# Synchronization of Multi-Agent Plans

Jeffrey S. Rosenschein
Computer Science Department
Stanford University
Stanford, CA 94305

## Abstract

Consider an intelligent agent constructing a plan to be executed by several other agents; correct plan execution will often require that actions be taken in a specific sequence. Therefore, the planner cannot simply tell each agent what action to perform; explicit mechanisms must exist for maintaining the execution sequence. This paper outlines such mechanisms. A framework for multiple-agent planning is developed, consisting of several parts. First, a formalism is adopted for representing knowledge about other agents' beliefs and goals, and is extended to allow representation of their capabilities. Communication primitives are defined that allow selective acceptance of goals and facts, and an explicit means of inducing an agent to perform an act is introduced. Finally, the ordering mechanisms (consisting of sequencing operators and a planning heuristic) are presented, along with a specific example of their use.

## Introduction

In recent years there has been growing interest in *distributed artificial intelligence* systems, collections of intelligent agents cooperating to solve goals. The motivation for a distributed approach to problem solving is two-fold: increased efficiency and increased capabilities. Certain tasks, such as sensing and control of air or ship traffic, have an inherently distributed nature and so lend themselves to a distributed solution [1] [2]. Even non-sensory tasks may be inherently distributed; the knowledge required to carry out these tasks might be split among several machines. Again, distributed problem solving is a natural way to proceed.

There exist two major paradigms for distributed artificial intelligence systems. The first paradigm is *planning for multiple agents*, where a single intelligent agent constructs a plan to be carried out by a group of agents, and then hands out the pieces of the plan to the relevant individuals. Randy Davis calls this paradigm "distributing the solution" [3]. The second paradigm is *distributed problem solving*, where a group of intelligent agents together construct, and possibly execute, the final plan.

This paper is concerned with the first paradigm, that of planning for multiple agents; in particular, we examine the problem of achieving synchrony among a group of agents who will be carrying out a centrally-produced plan. Imagine, for example, intelligent agents (located on various computers) that can construct and execute plans in the operating systems domain as well as communicate with each other (this is the domain being explored by the Stanford Intelligent Agents project). A user might tell the agent at Stanford that he wants a file X at MIT to be printed at CMU; the Stanford agent will construct a complete plan to accomplish this goal (containing certain actions to be taken by the MIT and CMU agents), and then tell MIT and CMU what to do. The plan might involve MIT sending the file to CMU, and CMU's printing it, but the Stanford agent must assure that these two actions occur in this order and are not reversed. One solution is for MIT to send file X to CMU, and then notify CMU that it has been sent; CMU waits for this notification, and then prints the file.

We present a method that formalizes the above solution, and thus can be used to maintain an ordering of actions performed by various agents. As a framework for the multiple-agent planning system, a formalism is adopted for representing beliefs and goals of agents, as well as their capabilities; primitives for inter-agent communication are defined. A planning heuristic for multi-agent synchrony is presented, along with its requisite operators. Finally, the example above is presented in greater detail.

## The Multi-Agent Formalism

### Beliefs and Goals

To construct plans for other agents, the planner must be able to represent and reason about their beliefs and goals. Though several alternatives are possible (such as "possible-worlds" formalisms [4] [5] [6] [7]), we choose the FACT and GOAL list formalism of Konolige and Nilsson [8]. In this approach, each agent has a FACT list that contains items that it believes, including the beliefs and goals of other agents (these last two are specified through the use of a metalanguage); the GOAL list contains the current goals of the agent. As an example, if A0 believes that A1 believes A1 has file FOO, and A0 also believes that A1 has the goal of deleting that file, the following items appear in A0's data base:

FACT(A1,'EXIST(FOO,A1)')
GOAL(A1,'DELETED(FOO)')

All planning will make use of STRIPS-like operators [9]. We allow instantiated operators to appear explicitly on any agent's GOAL list, rather than limiting this list to state descriptions. We differentiate between the two types of goals by calling the latter "operator-goals" and the former "state-goals".

### Capabilities

Previous work on multiple agents has assumed that all agents have identical capabilities, that is, that all agents have access to the identical operators. When agents are planning for differing operating system environments, this is clearly not the case. For example, an agent located on one machine may be able to run TEX on a file, while the agent on another machine that lacks TEX will not. We introduce the predicate HASCAP(agent,operator) to represent the capability of *agent* to carry out *operator*. Generality is provided by the use of partially instantiated operators in the HASCAP predicate. For example, if agent A0 believes that agent A1 can, in general, DELETE files, the following would appear in A0's data base:

HASCAP(A1,DELETE(A1,file))

We use the standard convention that the free variable "file" is universally quantified. HASCAP is also defined over more complex operator combinations; for example, the following axiom holds:

HASCAP(agent,AND(operator1,operator2)) ⇄ HASCAP(agent,operator1) ∧ HASCAP(agent,operator2)

### WILL-PERFORM as a Precondition

Cohen and Perrault [10] recognized the usefulness of making an agent's "wanting" to use an operator an explicit precondition of that operator. In this manner, one can get an agent to perform some action by making the action's preconditions true, including the precondition of making the agent "want" to carry out the action. We adopt a similar strategy recast into a more general form, and introduce the predicate WILL-PERFORM(agent,operator) to signify that *agent* will perform *operator*. WILL-PERFORM appears as an explicit precondition of all operators that do not occur spontaneously; so, for A0 to get A1 to apply the operator OP, A0 needs to make sure WILL-PERFORM(A1,OP) is true. The following axiom says that if an agent has the capability to perform an act and has the desire to perform the act, then he will perform it:

HASCAP(agent,oper) ∧ GOAL(agent,oper) ⊃ WILL-PERFORM(agent,oper)

Since the fact that this axiom is universally known is also known, the following axiom actually

appears in every agent's data base:

FACT(x,'HASCAP(agent,oper) ∧ GOAL(agent,oper) ⊃ WILL-PERFORM(agent,oper)')

This axiomatization of WILL-PERFORM models an agent's using an operator as an act of volition; if involuntary performance of acts is possible, WILL-PERFORM(agent,operator) could be made true without the agent actually possessing the operator as a goal. Other axioms would be introduced to model these cases.

An agent can apply an operator once WILL-PERFORM(agent,operator) becomes true; he will not necessarily check the truth of the operator's other preconditions or try to make them true. Given these assumptions, it is essential for the planner to assure that WILL-PERFORM's brother preconditions are true before WILL-PERFORM itself becomes true. Achieving this ordering of preconditions is identical to achieving synchrony, and will be discussed in further detail below.

## Communication Primitives

To integrate planning and communication, we need to adopt a coherent theory of planning communication acts themselves. The work of Cohen and Perrault sheds considerable light on this issue, and we use several of their communication operators (with modification) in the work that follows. For simplicity, the initiator of a communication act will be called the "speaker," and the receiver will be called the "hearer."

We use four communication operators: REQUEST, CAUSE-TO-WANT, INFORM, and CONVINCE. REQUEST and INFORM are illocutionary acts, that is, they model the speaker's communication act, but not the effect that act has on the hearer. CAUSE-TO-WANT and CONVINCE are perlocutionary acts, that is, they model the effects of communication acts. For example, the speaker might REQUEST some act of a hearer, but this will not directly cause the hearer to adopt that act as a goal; before the hearer adopts the goal, a CAUSE-TO-WANT must occur. This decoupling of the communication act from its effect allows for natural modeling of goal or fact refusal by the hearer (as contrasted with Konolige and Nilsson's single-step "asktoachieve" and "tell" operators). While Cohen and Perrault make CAUSE-TO-WANT and CONVINCE trivially triggered by REQUEST and INFORM respectively, we introduce the predicates ACCEPT and BE-SWAYED as explicit preconditions on the former operators. The communication operators are defined as follows:

REQUEST(x,y,act) -- x requests y to adopt act as a goal
P: WILL-PERFORM(x,REQUEST(x,y,act))
A: FACT(y,'GOAL(x,act)')

The effect of REQUEST is to let y know that x has "act" as a goal; x need not believe *a priori* that y can satisfy "act."

CAUSE-TO-WANT(x,y,act) -- x causes y to adopt act as a goal
P: FACT(y,'GOAL(x,act)') ∧ FACT(y,'HASCAP(y,act)') ∧
ACCEPT(x,y,act) ∧ HASCAP(y,CAUSE-TO-WANT(x,y,act))
A: GOAL(y,act)

CAUSE-TO-WANT causes y to adopt x's goal as its own, but only if y believes he has the capability to satisfy the goal and the ACCEPT predicate is true.

INFORM(x,y,prop) -- x informs y of prop
P: prop ; WILL-PERFORM(x,INFORM(x,y,prop))
A: FACT(y,'FACT(x,prop)')

INFORM should only take place if prop is true; its effect is to let y know that x believes prop.

The ";" appearing in INFORM's precondition list means that the item appearing before it should be satisfied before the item following it.

CONVINCE(x,y,prop) -- x convinces y to believe prop
P: FACT(y,'FACT(x,prop)') ∧ BE-SWAYED(x,y,prop)
∧ HASCAP(y,CONVINCE(x,y,prop))
A: FACT(y,prop)
D: FACT(y,NEGATE(prop))

CONVINCE causes y to adopt x's belief as its own, but only if BE-SWAYED is true; any contradictory belief is discarded. NEGATE is a function over strings such that NEGATE('x') gives the string '¬x'. Also, note the absence of WILL-PERFORM as a precondition of CAUSE-TO-WANT and CONVINCE; these operators will be applied when their preconditions are true, without any agent explicitly "wanting" them.

Agents' data bases contain axioms involving the ACCEPT and BE-SWAYED predicates; these axioms specify conditions under which the hearer will accept the speaker's facts or goals. For example, if agents A0 and A1 are in a master-slave relationship, we might have the following three axioms to indicate A1's subservience to A0's dictates:

MASTER(A0,A1)
MASTER(x,y) ⊃ ACCEPT(x,y,act)
MASTER(x,y) ⊃ BE-SWAYED(x,y,prop)

Other axioms might model A1's willingness to ACCEPT requests if his machine's load is low, or if he owes A0 a favor; he might BE-SWAYED by A0 if he knows A0 to be reliable, or to have particularly good information about this kind of fact (e.g. A0 will know best whether a file exists on his own machine).

## Ordered Preconditions

As explained above, the planner expects to make WILL-PERFORM(agent,operator) true in order to get *agent* to perform *operator*; once this predicate is true, the operator can be applied at any time. WILL-PERFORM will not be made true, however, until *agent* accepts the operator-goal *operator* (because of the above axiomatization of WILL-PERFORM). Thus, all other preconditions of *operator* should be true before the operator itself is adopted as a goal. Satisfaction of this principle will guarrantee multi-agent synchrony.

In general, an operator-goal should not be adopted by an agent until he *knows* that the other preconditions of the operator have been satisfied. To accomplish this, we introduce the predicates WAITING and HAS-DONE, and the operators PAUSE and WHEN-GET, defined as follows:

PAUSE(agent,precond,aim) -- agent decides to wait until precond is satisfied before adopting aim
P: WILL-PERFORM(agent,PAUSE(agent,precond,aim))
A: FACT(agent,'WAITING(precond,aim)')

WHEN-GET(agent,precond,aim) -- agent adopts aim when he knows that precond is satisfied
P: FACT(agent,'WAITING(precond,aim)') ∧ FACT(agent,precond)
∧ HASCAP(agent,WHEN-GET(agent,precond,aim))
A: GOAL(agent,aim)
D: FACT(agent,'WAITING(precond,aim)')

So, for example, to get agent A1 to wait until agent A0 has done act G before himself doing

act H, we would pass the following operator-goal to A1:

PAUSE(A1,HAS-DONE(A0,G),H)

This causes A1 to place WAITING(HAS-DONE(A0,G),H) in its data base. When A1 finds out (or more usually, is told) that A0 HAS-DONE G, WHEN-GET is triggered and A1 adopts H as a goal.

Note that the variable "precond" can actually be a conjunction of items; only when all the items are believed by the agent will WHEN-GET be triggered, since

FACT(agent,prop1) ∧ FACT(agent,prop2) ⇄ FACT(agent,AND(prop1,prop2)).

Our planner employs the following heuristic to guarantee multi-agent synchrony: assume there is an operator OP with preconditions P1 through PN (some J element subset "S" of which is not already true in the initial state), and WILL-PERFORM. The planner wants agent A0 to apply OP. Expansion of the plan on P1 through PN occurs before expansion of WILL-PERFORM; assume that the elements of S are made true by agents A1 through AJ, using operators O1 through OJ respectively. Then, instead of directly inducing OP's WILL-PERFORM operator-goal through a REQUEST and CAUSE-TO-WANT, the planner satisfies it through the PAUSE and WHEN-GET operators, whose "precond" variables are instantiated as the conjunction of J elements of the form HAS-DONE(Ai,Oi), where "i" ranges from 1 to J. Satisfaction of WHEN-GET's second FACT precondition is accomplished by INFORMs and CONVINCEs of the agents satisfying S, each of whom sends their own "HAS-DONE(Ai,Oi)" message. Finally, the planner must direct each of these agents to first apply Oi, and then inform A0 that they have done so (with a HAS-DONE message).

## An Example

A person using an Intelligent Agent at Stanford [ST] would like file REP.PRESS at MIT to be printed on the Dover printer at CMU. The agent at Stanford knows about the following two operators (in addition to the communication operators, PAUSE and WHEN-GET operators explained above):

DOVER(agent,file) -- agent prints file on the Dover
P: EXIST(file,agent) ; WILL-PERFORM(agent,DOVER(agent,file))
A: D-PRINTED(file,agent)

FTP-SEND(x,y,file) -- x sends file to y
P: EXIST(file,x) ; WILL-PERFORM(x,FTP-SEND(x,y,file))
A: EXIST(file,y)

The following items appear on the Stanford agent's FACT list (in addition to the HASCAP and WILL-PERFORM axioms listed above):

(1) FACT(x,'HASCAP(CMU,DOVER(CMU,file))')
(2) FACT(x,'HASCAP(MIT,FTP-SEND(MIT,CMU,file))')
(3) FACT(x,'HASCAP(CMU,PAUSE(CMU,precond,aim))')
(4) FACT(x,'HASCAP(CMU,WHEN-GET(CMU,precond,aim))')
(5) FACT(z,'HASCAP(x,REQUEST(x,y,act))')
(6) FACT(z,'HASCAP(y,CAUSE-TO-WANT(x,y,act))')
(7) FACT(z,'HASCAP(x,INFORM(x,y,prop))')
(8) FACT(z,'HASCAP(y,CONVINCE(x,y,prop))')
(9) EXIST(REP.PRESS,MIT)
(10) FACT(x,'MASTER(ST,CMU)')

(11) FACT(x,'MASTER(ST,MIT)')
(12) FACT(x,'BE-SWAYED(MIT,CMU,prop)')
(13) FACT(z,'MASTER(x,y) ⊃ ACCEPT(x,y,act)')
(14) FACT(z,'MASTER(x,y) ⊃ BE-SWAYED(x,y,prop)')

Axioms 1 through 8 list capabilities of the agents involved (actually, knowledge about these capabilities), with 5-8 stating that all agents have the basic communication primitives. Axioms 10 through 14 enlighten us about the hierarchy of control among the agents. Note that by the semantics of FACT, the axiom FACT(x,prop) in an agent's data base implies that prop is also in his data base (i.e. if an agent knows that everyone knows prop, then he knows prop).

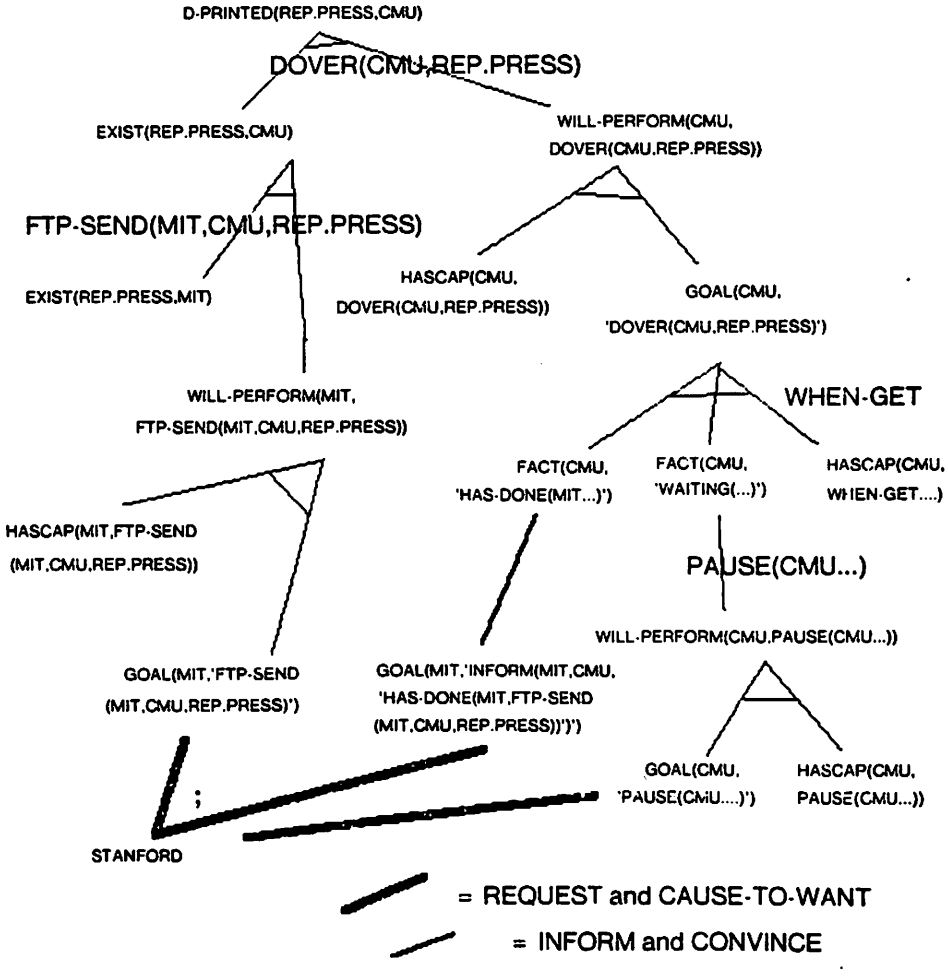
Figure 1 gives the expanded plan that ST constructs to fulfill the user's goal (the communication acts are represented schematically). It involves getting MIT to first send the file to CMU and then inform CMU that the file has been sent. In turn, CMU is told to wait until notified that MIT has carried out the FTP-SEND, and then to DOVER the file.

Construction of the plan proceeds as follows: working backwards from the D-PRINTED goal, ST chooses the DOVER operator to achieve it. Since the operator's preconditions are ordered, ST expands the first precondition (EXIST) before the second (WILL-PERFORM). The WILL-PERFORM in Figure 1's left branch does not trigger the planning heuristic, since its brother precondition "EXIST(REP.PRESS,MIT)" is true in the initial state. However, the WILL-PERFORM in the right branch does trigger the heuristic, since its brother precondition, "EXIST(REP.PRESS,CMU)" is not initially true. Thus, the goal "DOVER(CMU,REP.PRESS)" is not passed to CMU by a REQUEST and CAUSE-TO-WANT from ST. Instead, ST plans for CMU to get this goal through the PAUSE and WHEN-GET operators; both of these operators' *aim* variables are instantiated to "DOVER(CMU,REP.PRESS)", and their *precond* variables are instantiated to "HAS-DONE(MIT,FTP-SEND(MIT,CMU,REP.PRESS))". WHEN-GET's second FACT precondition is thus satisfied by a message from MIT, "HAS-DONE(MIT,FTP-SEND(MIT,CMU,REP.PRESS))". In turn, MIT is instructed to send this message to CMU after it has, in fact, done the FTP-SEND.

## Acknowledgment

The issues presented in this paper have been greatly clarified through many useful discussions with Mike Genesereth.

**FIGURE 1**


D-PRINTED(REP.PRESS,CMU)
DOVER(CMU,REP.PRESS)
EXIST(REP.PRESS,CMU)
WILL-PERFORM(CMU,
DOVER(CMU,REP.PRESS))
FTP-SEND(MIT,CMU,REP.PRESS)
EXIST(REP.PRESS,MIT)
HASCAP(CMU,
DOVER(CMU,REP.PRESS))
GOAL(CMU,
'DOVER(CMU,REP.PRESS)')
WILL-PERFORM(MIT,
FTP-SEND(MIT,CMU,REP.PRESS))
WHEN-GET
FACT(CMU,
'HAS-DONE(MIT...)')
FACT(CMU,
'WAITING(...)')
HASCAP(CMU,
WHEN-GET....)
HASCAP(MIT,FTP-SEND
(MIT,CMU,REP.PRESS))
PAUSE(CMU...)
WILL-PERFORM(CMU,PAUSE(CMU...))
GOAL(MIT,'FTP-SEND
(MIT,CMU,REP.PRESS)')
GOAL(MIT,'INFORM(MIT,CMU,
'HAS-DONE(MIT,FTP-SEND
(MIT,CMU,REP.PRESS))')')
GOAL(CMU,
'PAUSE(CMU....)')
HASCAP(CMU,
PAUSE(CMU...))
;
STANFORD
= REQUEST and CAUSE-TO-WANT
= INFORM and CONVINCE

## REFERENCES


1. Davis, R. and R. G. Smith, "Negotiation as a Metaphor for Distributed Problem Solving," Artificial Intelligence Laboratory Memo No. 624, Massachusetts Institute of Technology, Cambridge, MA (May 1981).

2. Steeb, R., S. Cammarata, F. A. Hayes-Roth, P. W. Thorndyke and R. B. Wesson, "Distributed Intelligence for Air Fleet Control," R-2728-ARPA, Rand Corporation, Santa Monica, CA (October 1981).

3. Davis, R., "A Model for Planning in a Multi-Agent Environment: Steps Toward Principles for Teamwork," A.I. Working Paper, Massachusetts Institute of Technology, Cambridge, MA (June 1981).

4. Moore, R. C., "Reasoning About Knowledge and Action," in IJCAI-5, pp. 223-227 (1977).

5. Moore, R. C., "Reasoning About Knowledge and Action," Artificial Intelligence Center Technical Note 191, SRI International, Menlo Park, California (1980).

6. Appelt, D., *Planning Natural Language Utterances to Satisfy Multiple Goals*, Ph.D. thesis, Stanford University, December 1981.

7. Appelt, D. E., "A Planner for Reasoning about Knowledge and Action," *Proc. of the First Annual Conference of the American Association for Artificial Intelligence*, Stanford, California (August 1980).

8. Konolige, K. and N. J. Nilsson, "Multiple-Agent Planning Systems," *Proc. of the First Annual Conference of the American Association for Artificial Intelligence*, Stanford, California (August 1980).

9. Nilsson, N. J., *Principles of Artificial Intelligence*, (Menlo Park: Tioga Publishing Co., 1980).

10. Cohen, P. R. and C. R. Perrault, "Elements of a Plan-Based Theory of Speech Acts," *Cognitive Science, 3* (3), pp. 177-212 (1979).