Reasoning about Multiple Plans in Dynamic Multi-agent Domains (Extended Abstract)

Jonathan Gratch

University of Southern California, Information Sciences Institute
4676 Admiralty Way, Marina del Rey, CA, 90292
gratch@isi.edu

Introduction

Classical planning algorithms such as TWEAK (Chapman, 1987) or SNLP (McAllester and Rosenblitt 1991) were developed to solve relatively simple problems in static, single-agent environments. Remarkably, recent applications have demonstrated that this basic technique, with modest extensions, can plan for large-scale, dynamic, multi-agent environments such as intelligent tutoring (Rickel and Johnson, 1997) and military planning (Wilkins and Myers, 1996; Hill *et al*., 1997). These planning "agents" go beyond the traditional focus on plan generation. These systems persist over extended time periods and demand a more dynamic approach to planning: goals change over time, plan generation must be interleaved with plan execution and plan repair. Systems that interact with other agents must distinguish between my activities, your activities, and our activities.

In contrast, a number of researchers -- coming from linguistics, multi-agent systems, and mixed-initiative planning -- argue for a much richer view of the planning process, which I will call the *intentional approach* (Bratman *et al* 1988; Cohen and Levesque, 1990; Pollack 1992; Grosz and Kraus, 1996; Ramshaw. 1991; Traum and Allen, 1994). Essentially, the intentional approach adds a layer atop standard plan representations that allows a system to reason about properties of different plans. For example, a having a plan doesn't change an agent's behavior unless it *intends to* perform the actions in the plan (Grosz and Kraus, 1996). Similarly, collaboration can be represented by allowing a planner to *intend that* other agents achieve their goals. The advantage of this additional complexity is it provides a framework for explicitly deliberate about many of the issues that arise in dynamic multi-agent environments. So, for example, Grosz and Kraus' *Shared Plan* formalism can be seen as a meta-level domain theory for collaborative planning.

Classical planning representations are clearly impoverished. However, it is unclear how to relate the intentional approach to more traditional planning methods. Intentional approaches are typically described as normative formalisms rather than implemented systems (Grosz and Kraus, 1996), or when implemented, do not utilize conventional planning algorithms (Ramshaw, 1991). This article describes a research program that aims to bridge the gap between these two formalisms. Specifically, I make the following four claims:

1) Planning agents operating in complex multi-agent environment must be able to reason about multiple plans and allow different plans to have different status (e.g. intended vs. hypothetical; executable vs. executing vs. executed; individual vs shared; individually known vs. commonly known; etc.). This means a planner must be able to represent some set of actions and constraints as "a plan" and represent and reason about its properties.

2) Many of these properties will modulate the way traditional planning algorithms behave towards the tasks and constraints in the plan. Thus, a planner will ignore threats that occur between hypothetical plans; a planner will not initiate a actions unless they belong to an executable plan; a planner will not retract a task if it has committed to performing it (unless it first retracts that commitment); etc.

3) The higher-level relations posited by intentional approaches must (and can be) re-expressed in the terminology of planning techniques, and that this will facilitate a richer understanding of these higher-level relations. For example, Grosz and Kraus introduce the meta-level predicate "intends that" to refer

to a commitment to helping another agent achieve its goals. This predicate can be given a number of different interpretations relating to how a planner handles "inter-plan interaction." For example, let's say a planner is representing two partial plans, one of its own activities and one of its collaborators. Let's further say that there can be threats between these two plans. If the planner "intends that" it's collaborator succeed, this could mean A) the planner will try to avoid introducing threats into the collaborator's plan, or B) the planner will be helpful, introducing actions or constraints into it's own plan that remove threats in the collaborator's, or C) the planner will be bossy, adding steps into the collaborator's plan (telling him what to do), or a myriad of other interpretations. The point being that each of these interpretations is a different "intentional stance" towards the plans of other agents (or towards other plans within the same agent) and we may wish to deliberate over which of these stances is appropriate.

4) The higher-level reasoning performed by intentional approaches can be implemented using traditional planning methods. In other words the higher-level domain theory can be expressed with the same representation language as the base-level theory (STRIPS operators) and reasoned with using a classical planner (i.e. metaplanning). The trick lies in providing the proper semantics for these meta-level actions such that executing them modulates the way the planner treats base-level plans.

This article sketches my approach and describes an implemented system that illustrates these ideas in the context of multi-agent military simulations.

The Planner

We briefly overview of the basic planner before describing how it can be extended to encompass some of the functionality of intentional planning approaches. While our extension could be incorporated into a variety of planning architectures, some of the details of our impact the example and discussion below.

Our planner incorporates a number of novel features, but for the purposes of this article it may be considered equivalent to IPEM (Ambros-Ingerson and Steel 1988), which shares many features with XII (Golden *et al*. 1994). Some differences between our approach and IPEM are briefly considered later in the article. IPEM was designed to support planning, execution and replanning for environments where actions have duration and the world can change in surprising ways.

IPEM plans by constraint posting in the same fashion as other classical planners such as SNLP (McAllester and Rosenblitt 1991). Constraints are added in response to threats to the current set of activities. For example, if an action has an open precondition, the planner attempts to resolve this threat by identifying an existing action that establishes the effect (*simple-establishment*) or introduce a new action (*step addition*). Both modifications add constraints. 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. Actions can also be decomposed hierarchically.

Following the terminology of hierarchical-task decomposition planning (*REF*), I will refer to actions as *tasks* and the set of constraints introduced by the planner as a *task network*. Besides this task 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 task 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.

Metaplanning

We extend the planning algorithm in several ways: (1) allow multiple plans to be represented in a single task network (the notion of a plan will be defined below); (2) define a set of higher-level relations over plans (to support intentional reasoning); (3) allow plans to modify the relationships in which plans participate (meta-planning).

To support these capabilities, we introduce the notion of a *plan designator*, a symbol that denotes the portion of the task corresponding to a plan. Plan designators can be used just as any other constant when defining a domain theory. They may be used as terms in the preconditions and effects of tasks. For example, if *P* is a plan designator, we may define a predicate *FLAWED* over plans and use *FLAWED(P)* as a precondition to some task in our domain theory. The difference between this and a statement such as *ON(A,B)* is that while *ON(A,B)* is typically interpreted as a relation among objects in an external environment, *FLAWED(P)* is to be interpreted as a relation over portions of the planner's task network (i.e. the subnetwork denoted by *P* participates in some flaw).

Plans and Plan designator semantics

A plan corresponds to a subset of the task network and a plan designator is the symbol that denotes this subset. We have a procedural semantics for determining this denotation that is closely tied to the operations a planner can use to make changes to the task network. A task network can be viewed as a set of constraints (task T1 is constrained to be in the plan, the fact F is protected from T3 to T4, etc.). For a given task network, each of these constraints belongs to exactly one plan (i.e., plans form a partition of the set of constraints comprising the task network).[1] Whenever new constraints are inserted into the task network, they are simultaneously added to a particular plan.

Deciding the goals a plan is to solve is necessarily domain-specific. Therefore, initial partial plans are created (along with the corresponding designation) as result of executing domain-specific procedures that can be associated with tasks. This "seed" plan will consists of constraints needed to represent a *goal* task and possibly a partial plan for achieving its preconditions. For example, in the military simulation domain, an agent can be commanded to achieve a goal. The radio message containing the command is translated into a partial plan using one of these domain-specific procedures (see below).

New constraints are added to this "seed" as the planner expands it. Step addition inserts a task into the network in order to establish some open precondition of another task (the establishee). Any constraints added as a result of this modification become part of the establishee's plan (more precisely, they becomes part of the plan containing the constraint that asserts the establishee). Similarly, simple establishment introduces protection and binding constraints into the network that becomes part of the establishee's plan. Task decomposition, which breaks an abstract task into a partially ordered sequence of subtasks, generates a set of constraints that become part of the abstract task's plan. The planner has three modifications for resolving threats to protection constraints (promotion, demotion, and separation). Constraints added by these modifications are added to the plan which contains the threatened protection constraint.

Intentions

Intentional planning approaches posit a set of higher-level "intentions" to support their reasoning about the planning process (e.g., intends-to, joint commitment, etc.). In my model, these intentions are treated as *meta-relations,* which are just relations containing plans (as defined above) and possibly other domain constructs represented by the planner (such as symbols denoting groups of individuals). For example, to represent group plans (plans that are to be executed by a group of individuals) the planner must possess a meta-relations that associates plans with groups.

To support metaplanning, we must have some way to express these meta-relations (intentions) using STRIPS operators. Just as a plan designator is a symbol denoting a plan, *meta-predicates* (predicates involving plan designators) are declarative representations of meta-relations. For example, *PLAN(P)* is a declarative statement that *P* is a plan and it may be used in preconditions and effects of tasks.

Finally, *meta-tasks* are tasks involving plans (i.e., one or more of their preconditions or effects is a meta-predicate). It is through meta-tasks that the planner manipulates meta-relations, and when defining meta-tasks, one must also define how its execution modifies the meta-relations of the plans it involves

---

[1] In our current work we are generalizing the notion of a plan designator to allow a given task to belong to multiple plans (see below).

(as illustrated in the next section). Meta-predicates are used to declare the preconditions and effects of these changes, allowing the planner to introspect on and control the planning process.

We emphasize this distinction between meta-predicates and meta-relations because one may not wish to declaratively represent all the meta-relations maintained by the planner (one reason for this could be efficiency concerns). For example, an ExecutePlan task might make a plan executable upon its initiation and make it unexecutable if the task fails, but we might choose not to explicitly represent the executability meta-relation as a meta-predicate.

To implement meta-predicates and meta-tasks, one must alter the planner to reflect the correspondence between meta-predicates and the underlying meta-relations they reflect. This correspondence is formed through the meaning of predicates on current world description. That is, one must provide a mechanism that assigns truth values to any meta-predicates in the CWD. For normal predicates such as *ON(A,B)*, this truth value is assigned by sensing the environment. With meta-predicates, we are essentially making the plan network "part of the environment," much as in the PRS planner (Georgeff and Lansky, 1987). That is, we must provide mechanisms for (1) examining the current meta-relations and assign truth values to corresponding predicates, and (2) receiving the commands of meta-tasks and making the appropriate change in the current meta-relations.

As there is a close tie between meta-relations and the planning architecture, we describe in detail a core set of "planner-specific" meta-relations that we have incorporated into our planner. We make no claims to the suitability of this set, but we have found them useful for implementing multi-agent domains, and they serve to illustrate the type of reasoning we wish to support.

Modifiability: Normally, the planner automatically modifies the plan in response to any flaws that may arise. Under some circumstances, one may wish to override this automatic behavior. For example, Cohen and Levesque introduce the notion of joint commitment to lock in a course of action until certain criteria are satisfied. The modifiability meta-relation allows the planner to engage in this deliberate committing and uncommitting to a course of action.

A plan is considered modifiable by default. If a plan is made unmodifiable then the planner is prevented from adding new constraints to the plan or removing existing constraints. Thus, if a task belonging to an unmodifiable plan has an open precondition flaw, the planner is prevented from resolving it (via simple establishment or step addition) as would normally occur. In contrast, if a task in an unmodifiable plan clobbers a task in another modifiable plan, the planner may still add constraints to the modifiable plan to resolve the conflict.

Flaws: A key property of plans is whether they contain any flaws. For example, one may be reluctant to execute a plan that has inconsistencies or missing steps. The flaw meta-relation and corresponding meta-predicate facilitates this type of reasoning. A plan is considered flawed if any of its constraints participate in a flaw.

Conjectures: Often, one would like to consider alternative plans to accomplish a goal before actually committing to a particular course of action. We use the conjecture meta-relation to support this type of reasoning. In particular, if we are considering two conjectured plans for achieving the same goal, we shouldn't allow constraints in one plan to clobber constraints in the other, as could happen since both plans are represented in the same network. If a plan is conjectured, the planner only checks for flaws between constraints within the plan or with constraints of other non-conjectured plans. Such a notion is key to support the type of exploration reasoning considered in some intentional approaches (Ramshaw, 1991; Ferguson, *et al.,* 1996).

Executability: Simply because a planner has created a plan doesn't mean it should execute it immediately, especially in a multi-agent setting where activities may require coordination. Using the executability meta-relation we can bring the execution of plans under deliberate control. If a plan is unexecutable, the planner is prevented from initiating the executing of any tasks in the plan. Tasks which have already been initiated will still be terminated if their effects are observed, but no new task may begin

Common Knowledge: Multi-agent planning requires the ability to communicate plans to other agents. The common knowledge meta-relation keeps track of to whom the plan has been communicated to.

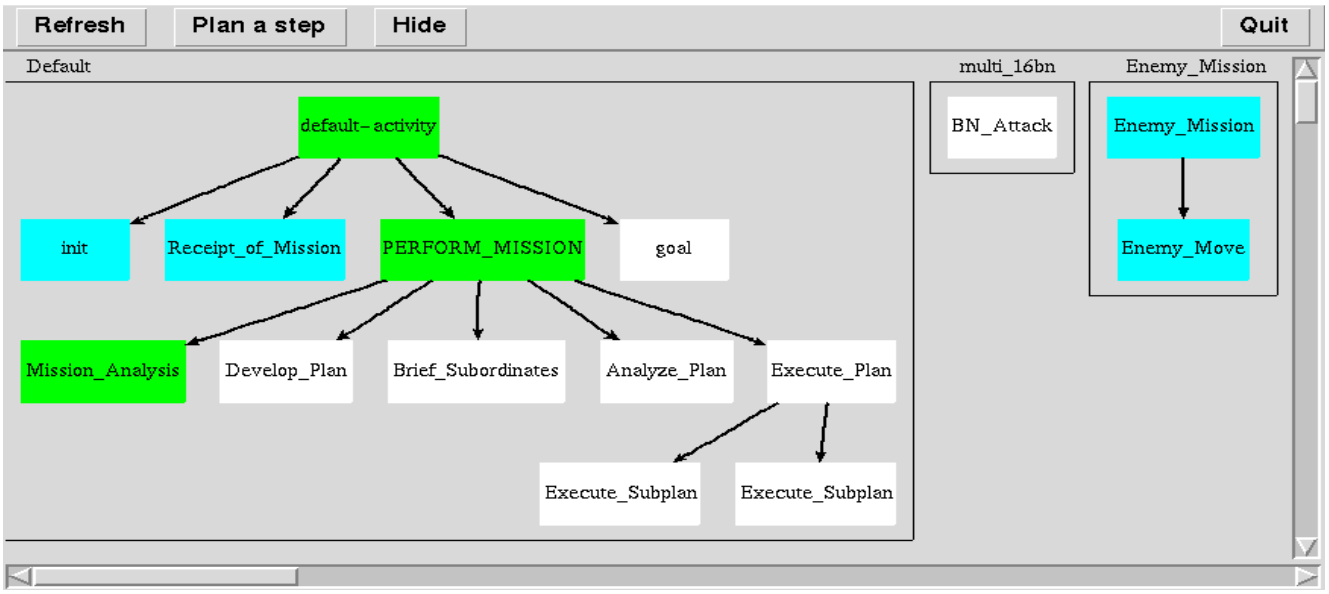

Figure 1: Battalion commander task network

Example

We illustrate the approach by working through an example of how metaplanning works in an application domain. The approach has been tested in the context of large-scale military simulations (Hill *et. al.* 1997) and we use this domain as a basis for discussion. Our system participated in a simulated military exercises known as Synthetic Theater of War '97 (Stow-97). This exercise involved about 5000 independently controlled simulated vehicles (planes, ships, tanks, soldiers, etc.) operating in a 700 by 500 kilometer virtual world and run across a distributed network of several hundred computers. The goal of this exercise was twofold: to support training for the United States Atlantic Command, and to demonstrate advanced simulation technology. This latter goal required generating appropriate and believable behavior, as judged by a group of "subject matter experts" (retired military personnel). Our contribution to this exercise was software for controlling several companies of helicopter entities that engaged in multiple battles against simulated ground vehicles during the course of the 48 hour exercise. Our participation was deemed a success by the standards imposed by the Stow-97 exercise.

In this domain, each company of helicopters consists of five vehicles that coordinate their activities as a group, and a commander agent that performs all high-level planning, replanning, and execution monitoring. The individual vehicle agents execute the commander's high-level plans, making appropriate reactions based on the current state of the environment (Tambe 1997). The commander agent is based on the planning architecture described above, whereas the vehicle agents can be viewed as RAP-like execution systems (Firby 1987) supplemented with Tambe's teamwork and agent tracking techniques. Most recently, we have extended the approach higher in the army command hierarchy, introducing a battalion commander agent that collaborates with its company commander agents to plan and execute a battalion deep attack mission.

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 communicates it to the two subordinate company 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 performed). If there are no problems, execution commences and each commander agent monitors the plan execution from its respective perspective.

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: A subordinate unit will have

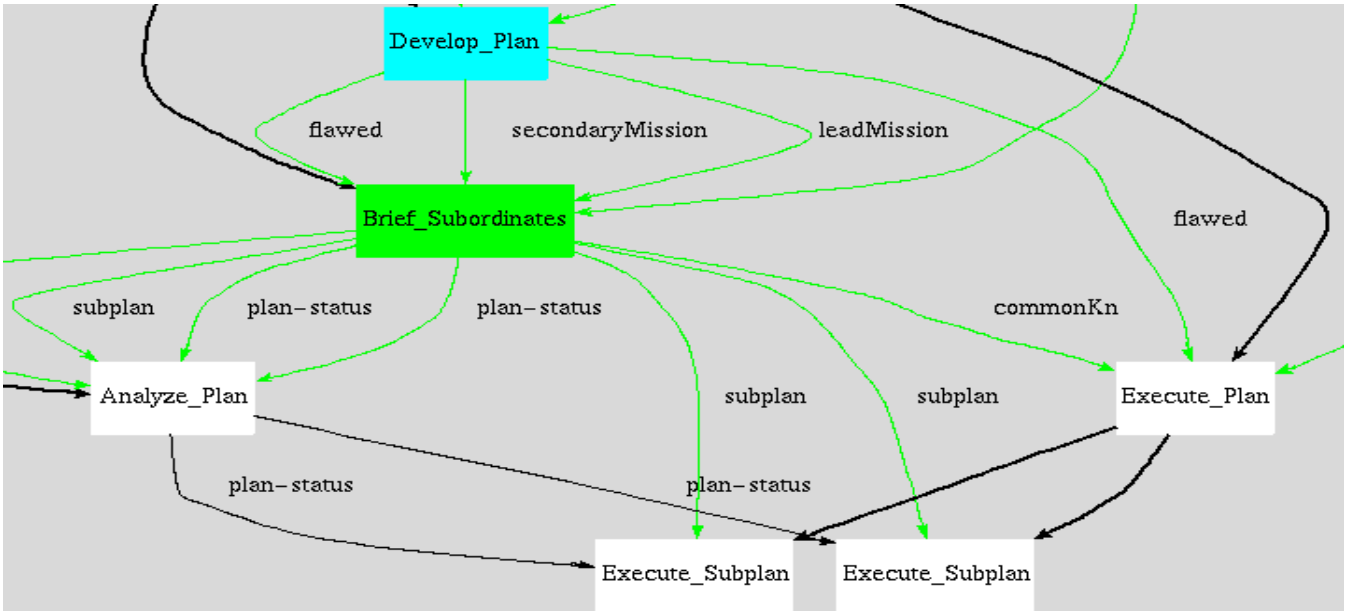

Figure 2: Some meta-task definitions

more detailed plans than it was ordered to perform. This means a subordinate has some latitude in executing and repairing plan while staying within the constraints mandated by their superiors. However, some plan failures may go beyond the scope (as when the require modifying the partial plan given to the subordinate). In these cases, 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 meta-level plan for establishing and modifying intentions on the base-level plans. Figure 1 illustrates the task network of the battalion commander at the early stages of mission planning. There are three plans. The meta-level 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. Figure 2 shows an expansion of part of this network illustrating the causal links between primitive tasks.

I don't have space to illustrate the planning process in any detail, however I will briefly describe the execution of two tasks in the meta-level battalion plan. This will give some flavor for how the meta-level establishes intentions that modulate the behavior of the planner at the base level. Figure 3 illustrates the definition of these two tasks: Receipt_of_Mission and Develop_Plan. These are standard STRIPS 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 run-time variables).

Whenever 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 satisfied by adding an instantiation of Receipt_of_Mission to the meta-level 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. Modifications can be made once the planner incorporates a Develop_Plan task into its network and begins its execution. At the start of Develop_Plan's execution, the *enable-modification* command changes the planner's intentional stance with respect to this base-level plan. The planner is now free to resolve flaws through its standard repertoire of 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

```
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 ?m1
?m2}
  :pre    plan-for{?group ?order ?plan}
          oriented-self{?group YES}
          flawed{?plan}
  :add    leadMission{?lead ?m1}
          secondaryMission{?second ?m2}
  :del    commonKn{?plan ?group}
          flawed{?plan}
  :commands
          :at-start   enable-modification{?plan}
           :at-end    disable-modification{?plan}
```

Figure 3: Some task definitions

not describe here). At this point, the Develop_Plan operator can be terminated, and 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 on the CWD by the reappearance of the *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 meta-reasoning, 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 immediately resolving the flaw. It can only modify the plan by deliberately enabling modifications at the meta-level. In the application, 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.

Issues and Conclusion

As the example illustrates, plan designators and meta-relations provide the planner an ability to deliberate over the process of planning as suggested by the intentional planning. A planning sytem can represent multiple plans, assign them "intentions" such as modifiability and executibility, and relate them to other domain constructs. Plans may be associated with groups of agents and represent the notion of common knowledge, relationships that are essential for coordinated behavior. An important advance is this intentional approach occurs within context of well-understood classical planning techniques.

This is an evolving project and there are important issues we have not adequately addressed. For example, IPEM supports fairly flexible *individual* replanning, however *collaborative* replanning requires some language for talking about the planning process with other agents (Ramshaw, 1991). In the current implementation, commanders recognize interactions between their subordinate units' plans, but have no means for telling the other agents about them in any detail. The only mechanism for collaborative replanning is for a commander to tell its subordinates to terminate their current plans and accept new orders. Similarly, there is also no mechanism for communicating about planning failures. For example, if a subordinate is unable complete a partial plan given by its commander, it cannot provide any useful feedback on why it failed. The connections between this work and intentional planning approaches also needs further elaboration. For example, which of the meta-level constructs suggested in the multi-agent literature can be represented using our technique? Are there some that require further representational extensions?

Finally, although our metaplanning approach could be incorporated into a variety of classical planners, it may be that different planning techniques are more or less conducive to its successful implementation. One key issue is how the planner searches the space of plans. Maintaining multiple (roughly independent) plans in the same network may favor different search methods than those commonly used. A novel feature of our planner is that it generates plans via local iterative repair search (Kautz and Selman 1996), rather than conventional systematic backtracking search of IPEM and other

classical planners. We believe that iterative repair search is more appropriate for planners that perform replanning and it also interacts well with the notion of multiple plans, but this conjecture needs further evaluation.

References

Ambros-Ingerson, J. A. and Steel, S. 1988. "Integrating Planning, Execution and Monitoring," in AAAI-88.

Bratman, M. E., Israel, D. J., and Pollack, M. E., 1988. "Plans and resource-bounded practical reasoning," *Computational Intelligence, 4,* pp 349-355.

Chapman, D. 1987. "Planning for conjunctive goals," *Artificial Intelligence, 32*, pp. 333-377.

Cohen, P. and Levesque, H., 1990. "Intention is choice with commitment," *Artificial Intelligence, 42(3)*.

Firby, J. 1987. "An investigation into reactive planning in complex domains," In AAAI-87.

Georgeff, M. P., and Lansky, A. L. 1987. "Reactive reasoning and planning," in AAAI-87, pp. 677-682.

Golden, K., Etzioni, O., and Weld, D. 1994. "Omnipotence without Omniscience: Efficient Sensor Management for Planning." in AAAI-94, Seattle, WA.

Grosz, B., and Kraus, S. 1996. "Collaborative Plans for Complex Group Action," Artificial Intelligence, 86(2).

Hill, R., Chen, J., Gratch, G., Rosenbloom, P., and Tambe, M. 1997. "Intelligent Agents for the Synthetic Battlefield," in AAAI-97/IAAI-97, pp. 1006-1012.

Kautz, H. and Selman, B. 1996. "Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search," in AAAI-96, Portland, OR, pp. 1194-1201.

Knoblock, C. 1995. "Planning, executing, sensing, and replanning for information gathering," in *Proceedings of IJCAI-95*, Montreal, pp. 1686-1693.

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

Pollack, M.E., 1992. "The uses of plans," *Artificial Intelligence, 57(1)*, pp 43-68.

Ramshaw, L. A., 1991. "A three-level model for plan exploration," in Proceedings of ACL91.

Rickel, J. and Johnson, L. 1997. Intelligent tutoring in virtual reality," in *Proc. of World Conference on AI in Education*.

Traum, D. R. and Allen, J. F., 1994. "Towards a formal theory of repair in plan execution and plan recognition," in *Procedings of UK planning and scheduling special interest group*.

Tambe, M. 1997. "Agent Architectures for Flexible, Practical Teamwork," in AAAI-97, pp. 22-28.

Wilensky, R. 1980. "Meta-Planning," Proceedings AAAI-80, Stanford, CA, pp. 334-336.

Wilkins D. E. and Myers, K. L. 1996. "Asynchronous dynamic replanning in a multiagent planning architecture," in *Advanced Planning Technology,* Austin Tate (ed.), AAAI Press.