Appears in the *proceedings of the 3rd international joint conference on autonomous agents and multi agent systems*

A Planner-Independent Collaborative Planning Assistant

Hyeok-Soo Kim Jonathan Gratch
Institute for Creative Technologies
University of Southern California
13274 Fiji Way, Marina del Rey, CA, 90292, USA
kimhs@ict.usc.edu gratch@ict.usc.edu

Abstract

This article introduces a novel approach to the problem of collaborative planning. We present a method that takes classical one-shot planning techniques - that take a fixed set of goals, initial state, and a domain theory - and adapts them to support the incremental, hierarchical and exploratory nature of collaborative planning that occurs between human planners, and that multi-agent planning systems attempt to support. This approach is planner-independent - in that it could be applied to any classical planning technique - and recasts the problem of collaborative planning as a search through a space of possible inputs to a classical planning system. This article outlines the technique and describes its application to the Mission Rehearsal Exercise, a multi-agent training system.

1. Introduction

This paper presents techniques to enhance the generality and modularity of collaborative (e.g., mixed-initiative) planning systems. By allowing human users and planning systems to jointly develop plans, such systems increase users' trust in the end product as well as complement the intuitive planning skills of human experts with the superior bookkeeping capabilities of automated planning systems. Indeed, collaborative planning systems have been applied to a number of significant applications including disaster planning [1], and tutoring systems for teaching people how to plan and make decisions [14]. These systems differ in terms of who has authority or initiative, what knowledge is available to the agent and the high-level communicative goals of the system (e.g., tutoring goals vs. task goals) [7]. But they share a great deal, in particular the need to plan, including the ability to generate plans, provide feedbacks on the feasibility of different options, monitor their execution, and replan in response to unexpected events or user interventions.

These are large and complex systems that tightly integrate a number of capabilities. Beyond plan generation, they communicate with the user, often through natural language, model aspects of the user's mental state, recognize user intentions, execute plans and monitor the external environment. Further, users often demand flexible control over the planning process, at times micromanaging and at other times expecting the system to handle all the details. Facilitating this close and varying level of control over the planning process complicates the problem of cleanly separating the communication module from the planning process. For example, rather than simply accepting a goal and initial state, the planner must support a wide range of possible inputs. This lack of modularity limits the generality of these methods and contributes to the obsolescence of their component technologies. This is most obvious with regard to the planning techniques that underlie these systems, arguably their most essential component technology.

Due to this tight connection between planning and communication components, most planning techniques underlying collaborative planning systems are antiquated or rudimentary. Planning techniques are, on the other hand, evolving rapidly. A quick review of the recent AI Planning system competitions illustrates that the top performing planners are in constant flux. For example, in 1998, IPP was the winner of the ADL track and also showed good performance in the STRIPS track, and HSP solved the most problems in the STRIPS track [12]. In 2000, IPP was replaced by FF (faster but total order) [4]. In 2002, MIPS solved the highest number of problems in the fully automated track, and was the only system that produced solutions in each track, while FF also out-performed its competitors in the numeric and STRIPS domains [11]. To our knowledge, none of these techniques have been incorporated into state-of-the-art collaborative planning systems.

Tying a collaborative planner to a specific planning algorithm can also significantly limit its generality. The developers of planning systems point to the *domain -independence* of their planning techniques as an argument for their wide applicability. There is a strong reason to doubt this claim. Planning performance varies dramatically across application domains. Different planning systems excel on different domains and some, supposedly domain-independent, planners cannot even represent distinctions essential for certain domains. For example, at the AI planning system competition in 2002, FF planner exhibited outstanding performance against its competitors in most of the Numeric and STRIPS problems, but it didn't compete in the temporal domains. On the other hand, TALPlanner out-performed its competitors in the temporal domains, but didn't participate in the numeric domains [11].

We argue that *planner-independence* is a far more crucial design goal than domain-independence in the design of collaborative planning systems, though is not immediately obvious how to achieve this goal. A planner-independent system embodies the design philosophy that the planner should be a modular component that can be easily replaced depending on the characteristics of the application or as improved techniques become available. Unfortunately, due to close input a user has over the planning process, planning and user-interface modules are tightly intertwined in collaborative planning systems. In contrast, the typical interface to conventional planning system is at the level of specifying a domain theory, initial state and a set of goals, which is really an inappropriate level to separate planning and communicative functions.

The question we address in this paper is what is the right level of abstraction (what is the right API) to separate the planning system from other modules in a collaborative planning system, specifically focusing on the connection to the user-interaction module, which tends to have the tightest interaction. Our approach is to consider the generic planning services necessary to support collaboration, define a level of abstraction in terms of these basic services, and map between these services and the low-level API of planning systems in a planner-independent fashion.

Section 2 lays out a high-level description of collaborative planning systems and section 3 elaborates our basic approach and describes how we map from the basic planning services necessary for collaboration to low-level calls to a conventional planning system. We conclude by discussing evaluation and direction for future research in section 4 and 5.

2. Collaborative Planning Systems

Collaborative planning systems allow a human user and an intelligent system to closely interact during the process of plan generation, execution, and repair. Such systems have been explored in the research community under a variety of titles including mixed-initiative planning systems, human interactive planning systems, planning systems with adjustable autonomy, and tutoring systems. They differ in many respects but they share two tightly intertwined capabilities: they must communicate with the user about the planning processes, and they must be able to develop and reason about plans. Here we review this work in terms of terminology we will use in this article.

2.1 Communication

The interaction between a user and a system can be seen as a dialogue, and researchers in this area have been heavily influenced by linguistic theories, regardless of whether the system actually communicates via natural language or through a more stylized interface. In such systems, the interaction is controlled by a *dialogue manager* and the content of the interaction is frequently characterized in terms of *speech acts* [3], a formulism for characterizing and classifying natural language utterances. For example, a request like "where is the helicopter" is represented as an "information request" about some attribute of a domain entity. Speech acts have a certain structure and impose certain communicative obligations on the listener. For example, upon receiving an information request, the system is obligated to respond to the request with some assertion. Standard speech acts include *inform, order, request, accept, reject*, and *counter-propose*. Depending on the system's capabilities, these are frequently augmented with *dialogue acts* that manage turn-taking and shifts in initiative between the system and the user [17].

Speech acts provide an abstract structure, but by analyzing human-to-human collaborative planning dialogues, collaborative planning researchers have also classified general features of the content of these conversations. Human interactive planning dialogues revolve around aspects of planning process, including the development and refinement of plans, the evaluation and comparison of alternatives, the clarification of features of the environment, the identification of plan problems or threats, and the clarification of aspects of the planning dialogues. By combining these aspects with speech acts, these systems can classify a range of utterances. For example, a user might inform the system of a course of actions, order the system to adopt it, request the system to develop an alternative, accept the alternative, order its execution, etc.

2.2 Planning and Planning Service Requests

Speech act theory facilitates understanding, but to service such speech acts, the system must also support a range of plan reasoning capabilities. The classic definition of the planning problem is simple to generate a satisfying plan given a set of goals and initial state. However, researchers in collaborative planning recognize this definition is far too restrictive. In terms of plan generation, users frequently demand tighter and incremental control over the planning process. As in many real-world collaborative planning domains, plans are typically hierarchical and users interact with the system to refine their plans, explore or get advice about different courses of action, and receive assistance in initiating and monitoring the execution of the plan. If a user requests a course of actions, the system must be able to develop one. If the user requests a comparison of alternatives, the system must have the means to provide it. If the system wishes to take initiative this must be motivated by some inferences concerning the planning process.

Consider the following hypothetical interchange from the Mission Rehearsal Exercise (MRE), a multi-agent training system [14]:

USER: How can I reinforce the platoon in downtown Celic?

SYSTEM: There are two feasible options. Send two squads forward or send one squad to secure our route and speed our subsequent movement.
USER: What is the disadvantage of sending two squads?
SYSTEM: It will fracture our forces and limit our future options.
USER: Send first squad to secure the route.
SYSTEM: Sir. First squad needs to secure the landing zone. I suggest we send forth squad.
USER: We don't need to secure the landing zone anymore. Send first squad.
…

This example illustrates several planning capabilities that aren't obviously mapped to traditional planning problems. The planning process is hierarchical and incremental. Rather than generating a complete plan, the system proposes single step refinements. Rather than generating a single satisfying plan, multiple qualitatively different options are explored in parallel (e.g., sending one versus two squads). Rather than receiving a simple list of plan steps, the user may ask for evaluative information (e.g., what is the disadvantage), and the system may take the initiative to offer advice (e.g., first squad is preoccupied). Finally, rather than accepting a fixed goal state, planning requirements may be changed in mid-stream (e.g., as when the user drops the goal that the landing zone must be secure). Collaborative planners typically also incorporate functions to execute plan steps, monitor their execution, and detect when plan repairs are necessary.

We use the term *planning service requests* to refer to the collection of planning capabilities like the above that are necessary to address a user's plan-related speech acts and to inform the systems inferences about dialogue initiative. Collectively, planning service requests define an API that the underlying plan reasoning system must support.

3. Planner-Independent Collaborative Planning Assistant (PICOPA)

Figure 1 illustrates our view of how to impose greater modularity between the planning and the communicative modules in a collaborative planning system. The key idea is to define a set of abstract planning service requests that can serve as a bridge between the dialogue manager and a traditional planning system. Under this view, the question we must address is how to provide a general and planner -independent mapping between these service requests and the capabilities of planning systems.

Our approach is motivated by the "in practice" efficiency of recent planning techniques within the context of collaborative planning domains. Although planning in general is hard (indeed, undecidable), for suitably constrained applications recent planning techniques can, in practice, "solve" the planning problem. This is particularly true for the relatively constrained propositional domain theories used by many collaborative planning systems.

Our admittedly strong assumption in this paper is that planning is decidable and relatively efficient in practice, though we discuss how to relax this assumption later in the paper. If plans can be generated in milliseconds, however, it opens up new possibilities for collaborative planning. One could imagine repeatedly calling a planner to solve variations of a planning problem to explore features of a domain. For example, if the user wanted to consider the differences between evacuating injured personnel with a helicopter versus an ambulance, one could simply solve the planning problem twice - once with the domain theory only with the helicopter-related actions and once only with the ambulance-related actions - and summarize the outcomes to the user.

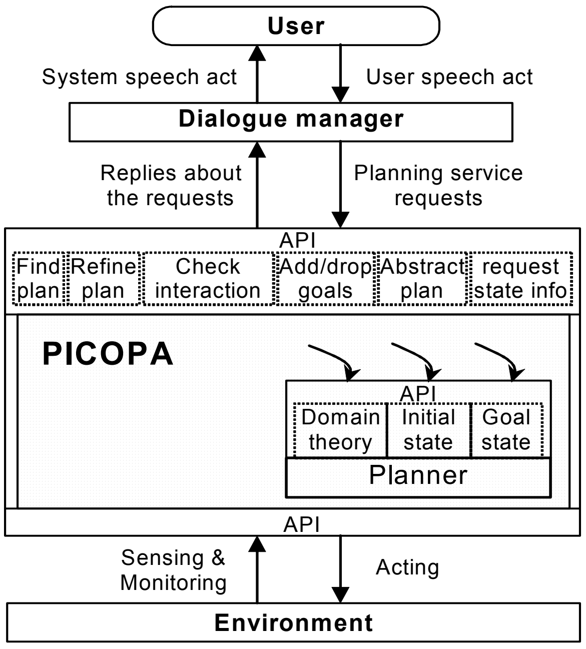

Figure 1: PICOPA architecture

We build on this observation, showing how a more flexible repertoire of planning service requests can be constructed by solving a set of suitably varied planning problems. By systematically varying a traditional planner's domain theory, initial state and goal state, the system can, by brute force, provide planner-independent mappings between the planning service requests demanded by collaborative planning systems and the capabilities of traditional planners. The planning system itself can be treated as a black box and alternative planners could be incorporated, assuming they all take as input an initial and goal state description and a domain theory consisting of a set of primitive actions.

In the remaining of this section, we describe how PICOPA maps high-level planning service requests (e.g., refine the current plan or check interaction) onto a sequence of traditional planning problems. To handle such requests, the system needs a mechanism that maps them into appropriate inputs for a classical planning system and also provides the dialogue manager interpretable feedback from the result of the planning system. First we introduce new representational constructs, *hierarchical action sets* and *the current plan set*, to facilitate this mapping.

3.1 Hierarchical action set

A hierarchical action set is a novel domain representation that takes advantage of the speed of modern non-hierarchical planning algorithms but retains the control and flexibility of collaborative hierarchical planning. It is an AND/OR graph that consists of both abstract and primitive actions and it represents hierarchical decomposition rules that refine a high-level action to a set or multiple alternative sets of lower-level ones.

Though superficially similar to hierarchical action structures in conventional hierarchical planning systems [6], there are significant differences. An abstract action represents an *unordered set* of primitive actions that are potentially useful for achieving a goal rather than a high level *sequence* of actions to perform. Decomposing an abstract action corresponds to building the set of primitive actions into several more focused subsets, rather than yielding a more detailed lower level description.

For example, consider the hierarchical action set for obtaining a shelter shown in Figure 2. Two different decompositions of the root action, rent an apartment and buy a house, subdivide the entire set of six primitive actions to two smaller and qualitatively distinct subsets, {searching classified ads, visiting apartments, placing deposit} and {getting a real estate agent, getting loan pre-approval, visiting open houses}, respectively. If one prefers to rent an apartment, the system retains the primitive actions under rent an apartment in the current domain and excludes the primitive actions related to buying a house. At any point in the plan refinement process, the current set of primitive actions is passed to a non-hierarchical classical planning system to check the existence of a complete plan. This process allows a user to hierarchically construct a plan under his/her control and preference while continually reorganizing the domain theory to reflect to his/her choices. This hierarchical action set also makes it easy to transform a requested planning service into a set of conventional planning problems by generating appropriate inputs, especially domain theories, for traditional planning systems.

For a given domain, there can be multiple hierarchical action sets, each of which has its own goals. These goals are used for selecting appropriate hierarchical action sets according to the given goals. For example, let's assume there are four actions sets, A, B, C, and D, and each has a goal, a, b, c, and d respectively. When a user wants to generate a plan to achieve b and d, the system will automatically select B and D as initial hierarchical action sets. To avoid redundancy and ambiguity in selecting initial action sets, a set of goals associated with an action set cannot be a subset of goals of another action set, but two action sets can have goals in common. Hierarchical action sets satisfy the downward and the upward solution properties, so once an abstract solution is found all other abstract plans can be pruned away and all the descendents of any inconsistent abstract plan can be pruned away as well [15].

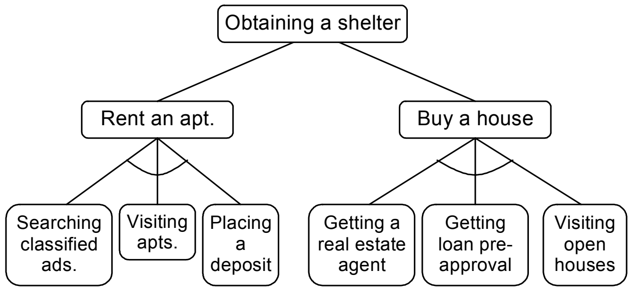

Figure 2: An example of a hierarchical action set for obtaining a shelter

3.2 Current plan set

To keep track of development of a plan while exploring hierarchical action sets and frequently changing the current domain theory, the system uses an array, *the current plan set*, to represent the current plan developed so far. Initially, it consists of the root actions of selected hierarchical action sets, such that the union of the effects of the root actions should contain all of the user's goals. If different combinations of action sets could satisfy these goals, these are considered potential choice points for the user. The system first checks whether or not each combination has a consistent and complete solution by sending the planner a new domain theory that consists of all the primitive actions in each combination in turn. If the planner finds a solution with a combination, the hierarchical action sets in the combination will be considered as valid candidate hierarchical action sets. The dialogue manager then informs the survived candidates to the user and lets him/her choose one among them. The root actions of the selected hierarchical action sets become initial members of the current plan set.

The current plan set is updated whenever each abstract action is replaced by one of its refinements after checking consistency between the newly introduced actions and the existing ones. This process continues until all the actions in the current plan set become primitive.

The above two new domain representations also make it possible for users to select their own strategies rather than the system imposing a specific refinement strategy, such as least commitment or fewest alternative first (FAF) [13]. The system just provides the necessary information to implement these strategies, i.e., existence of alternative refinements or the consistency of each refinement with the rest of the current plan.

3.3 Solution matrix

PICOPA handles planning service requests by systemically solving variants of the current plan set and combining the results in formulating an answer to the request. The solution matrix represents this computation.

For example, in Figure 3, the user wishes to decompose two abstract actions, A_1 and C_2, in parallel (a user can decompose them simultaneously). PICOPA must infer which combinations of refinements are consistent with each other. The solution matrix in Table 1 illustrates how this service request is translated into a series of planning problems that are specializations of the current plan set. To check the consistency of each problem, PICOPA constructs a domain theory for each possible combination of refinement with primitive actions and the primitive descendents of the abstract actions in the combination. For example, the first row corresponds to specializing A_1 into (A_{111}, A_{112}) and C_2 into (C_{211}, C_{212}), remaining B_3 and D_1. After examining each domain theory by passing it to a conventional planning system along with initial and goal state, PICOPA generates a solution matrix that summarizes the results. If the user chooses one among possible candidates, the current plan set is updated to reflect this choice. Table 1 shows the first three of four possible combinations are found to be consistent. If the user chooses the second one - (A_{111}, A_{112}) and (C_{221}, C_{222}) - the current plan set is specialized from {A_1, B_3, C_2, D_1} to {A_{111}, A_{112}, B_3, C_{221}, C_{222}, D_1}. By showing if a part of the plan has a strong connection with its other part (e.g., (A_{121}, A_{122}) is inconsistent with (C_{221}, C_{222}), namely, (A_{121}, A_{122}) should be only with (C_{211}, C_{212})), or if a decision limits the future development of the plan (e.g., selecting (A_{121}, A_{122}) restricts to decomposing C_2 only into (C_{211}, C_{212})), the solution matrix prevent users from bad decisions as well as provides useful information for users to make right decisions.

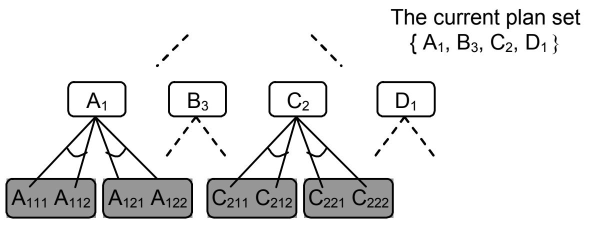

Figure 3: Decompositions of A_1 and C_2

A_1	B_3	C_2	D_1	consistency
(A_{111}, A_{212})	All the primitive descendents of B_3	(C_{211}, C_{212})	All the primitive descendents of D_1	Yes
(A_{111}, A_{112})		(C_{221}, C_{222})		Yes
(A_{121}, A_{122})		(C_{211}, C_{212})		Yes
(A_{121}, A_{122})		(C_{221}, C_{222})		No

Table 1: A solution matrix (A_1 and C_2)

3.4 Planning service requests

There are a wide range of planning service requests that are necessary to support collaborative plan-related interactions between human users and the system. Collectively, planning service requests define an API that cleanly separate the underlying planner from other system components. Several researchers in the collaborative planning community have proposed particular APIs. For example, Allen and Ferguson formalized a set of planning service requests – they call them “problem solving operations”, - necessary for collaborative planning in terms of a well defined API shown in Table 2 [2]. Here we describe how to map these high-level planning service requests into classical planning problems in terms of hierarchical action set, current plan set, and solution matrix. In order to facilitate the evaluation of our approach and easily compare it with other related works, the planning service requests contain all the problem solving operations that Allen and Ferguson marshaled in their recent paper, and several supplementary ones. Up to now we’ve focused on decomposition. Now we’ll generalize this discussion to a number of planning service requests.

Introduce/Refine objective
Modify/correct goal or solution
Evaluate plan
Specify/Extend solution
Create/Compare/Reject option/solution
Undo operation
Cancel plan

Table 2: Collaborative problem solving operations (Allen and Ferguson)

Introduce objective: when the system gets a set of goals from a user to generate a plan, PICOPA first identifies hierarchical action sets relevant to the stated user goals. The union of the goals of selected hierarchical action sets should contain all of the user’s goals, and their root actions become the initial current plan set. In the example in Figure 4, if the given goals are to achieve ***q*** and ***z***, then there exist two possible hierarchical action sets, {A, B} or {A, C}. Note that there may be multiple consistent combinations of hierarchical action sets that are treated as alternatives for the user to select amongst.

Refine plan (objective): After a set of user goals are introduced and the relevant high-level tasks are decided, the user and PICOPA should concretize the tasks until they have a satisfiable plan in an executable level by navigating the hierarchical action sets from part to part according to their current concern and the degree of exigency of the tasks. When the user requests to expand an abstract action in the current plan set, PICOPA shows the possible decompositions of the action and checks if each decomposition is consistent with remaining part of the current plan. This process can be done by generating planning problems that differ only in their domain theories (they share the same initial and goal states) and sending them to the underlying planner. Each domain theory consists of the primitive descendents of the selected alternative and ones of the remaining actions in the current plan set. The user can then select one among the alternatives that passed the consistency check. From the example in Figure 4, let's assume that the current plan set is {A_{11}, A_{12}}. If the user would like to expand A_{12} first, then the system checks the consistency of two alternative refinements of A_{12} with the remaining part in the current plan set (in this case, only with the step A_{11}). PICOPA constructs primitive planning problems for each of these alternatives by specializing the domain theories - {A_{1111}, A_{1112}, A_{1121}, A_{1122}, A_{1211}, A_{1212}} and {A_{1111}, A_{1112}, A_{1121}, A_{1122}, A_{1221}, A_{1222}}. The solution matrix in Table 3 shows the consistency of the two different decompositions of A_{12} with the remaining plan steps. This table illustrates that (A_{1221}, A_{1222}) is inconsistent with A_{11}, meaning that the user can choose only (A_{1211}, A_{1212}) as a decomposition of A_{12}. After the user chooses one of the consistent options, the current plan set is replaced to the newly selected one, i.e., from {A_{11}, A_{12}} to {A_{11}, A_{1211}, A_{1212}}.

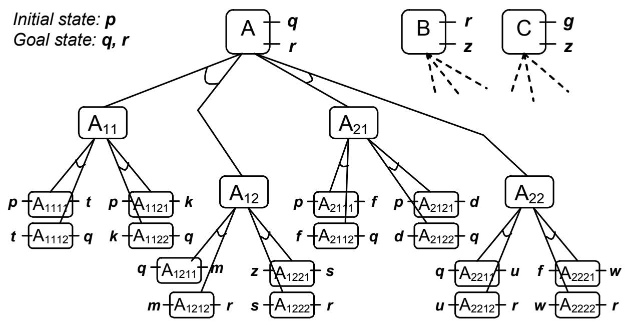

Figure 4: Hierarchical Action Sets. The preconditions appear left of the action, and the effects right.

A_{11}	A_{12}	Consistency
All the primitive descendents of A_{11}	(a_{1211} a_{1212})	Yes
	(a_{1221}, a_{1222})	No

Table 3: Solution matrix for refining A_{12}

Check interaction (Compare/Evaluate options): A key difficulties in collaborative planning is detecting and communicating interdependencies between different portions of the plan. For example, how a user decides to obtain a shelter may constrain his or her options for obtaining transportation due to the limited budget. A novel feature of PICOPA allows users to systematically explore interactions among alternative plan refinements by evaluating the consistency of combinations of possible refinements. For example, the solution matrix in Table 1 evaluates the pair-wise interactions between refinements of A_1 and C_2. The results indicate interactions between these tasks (how the user chooses one as a refinement of A_1 will limit his or her options for refining C_2). Although the system cannot detect exactly why the interaction occurred (the planner only informs "no plan" with the inputs, not why the failure occurred), it provides useful feedback to the user: e.g., "if you refine A_1 into A_{121} and A_{122}, you will constrain your options for refining C_2". Depending on the time available, PICOPA can progressively deepen its interaction checks, initially exploring pair-wise interactions, moving on to 3-way interactions, etc.

Modify/Correct solution (Undo operation): In addition to refinement, the system allows non-chronological backtracking over refinement decisions. When the system cannot develop a plan anymore from the current plan set due to some new constraints or circumstantial changes created in the middle of plan generation or when a user changes his/her previous decision, the related actions in the current plan set are replaced by their parent action for reconsideration by reverting to the previous decision point in the planning process, as well as any siblings that were previously pruned away from the hierarchical action sets are recalled into the sets.

Add or drop goal (Modify/Correct goal): Users can add or drop goals even in the middle of plan generation. When users want to drop goals, it doesn't invalidate the current plan set, though it may now contain unnecessary plan steps. These plan steps are automatically removed through the process of eliminating unnecessary plan steps that will be explained right after. In the example in Figure 4, let's assume that the original goals are [***q*, *r***] and the current plan set is {A_{11}, A_{12}}. After dropping a goal, [***r***], A_{12} becomes unnecessary and is automatically removed from the current plan set.

On the other hand, if a user requests to add a goal, PICOPA finds an appropriate hierarchical action set, the goals (effects) of which contain the goal desired to be added, and then adds the root of the hierarchical action set into the current plan set. In the example in Figure 4, if a user wants to add a goal, [***g***], PICOPA adds the root action (C) to the current plan set. However, this may cause unnecessary processing when the goal can be achieved from the current plan without adding any additional plan steps. For example, let's assume the original goal is [***q***] and the current plan set consists of {A_{11}, A_{12}}. If PICOPA is requested to add a new goal [***r***], it will reorganize the current plan set by adding the root action (B) that has [***r***] as an effect. Since there are duplicated plan steps that achieve [***r***], one of them, A_{12} or B, should be removed from the current plan set. To prevent this problem, whenever the system gets a request to add a goal, it first checks if the system can achieve the added goal from the current plan set without any change. If not, then PICOPA finds an appropriate hierarchical action set that contains the added goal as an effect and adds the root actions into the current plan set.

There are two additional services that are applied regardless of user requests. The following two services are performed whenever a current plan set is updated and a plan step is executed respectively.

Eliminate unnecessary plan steps: In many cases, goals of one part of the plan can be achieved fortuitously by some other portion of the plan, in other words, there may exist redundant plan steps to achieve a same goal. It may also happen that a goal can be achieved by an unexpected event in the middle of plan execution. In some other cases, a hierarchical action set may contain unnecessary actions for a certain circumstance since a hierarchical action set defines a very general way to achieve a set of goals. From the above two cases, it is intuitively realized that PICOPA is required to detect and eliminate redundant or unnecessary plan steps so as not to produce inefficient plans. Let's consider the following two situations in a planning problem to build a house. First, a house generally needs to be air-conditioned and winterized. However, if the user wants to build a house in Hawaii he doesn't need to spend extra money for protection against the cold. Second, if he goes to buy a bath tub, and the store also sells lawn and garden products, he doesn't need to go to a lawn and garden store. When only a part of the selected hierarchical action set is required for the given problem or a goal (sub-goal) is fortuitously achieved, it can makes a part of the current plan set superfluous. PICOPA detects the existence of these plan steps, that have no primitive descendent in the implicitly returned plan and remove them from the current plan set. . For example, in Figure 1, let's assume that the goal state is [***q***] instead of [***q***, ***r***]. Since there is no hierarchical action set that achieves only [***q***] the system inevitably selects {A} as an initial current plan set that is to additionally achieve ***r*** as well as ***q***. Then, the user chooses {A_{11}, A_{12}} as the decomposition of {A}. Since A_{12} is just to achieve [***r***], the implicitly returned plan from the underlying planning system doesn't contain any action that is descent of A_{12}. So, A_{12} is identified to be unnecessary and removed from the current plan set i.e., the current plan set is changed from {A_{11}, A_{12}} to {A_{11}}.

Monitor execution and replan: PICOPA monitors the current world to detect external events. If the system detects any unexpected change in the current world, it examines the validity of the current plan with the change. When the plan is no longer consistent with the changed world state, the system must throw out the current plan and regenerate a new plan from scratch to achieve the user goals from the new situation. Though PICOPA cannot detect what caused the plan failure because it doesn't have any knowledge about internal planning process of the base planning system, it can detect if the current plan is valid with the changed world by checking the interaction between the unexpected event and the remaining part of the plan (unexecuted plan steps).

In addition to the planning service requests listed above, the system may meet a circumstance that needs to reformulate domain theories rather than reconstructing domain theories with related actions.

Specify solution: When the user wants to specify a resource for an action (assigning the first squad to secure the landing zone) or prefers to execute an action before other action (pacifying the crowd before moving the injured boy to hospital) it is necessary to reformulate the definition of the action. For specifying a resource to an action, the target variable in the definition of the action should be replace by the specific instance rather than a planner instantiates the variable with arbitrary one of the possible resources. There is still the difficulty of mapping a variable binding in an abstract level into ones in the primitive level, but this ambiguity can be clarified by additional interactions with the user to find which primitive actions should be assigned with the specific resource.

When users want to post ordering constraints between actions, such a request could be handled by adding auxiliary constraints to the base planner's domain theory. For example, one way to enforce abstract action A to occur before abstract action B is to add "A-complete" effects to each of A's set of primitive actions and "A-complete" preconditions to each of B's actions. We believe that many additional requests can be covered by similar representational manipulations.

4. Evaluation

A primary empirical question for this work is whether the approach, given its emphasis on "in practice efficiency" of planning, will scale to practical collaborative planning domains. In the MRE team training system, a restricted plan scripting language sufficed to drive mixed-initiative interaction [14]. We discovered that IPP [9], a contemporary planning system, could do full plan generation on the same domain theory in less time than it took to find a scripted solution (typically less than 100 ms to generate a plan from scratch). FF [8] can also generate a plan in less time than IPP does, but the totally ordered plan from FF causes unnecessary ordering constraints that limit the flexibility of its execution. Further, our early experiments with JADE [5], a complex collaborative force deployment planning domains, shows that IPP and FF can solve problems in at most a couple of seconds. This is not to say that planning is a solved problem, but for many of the domains considered by collaborative planning, the recent new planning techniques are far more than adequate. With this confidence, we have been applying HICOPA to MRE, expecting improvement of planning capability as well as independency of planning component. We will then contrast the coverage of API and speed against the current existing system. This possibility also encourages us to evaluate the HICOPA by testing it with other collaborative planning domains, such as ones in JADE or O-Plan2. Although due to the limitation in getting these systems we can only test it with their example domains, it seems sufficient to verify our approach by validating that PICOPA can cover API across multiple domains. Finally, we will verify the adaptability and compatibility of the PICOPA with diverse classical planning systems and apply it onto various types of planning domain.

5. Conclusions

This paper presented a collaborative plan assisting agent (PICOPA) allowing human users and planning systems to jointly develop plans by directly using classical AI planners. We proposed a mapping mechanism from planning service requests necessary for collaborative interaction into lower-level calls to a conventional planning system by generating appropriate inputs for the planning system. Therefore, PICOPA can utilize existing classical AI planners for collaborative planning problems according to the characteristics of the application or as new techniques become available rather than developing a custom planner for a specific system or application. This approach can make classical planning researches relevant to the collaborative planning community.

The current system depends on the strong assumption that the base planner can quickly either solve a planning problem or detect that no solution exists. While this may be reasonable for simple domains, it could be unreasonable in general. We can relax this requirement by extending solution matrices of PICOPA to a 3-value logic (consistent, inconsistent, unknown) where "unknown" means that the planner could not terminate within some small pre-specified time limit. We would then have to suitably qualify the results of planning service requests, making responses, in essence, heuristics rather than definitive results. For example, rather than stating that a refinement was inconsistent, PICOPA could state that it may be inconsistent. The user could increase the quality of this feedback by extending the time limit, or the system could periodically fill in solution matrices as time becomes available.

PICOPA is being applied within the context of the Mission Rehearsal Exercise collaborative planning system, a rich virtual training environment that includes human trainees interacting with graphically embodied virtual agents that can communicate with through natural language about tactical decisions. PICOPA shows promise to extend the planning capabilities of the exiting MRE system while at the same time allowing the planning component to be more modular and easier to update with more advanced planning techniques as they become available.

6. Acknowledgements

This paper was developed with funds of the Department of the Army under contract number DAAD 19-99-D-0046. Any opinions, findings and conclusions or recommendations in this paper are those of the authors and do not necessarily reflect the views of the Department of the Army.

7. References

[1] J. Allen, D. Byron, M. Dzikovska, G. Ferguson, L. Galescu, and A. Stent. Towards conversational human-computer interaction. , *AI Magazine*, 22(4): 27--37, 2001.

[2] James Allen and George Ferguson. Human-machine collaborative planning. *Proceedings of the Third International NASA Workshop on Planning and Scheduling for Space*, Houston, TX, October 27-29, 2002.

[3] J. A. Austin. How to do things with words. Harvard University press, Cambridge, Massachusetts, 1962.

[4] Fahiem Bacchus. Results of AIPS-2000 planning competition. 2000. url:http://www.cs.toronto.edu/aips2000/SelfContainedAIPS-2000.ps.

[5] Cox, M. T., and Veloso, M. M.. Controlling for unexpected goals when planning in a mixed-initiative setting. In E. Costa & A. Cardoso (Eds.), *Progress in Artificial Intelligence: Eighth Portuguese Conference on Artificial Intelligence*: 309—318, Berlin: Springer, 1997.

[6] Kutluhan Erol, James Hendler, and Dana S. Nau. UMCP: A sound and complete procedure for hierarchical task-network planning. In AIPS-94, pages 249-254, 1994.

[7] Marti A. Hearst. Trends & Controversies: Mixed-initiative interaction. IEEE Intelligent System, 14(5):14-23, 1999.

[8] Jorg Hoffmann and Bernhard Nebel. The FF Planning System: Fast Plan Generation Through Heuristic Search. Journal of Artificial Research, Volume 14: pages 253-302, 2001.

[9] J. Kohler, B. Nebel, J. Hoffmann, Y. Dimopoulos. Extending Planning Graphs to an ADL subset. ECP-97: pages 273-285, Springer LNAI 1348, 1997.

[10] Hyeok-Soo Kim and Jonathan Gratch. A planner independent approach to human interactive planning. Eighteenth International Conference on Artificial Intelligence, Workshop on Mixed-Initiative Intelligent Systems, Pages 87--93, 2002.

[11] IPC, the results of 2002 International Planning Competition. 2002. url:http://www.dur.ac.uk/d.p.long/competition.html

[12] Drew McDermott. The 1998 Planning system competition, Artificial Intelligence, 21(2):35--55, 2000.

[13] Reiko Tsuneto, Dana Nau, and James Hendler. Plan-refinement strategies and search-space size. *Proceedings of the European Conference on Planning*, pages 414--426, 1997.

[14] J Riclel, S Marsella, J Gratch, R Hill, D Traum, and W Swartout. Toward a new generation of virtual humans for interactive experiences. *IEEE Intelligent Systems* 17(4):32--38, 2002.

[15] Stuart Russell and Peter Norvig. *Artificial Intelligence A Modern Approach*. Prentice-Hall, Upper Saddle River, New Jersey, 1995.

[16] W. Swartout, R. Hill, J. Gratch, W. Johnson, C. Kyriakakis, K. Labore, R. Lindheim, S. Marsella, D. Miraglia, B. Moore, J. Morie, J. Rickel, M. Thiebaux, L. Tuch, R. Whitney, and J. Douglas. *Toward the holedeck: Integrating graphics*, sound, character and story. In Proceedings of 5th International Conference on Autonomous Agents, 2001.

[17] David R. Traum. *Speech acts for dialogue agents*. In Rao, A. and Wooldridge, M., editors, Foundations of Rational Agency. Kluwer, 1999.