Proc. of the 8th Conference on Computer Generated Forces and Behavioral Representation, Orlando, FL, May 1999

Continuous Planning and Collaboration for Command and Control in Joint Synthetic Battlespaces

Jonathan Gratch
Randall W. Hill, Jr.
University of Southern California
Information Sciences Institute
4676 Admiralty Way, Suite 1001
Marina del Rey, CA 90292-6695
310-822-1511
{gratch,hill}@isi.edu

Keywords:
Planning, Command and Control

ABSTRACT: *In this paper we describe our efforts to model command and control entities for Joint Synthetic Battlespaces. Command agents require a broader repertoire of capabilities than is typically modeled in simulation. They must develop mission plans involving multiple subordinate units, monitor execution, dynamically modify mission plans in response to situational contingencies, collaborate with other decision makers, and deal with a host of organizational issues. We describe our approach to command agent modeling that addresses a number of these issues through its continuous and collaborative approach to mission planning.*

1. Introduction

Two of the challenges in modeling command and control (C2) entities in a Joint Synthetic Battlespace (JSB) are to provide operational realism and cost effective deployment of these entities. The lack of realism in existing synthetic C2 forces usually stems from being unable to: (1) behave in a goal-directed manner, particularly in situations not strictly covered by doctrine; (2) dynamically modify mission plans in response to a situational contingency; (3) collaborate in group decision making and behavior; and (4) vary organizational behavior by modeling differences in role, organization, and perspective among individual entities. To be cost effective, synthetic C2 forces need to: (5) behave autonomously, reducing the requirement for human control in JSB exercises; (6) represent a wide range of organizations and scenarios. In this paper we describe our efforts to support such abilities through the development of a flexible, integrated model of C2 behavior involving multiple echelons in the domain of Army Attack Helicopter Battalion operations. Here we focus on two key aspects of the behavior required to accomplish this goal: *continuous planning* and *collaborative behavior*.

Continuous Planning

To realistically model C2 behavior requires a continuous planning capability. Planning is a reasoning process that generates an ordered set of tasks from a goal description (addresses issue 1). For the purposes of modeling the C2 activities in a JSB, it is not sufficient to establish a mission plan, launch the aircraft, and wait until the aircraft return to determine success. Plans sometimes fail, situational interrupts (mission exceptions) occur, and opportunities arise, all making it necessary to re-plan the mission during the execution phase. C2 entities must continually assess the situation, monitor the execution of mission plans, and react to conditions affecting their goals and plans before and during the mission by dynamically replanning so that their goals may be achieved in new situations (issue 2). When a continuous planning capability is combined with a sufficiently deep domain theory of air operations, C2 entities become more capable of autonomous behavior (issue 5), covering a wide range of scenarios (issue 6).

Collaborative Behavior

Mission planning and execution is a collaborative enterprise (issue 3) involving C2 entities distributed across multiple echelons. Not only must these entities be capable of continuous planning, but they must also model

each other's goals (issue 1) and plans, and reason about how decisions they make will affect others. Factors such as the entity's role, perspective and the "management culture" of the organization will affect the content and form of communications among [14], and will ultimately change the overall behavior of the organization (issue 4). Without collaboration, autonomous group behavior is not possible (issue 5). Like continuous planning, collaboration requires an understanding of C2 operations domain, to include models of domain-specific C2 communications, organizations, and relationships. Given these models, it is possible to represent a range of different organizations (issue 6).

This combination of capabilities has been developed in constructing the Soar cognitive architecture [12]; intelligent forces (IFORs) for individual synthetic aircraft; and command forces (CFORs), which command organizations of individual aircraft. We have developed and demonstrated these capabilities within the domain of the Army Attack Helicopter Battalion. We will illustrate the capabilities of our system with examples from this domain.

2. Related work

Before proceeding to the technical details of our approach, it is useful to set some context by contrasting our methods with alternative simulation technology. Prior models of planning have almost exclusively focused on the problem of plan generation, without consideration of the dynamics of how the plan would be executed or modified. For example, the Defense Advanced Research Projects Agency's (DARPA) JFACC program has supported planning tools that focus primarily on plan generation and plan evaluation. In contrast, models that do account for dynamics tend to focus exclusively on reactions to the exclusion of deliberate planning. For example, the finite state models used by semi-automated forces (SAFs) are excellent for reacting to immediate threats, but require a human in the loop to ensure goal-directed behavior. Each of these extremes can be reasonable (the JFACC has many hours to generate and perfect a plan, whereas individual vehicles often must immediately react). However, modeling forces at the company and battalion level (where C2 can make the most timely use of enhanced situation monitoring platforms) requires tight integration of plan generation, plan execution/monitoring, and plan repair.

Modeling C2 nodes also brings to the forefront organizational issues that have been largely ignored in past simulation efforts. SAF models rely on human controllers to de-conflict interactions between units and to manage coordinated activities. However, as we extend simulation to higher levels of command, we must model the reasoning involved in coordination. For example, when an battalion commander adjusts a mission to the evolving situation, he must coordinate his various subordinate units and reason about the constraints that arise across these units and other friendly forces in the area of operations. Realistic models must account for the fact that the U.S. doctrine allows considerable autonomy to commanders when missions change dynamically. So during execution, a battalion commander might alter his plans without waiting for approval as long as the changes are consistent with the overall intent.

Our modeling technology resembles some of the more innovative approaches to command and control modeling. For example, SRI's CPEF system also provides a similar integrated model of plan generation, monitoring, and repair [11]. There are key differences, however, in scope and focus. CPEF is designed as a human in the loop decision support tool for staff at the JFACC. The most closely related work is the Command Entity (CE) developed by Calder *et al*. as part of DARPA's Command Forces program [2]. Like Soar/CFOR, CE is a relatively domain independent approach to C2 modeling and has been successfully applied to modeling battalion and company level commanders for army ground operations. CE is based on constraint satisfaction methods [9], a fundamentally different reasoning methodology than the planning approach underlying our technology. Planning and constraint satisfaction methods offer unique advantages and disadvantages and it is hard to characterize one as being more or less suitable to C2 modeling. The general view in the AI reasoning community is that constraint satisfaction methods are more efficient (allowing their application to larger problems) but they are not as easy to use, either in terms of incorporating domain knowledge into them or in terms of understanding their reasoning process. Recent advances in planning research suggest ways of combining these advantages, representing the domain as a planning problem but automatically translating it into a constraint satisfaction problem [13]. Although this hybrid approach is still too restrictive for the type of reasoning discussed in this paper, it suggests a long-term development scheme where we can present domain developers with the advantages of the planning perspective today, and shield them from future changes in the underlying implementation.

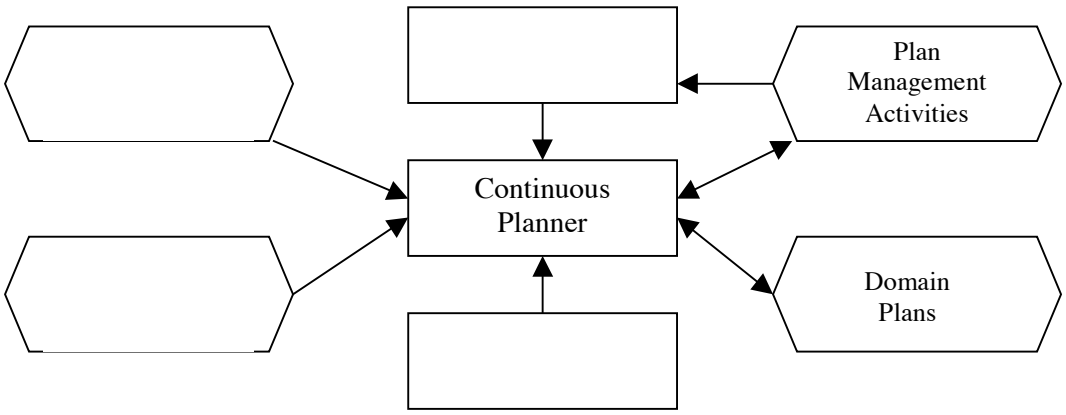

Figure 1: Soar/CFOR Architecture

3. Continuous Planning

The details of the technical approach are described by function. We first describe how Soar/CFOR addresses the issue of continuous planning. Next, we discuss the planner's support for modeling a distributed organization of C2 nodes. Next, we describe intelligent forces (IFORs). Finally we describe our approach to knowledge / software engineering. Figure 1 illustrates the basic architecture of a Soar/CFOR C2 agent.

Planning is a key capability that separates command agents from the more reactive automated forces typically modeled in simulations such as ModSAF. Planning requires predicting the outcome of events into the future and proactively deciding on appropriate courses of action.

Prior work in deliberative mission planning has tended to focus on plan generation. In contrast, researchers who have tried to embed planning agents in realistic and dynamic environments have increasingly advocated the view that plan generation, plan execution/monitoring, and plan repair are fundamentally inseparable (the 1998 AAAI workshop on Distributed Continual Planning was devoted to this integrated view of planning). Our Soar/CFOR planner builds on planning algorithms that support this integrated view: specifically, IPEM [1] and XII [4]. These planners share many features with traditional AI planning systems like SIPE [15] but interleave the plan generation process with execution and repair activities.

Plans in Soar/CFOR are hierarchically organized sequences of tasks. Each task corresponds to some process and the task description includes (1) initiation or preconditions of the process, (2) completion conditions, (3) interruption conditions, and (4) the responsible entity (who performs the process). Task conditions are used to assess the validity of generated plans and enable the planner to monitor the plan's proper execution. Tasks can be further decomposed into subtasks. Typically, this decomposition is context dependent: depending on the current or projected situation, different subtasks may be appropriate. (Different decompositions can be thought of as alternative courses of action to accomplish the high-level task.) If the planner chooses a particular decomposition, the context validating this decision recorded in the plan structure so that, if the context changes, the planner can verify whether the decomposition is still appropriate. For example, a helicopter company may choose to fly in a column (maximizing speed) because it believes there are no enemy forces in its avenue of approach. If subsequent intelligence contradicts this assumption, a C2 agent can recognize that a slower but safer formation is more appropriate.

The Soar/CFOR planner supports dynamic plan monitoring and repair by performing continuous situation monitoring. On-board sensors and situation reports are assessed by domain-specific routines to maintain a current awareness of the situation. The planner continuously compares this awareness against its current plans, and uses these comparisons as inputs into the reasoning underlying plan monitoring and repair. For example, if a battalion commander, modeled by Soar/CFOR, receives a report that one of its companies has reached the holding area, the planner recognizes that this information satisfies the completion condition of the ingress task. This in turn allows the planner to infer that the ingress has terminated and the company is now prepared to engage. In contrast, a report that the company is delayed might violate the current constraints in the plan and force some repair activities. So if the delayed flight was to engage in a coordinated attack with another company, the battalion commander might delay the second company's departure, or even cancel the entire mission.

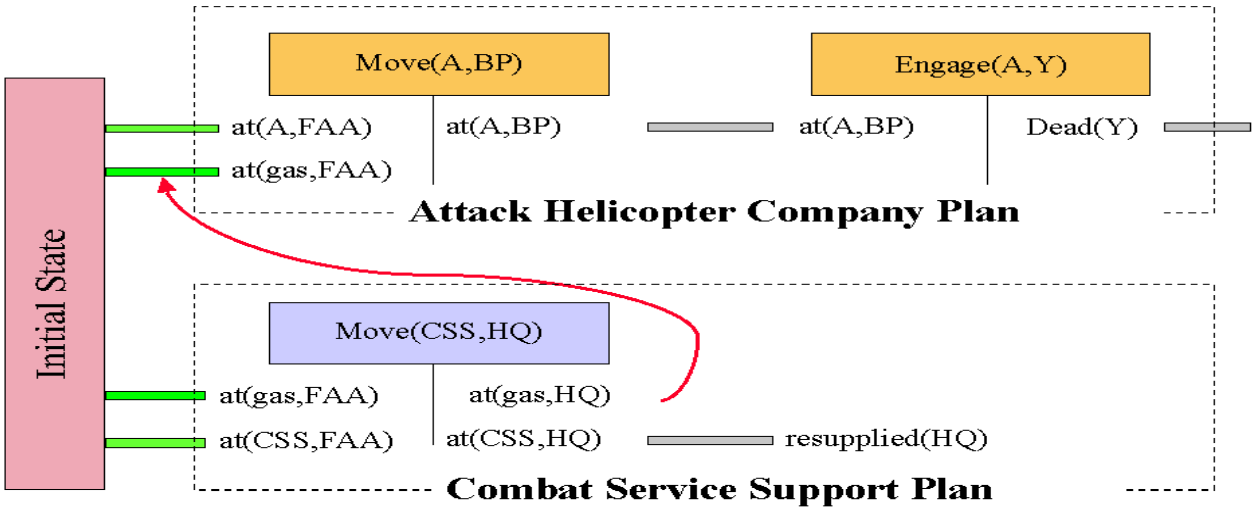

Figure 2: An example of two interacting plans

Algorithmic Details

More specifically, Soar/CFOR plans by constraint posting in the same fashion as other classical planners such as SNLP [10]. Constraints are added in response to threats in the current plan network. For example, if an action has an open precondition, the planner tries to resolve this threat by identifying an existing action that establishes the effect (*simple-establishment*) or introduce a new action (*step-addition*). Both activities add constraints to the current plan. Simple-establishment asserts a *protection constraint* that protect the effect from the moment it is created until it is used by the precondition, and binding constraints that ensure the effect unifies with the open-precondition. Step-addition posts a constraint to include the new action in addition to the constraints posted by simple-establishment. Unlike SNLP, actions have duration: they must be explicitly initiated and terminated and actions may fail. Actions can also be decomposed hierarchically.

We refer to actions as *tasks* and the set of constraints introduced by the planner as a *plan network*. Besides this network, the planner maintains a declarative representation of the perceived state of the world or *current world description* (CWD). The CWD allows the planner to monitor the execution of task and detect any surprising changes in the environment. The planner may only initiate tasks whose preconditions unify with the CWD (and are not preceded by any uninitiated tasks). Similarly, tasks are terminated when all of their effects appear in the CWD. Task initiation and termination may be interleaved with other planning operations. As the CWD reflects the perceived state of the world, it may change in ways not predicted by the current plan network. For example, some external process modifying the environment is detected by changes to the CWD not predicted by the current set of executing tasks. These changes may provide opportunities (as when an unsatisfied precondition is unexpectedly observed in the world). They may also threaten constraints in the plan network, forcing the planner to modify the task network to resolve them.

Figure 2 illustrates a set of tasks maintained in a task network. Each task has a set of preconditions (predicates listed at the bottom left of each task) and a set of effects (predicates listed at the bottom right of each task). A valid plan must ensure that each precondition is established by some effect. In the figure, horizontal bars illustrate correspond to the protection constraints. Each protection constraint represents the fact that an effect is being used to establish a precondition, and that the effect must be protected during the duration of the protection constraint. In the figure, one protection constraint is possibly violated: the fact that the CSS unit is moving the gas station threatens the protection constraint that the attack helicopter needs the gas. This is only a possible violation because the plan does not specify an ordering between the two movement tasks. If the helicopters move first, there is no problem. Planning works by identifying such potential threats and resolving them (in this case by deciding that the helicopters must depart before the gas station can be moved).

Soar/CFOR has some general plan repair operators that allow it to non-chronologically retract problematic constraints from the plan network. The CFOR planner augments this capability by incorporating a validation-structure approach to plan repair [8]. The details of this

are unimportant to understand the basic ideas in this article. The main point is that the planner has a number of operations that allow it to modify its current plans. Some of these operations add constraints to the plan network, while other operations retract constraints.

4. Collaborative Behavior

Soar/CFOR supports the modeling of distributed planning nodes in an organization. We have demonstrated this capability in the context of Army aviation planning where a battalion command agent plans collaboratively and autonomously (no human in the loop) with his subordinate attack company agents and a combat service support (logistics) agent. The Soar/CFOR planner provides domain-independent reasoning capabilities for organizational modeling and thus provides considerable leverage in modeling military command organizations.

Soar/CFOR is designed to model an organization of agents that plan in a distributed and asynchronous manner. Different organizational structures are easily represented as input to the planner: one can manipulate the number and type of elements, how they exchange information, and the authority relationships between them. The architecture also supports differing levels of autonomy between commanders and their subordinates, thereby facilitating the modeling or more or less rigid organizational structures. For example, current military doctrine specifies a relatively rigid and hierarchical distributed planning process. This doctrine is represented in Soar/CFOR as a data structure, rather than being reflected in the planning architecture, making it is relatively easy to program in alternative organizational structures.

Three novel characteristics of the planner support this reasoning. First, Soar/CFOR has the ability to maintain multiple plans in memory and reason about their interactions. Figure 2 illustrates that the planner is reasoning about two plans (an attack helicopter plan and a CSS plan), and that it can detect potential interactions between plans of different agents. This allows a command agent to not only reason about his own activities, but also represent (to some level of detail) the activities of other friendly units and the projected activities of enemy units. This provides the command agent a more coherent picture of the overall situation and allows the agent to understand the interrelationships between plans and the consequences of possible plan changes on other units. For example, consider a situation where JSTARS identifies a large movement of enemy ground forces. By modeling the activities in its area of operations, a command agent can identify which units are impacted by the new information, which units to re-task, and how these changes affect other friendly forces.

Second, Soar/CFOR maintains explicit representations of plan management activities. These are activities that provide structure to the process of planning and implement protocols for how and when distributed planning agents should exchange information. For example, the Army has spent considerable effort formalizing the planning process in what has become known as the Military Decision Making Process (MDMP). MDMP breaks planning into a sequence of tasks such as mission analysis, course of action development, course of action analysis, etc. These tasks differ from those usually considered by traditional planning systems as they refer to stages of the planning process, rather than primitive tasks an agent performs in the world. (Plan management is typically viewed as a form of meta-reasoning and has been traditionally either ignored or modeled with very different algorithms and data structures than those used in planning.) In the Soar/CFOR planner, these plan management activities are represented as an explicit plan and are modeled using the same data structures as other domain activities. The inputs and outputs of these plan management tasks, in turn, determine the flow of information between agents in the organization. The advantage of this scheme is that (1) interactions between planning agents can be programmed as easily as other domain activities, (2) they can be programmed using the same data structures, and (3) they provide a uniform medium for supporting visualization and traceability of the reasoning process.

Finally, Soar/CFOR supports the modeling of different management styles, what we refer to as *planning stances*. Specifically, a domain modeler can vary the degree to which a C2 planning agent will be cooperative or antagonistic to the activities of other agents. For example, a commander will try to support or at least de-conflict with the plans of other commanders, but will try to defeat the plans of opposition forces. Soar/CFOR supports a spectrum of such different styles. Additionally, Soar/CFOR allows a domain modeler to represent several distinctions related to authority and autonomy. For example, one can indicate who a C2 agent has the authority to command, and one can indicate whose commands must be followed to the letter and whose are open to counter proposals.

Algorithmic Details

These three characteristics—reasoning about multiple plans, maintaining plan management plans, and modeling management styles—are supported by a *plan manager* that augments the planner's basic reasoning capabilities.

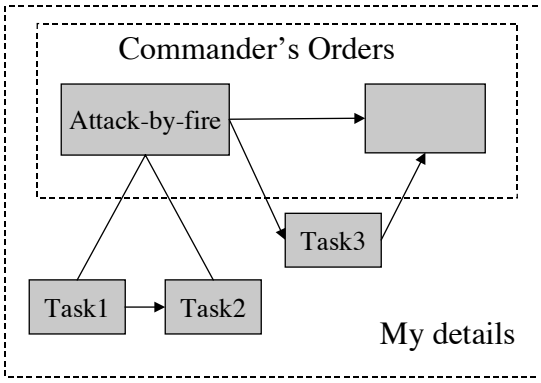

Figure 3: Modeling authority

The plan manager keeps track of the fact that different tasks in the plan network correspond to the activities of different agents. Tasks are organized into a higher-level data structure called "a plan." Plans are intended to refer to clusters of activities that are meaningful in a particular domain. In a multi-agent application, different plans most naturally refer the planner's understanding of the activities of different agents (e.g., my plans vs. my enemy's plans). The plan manager reasons about interactions between plans and can alter the way the planner behaves towards different plans in the plan network.

Planning stances are implemented by constraining the way the planner may modify different plans in the plan network. For example, by default the planner will try to resolve every perceived problem in every plan it represents. However, one may not have the authority to make changes in some units' plan, and one generally wants there to be problems in the plans of adversaries. The plan manager realizes these different behaviors by changing the control properties associated with the different plans in memory.

Figure 3 illustrates how overlapping plans can be used to model a notion of authority. In military operations, one has to accept orders from a commander. These orders must be obeyed, but one has some flexibility in fleshing out the details. A subordinate planning agent should distinguish between the part of the plan that is fixed and the part that it has the authority to alter, if for example, the plans must be repaired during the execution of the orders. This can be modeled by representing overlapping plans. One plan contains the initial orders and is deemed unmodifiable but executable through a suitable choice of plan properties. This plan is contained within a larger plan that allows modifications. Any changes made by the subordinate agent only appear in the larger plan, and the initial orders must remain unchanged.

Up to this point we've only discussed how to represent different planning stances. However a plan management strategy demands the ability to change stances dynamically as plans are generated and executed. For example, to implement the military decision-making process, an agent must take a modifying stance towards the mission plan until it has evolved to a satisfactory level. At that point, it must commit to these plans (taking an unmodifying stance), share them with the troops, and make them available for execution (taking an executing stance). If plans break down, the commander must return to a modifying stance till the plan is repaired.

Dynamic stances are modeled by allowing plan properties to be mentioned and modified by tasks in the plan network. In this way we can create explicit plan management plans that are generated and executed just as any other plan handled by the planner. The only difference is that the preconditions and effects of such "plan management tasks" refer to properties maintained by the plan manager and their execution signals the plan manager to alter the current set of plan properties. This will be illustrated in the next section.

5. Illustration

We illustrate some of capabilities of the planner by describing a partial planning trace, focusing on the plan management capabilities of the planner. The example is taken from a battalion-level helicopter attack mission which we have modeled in simulation. In this case, a battalion consists of two attack companies and a combat service support unit. Each company of helicopters consists of five vehicles modeled using Soar/IFOR and a company command agent modeled using the Soar/CFOR planner. The battalion commander and a combat service support unit are also modeled using Soar/CFOR. We illustrate the capabilities of the planner by examining the planning performed by the planning agents in the course of a typical exercise. During such an exercise, the battalion command agent receives orders from its commanding unit (a brigade commander). This consists of the goal of the battalion's mission, a partial plan for achieving it, and a list of enemy activity. The command agent generates an abstract plan to achieve the goal and sends it to the subordinate commanders, who in turn further elaborate their portion of the battalion plan (each avoiding the introduction of threats into their sibling's plan). The elaborated company plans are transmitted back to the battalion commander, who verifies there are no conflicts between the company plans (only the most rudimentary plan merging is currently implemented). If there are no problems, execution begins and each commander agent monitors the execution from its respective perspective.

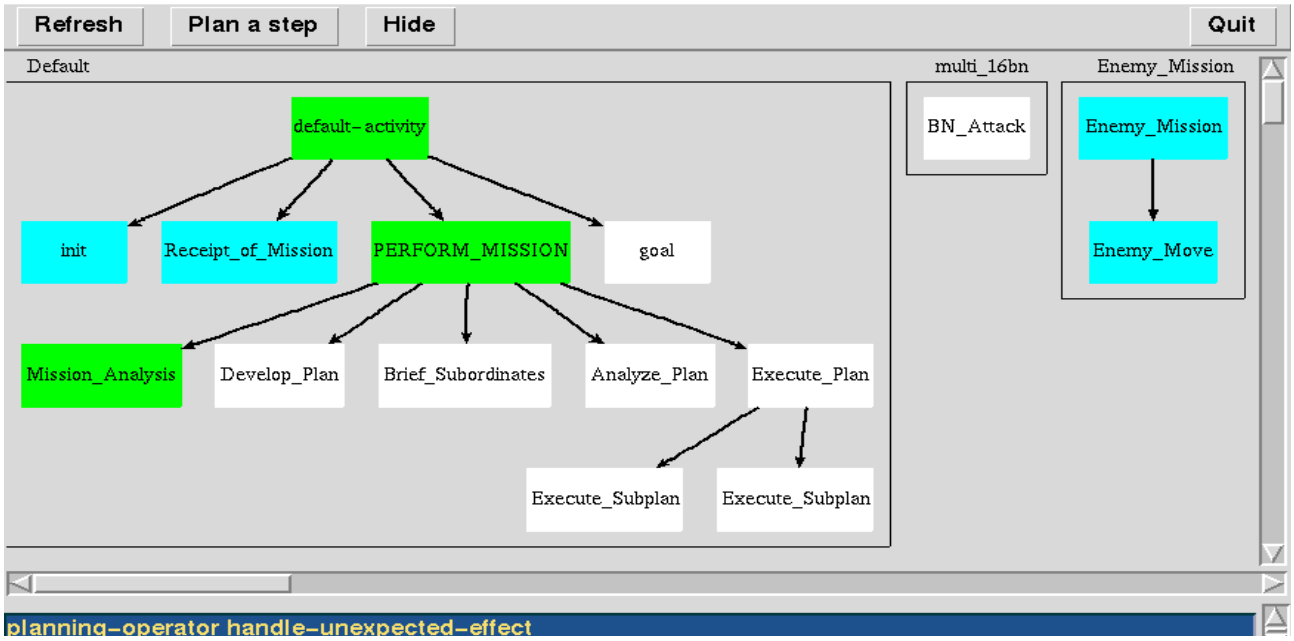

Figure 4: A battalion commander's plan network

Changes in the environment can invalidate current plans and replanning occurs in a layered fashion. Plans become more specific as one moves down the chain of command. This means a subordinate has some latitude in executing and repairing a plan while staying within the constraints mandated by their superiors. This latitude is implemented by the appropriate definitions of plan management tasks. If a plan failure exceeds the scope of this authority (as when they require modifying the partial plan given to the subordinate), the unit's commander must detect the flaw, repair the plan, and communicate the change to its subordinates.

Each commander represents several plans in a single task network: there are base-level plans for each of the agents the commander knows about. For example, a company commander will have a base-level plan for its own activities, those of its sibling company, and those of any enemies it has been informed of. Each commander also maintains a plan management plan that explicitly implements the military decision making process. Figure 4 illustrates the plan network of the battalion commander at the early stages of mission planning. There are three plans. The plan management plan is in the box to the left and is in partial stage of execution. The box in the middle is the preliminary battalion base-level plan (only a single abstract step at this point in the plan generation process). The box on the right is the current expected plans of the enemy forces related to this mission.

We will briefly describe the execution of two of the battalion's plan management tasks to give some flavor for how plan management modulate the behavior of the planner at the base level. These two task definitions are illustrated in Figure 5: Receipt_of_Mission and Develop_Plan. These are standard STRIPS [3] operator definitions with one minor difference: the *commands* field specifies the procedures that are to be executed at certain specified times during the execution of the task. Commands can occur at task initiation, termination or failure and commands may generate bindings (thereby implementing a primitive form of information gathering).

When a commander agent is sent a new mission, a domain specific rule asserts the new goal of extracting the plan contained with this order: *plan-for(?me ?order ?plan)*. This is achieved by adding a Receipt_of_Mission task to the management plan. When initiated, the task invokes a sequence of commands that create a new plan structure and populate it with the partial plan contained within the commander's order. The *disable-modification* command makes this plan initially unmodifiable (i.e., the planner is prevented from resolving flaws in this plan, local or otherwise). Modifications can be made once the planner initiates a Develop_Plan task. At the start of Develop_Plan's execution, the *enable-modification* command changes the planner's stance with respect to this base-level plan. The planner is now free to resolve flaws through its standard repertoire of plan modification methods (simple-establishment, promotion, etc.). If all goes well, all flaws will eventually be eliminated from the plan, satisfying the effect that the plan is not flawed (as well as other effects that I will not describe here). At this point, the Develop_Plan operator can be terminated, and

```
ReceiptofMission {?recipient ?sender ?order ?suborder ?plan}
  :pre      order{?sender ?recipient ?order}
  :add      suborder{?order ?recipient ?suborder}
            order{?recipient ?recipient ?suborder}
            plan-for{?recipient ?suborder ?plan}
            plan{?plan}
            plan-status{?recipient ?plan UNAPPROVED}
  :bindings {{?recipient != ?sender} {?order != ?suborder}}
  :commands
            :at-start ?plan  = create-plan{}
            :at-start ?suborder = extract-order{?recipient ?order}
            :at-start populate-plan{?plan ?suborder}
            :at-start disable-modification{?plan}
```

```
DevelopPlan{?group ?order ?plan ?lead ?second}
  :pre      plan-for{?group ?order ?plan}
            oriented-self{?group YES}
            flawed{?plan}
  :add      leadMission{?lead}
            secondaryMission{?second}
  :del      commonKn{?plan ?group}
            flawed{?plan}
  :commands
            :at-start   enable-modification{?plan}
            :at-end     disable-modification{?plan}
```

Figure 5: Definition of some plan management tasks

the *disable-modification* command is executed, locking in the changes made to this base-level plan.

Sometimes things do not go as expected. For example, perhaps after the group begins executing the plan it receives new information about enemy activity that violates some protection constraint in the base-level plan (say a location that was assumed to be safe to land is now threatened). This threat is represented as an unexpected event that asserts a *FLAWED(P)* predicate, which in turn violates the effect established by Develop_Plan (which, in fact, violates a maintenance constraint in ExecutePlan, causing this task to fail). Without plan management, the planner would immediately try to resolve the flaw in the base-level plan *P*, ignoring the other members currently executing this plan, (who may be unaware of the flaw). In contrast, as *P* is unmodifiable, the planner is prevented from resolving the flaw until it deliberately enables modifications through the management plan. The planner will deliberately add a Repair_Plan step that corrects the plan (by re-enabling and then disabling modification), and transmit the repaired plan, thereby reestablishing common knowledge of the group activities.

6. Conclusions

In this paper we described our efforts to model command and control entities for Joint Synthetic Battlespaces. Command agents require a broader repertoire of capabilities than is typically modeled in simulation. The Soar/CFOR agent architecture supports these broader requirements, allowing command agents to develop mission plans involving multiple subordinate units, monitor execution, dynamically modify mission plans in response to situational contingencies, collaborate with other decision makers, and deal with a host of organizational issues. Ongoing work seeks to extend these capabilities and demonstrate the generality of the system by modeling other command entities such as Air Force airborne command elements

7. Acknowledgements

We gratefully acknowledge the support of the Defense Advanced Research Projects Agency (DARPA) under the Advanced Simulation Technology Thrust (ASTT) program, via subcontract L74158 with the University of Michigan, under prime contract N61339-97-K-008.

8. References

[1] Ambros-Ingerson, J. A. and Steel, S. 1988. “Integrating Planning, Execution and Monitoring,” in AAAI-88.

[2] R. Calder,, R. Carreiro,, J. Panagos, G. Vrablik, B. Wise. “Architecture of a Command Forces Command Entity.” Proceedings of the Sixth Conference on Computer Generated Forces and Behavioral Representation. Orlando, FL, 1996.

[3] Fikes, R. E. and Nilsson, N. J., 1971. STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence 2 (3-4). 189-208.

[4] Golden, K., Etzioni, O., and Weld, D. 1994. “Omnipotence without Omniscience: Efficient Sensor Management for Planning

[5] J. Gratch. "Task-decomposition planning for command decision making," Proceedings of the Sixth Conference on Computer Generated Forces and Behavioral Representation, STRICOM-DMSO, July, 1996, pp. 37-45.

[6] R. Hill, J. Chen, J. Gratch, P. Rosenbloom, M. Tambe, "Soar-RWA: Planning, Teamwork, and Intelligent Behavior for Synthetic Rotary Wing Aircraft," *Proceedings of the 7th Conference on*

Computer Generated Forces & Behavioral Representation, Orlando, FL., May 12-14, 1998.

[7] R. Hill, J. Chen, J. Gratch, P. Rosenbloom, M. Tambe, "Intelligent Agents for the Synthetic Battlefield: A Company of Rotary Wing Aircraft," *Proceedings of Innovative Applications of Artificial Intelligence* (IAAI-97), Providence, RI, July 1997.

[8] Kambhampati, S. and Hendler, J. 1992. A validation-structure-based theory of plan modification and reuse. *Artificial* Intelligence 55, pp 193-258.

[9] McAllester, D. and Rosenblitt, D. 1991. "Systematic Nonlinear Planning," in AAAI-91.

[10] Myers, K. L., 1998. Towards a Framework for Continuous Planning and Execution. AAAI Fall Symposium on Distributed Continual Planning, Orlando FL.

[11] Newell, A. 1990. *Unified Theories of Cognition*. Harvard Press.

[12] Mackworth, A. The logic of constraint satisfaction. *Artificial Intelligence* 58, 1992.

[13] Selman, B. and Kautz, H. The role of domain-specific knowledge in the planning as satisfiability framework. Proceedings of the Fourth International Conferences on Artificial Intelligence Planning Systems (AIPS). Pittsburgh, PA, 1998.

[14] Rasmussen, J., Pejtersen, A., and Goodstein, L. (1994). Cognitive Systems Engineering. New York: John Wiley & Sons, Inc.

[15] D. Wilkins. Practical Planning. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1988.

Author Biography

JONATHAN GRATCH is a computer scientist at the University of Southern California Information Sciences Institute (USC-ISI) and a research assistant professor in the computer science department at USC. He completed his undergraduate education in computer science at the University of Texas at Austin in 1986. He received his Ph.D. in 1995 from the University of Illinois in Urbana Champaign. His research interests are in the areas of planning, learning and decision theory.

RANDALL W. HILL, JR. is a project leader at the University of Southern California Information Sciences Institute (USC-ISI) and a research assistant professor in the computer science department at USC. He received his B.S. degree from the United States Military Academy at West Point in 1978 and his M.S. and Ph.D. degrees in computer science from USC in 1987 and 1993, respectively. His research interests are in the areas of integrated intelligent systems, cognitive modeling, perception, and intelligent tutoring systems.