. Corkill, "Coherent cooperation among communicating problem solvers," *IEEE Trans. Computers*, vol. C-36, pp. 1275–1291, Nov. 1987.

[9] E. H. Durfee and T. A. Montgomery, "MICE: A flexible testbed for intelligent coordination experiments," in *Proc. 9th Workshop on Distributed Artificial Intelligence*, pp. 25–40, Sept. 1989.

[10] E. A. Emerson, "Temporal and modal logic," in J. van Leeuwen, Ed., *Handbook of Theoretical Computer Science*. Amsterdam, The Netherlands: North-Holland Publishing Company, 1989.

[11] E. A. Emerson and J. Y. Halpern, " 'Sometimes' and 'Not Never' revisited: On branching versus linear time temporal logic," *J. ACM*, vol. 33, pp. 151–178, 1986.

[12] R. F. Franklin and L. A. Harmon, "Elements of cooperative behavior," *Tech. Rep.*, Environmental Research Institute of Michigan, Ann Arbor, MI, Aug. 1987.

[13] L. Gasser, N. Rouquette, R. Hill, and J. Lieb, "Representing and using organizational knowledge in distributed AI systems," in M. Huhns and L. Gasser, Eds., *Distributed Artificial Intelligence II*. London, UK: Pitman Publishing Limited, 1989, ch. 3, pp. 55–78.

[14] M. P. Georgeff and A. L. Lansky, "A representation of parallel activity based on events, structure and causality," in M. Georgeff and A. Lansky, Eds., in *Proc. 1986 Workshop on Reasoning about Actions and Plans*, pp. 123–160, 1987.

[15] C. L. Hamblin, *Imperatives*. Oxford, UK: Basil Blackwell Ltd., 1987.

[16] P. J. Hayes, "The frame problem and related problems in artificial intelligence," in B. L. Webber and N. J. Nilsson, Eds., *Readings in Artificial Intelligence*. San Mateao, CA: Morgan Kaufmann, 1981, pp. 223–230.

[17] M. N. Huhns *et al.*, "DAI for document retrieval: The MINDS project," in M. N. Huhns, Ed., *Distributed Artificial Intelligence* London: Pitman/Morgan Kaufmann, 1987, pp. 249–283.

[18] D. B. Lenat and R. V. Guha, *Building Large Knowledge Base Systems*. Reading, MA: Addison Wesley, 1989.

[19] J. McCarthy and P. J. Hayes, "Some philosophical problems from the standpoint of artificial intelligence," in *Machine Intelligence 4*. American Elsevier, 1969.

[20] B. Moszkowski, *Executing Temporal Logic Programs*. Cambridge, UK: Cambridge University Press, 1986.

[21] R. Perrault, "An application of default logic to speech act theory," *Tech. Rep. 90*, Center for the Study of Language and Information, Stanford, CA, Mar. 1987.

[22] S. J. Rosenschein, "Formal theories of knowledge in AI and robotics," *New Generation Computing*, vol. 3, 1985.

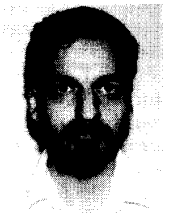
[23] Y. Shoham, *Reasoning About Change: Time and Causation from the Standpoint of AI*. Cambridge, MA: MIT Press, 1988.

[24] D. Smith and M. Broadwell, "The pilot's associate—An overview," in *Proc. SAE Aerotech Conf.*, Los Angeles, CA, May 1988.

[25] L. M. Stephens and M. Merx, "The effect of agent control strategy on the performance of a DAI pursuit problem," *IEEE Trans. Syst., Man, Cybern.*, 1990, submitted for publication.

[26] S. Tutiya, D. Israel, and J. Perry, "Action as meaning," in *Proc. First Conf. Situation Theory and Its Applications*, Stanford, CA, 1989.

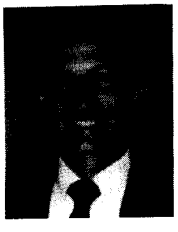
[27] J. F. A. K. van Benthem, *The Logic of Time*. Dordrecht, Netherlands: D. Reidel, 1984.



Munindar P. Singh received the Ph.D. degree in computer science from the University of Texas at Austin.

He works on theoretical aspects of artificial intelligence, especially in the area of multiagent systems. His research has been on theories of intention, know-how, and communication. He is also interested in the logics of belief and time.

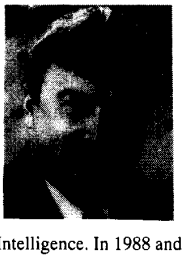
He is currently a member of the technical staff at MCC.



Michael N. Huhns received the B.S.E.E. degree from the University of Michigan in 1969, and the M.S. and Ph.D. degrees in electrical engineering from the University of Southern California in 1971 and 1975, respectively.

He was an associate professor of electrical and computer engineering at the University of South Carolina, where he also directed the Center for Machine Intelligence. He joined MCC in 1985, where he is a senior member of the artificial intelligence laboratory conducting research on the Argo, Antares, RAD, and Carnot projects. He is the author of 84 technical papers in machine intelligence and an editor of the books *Distributed Artificial Intelligence*, volumes I and II. His research interests are in the areas of machine learning, distributed artificial intelligence, database schema integration, and computer vision.

Dr. Huhns is a member of Sigma Xi, Tau Beta Pi, Eta Kappa Nu, ACM, and AAAI.



Larry M. Stephens received the B.S. degree in electrical engineering from the University of South Carolina in 1968 and the M.S. and Ph.D. degrees in electrical engineering from the Johns Hopkins University in 1974 and 1977, respectively.

As a US naval officer he was assigned to the Naval Reactors Program, Washington, DC.Since 1978 he has been a member of the faculty of the University of South Carolina and is currently a professor of electrical and computer engineering and a member of the department's Center for Machine Intelligence. In 1988 and 1989 he was on leave from his academic position and served as a consultant to MCC, Austin, TX, where he participated in research projects on distributed knowledge-based systems, reasoning architectures for synthesis tasks, and plausible inferencing. His current research interests include the fundamentals of knowledge representation and common-sense reasoning.

Dr. Stephens is a member of Tau Beta Pi, Eta Kappa Nu, Phi Beta Kappa, AAAI, and Sigma Xi.